Distributed Machine Learning: A Comprehensive Review and Theoretical Approach to Enhancing Data Privacy & Security

Mark Lekina Rorat and Destin Niyomufasha

Dartmouth College

{*mark.l.rorat.24, destin.niyomufasha.26*}*@dartmouth.edu*

February 28, 2022

Introduction

Distributed Machine Learning (DML) is a paradigm that leverages multiple computational resources to train machine learning models. This approach involves partitioning the training process across different machines, processors, or nodes, and employing coordination mechanisms for effective synchronization. In this paper, we will first review common patterns used when designing distributed training systems, in the fashion of Langer *et al.*[4]. Next, we will discuss security and data privacy concerns in the realm of distributed training. Finally, we will propose a new paradigm that combines the concepts of proof of learning and encryption to try and mitigate the aforementioned problems.

Background Knowledge

Model parallelism vs. data parallelism

Two primary strategies underpin distributed training: *model parallelism* and *data parallelism*. Model parallelism involves partitioning a machine-learning model into smaller parts or segments that can be processed in parallel. This strategy is usually straightforward to implement in encoder-decoder systems common in Natural Language Processing (NLP), such as the Transformer model.

The partitioning can be vertical, *i.e.* applying splits between neural network layers, or horizontal, where the layers themselves are partitioned. However, model parallelism presents several challenges. It requires meticulous synchronization of model segments to manage dependencies and ensure seamless data flow between partitions. It also incurs high communication overhead, particularly for data-intensive exchanges between partitions. Furthermore, scalability issues arise as the number of partitions increases, necessitating the restructuring of the entire configuration. For this reason, modern distributed training leans more towards data parallelism. Data parallelism involves splitting the dataset into smaller chunks and distributing these chunks across multiple nodes. Each node then performs computations on its portion of the data, often using a copy of the same model. The key idea behind data parallelism is that the sum of per-parameter gradients computed using subsets of a mini-batch matches the per-parameter gradients for the entire input batch, i.e.,

$$\frac{\partial L(x;w)}{\partial w} \approxeq \frac{\partial L(x^0;w)}{\partial w} + \cdot + \frac{\partial L(x^n;w)}{\partial w}$$

Centralized vs. Decentralized Optimization

Regarding the updating of model parameters based on computed gradients in distributed training, two strategies are common: *centralized optimization* and *decentralized optimization*. Centralized optimization involves architectures where the optimization process is coordinated through a central node or a set of central nodes known as the *parameter server*. The parameter server acts as the optimization instance that updates model parameters and sends the new model to worker nodes. These workers perform backpropagation and send computed gradients to the parameter server. This approach ensures that all worker nodes are synchronized with the same global view of the model parameters. However, in this setup, parameter servers can be a bottleneck and a single point of failure. On the other hand, decentralized optimization refers to the approach where the optimization process is spread across multiple nodes without a central coordinator. Each node directly communicates with one or more other nodes in the network to exchange information (e.g., gradients, parameters) and update its local model. This approach significantly reduces bottlenecks associated with centralized approaches and improves fault tolerance, as there's no single point of failure. However, nodes may have different views of the model at a given time, which can lead to convergence issues. To mitigate this, decentralized systems incorporate the concept of exploration and exploitation phases. During the *exploration phase*, workers iteratively evaluate the loss function using different mini-batches and independently update their local models. In the

exploitation phase, workers share their model updates with the master node, which merges the updates to distill adjustments that work better on average across the investigated portion of the training dataset, and shares the new parameters with the workers.

Synchronous vs Asynchronous Scheduling

Scheduling strategies play a crucial role in the efficient training of deep neural network models. Under this classification, we have synchronous and asynchronous scheduling. *Synchronous scheduling* involves simultaneous updates where all worker nodes must complete tasks before the system proceeds to the next step. This ensures that updates are based on the same model version, leading to consistency in model updates and potentially more stable convergence behaviors. However, this approach can lead to inefficiencies due to resource under-utilization if some nodes are slower than others, as faster nodes must wait for stragglers to complete before proceeding. *Asynchronous scheduling*, on the other hand, allows for independent updates. Nodes update the shared model as soon as they complete their tasks, without waiting for other nodes. This speeds up the overall process and maximizes resource utilization and efficiency by eliminating idle time, making it well-suited for environments with heterogeneous computing resources. However, this can lead to inconsistencies in the model state, as updates may be applied based on stale parameters, potentially causing slower convergence or divergence if not properly managed. *Bounded asynchronous scheduling* is a variant approach that allows for updates from nodes to be applied asynchronously but within a certain bounded delay. This ensures that no update is based on information that is too outdated. It aims to strike a balance between the efficiency of asynchronous methods and the consistency of synchronous methods, potentially reducing idle time while avoiding significant divergence in model states. Implementing and maintaining a bounded delay reduces the time it takes the distributed model to converge, but at the cost of adding a layer of complexity to the system.

Communication Patterns

The communication pattern used for exchanging parameters is a key factor that influences the efficiency and effectiveness of the training process. Several communication patterns are commonly used, each with its advantages and potential drawbacks. We have already discussed the *parameter server* pattern, where one or more nodes hold the global model parameters and workers send gradients and receive updates from the server. The *All-Reduce* algorithm involves the collective

application of an operation on data (gradients) from all nodes. In the context of distributed machine learning, it aggregates gradients from all nodes to compute the global gradient. This pattern is ideal for synchronous data-parallel training, where gradients need global aggregation. The *Ring All-Reduce* algorithm is a variant of the All-Reduce pattern where nodes form a logical ring and exchange data circularly. Nodes in a ring exchange data with a neighbor, perform partial reductions and propagate the updates. This pattern improves efficiency as it reduces communication overhead and evenly distributes load to workers, making it ideal for large-scale training. Finally, *gossip protocols* involve nodes randomly exchanging parameters or gradients with neighboring nodes for decentralized convergence. This pattern is resilient to node failures and offers good scalability. It is ideal for decentralized training where strict order or central coordination is challenging. However, convergence may be slower or less predictable.

Security and Data Privacy in Distributed ML

Having establishing a framework for classifying distributed training methodologies, we will now discuss data privacy and security. Whereas we primarily refer to centralized systems in this section, the concepts still apply to decentralized systems. While offering significant advantages in terms of scalability and computational efficiency, distributed training also raises critical concerns regarding data privacy and security of the distributed model against adversarial attacks. One approach to address these issues is the concept of *Proof of Learning* (PoL).

Proof of Learning

The key idea of PoL, as described by Jia *et al.*[3] is to verify the integrity of the training procedure used to obtain a machine-learning model as a means of ensuring adversarial workers do not poison the central model by sending corrupt gradient updates to the parameter server or other workers. Specifically, it aims to prove that the provided parameters are indeed the result of a specific optimization procedure. In this setup, the prover (worker) provides an encrypted PoL, and the verifier (parameter-server) uses the provided proof to verify the parameters. The PoL generated by the prover is constituted of several components: a set of model-specific information obtained during training, information about specific data points sampled at intervals, and auxiliary information, such as hyperparameters, model architecture, and optimizer choices. The verifier uses this information to determine the authenticity of the weights or gradients received from the prover. PoL

works because an adversary seeking to illegitimately manufacture a proof-of-learning would need to perform at least as much work as is needed for training itself. This provides a concrete mechanism for model ownership resolution and enhancing data security in distributed training.

Problems with Proof of Learning

While PoL offers a promising approach to enhancing security in distributed training, it is not without its challenges. One of the primary assumptions of PoL is that the verifier poses no threat. This may not always hold. For example, in a centralized training scenario, a malicious actor could potentially gain access to the parameter server and (consequently) the gradient updates from each worker. Contrary to the common assumption that only sending gradients (and not data) to the parameter server is safe, it has been shown that both the training data and label set can be approximated from the gradients. Zhu *et al.*[6] introduce an optimization algorithm, *Deep Leakage from Gradients* (DLG), with the learning objective of minimizing the distance between the actual gradients and dummy gradients generated from dummy inputs and labels. This algorithm can accurately approximate the exact training inputs and labels. This poses a significant risk as it could lead to the exposure of sensitive information. Building upon their work, Dang *et al.*[1] propose a method, *Revealing Labels from Gradients* (RLG), to obtain the entire set of labels used in computing a weight update. The existence of these strategies poses a significant risk to data privacy, as it could lead to the exposure of sensitive information.

Proposal: Proof of Learning + Homomorphic Encryption

To mitigate this problem, we propose making adjustments to the PoL generation algorithm. In our proposed approach, we aim to mitigate the problem by making adjustments to the Proof of Learning (PoL) generation algorithm. Our solution involves modifying the Proof of Learning (PoL) generation algorithm to include *homomorphic encryption* of weights before outsourcing. Homomorphic functions are functions that satisfy the equation $f(x \times y) = f(x) \times f(y)$. Therefore, homomorphic encryption can be defined as a form of encryption that allows computations to be carried out on ciphertext, thus generating an encrypted result that, when decrypted, matches the result of operations performed on the plaintext. Homomorphic encryption allows the recipient of encrypted data to encrypt the result of some computation without knowing the inputs. This property enables secure outsourcing of computation and is particularly relevant in the case where

workers need to guarantee the privacy of sensitive data. This method has been applied to neural networks with some success, as demonstrated by Gilad-Bachrach *et al.*[2].

Implementation and Proposed Experiments

One key challenge that we anticipate is the difficulty of finding homomorphic equivalents for the non-linear functions activations. Whereas homomorphic encryption supports simple operations such as additions and multiplications, it is much harder to encode non-linear activation functions such as the sigmoid and hyperbolic tangent. Fortunately, Lee *et al.*[5] show that we can replace non-linear activation functions with state-of-the-art approximation methods to evaluate these non-arithmetic functions. This approach opens up the possibility of applying *Fully Homomorphic Encryption* (FHE) to advanced deep privacy-preserving machine learning models. The novelty of this approach lies in its combination of existing strategies that independently tackle security against attacks (PoL) and privacy (gradient encryption). In summary, using an encryption scheme before broadcasting computed weights and implementing PoL verification at the parameter server prevents a compromised parameter server from accessing private data from the model updates, while at the same time guards the shared model from poisoning attempts by malicious worker nodes. Given the time and resource constraints, we could not implement a fully distributed training system to test out our proposal, thus our proposed approach is entirely theoretical. We hypothesize that our approach would work, given the success of both of the constituent strategies that we have brought together. We are, however, wary of potential incompatibility between the proposed homomorphic encryption scheme and a potential decline in the probability of the PoL verifier accepting valid proofs due to noise added during encryption.

Conclusion

This paper has explored the complexities and challenges of distributed training. We have discussed the trade-offs between model parallelism and data parallelism, centralized and decentralized optimization, and synchronous and asynchronous scheduling. We have also examined the impact of communication patterns on the efficiency and effectiveness of the training process. In the realm of security and data privacy, we have highlighted the risks associated with adversarial attacks and data leakage. To address these concerns, we proposed a novel approach that integrates the concepts of Proof of Learning and homomorphic encryption to ensure the integrity of the training procedure

and protect sensitive data. However, our proposal is not without its challenges, chief of which is that the approach is untested. Implementing homomorphic encryption in non-linear functions and ensuring the compatibility of our approach with the Proof of Learning verification process are significant hurdles to overcome. Despite these challenges, we believe that our approach offers a promising direction for enhancing data security and privacy in distributed ML.While our proposed approach remains theoretical at this stage, we are optimistic about its potential to contribute to future advancements in secure and privacy-preserving distributed machine learning.

References

[1] Trung Dang, Om Thakkar, Swaroop Ramaswamy, Rajiv Mathews, Peter Chin, and Françoise Beaufays. Revealing and protecting labels in distributed training. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 1727–1738. Curran Associates, Inc., 2021.

[2] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016.

[3] Hengrui Jia, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1039–1056. IEEE, 2021.

[4] Matthias Langer, Zhen He, Wenny Rahayu, and Yanbo Xue. Distributed training of deep learning models: A taxonomic perspective. *IEEE Transactions on Parallel and Distributed Systems*, 31(12):2802–2818, 2020.

[5] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.

[6] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.