Byzantine Attacks in Distributed Training

Andreea Zaharia
Department of Computer Science and Technology
University of Cambridge
az396@cam.ac.uk

Abstract

This project investigates Byzantine tolerance in distributed data-parallel training, by evaluating Byzantine Tolerant All-Reduce Stochastic Gradient Descent [Gorbunov et al., 2021] on a new model and learning task, while also introducing two new gradient Byzantine attacks. We find the conclusions from the paper are supported by our experiments and the defense successfully combats the two new attacks.

1 Introduction

The amount of data and the size of the architectures required for many machine learning workloads are continuously increasing. This is a desirable trend, as larger datasets lead to improved accuracy, across all tasks that benefit from pre-training the model [Goyal et al., 2017], but it also means that more resources are needed for productionalising state-of-the-art machine learning models. Thus, to support this growth, distributed training is becoming ever more important.

Distributed training, in line with any distributed system, can be subject to attacks and failures that threaten its safety. Among these, there is the category of Byzantine attacks, in which malicious workers might derail the training, by replying with unsuitable gradients or aggregating improperly. This project is an investigation into Byzantine attacks and Byzantine tolerance in the distributed training of deep learning models. Its objectives are to evaluate the Byzantine-Tolerant All-Reduce Stochastic Gradient Descent (BTARD-SGD) algorithm, presented by Gorbunov et al. [2021], on a different convolutional model and in the presence of additional types of Byzantine attacks, in a controlled simulation environment. This report sets to highlight the potential impact of a minority of Byzantine workers in data-parallel distributed training, while also opening up a discussion about the properties of Byzantine-tolerant variations of stochastic gradient descent.

The repository containing the implementation, documentation of planning and decisions, the setup guide, and the source code for this report is at https://github.com/andreea-zaharia/btard-l46-project.

2 Background

2.1 Byzantine conditions in distributed systems

In a real-life distributed setup, worker nodes are not perfectly reliable. Nodes might suffer from fail-stop failures, in which they would cease computation on their replica and stop replying, or from Byzantine failures and attacks, in which they could have any unexpected behaviour, including replying with wrong information in the correct format. Byzantine failures can be caused by malfunctioning due to hardware failure or software errors, while Byzantine attacks are executed with malicious intentions.

Byzantine failures and attacks have always troubled the world of distributed systems, being a central topic for the research in distributed consensus ever since the publication of the Castro–Liskov algorithm [Castro et al., 1999]. As the field of machine learning systems builds on distributed systems, it also inherits its problems. Thus, distributed learning can also suffer from Byzantine attacks.

2.2 Data parallelism in distributed training

There are many strategies by which training can be distributed across multiple devices, specifically GPUs, and they can be grouped under two categories: data-parallel and model-parallel. In data-parallel approaches, the model is typically replicated on each device and they individually train on disjoint subsets of the dataset. In model-parallel approaches, the model is split into subsets of consecutive layers and each subset is placed on a different device to be trained separately, in an effort to maximise the overlap between communication (i.e., of gradients across GPUs) and computation (i.e., the backward pass). Most model-parallel systems are in fact hybrid-parallel, relying on data parallelism between workers at the same stage [Rajbhandari et al., 2020, Narayanan et al., 2021].

2.3 Mini-batch stochastic gradient descent and all reduce

Stochastic Gradient Descent (SGD), used to update the weights of a neural network during training, is often adapted in deep learning to use mini-batches. Mini-batched SGD (often referred to simply as SGD) uses $|\mathcal{B}|$ uniformly sampled examples and proves more stable than approximate gradient methods. The update performed during training in mini-batch stochastic gradient descent is:

$$w^{(t+1)} = w^{(t)} - \frac{\eta_t}{|\mathcal{B}|} \cdot \sum_{x \in \mathcal{B}^{(t)}} \underbrace{\nabla f\left(x, w^{(t)}\right)}_{\text{Example-level gradient.}} \tag{1}$$

where w are the weights, t is the time step, $\mathcal{B}$ is a mini-batch sampled from the labelled training set, η is the learning rate, and f is the loss function.

In distributed training, gradients are computed separately by each replica of the model and then aggregated, or reduced, to a single *global* result, which needs to be distributed to all processing units. This is achieved with the all-reduce pattern in most synchronous data-parallel algorithms and frameworks. All-reduce solutions are synchronous, although each replica operates at its own pace on a different data slice because there is a synchronisation barrier enforcing that all workers have the correct (i.e., same) gradient before progressing to the next step of training.

Communication across nodes can be either centralised or decentralised. In the centralised approach, the replicas compute their gradients independently and submit them to a designated server, called the parameter server, which performs the *reduce* operation and distributes the result. Some setups might have a few parameter servers, rather than just one.

Butterfly All-Reduce is an alternative all-reduce algorithm, proposing a topology that eliminates instances of one-to-all communication from any node while improving performance up to nine times for vector lengths in the order of 10^8 [Li et al., 2017] compared to classical all-reduce.

2.4 Byzantine defenses in distributed learning

The problem can be further split depending on whether the distributed learning is done in a setup with one parameter server (or a few of them) or whether it is done in a decentralised setup.

Research before Gorbunov et al. [2021] has focused on the parameter server case, which inherently has trust assumptions in the servers. One way to approach Byzantine-tolerant stochastic gradient descent in a parameter server setup is replacing the averaging method typically used in SGD with a different aggregation rule. One notable such solution is Krum, introduced by Blanchard et al. [2017]. This has been later proven in El El Mhamdi et al. [2018] that it might yield ineffectual models, despite converging, and in Baruch et al. [2019] that it can still fail under an attack that generated a set of parameters that differ from the mean of each parameter by only a small amount. The Trimmed Mean category of defenses [Xie et al., 2018, Yin et al., 2018] is similar to Krum in that it proposes changing the aggregation rule, while also being detectable by convergence attacks in which Byzantine workers submit parameters that are not very far away from the mean. Prior art without such limitations is restricted to Karimireddy et al. [2021], which introduced the Centered Clip aggregation rule that Gorbunov et al. [2021] later generalise, with modifications, to a decentralised setup.

Work on Byzantine-tolerant decentralised distributed training, such as setups leveraging ring all-reduce [Sergeev and Del Balso, 2018] and butterfly all-reduce, is not as advanced, especially when considering deploying to real-world contexts. The solutions proposed by prior art have high computational complexity, preventing them from scaling to achieve the computational power needed for modern deep learning tasks. Some examples are the algorithm introduced by Gupta et al. [2021], seeking exact consensus by communicating full vectors to other agents in the network at each step, and Byzantine-Resilient Distributed Coordinate Descent (ByRDiE) [Yang and Bajwa, 2019], which relies on full gradient computation.

Byzantine faults and attacks are a threat to federated learning as well. However, both Gorbunov et al. [2021] and this project are restricted in scope to the distributed training scenario. The approach in Gorbunov et al. [2021] does not generalise to federated learning because of the assumption that the dataset is *public*, such that all workers can access all parts of the data; this assumption is necessary for verifying the computation of a suspected attacker before banning them. Prior work has proposed solutions for federated learning with a parameter server in Byzantine conditions, in cases of both independent and identically distributed (IID) data [Chen et al., 2017, Blanchard et al., 2017] and non-IID data [Prakash and Avestimehr, 2020].

3 The threat model of Byzantine attacks

This project is concerned with data-parallel distributed training, using mini-batch stochastic gradient descent and a decentralised all-reduce pattern. Henceforth, all analyses and experiments are written with such a system in mind.

By the threat model for Byzantine attacks, the gradient g at rank i and time step t is the following:

$$g_i^t = \begin{cases} X, & \text{if } i^{\text{th}} \text{ worker is Byzantine,} \\ \nabla f_i\left(x, w^{(t)}\right), & \text{otherwise.} \end{cases} \tag{2}$$

where w are the weights, x is training data, f is the loss function, and X can take an arbitrary value.

On the one hand, the arbitrary value X can be an erroneous result of a genuine attempt at computing the gradient $\nabla F_i\left(x^t\right)$, when the Byzantine failure resulted from malfunctioning hardware, software bugs, or stalled processes on node i. On the other hand, X can be carefully chosen with malicious intentions, to corrupt the model learnt or to prevent convergence. The attacks can be executed at every step or at strategic points in the training—for example, closer to the start or closer to convergence—and the attackers can operate independently or together.

As there is currently no universal solution for Byzantine tolerance in SGD, in all existing solutions the threat model is adjusted with additional assumptions restricting X as needed for the convergence proof of the algorithm to hold. Many other attack strategies become available as we assume the malicious workers possess more knowledge about their peers. A consequence of assuming omniscience is that not only could Byzantine attackers modify gradients, but they could also collude with the other malicious workers and run more complex time-coupled attack strategies. Some attacks only require the Byzantine workers to have access to *communicate* with the other malicious workers, without being truly omniscient [Baruch et al., 2019, Xie et al., 2020].

4 Byzantine-Tolerant All-Reduce Stochastic Gradient Descent

Gorbunov et al. [2021] introduce Byzantine-Tolerant All-Reduce Stochastic Gradient Descent (BTARD-SGD), a protocol for decentralised distributed data-parallel training. BTARD-SGD is designed around communication efficiency, having minimal overhead that does not depend on the number of parameters.

4.1 Assumptions

Several assumptions are required for the inner working of the algorithm and for its convergence proof to hold, as acknowledged in Gorbunov et al. [2021]. We can summarise these as follows:

1. All peers can access the entire training data.

2. A worker cannot interfere with the gradient computation of another worker.
3. Any worker can be an attacker. There are no node-level trust assumptions.

For this project, we extend this list of assumptions with those that are consequences of choices made for the experiment's design. These are discussed in Section 5.1.

Assuming that Byzantine attackers do not accuse their peers when it is their turn to validate, we obtain an upper bound of $\lceil \frac{n}{2} \rceil - 1$ simultaneous Byzantine workers, where n is the number of workers.

4.2 Butterfly Clip

Butterfly Clip, devised and coined by Gorbunov et al. [2021], is a procedure combining Butterfly All-Reduce with the Centered Clip robust aggregation rule. Butterfly All-Reduce has been chosen as a building block for BTARD-SGD because workers aggregate disjoint parts of the gradient vector; this is beneficial and desirable as it isolates incorrect contributions. Iterative centered clipping (Centered Clip) is a robust aggregator for approximating the ideal update, up to some error, with only $\mathcal{O}(n)$ computation, where n is the number of peers [Karimireddy et al., 2021].

While combining these two techniques sets the basis for a robust aggregator that can detect attacks at a certain step, further refinements are needed to establish a defense against attacks that are executed over many iterations. For this, BTARD-SGD develops the validation and accusation procedures, that can detect incorrect computation in a decentralised fashion.

4.3 Gradient validation and peer accusation

At each step, workers compute a checksum of their computed gradients and broadcast it to announce the completion of their computation and to commit to their result. The algorithm designates m peers to validate the computation at some arbitrary interval of steps, using a random selection process such that the malicious workers cannot predict when they will be validated next. The validator must then recalculate the gradients of a random selection of other peers and accuse them if they calculated a different gradient for them.

Upon an accusation, all workers will recompute the gradient of the accused to confirm or infirm that the suspect is Byzantine. If consensus among workers is that the suspect has computed a correct gradient, the accuser will be banned instead since a false accusation is evidence of malice.

This procedure guarantees that gradients making it to the aggregation stage are correct. Lastly, peers also need to validate the result of their aggregation when they receive it. Due to computation overhead, gathering all gradients to confirm the result of Centered Clip is not an option. The alternative test devised by Gorbunov et al. [2021] proposes sampling, with the same global seed, a random vector z in the space of model gradients and computing the inner product:

$$s_i = \left\langle z, (g_i - x) \min\left\{1, \frac{\tau}{\|g_i - x\|}\right\}\right\rangle \tag{3}$$

where τ is the constant of Centered Clip that dictates how aggressive the outlier removal shall be.

The aggregation succeeds if the sum of the inner products from all workers is zero, i.e. $\sum_{i=1}^{n} s_i = 0$.

A full iteration of BTARD-SGD combining these procedures is illustrated in Figure 1.

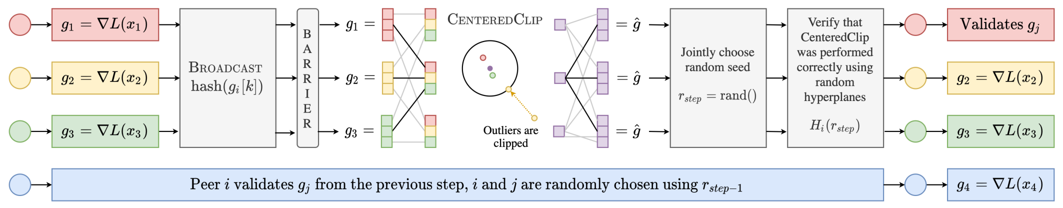

Figure 1: A schematic summary of one step of Byzantine-Tolerant All-Reduce, where g_i are gradients, r_{step} is the random seed, and $H_i(r_{\text{step}})$ computes the random vector z. Image source: https://github.com/yandex-research/btard.

5 Experiment

In this project we set out to investigate experimentally the following questions:

- How does BTARD-SGD perform on a new learning task, with much fewer parameters?
- How does an attack that only perturbs the parameters' values by a small amount perform?
- What is the difference in impact between an aggressive attack and a mild attack?

5.1 Experiment design

In this project we build on the implementation published with Gorbunov et al. [2021], which can be found at https://github.com/yandex-research/btard, reconfiguring parts of it for simulation on a single physical device. We implement new experiments with BTARD-SGD on a new small convolutional model for MNIST classification, diversifying the model sizes used in its evaluation. This project also contributes two new attacks, *Label Shuffle* and *Noise Addition*, which we evaluate alongside three attacks described in the original BTARD-SGD paper.

Simulating a data-parallel training setup with six workers, we run the following experiments:

A The *aggressive* attack: Two Byzantine workers attacking every step, starting at step 100.

B The *mild* attack: One Byzantine worker attacking every 5^{th} step, starting at step 100.

Starting at step 100, in this case, means that the malicious intervention of the Byzantines occurs closer to convergence, even though the simulation runs until 900 steps per experiment. We evaluate BTARD with constant $\tau = 5$, meaning moderate clipping. Each experiment consists of iterating over three different randomly drawn seeds and repeating all runs three times.

The baseline for comparison will be the same model, trained by the same number of workers in the same configuration, with all workers being honest and non-faulty.

5.2 Model and learning task

The model chosen is a simple neural network, with two convolutional layers, each followed by a pooling layer to reduce the parameter space. Its full architecture is depicted in Figure 2. The loss function is cross-entropy and the optimiser is SGD with Nesterov momentum. The learning rate is scheduled with cosine annealing, with a base learning rate of 0.05. The batch size is 256 per worker and the number of epochs per worker is restricted to four, both selected to lower the total running time of the experiment.

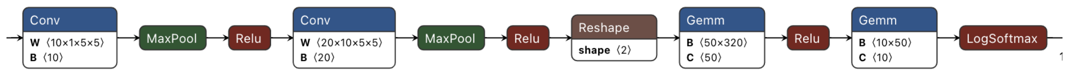

Figure 2: Visualisation of the convolutional neural network architecture implemented.

The model is trained for handwritten digit classification, on the MNIST dataset of handwritten digits [LeCun et al., 1998a,b]. With 60,000 data points in the training set and 10,000 in the test set, MNIST is a relatively small dataset, allowing for reasonable development iteration times and suitable for simulation in a single GPU environment.

5.3 Restrictions for workers' behaviour

To best control the simulation and obtain clear results, we make the following system design choices that translate to assumptions about the possible behaviour of the workers:

1. All malicious workers start attacking at the same time.
2. After a worker is banned, it starts sending correct gradients as if it became a normal peer.
3. All Byzantine attackers perform the same type of attack.
4. Byzantine nodes may only *collude* at the beginning and cannot undergo additional malevolent communication later.

The alternative to assumption (2) would have been dropping said worker completely, but that would have posed difficulties in evaluating convergence of the training as the number of replicas would have changed significantly. Additionally, the training data assigned to that node would have to be redistributed or abandoned.

While these restrictions limit how many of the possible attack schemes can be represented through configurations of this implementation, they enable solid simulations both for evaluating the convergence impact of BTARD-SGD and for comparing the different attacks.

5.4 Attacks

The types of workers participating in the experiments ran in this project are the following:

Normal: workers compute the cross-entropy loss function for the targets and run the backward pass.

Amplified Sign Flip: attackers send the opposite of the true gradient, amplified by 1000 to weigh more in the aggregated gradient if clipping is not used.

Amplified Constant Direction Response: attackers all send large vectors, amplified by 1000, in the same arbitrary direction. The random seed is drawn and passed to all malicious workers just at the start of the training.

Label Flip: attackers compute the gradient based on the cross-entropy loss function for flipped labels. Each label value is replaced with $(9 - \text{label})$.

Label Shuffle: attackers compute the gradients correctly but pass to the cross-entropy loss function an arbitrary permutation of the targets.

Noise Addition: attackers compute the gradients but add random noise to every other parameter p drawn from a Gaussian distribution with $\mu = p$ and $\sigma = 100$.

Amplified Sign Flip, *Amplified Constant Direction Response*, and *Label Flip* are used for evaluating BTARD in Gorbunov et al. [2021], in their experiments on both ResNet and ALBERT. *Label Shuffle* and *Noise Addition* are new attack types developed for this project, which do not appear in the original paper. *Noise Addition* has been introduced to illustrate how a small variation of the parameters impacts the loss and accuracy. This complements attacks such as the *Amplified Constant Direction Response*, which generates parameters that are far away from the mean.

5.5 System model and setup

The code is implemented in Python and all dependencies are in the open-source domain. The machine learning framework used is PyTorch, with Netron for the visualisation of the model (Figure 2) and TensorBoard for visualisation of the results. The NVIDIA CUDA Toolkit is necessary to use Nvidia GPUs for general-purpose computation. The project was run in Google Colab, as it is a simple and effective way to access a CUDA-compatible GPU for small projects. More information about the setup, software dependencies, and decision-making related to the software engineering part of this project is included in the project repository: https://github.com/andreea-zaharia/btard-l46-project.

The experiment as implemented requires a single GPU, on which it simulates multiple virtual workers as independently spawned processes, using `torch.multiprocessing`. Each process instantiates a different type of worker that remains the same throughout the training, thus a normal worker cannot suddenly become Byzantine beyond the pre-determined start point of the attack. This is a limitation of the implementation, as it does not account for Byzantine failures due to malfunctioning nodes, but it helps control the simulation and facilitate interpretation of the results. Another limitation introduced by the simulation environment is that the number of workers that can be simulated cannot exceed 6. Thus, to respect the condition that $2f + 1 \leq n$, the number of Byzantine workers cannot exceed 2.

6 Results

As we seek to evaluate how the defense behaves on the small model classifying MNIST, a reference point for our analysis is the evaluation presented in Gorbunov et al. [2021], where they run experiments on a ResNet architecture for classification on CIFAR-10 and a transformer model, ALBERT.

In our experiments on the new model, we observe a similar impact on convergence, which supports the merits of the algorithm. The speed of convergence is not worsened but the final accuracy is slightly lower. We will use this drop in accuracy to rank the effectiveness of the individual attacks.

We observe in our experiments a slightly different ranking in the impact of the different attacks presented in the paper, with label flipping being more impactful than sign flipping in both the aggressive and the mild attack schemes. We also find that the newly added attacks have a noticeable impact compared to the baseline, but cause smaller changes in loss and accuracy than the other attacks. This confirms our expectations, as we intended to devise a type of attack performing small magnitude changes. We observe that the algorithm performs well in the presence of these new attacks, too. However, we note a significant negative impact on test accuracy with the aggressive Amplified Constant Direction attack.

The entire experiment took around 1.5 hours on 1x Tesla P100 16GB cloud runtime setup, through Google Colab, with the configuration and the task described in Section 5.

6.1 The aggressive attack: two Byzantine workers attacking every step

Figures 3 and 4 show the loss and accuracy, respectively, for all attacks on this model with the configuration described in Section 5.1. We note that all attacks have a noticeable impact on loss and accuracy as the two Byzantines start their attack in tandem, but BTARD bans all of them quickly and eventually converges to good accuracy on the MNIST task.

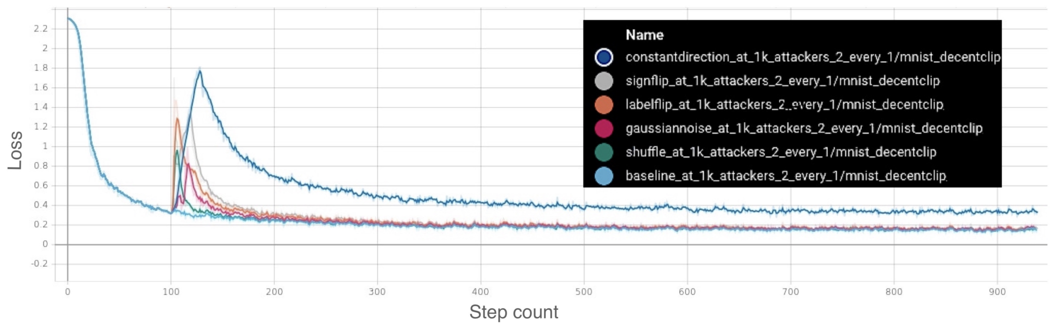

Figure 3: Learning curves of the loss function for BTARD-SGD with moderate clipping. All attacks. Training with 6 workers, including two Byzantines starting to attack at step 100.

On this model, the *Amplified Constant Direction Response* attack has the most significant impact. The training still converges, but at the end of the training, at the 900^{th} step, the test accuracy remains over 0.01 below all the others. For the MNIST classification task, this is a notable difference. The other attacks have only a minor negative impact on convergence, which can be reasonably discussed as negligible.

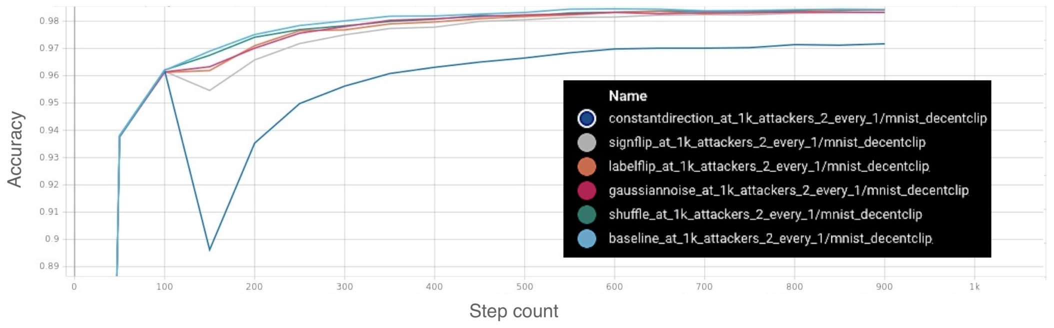

Figure 4: Test accuracy for BTARD-SGD with moderate clipping. All attacks. Training with 6 workers, including two Byzantines starting to attack at step 100.

6.2 The mild attack: one Byzantine worker attacking every five steps

In the mild attack, the attack strategies do not stray far from the baseline, being nearly indistinguishable from noise other than at the attack start step and soon after. For both loss (Figure 5) and accuracy (Figure 6) we note the higher magnitude impact of the *Amplified Constant Direction Response* attack, but this time its course is corrected towards convergence, preventing it from yielding a drop in test accuracy at the end of training.

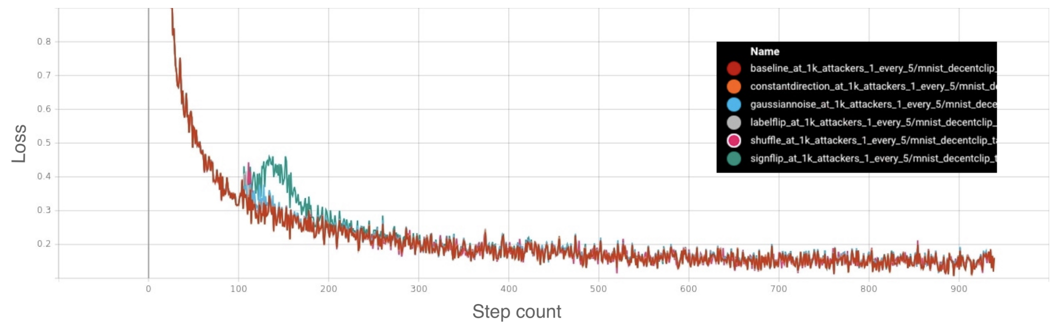

Figure 5: Learning curves of the loss function for BTARD with moderate clipping, zoomed in. All attacks. Training with 6 workers, including a Byzantine attacking every fifth step, starting at step 100.

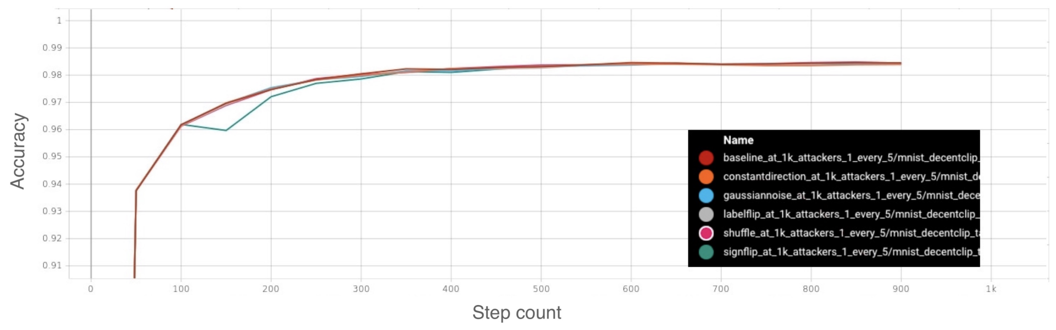

Figure 6: Test accuracy for BTARD-SGD with moderate clipping, zoomed in. All attacks. Training with 6 workers, including a Byzantine attacking every fifth step, starting at step 100.

7 Directions for future work

Directions for future work could cover an investigation into how violating individual assumptions alters the convergence in practice, such as by decoupling the start times across attackers and combining attackers of different types in the same experiment. A related project could also examine the performance of Butterfly Clip in the presence of aggregation attacks, which were set outside the scope of this project. Extending to an actual multi-GPU setup would allow larger-scale experiments for a greater variety of attack schemes with the same, or more, attacker types.

8 Conclusion

This project investigated Byzantine attacks and defenses in data-parallel distributed training, with public datasets. We evaluate Byzantine-Tolerant All-Reduce Stochastic Gradient Descent, proposed by Gorbunov et al. [2021], on a new model and learning task, introducing new gradient attacks employed by Byzantine workers, in a controlled simulation environment. The conclusions we draw are largely in agreement with the experiments in the original paper, with minor differences that do not affect the correctness or scalability of the algorithm. Lastly, BTARD-SGD succeeds in banning the attackers employing the new attack strategies introduced, *Label Shuffle* and *Noise Addition*.

References

Moran Baruch, Gilad Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. *arXiv preprint arXiv:1902.06156*, 2019.

Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 118–128, 2017.

Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

Yudong Chen, Lili Su, and Jiaming Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–25, 2017.

Mahdi El El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in byzantium. *arXiv e-prints*, pages arXiv–1802, 2018.

Eduard Gorbunov, Alexander Borzunov, Michael Diskin, and Max Ryabinin. Secure distributed training at scale. *arXiv preprint arXiv:2106.11257*, 2021.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Nirupam Gupta, Thinh T Doan, and Nitin H Vaidya. Byzantine fault-tolerance in decentralized optimization under 2f-redundancy. In *2021 American Control Conference (ACC)*, pages 3632–3637. IEEE, 2021.

Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. Learning from history for byzantine robust optimization. In *International Conference on Machine Learning*, pages 5311–5319. PMLR, 2021.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a.

Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST database of handwritten digits, May 1998b. URL http://yann.lecun.com/exdb/mnist/. retrieved on 23 Dec 2021.

Zhenyu Li, James Davis, and Stephen Jarvis. An efficient task-based all-reduce for machine learning applications. In *Proceedings of the Machine Learning on HPC Environments*, pages 1–8. 2017.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters. *arXiv preprint arXiv:2104.04473*, 2021.

Saurav Prakash and Amir Salman Avestimehr. Mitigating byzantine attacks in federated learning. *arXiv preprint arXiv:2010.07541*, 2020.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116*, 2018.

Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. In *Uncertainty in Artificial Intelligence*, pages 261–270. PMLR, 2020.

Zhixiong Yang and Waheed U Bajwa. Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning. *IEEE Transactions on Signal and Information Processing over Networks*, 5 (4):611–627, 2019.

Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR, 2018.