Learning to Learn Gradient Aggregation by Gradient Descent

Jinlong Ji[1] , **Xuhui Chen**[1,2] , **Qianlong Wang**[1] , **Lixing Yu**[1] and **Pan Li**[1*]
[1]Case Western Reserve University
[2]Kent State University
{jxj405, qxw204, lxy257, pxl288}@case.edu, xchen2@kent.edu

Abstract

In the big data era, distributed machine learning emerges as an important learning paradigm to mine large volumes of data by taking advantage of distributed computing resources. In this work, motivated by *learning to learn*, we propose a meta-learning approach to coordinate the learning process in the master-slave type of distributed systems. Specifically, we utilize a recurrent neural network (RNN) in the parameter server (the master) to learn to aggregate the gradients from the workers (the slaves). We design a coordinatewise preprocessing and postprocessing method to make the neural network based aggregator more robust. Besides, to address the fault tolerance, especially the Byzantine attack, in distributed machine learning systems, we propose an RNN aggregator with additional loss information (ARNN) to improve the system resilience. We conduct extensive experiments to demonstrate the effectiveness of the RNN aggregator, and also show that it can be easily generalized and achieve remarkable performance when transferred to other distributed systems. Moreover, under majoritarian Byzantine attacks, the ARNN aggregator outperforms the Krum, the state-of-art fault tolerance aggregation method, by 43.14%. In addition, our RNN aggregator enables the server to aggregate gradients from variant local models, which significantly improve the scalability of distributed learning.

1 Introduction

With the onset of the big data era, governments, companies and research institutions for the first time have the opportunity to gain deep insight into their problems and develop better solutions by leveraging massive available data [Dean *et al.*, 2012; Chen *et al.*, 2017]. However, developing machine learning algorithms to analyze a huge amount of data usually requires intensive computing resources, which may not be provided by any single user. Moreover, the massive data are typically generated by multiple parties and stored in a geographically distributed manner. Such scenarios with distributed computing power and datasets have spurred the study of distributed machine learning techniques, such as multi-agent optimization [Nedic and Ozdaglar, 2009], multi-party learning [Hamm *et al.*, 2016], and federated learning [Konečnỳ *et al.*, 2016].

The most typical distributed machine learning paradigm is the master-slave pattern, and has been adopted in many industrial-level applications [Abadi *et al.*, 2016; Chen *et al.*, 2018b]. Particularly, each worker (the slave) in the system is a computing node, which has access to its own share of the data. All workers learn parallelly and only send their local model information (e.g., gradients of the model) to a parameter server, which can significant reduce the communication cost, while the parameter server (the master) aggregates all the received information with hand-designed methods so as to update the global model. This training procedure is applicable to a rich diversity of machine learning algorithms, such as linear regression/classification, support vector machine and deep neural network.

As mentioned above, the aggregation method plays a vital role in the distributed learning process. So far, the aggregation methods are linear combination like averaging [Polyak and Juditsky, 1992] and its variants [Zhang *et al.*, 2015; Lian *et al.*, 2015]. However, it still requires study on how to make the aggregators more robust in distributed learning. Particularly, distributed systems are vulnerable to computing error from the workers, especially when adversarial attackers compromise one or more computing nodes to launch Byzantine attacks. In such scenarios, the distributed learning process may not converge and the system may crash [Blanchard *et al.*, 2017]. In addition, traditionally, in master-slave type of distributed machine learning systems, the linear combination aggregator in the parameter server can only work when all workers hold the same local model. However, in many distributed systems, the workers may vary in computing power and memory, and hence can only afford models of different complexities. Therefore, the current aggregators impede the large-scale implementation of distributed machine learning.

In this paper, we take the *learning to learn* approach to coordinate the distributed learning among the workers. Particularly, *Learning to learn* is a meta-learning strategy [Thrun and Pratt, 1998], where the abstract knowledge extracted from the learning (or optimization) process can enable fast and effi-

*Contact Author

cient learning. We develop a deep meta-learning aggregation method for distributed machine learning. Specifically, we model the aggregation process in distributed learning as a machine learning task. Then we use a recurrent neural network (RNN) to aggregate the gradients collected from the workers. To learn this neural network based aggregator, we choose to optimize it along the horizontal direction (i.e., the number of steps) of the distributed learning process. To make the aggregator robust, we design a coordinatewise preprocessing and postprocessing method. Once the aggregator is well-trained, it makes decisions on how to aggregate the scattered local gradients by the information learned from its previous optimization process. Moreover, in order to address the Byzantine attacks, we improve the RNN aggregator with additional loss information to make the system robust and stable.

The contributions of our work are summarized as follows:

- To the best of our knowledge, this is the first study proposing a *learning to learn* approach to coordinate a distributed learning process. Instead of adopting a predefined deterministic aggregation rule, we model the aggregating process as a learning problem and utilize a recurrent neural network to optimize it. In addition, we design a preprocessing and postprocessing method to enhance the stability of the optimization process. We also investigate the robustness and stability of distributed learning, and propose an improved RNN aggregator with additional information loss to address the fault tolerance issues, especially Byzantine attacks.
- The experiment results demonstrate that the proposed RNN aggregators outperforms the classical averaging aggregator by a large margin on both the MNIST and the CIFAR-10 datasets. Our experiments also demonstrate that the proposed aggregators can be easily generalized and achieves remarkable performance when transferred to other distributed systems with different learning models and different datasets, respectively. In addition, compared to the state-of-the-art fault tolerance algorithm Krum, the proposed ARNN aggregator shows its competitive performance and outperforms Krum in the majoritarian Byzantine attacks.
- The empirical results prove that the proposed scheme enables the aggregator to utilize abstract information across variant local learning models, which significantly improve the scalability of the traditional distributed learning system.

The rest of the paper is organized as follows. In the next section, we briefly introduce the most related works. Then we detail the design of recurrent neural network based aggregator and its variant, RNN aggregator with additional loss information. After that, we evaluate the performance of the proposed RNN aggregator through extensive experiments. Finally, we conclude the paper.

2 Related Works

2.1 Learning to Learn

The idea of *learning to learn* was inspired by psychology studies [Ward, 1937], where researchers discovered that the human beings are born to be able to learn new skills via previous learning experiences. The idea of *learning to learn* has been widely explored in various machine learning systems such as meta-learning, life-long learning and continuous learning, as well as many optimization problems, including learning to optimize hyperparameters [Maclaurin *et al.*, 2015], learning to optimize neural networks [Ge *et al.*, 2017] and learning to optimize loss functions [Houthooft *et al.*, 2018].

Particularly, *learning to learn* provides an efficient approach to designing learning systems through learning. For instances, Bengio et al. proposed learning local Hebbian updates by gradient descent [Bengio *et al.*, 1992], which utilizes supervised learning to learn unsupervised learning algorithms. Moreover, learning to learn by gradient descent by gradient descent [Andrychowicz *et al.*, 2016] and learning to learn without gradient descent by gradient descent [Chen *et al.*, 2016] employ supervised learning at the meta level to learn supervised learning algorithms and Bayesian optimization algorithms, respectively. In this paper, motivated by these previous works, we utilize supervised learning at the meta level to learn an aggregation method for distributed machine learning.

2.2 Aggregation Methods

The study of aggregation methods has drawn intensive attention from researchers in machine learning. For example, some researchers study the aggregation of gradients with reduced communication and computation costs [Chen *et al.*, 2018a], and some others focus on accelerating the convergence process [Tsitsiklis *et al.*, 1986]. There are also some works that develop aggregation schemes aiming to improve the stability and robustness of the learning process [Scardovi and Leonard, 2009]. Note that most aggregation methods are pre-defined and deterministic, and hence are difficult to be generalized. They are also generally sensitive to errors in the learning system.

In this work, we propose a deep meta-learning system, i.e., an RNN aggregator that can achieve better optimization performance compared with existing deterministic approaches like the averaging aggregator, as proposed by [Konečnỳ *et al.*, 2015]. We also address the fault tolerance issue, especially Byzantine attacks, in the proposed RNN aggregator. Moreover, our proposed aggregation method can work across different local models, which enables distributed learning systems to scale.

3 Distributed Machine Learning Model

In this section, we present the *master-slave* type of distributed learning system, and introduce a general learning algorithm based on distributed gradient descent.

As shown in Figure 1, a distributed learning system is composed of a parameter server $\mathcal{S}$ and a group of workers $\{\mathcal{W}_1, \mathcal{W}_2, \cdots, \mathcal{W}_M\}$. Distributed machine learning can account for a variety of learning problems, including in federated learning [Konečnỳ *et al.*, 2016] and multi-party learning [Hamm *et al.*, 2016], where each worker $\mathcal{W}_m$ can only access its own dataset $\mathbf{x}_m$, and parallel train on a large-scale dataset,

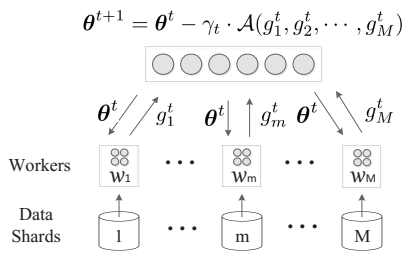

Figure 1: The architecture of the master-slave type of distributed learning system.

where each worker $\mathcal{W}_m$, an individual computing unit, is associated with a specific shard of data $\mathbf{x}_m$.

The objective of distributed machine learning is to let the M workers solve the following problem in a distributed and collaborative fashion :

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}) := \sum_{m \in \mathcal{M}} \mathcal{L}_m(\boldsymbol{\theta}, \mathbf{x}_m),$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is the shared global model, and $\mathcal{L}$ and $\{\mathcal{L}_m, m \in \mathcal{M}\}$ are objective functions with $\mathcal{M} := \{1, 2, \cdots, M\}$. Specifically, $\mathcal{L}_m$ is computed by the worker $\mathcal{W}_m$ as follows,

$$\mathcal{L}_m(\boldsymbol{\theta}, \mathbf{x}_m) := \sum_{x_i \in \mathbf{x}_m} \ell_m(\boldsymbol{\theta}, x_i),$$

where ℓ_m is the loss function (e.g., hinge loss or cross-entropy loss) and $\mathbf{x}_m$ is the data shard that the worker $\mathcal{W}_m$ can access. Generally, a learning model is determined by the structure of the model and the corresponding paramters. In the following, we will slightly abuse the notation of $\boldsymbol{\theta}$ as the parameter vector of the model.

Note that most learning algorithms are based on a common optimized method called stochastic gradient descent (SGD). Here, we introduce the distributed SGD algorithm for distributed learning system [Rajkumar and Agarwal, 2012]. Particularly, all the workers jointly learn a common model iteratively. In tth iteration, the parameter server $\mathcal{S}$ broadcasts the current global model $\boldsymbol{\theta}^t$ to the workers. Then, each worker $\mathcal{W}_m$ computes its local gradient based on its own data shard and uploads it to the server $\mathcal{S}$. Once $\mathcal{S}$ receives the gradients from all the workers, it updates the common global model by aggregating all the gradients via an aggregation function $\mathcal{A}$. We summarize the distributed SGD algorithm by two main steps in each iteration t :

- **Local gradient calculation**: each worker $\mathcal{W}_m$ calculates the local gradient g_m^t based on the current global model $\boldsymbol{\theta}^t$ and its local data shard $\mathbf{x}_m$, i.e.,

$$g_m^t = \frac{\partial}{\partial \boldsymbol{\theta}^t} \mathcal{L}_m(\boldsymbol{\theta}^t, \mathbf{x}_n).$$

- **Global gradients aggregation**: the server $\mathcal{S}$ aggregates the gradients received from all the workers and updates the global model :

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \gamma_t \cdot \mathcal{A}(g_1^t, g_2^t, \cdots, g_M^t),$$

where γ_t is the step size.

Noticeably, in the literature, the aggregation rule $\mathcal{A}$ is typically averaging [Polyak and Juditsky, 1992] and its variants [Zhang *et al.*, 2015; Lian *et al.*, 2015; Tsitsiklis *et al.*, 1986].

4 Learning to Learn Gradients Aggregation by Gradient Descent

In this section, we model the gradients aggregation in distributed machine learning as a learning problem and develop an deep neural network aggregator.

4.1 Recurrent Neural Network Based Aggregator

Motivated by the idea of *learning to learn*, instead of hand-designed aggregation methods, our goal is to develop a learning based aggregator that is able to learn at the meta-level to aggregate the information received from the workers in distributed learning.

To achieve this goal, we consider directly parameterizing the aggregation process as a learning model $\mathcal{A}$ with $\boldsymbol{\phi}$ being its parameter vector. The aggregator $\mathcal{A}$ learns from the parameter server's aggregation experiences and updates itself. Thus, in the tth iteration, the aggregator works as follows:

1. The aggragator $\mathcal{A}$ at the parameter server $\mathcal{S}$ takes the uploaded gradients $(g_1^t, \cdots, g_i^t, \cdots, g_M^t)$ as input, which we denote by $g_{\mathcal{W}}^t$.
2. Given the internally stored hidden knowledge $\mathbf{h}_t$ and the current parameter vector $\boldsymbol{\phi}$, which was learned from the past learning experiences, $\mathcal{A}(\boldsymbol{\phi}, g_{\mathcal{W}}^t)$ outputs its predicted global model $\boldsymbol{\theta}^{t+1}$.
3. The aggregator $\mathcal{A}$ updates its internally stored hidden state to obtain $\mathbf{h}_{t+1}$.

We summarize the process in the tth iteration as follows:

$$\mathbf{h}_{t+1}, \boldsymbol{\theta}^{t+1} = \mathcal{A}(\mathbf{h}_t, g_{\mathcal{W}}^t, \boldsymbol{\phi}).$$

We notice that this procedure can easily map onto a recurrent neural network (RNN), which learns to utilize its internal memory to store information about previous processes and function evaluations, and learns to access this memory to produce current output. To address the gradients vanishing or exploding issues when dealing with long sequential data, in this paper, we apply two popular and efficient variants of the standard RNN, RNN with Long Short-Term Memory units (LSTM) [Hochreiter and Schmidhuber, 1997] and RNN with Gated Recurrent units (GRU) [Chung *et al.*, 2014] .

Subsequently, we can formulate the proposed aggregation procedure as :

$$\mathbf{h}_{t+1}, \boldsymbol{\theta}^{t+1} = \mathcal{A}_{\text{RNN}}(\mathbf{h}_t, g_{\mathcal{W}}^t, \boldsymbol{\phi}),$$

where $\mathcal{A}_{\text{RNN}}(\cdot)$ represents the aggregation method at the parameter server $\mathcal{S}$, and the $\boldsymbol{\phi}$ represents all the parameters in the RNN network.

4.2 Distributed Learning with an RNN Aggregator

With the RNN aggregator $\mathcal{A}_{\text{RNN}}$, the distributed learning process discussed before works as follows:

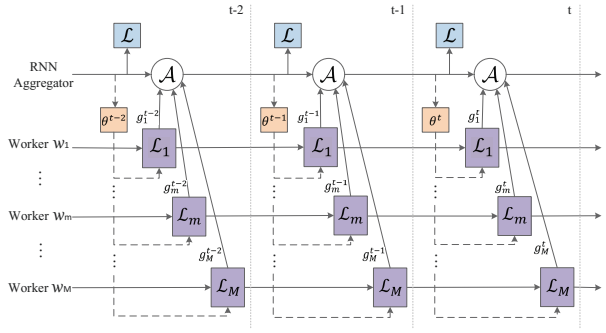

Figure 2: The computational graph used for computing the gradient of the RNN aggregator.

- Local gradient calculation:

$$g_m^t = \frac{\partial}{\partial \boldsymbol{\theta}^t} \mathcal{L}_m(\boldsymbol{\theta}^t, \mathbf{x}_n),$$

- Global gradients aggregation:

$$\mathbf{h}_{t+1}, \boldsymbol{\theta}^{t+1} = \mathcal{A}_{\text{RNN}}(\mathbf{h}_t, g_{\mathcal{W}}^t, \boldsymbol{\phi}).$$

Therefore, the objective of a distributed learning with the RNN aggregator is to find $\boldsymbol{\phi}^*$:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}_{\text{RNN}}(\boldsymbol{\theta}^t, \mathbf{x}_n, \boldsymbol{\phi})$$

Loss Function

A common loss function that is usually used is the loss of the final iteration T:

$$\mathcal{L}_{\text{RNN}}(\boldsymbol{\theta}^t, \mathbf{x}_n, \boldsymbol{\phi}) = \mathbb{E} \sum_{m \in \mathcal{M}} \mathcal{L}_m(\boldsymbol{\theta}^t, \mathbf{x}_n, \boldsymbol{\phi}). \tag{1}$$

The objective function in the Equation (1) depends only on the parameters in the final iteration. In order to efficiently train the aggregator $\mathcal{A}_{RNN}(\boldsymbol{\phi})$, it is convenient to have an objective that depends on the entire trajectory of optimization for some horizon T. Thus, we employ the same loss function as that in previous works [Andrychowicz *et al.*, 2016], which is :

$$\mathcal{L}_{\text{RNN}}(\boldsymbol{\theta}^t, \mathbf{x}_n, \boldsymbol{\phi}) = \mathbb{E} \sum_{m \in \mathcal{M}} \sum_{t=1}^{T} \mathcal{L}_m(\boldsymbol{\theta}^t, \mathbf{x}_n, \boldsymbol{\phi})$$

We minimize the value of $\mathcal{L}_{\text{RNN}}(\boldsymbol{\theta}^t, \mathbf{x}_n, \boldsymbol{\phi})$ using gradient descent on $\boldsymbol{\phi}$. $\partial \mathcal{L}_{\text{RNN}} / \partial \boldsymbol{\phi}$ is estimated based on the computational graph in Figure 2 with Back-propagation Through Time (BPTT). Note that the solid edges in the graph allow the gradients to flow along, while the dashed edges drop the gradients on them.

Preprocessing and Postprocessing

The magnitudes of the aggregator's inputs and outputs may vary dramatically among the workers because their data shards can be very different. However, neural networks usually work robustly only for inputs and outputs that are neither very small nor very large. Therefore, we rescale the inputs and outputs of the aggregator using suitable constants, which are shared across all iterations and workers, so as to efficiently train a stable aggregator.

In particular, we first develop an algorithm to preprocess the aggregator's inputs. Since the input gradients group $g_{\mathcal{W}}^t$ are from several workers, we design a coordinatewise gradients rescaling method to eliminate significant differences among the local gradients.

Recall that the aggregator's input $g_{\mathcal{W}}^t$ is defined as follows:

$$\begin{aligned} g_{\mathcal{W}}^t &:= \{g_1^t, g_2^t, \cdots, g_{\mathcal{M}}^t\} \\ &= \begin{Bmatrix} g_{1,1}^t & g_{2,1}^t & \cdots & g_{\mathcal{M},1}^t \\ g_{1,2}^t & g_{2,2}^t & \cdots & g_{\mathcal{M},2}^t \\ \vdots & \vdots & \vdots & \vdots \\ g_{1,d}^t & g_{2,d}^t & \cdots & g_{\mathcal{M},d}^t \end{Bmatrix} \\ &= \{g_{*,1}^t, g_{*,2}^t, \cdots, g_{*,d}^t\}^T, \end{aligned}$$

where d is the dimension of g_m^t. We conduct the following preprocessing:

$$A_i = \|g_{*,i}^t\|_2$$

$$g_{m,i}^t := \frac{g_{m,i}^t}{A_i}$$

Then we also rescale $g_{m,i}^t$ as:

$$g_{m,i}^t := \begin{Bmatrix} g_{m,i}^t & \text{if} & |g_{m,i}^t| \geq e^{-p} \\ e^{-p} & & \text{otherwise} \end{Bmatrix},$$

where $p > 0$ is a control parameter to disregard very small gradients.

On the other hand, we rescale the output of the aggregator as follows:

$$\boldsymbol{\theta}_i^{t+1} := \boldsymbol{\theta}_i^{t+1} \cdot A_i.$$

4.3 RNN Aggregator with Additional Loss Information

As discussed in the previous section, in a large-scale distributed machine learning system, some workers may have computing errors. What is worse, adversarial attackers may compromise one or more workers and upload forged misleading gradients to hamper the learning process. To address these issues, we utilize additional information to improve the efficiency and robustness of the RNN aggregator.

We sample $n_r (n_r \ll n_b^m)$ data drawn from a public training dataset, where n_b^m is the training batch size of worker $\mathcal{W}_m$, and then calculate additional loss information $f_r^m = \frac{1}{n_r} \sum_{i=1}^{n_r} \ell_m(x_i; g_m^t)$ for each worker. This additional loss information f_r^m can help estimate the descendant of the local gradient g_m^t.

In particular, as shown in Figure 3, in the tth iteration, the parameter sever $\mathcal{S}$ takes local gradient g_m^t and its corresponding additional loss information as input data from worker $\mathcal{W}_m$ to train the RNN aggregator $\mathcal{A}_{\text{RNN}}$.

5 Experiments

In this section, we evaluate the performance of the proposed RNN aggregator and RNN aggregator with additional loss information (ARNN aggregator).

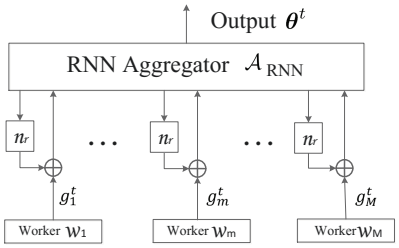

Figure 3: RNN aggregtor with additional loss information.

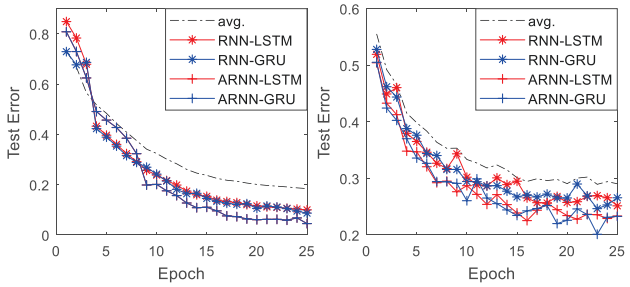

Figure 4: Performance among RNN aggregators, ARNN aggregators and averaging aggregator.

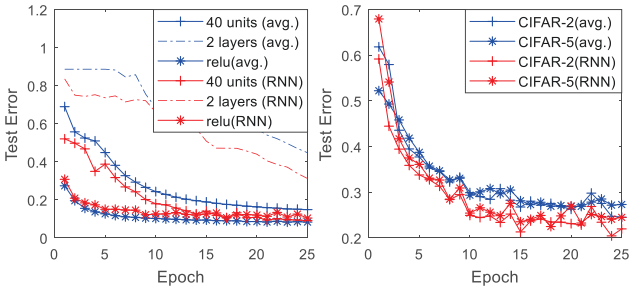

Figure 5: Generalization Performance of LSTM-based RNN aggregator. **Left:** Generalized to different models. **Right:** Generalized to different datasets.

5.1 Experiments Setup

We conduct experiments on two image classification tasks: handwritten digits classification on MNIST dataset and object recognition on CIFAR-10 dataset. We equally divide the training datasets into 10 subsets and assign each subset to an individual worker.

In all experiments, both the RNN aggregators and ARNN aggregators use two-layer LSTMs (or GRUs) with 20 hidden units in each layer and they are trained with BPFT. The optimization process is performed by using ADAM. In addition, we use early stopping to avoid overfitting while training the aggregators. After some fixed number of learning steps, we freeze the aggregator's parameters and evaluate its performance. We pick the best aggregator according to its performance on validation data and report its performance.

5.2 Performance on MNIST and CIFAR-10

In Figure 4, we compare the learning curves for different aggregators, including RNN aggregator with LSTM units, RNN aggregator with GRU units, ARNN aggregator with LSTM units, ARNN aggregator with GRU units and classical averaging aggregators. In the MNIST task, the base network in each worker is multiple layer perceptrons (MLP) with one hidden layer of 20 units using a sigmoid activation function. Meanwhile, in the CIFAR-10 task, each worker holds a model, including two convolutional layers with max pooling followed by a fully-connected layer. In general, we find that the performance of the LSTM network is similar to GRU network in RNN aggregators, as well as in ARNN aggregators. Therefore, in the following, we only investigate the LSTM based aggregators. In addition, both the RNN aggregators and ARNN aggregators outperform the current state-of-the-art averaging aggregator, and ARNN aggregators are more efficient than the RNN aggregators.

5.3 Generalization Performance

An efficient and effective aggregation method should be able to work on different distributed learning systems. Thus, it is important to consider the generalization performance of the proposed aggregators. Specifically, we take LSTM based RNN aggregators as an example and investigate the generalization performance of the proposed scheme in Figure 5.

Generalization Performance on Different Models

We apply the LSTM-based RNN aggregator trained from a system with the previous setting to other distributed systems with different worker models. The modifications of the workers' local models are (1) 40 units: an MLP using one hidden layer of 40 units with the sigmoid activation function, (2) 2 hidden: an MLP using two hidden layers, each of which has 20 units with the sigmoid function, and (3) ReLU: an MLP using one hidden layer of 20 units with the ReLU activation function. As shown in the left plot of Figure 5, the learning cures demonstrate that our learned LSTM-based RNN aggregator works well on these generalization tasks. The RNN aggregator still outperforms the averaging aggregator in (1) and (2), and has competitive performance in (3).

Generalization Performance on Different Datasets

We apply the LSTM-based RNN aggregator trained from a system with the previous setting, and transfer it to other datasets. We utilized two modified datasets CIFAR-2 and CIFAR-5. (For example, the CIFAR-2 dataset only contains data corresponding to 2 of the labels.) The learning curves are shown in the right plot of Figure 5 demonstrate that our learned LSTM-based RNN aggregator works well on transferring among the disjoint subset of the data, and they can compete with classic averaging aggregation rules.

5.4 Fault Resilience

Distributing a computation over a group of nodes may induce a high risk of failure due to unreliable communication and erroneous computation. In this section, we investigate the fault resilience of RNN aggregators and the improved ARNN aggregators under Byzantine attack, where attackers compromise the workers and actively upload forged gradients [Xie *et*

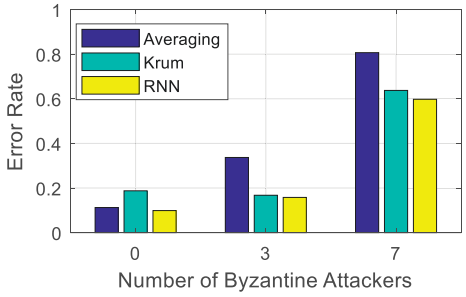

Figure 6: Performance Comparisons among LSTM aggregator, averaging aggregator and Krum aggregator under Byzantine attack.

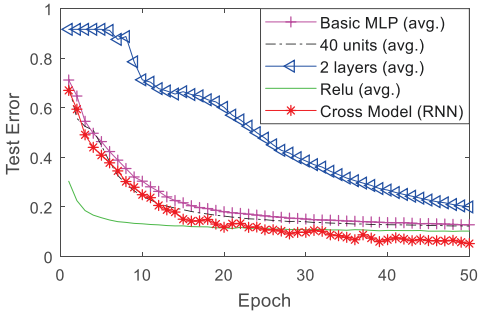

Figure 7: Performance of RNN aggregator across different learning models.

al., 2018] to jeopardize the distributed learning process[1].

We compare our RNN aggregator and ARNN aggregator with the averaging aggregator and the Krum aggregator, which is a state-of-art error tolerant aggregation algorithm [Blanchard *et al.*, 2017], in distributed learning systems with 10 honest workers, 3 out of 10 Byzantine workers, and 7 out of 10 Byzantine workers. As shown in Figure 6, the averaging aggregator is unable to tolerant any Byzantine attack. When only a few computing nodes are Byzantine workers, the RNN aggregator and ARNN aggregator have slightly better performance than the Krum aggregator. When Byzantine workers dominate the honest workers, the Krum aggregator and RNN aggregator cannot resist the attackers and the learning system crashes. However, the ARNN aggregator still works well and outperforms the Krum by 43.14%. The reason behind this is that the Krum aggregates the reliable gradients only and filters our the outlier gradients, which will be fooled by the attackers and filter out the correct gradients when the attackers take the majority. In contrast, the ARNN aggregator utilizes the additional loss information to learn which of the gradients are from the honest participants and aggregates them only.

5.5 System Scalability

In this section, we discuss the performance of RNN aggregator collecting gradients from variant local models. We set up 4 distributed learning systems with different types of local models: an MLP with 1 hidden layer of 20 nodes and sigmoid activation function, an MLP with 1 hidden layer of 40 nodes and sigmoid activation function, an MLP with 2 hidden layers of 20 nodes and sigmoid activation function, and an MLP with 1 hidden layer of 20 nodes with ReLU activation function. Each system has 10 workers. We randomly choose 3 workers from each system and send the gradients to RNN aggregator to update an MLP with 1 hidden layer of 20 nodes and sigmoid activation function.

We demonstrate the learning curves of 4 distributed learning systems in Figure 7 and find that the RNN aggregator can efficiently utilize gradients from different local models to improve the global learning model's performance. Therefore, our proposed RNN aggregator can easily improve the scalability of distributed machine learning systems by involving workers with different types of local learning models. Meanwhile, the experiment also demonstrates that the RNN aggregator is not only able to coordinate workers in one distributed system, but also it can work hierarchically and be used to manage multiple distributed systems. For instance, the RNN aggregator may work as an assistant to utilize information from other distributed learning systems to improve its own learning performance. Note that the ARNN aggregator would obtain the same performance as the RNN aggregator since it mainly differs in the resilience to Byzantine attackers.

[1]In the case of unreliable communications, local gradients may be lost during transmissions, which can be considered equivalent to the scenario that the parameter server receives zero gradients. In the case of erroneous computations, the parameter server receives incorrect gradients from workers. Both cases can be included by the Byzantine attacks. Therefore, in this paper, we focus on evaluating the Byzantine attack tolerance of the proposed RNN aggregator and ARNN aggregator.

6 Conclusions

In this work, different from traditional linear combination aggregation methods, we attempt an alternative approach, *learning to learn*, to coordinate the *master-slave* type of distributed learning process. Specifically, we model the gradients aggregating process at the parameter server as a learning task, which enables us to utilize the recurrent neural network to learn to aggregate the gradients. In addition, we improve the RNN aggregator with additional loss information to address fault tolerance issues, especially Byzantine attacks, in the distributed machine learning system.

The results on the MNIST and CIFAR datasets show that our proposed aggregation methods can outperform classic averaging aggregation method, and Krum, the state-of-the art fault resilience aggregation method. Moreover, the RNN aggregator is the first to utilize gradients from variant models to increase the distributed learning performance.

References

[Abadi *et al.*, 2016] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[Andrychowicz *et al.*, 2016] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent.

In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[Bengio *et al.*, 1992] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, pages 6–8. Univ. of Texas, 1992.

[Blanchard *et al.*, 2017] Peva Blanchard, Rachid Guerraoui, Julien Stainer, et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, pages 119–129, 2017.

[Chen *et al.*, 2016] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. *arXiv preprint arXiv:1611.03824*, 2016.

[Chen *et al.*, 2017] Xuhui Chen, Jinlong Ji, Kenneth Loparo, and Pan Li. Real-time personalized cardiac arrhythmia detection and diagnosis: A cloud computing architecture. In *2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 201–204. IEEE, 2017.

[Chen *et al.*, 2018a] Tianyi Chen, Georgios B Giannakis, Tao Sun, and Wotao Yin. Lag: Lazily aggregated gradient for communication-efficient distributed learning. *arXiv preprint arXiv:1805.09965*, 2018.

[Chen *et al.*, 2018b] Xuhui Chen, Jinlong Ji, Changqing Luo, Weixian Liao, and Pan Li. When machine learning meets blockchain: A decentralized, privacy-preserving and secure design. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1178–1187. IEEE, 2018.

[Chung *et al.*, 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[Dean *et al.*, 2012] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[Ge *et al.*, 2017] Rong Ge, Jason D Lee, and Tengyu Ma. Learning one-hidden-layer neural networks with landscape design. *arXiv preprint arXiv:1711.00501*, 2017.

[Hamm *et al.*, 2016] Jihun Hamm, Yingjun Cao, and Mikhail Belkin. Learning privately from multiparty data. In *International Conference on Machine Learning*, pages 555–563, 2016.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Houthooft *et al.*, 2018] Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[Konečnỳ *et al.*, 2015] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[Konečnỳ *et al.*, 2016] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[Lian *et al.*, 2015] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[Maclaurin *et al.*, 2015] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[Nedic and Ozdaglar, 2009] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[Polyak and Juditsky, 1992] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[Rajkumar and Agarwal, 2012] Arun Rajkumar and Shivani Agarwal. A differentially private stochastic gradient descent algorithm for multiparty classification. In *Artificial Intelligence and Statistics*, pages 933–941, 2012.

[Scardovi and Leonard, 2009] Luca Scardovi and Naomi Ehrich Leonard. Robustness of aggregation in networked dynamical systems. In *Robot Communication and Coordination, 2009. ROBOCOMM'09. Second International Conference on*, pages 1–6. IEEE, 2009.

[Thrun and Pratt, 1998] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.

[Tsitsiklis *et al.*, 1986] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[Ward, 1937] Lewis B Ward. Reminiscence and rote learning. *Psychological Monographs*, 49(4):i, 1937.

[Xie *et al.*, 2018] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Zeno: Byzantine-suspicious stochastic gradient descent. *arXiv preprint arXiv:1805.10032*, 2018.

[Zhang *et al.*, 2015] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, pages 685–693, 2015.