FedRL: Improving the Performance of Federated Learning with Non-IID Data

Yufei Kang, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

Timothy Zeyl
Huawei Technology Canada Ltd.

***Abstract*—Federated learning preserves data privacy by training machine learning models in a distributed fashion, where local models are trained on the client devices and aggregated on the server. Prevalent aggregation algorithms in federated learning perform well in homogeneous settings, but suffer from inadequate convergence in heterogeneous settings due to non-IID data distribution. In this paper, we explore the shortcomings of existing work and recognize that the memory loss of optimizers in aggregation steps limits convergence performance. In response, we propose *FedRL*, a new adaptive aggregation algorithm with the supervision of a policy-based deep reinforcement learning agent. Using real-world datasets, we evaluate the effectiveness of *FedRL* by comparing to state-of-the-art adaptive aggregation algorithms in the literature, and show its superiority in accelerating convergence to a target accuracy.**

I. Introduction

Federated Learning [1] represents an emerging machine learning paradigm which allows client devices to train a shared machine learning model under the orchestration of a central server. In each iteration, remote clients perform the optimization locally, and send the model updates only to a server for aggregation. Though federated learning preserves the privacy of user-generated data, convergence to a target accuracy for the global model may be much slower, partly due to the overhead it takes to communicate with the server, and partly due to the non-IID distribution across client devices.

To reduce communication overhead, only a small portion of all federated learning clients connect with the server in each round to synchronize their models, and only after multiple batches of local updating. In each iteration, clients first download the global model and run several batches of stochastic gradient descent (*SGD*) on their local datasets, and then communicate their model updates to the server. The server then leverages the federated averaging (*FedAvg*) [1], which is typically used as the de facto aggregation algorithm, to simply compute the average of model updates it receives from clients to update the global model.

While federated averaging converges reasonably well in homogeneous settings and has a low communication cost, it does not fully resolve the essential problems associated with statistical heterogeneity. Empirically, diverging and slow convergence have been observed in settings where data distribution is non-IID (not independent and identically distributed) [1]–[3]. In response to such disadvantages with federated averaging, several new algorithms integrated with optimization practices have been proposed, including *FedAvgM* [3], *FedAdam* and *FedAdagrad* [4]. While these algorithms benefit from the craft of learning rate adaptation, they experience performance losses in convergence with some training workloads and certain data distributions. As the optimization techniques involved were not developed for federated learning scenarios, non-IID data distributions were not addressed specifically. In fact, the non-federated analogues of these optimizers struggle to achieve adequate convergence behavior.

To address these challenges, in this paper, we explore the underlying reasons for these limitations, as well as the potential use of reinforcement learning agents. We then propose a new aggregation algorithm, called *FedRL*, to adaptively optimize the model according to training progress, with the objective of incurring fewer communication rounds and speeding up the convergence in the presence of non-IID data distributions.

We begin our exploration by considering the pragmatic optimizers in the literature as well as adaptive optimization methods that make use of exponential moving average. As we shall show, these optimizers all have problems converging to the optimum due to improper hyper-parameter adjustments, which results from the lack of long-term memory. This happens even in some convex settings [5], and could be more severe in the complicated federated learning situation.

We recognize that these practices were unable to achieve a balanced tradeoff between the adaptivity from optimization methods with exponential moving average, and the experience from past optimization trajectories in federated learning. In *FedRL*, we strike that balance by building complementary connections with a policy-based deep reinforcement learning agent. By learning from past client update and the corresponding model accuracy improvement, the agent develops adaptivity control over federated optimization. Combined with the optimization method with exponential moving average, *FedRL* is capable of proper adaptive aggregation given synchronous information of the models. We conduct experiments on real-world datasets and demonstrate that *FedRL*, whose aggregation method has long-term memory and flexible adaptivity, offers superior performance in both convergence rates and model accuracy in federated learning settings with non-IID data.

Highlights of our original contributions in this paper are as follows. To begin with, we recognize that the memory loss of

978-1-6654-3540-6/22

optimizers in the aggregation step limits the convergence of the federated learning algorithm in a heterogeneous setting. In response, we propose a policy-based deep reinforcement learning agent to lead the aggregation process, which adds long-term memory to adaptive optimization methods. Moreover, we present a new and robust federated learning algorithm, called *FedRL*, whose aggregation is supervised by an experienced reinforcement learning agent to handle the convergence challenge in non-IID settings. We perform an extensive set of experiments on three real-world datasets, and show that *FedRL* exhibits better convergence against heterogeneous data. Finally, we verify the robustness and generalization of *FedRL* on untrained data distributions and dynamic conditions where clients generate data as training progresses. Without the need for re-training, *FedRL* keeps its advantage over comparisons on all scenarios.

II. Related work and Challenge

Federated Learning *FedAvg* is typically used as the de facto algorithm. While it has achieved some success in homogeneous environments, in the presence of data heterogeneity, it is observed to diverge and slowly converge [1]–[3]. As such statistical heterogeneity is common in the real world, researchers feel an imperative to improve the aggregation algorithm for non-IID environments and made several attempts.

Some studies propose to improve it on the local training side. *FedProx* [6], *FedDane* [7] and *SCAFFOLD* [2] regularize the local objective functions or introduce additional control variates to avoid local models drift towards their local minima. With slightly improved model performance, clients have to pay more computation cost on local training and parameter tuning. Some other researchers propose to apply momentum or adaptivity to the clients [8] [9]. However, client optimizer states are separately updated when they perform momentum or adaptive optimization methods, requiring extra communication overhead for optimizer states synchronization.

Some researchers apply neuron matching in either coordinate level [10] or layer level [11] before averaging. However, the structural limitations restrict it to specific neural architectures and cannot work generally in machine learning tasks.

Enabling server momentum, *FedAvgM* [3] runs accumulation of the model updates history to dampen oscillations. Later, introducing adaptivity, *FedOpt* aggregation framework that incorporates adaptive optimizers is proposed. With *Adam* and *Adagrad* optimization methods applied, *FedAdam* and *FedAdagrad* are presented.

Although these adaptive optimization adoptions improve convergence compared to *FedAvg*, they have difficulty adjusting parameters when presented with more complicated non-IID distributions, incurring convergence problem. To illustrate the challenge, we conduct an experiment using *FedAdam* and *FedAdagrad* in a heterogeneous setting on the CIFAR-10 dataset and plot the results in Fig. 1. For a fair comparison, we also include *FedAvg*.

We train a modified VGG-16 model with PyTorch on the CIFAR-10 dataset, which consists of 60000 32×32 color images in 10 classes. We consider 100 federated learning clients in total, each running on a different thread in parallel, and allow 10% of the clients to communicate their model updates to the central server at each communication round. The model is trained in both the IID and non-IID settings, respectively. For the IID setting, the 60,000 samples are randomly distributed among 100 clients, such that each client owns 600 samples. For the non-IID setting, each client still owns 600 samples, yet 50% of which come from a dominant class and the remaining half belong to other classes.

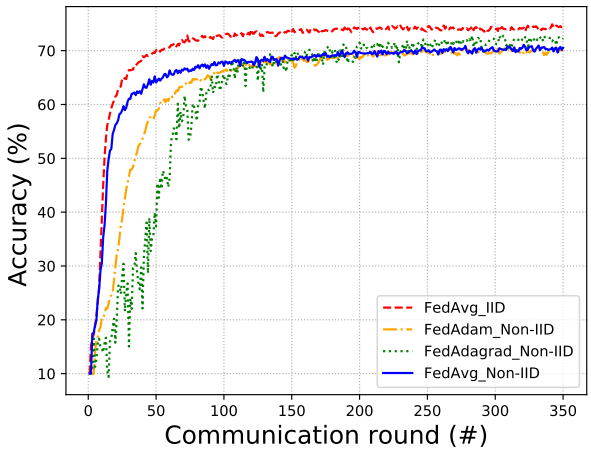


Fig. 1. Training a CNN model on non-IID CIFAR-10 data.

As the results shows in Fig. 1, in the non-IID setting, *FedAdam* and *FedAdagrad* performed slightly better than FedAvg in terms of the accuracy and convergence speed. However, when compared with the performance of FedAvg in IID setting, where clients have the same quantity of data but with an ideal distribution, we can see a performance loss of these algorithms in both accuracy and convergence speed.

Before exploring the cause for such a weakness in a federated learning setting, we first have a quick review of adaptive optimization methods in a centralized setting.

Adaptive Optimization Methods In recent years, to alleviate the slow convergence of *SGD* in nonconvex settings, automatic learning rate decay is introduced and *Adagrad* is proposed as the path-breaker. While it demonstrated an advantage over *SGD*, *Adagrad* suffers in dense setting due to its uncontrollably rapid learning rate decrease [12].

Here cut in the method with exponential moving average (*EMA*) [13]. By leveraging gradients scaled down by square roots of exponential moving averages of squared past gradients, it gets rid of the burdensome gradients computed several steps ago, and therefore prevents unrestrained learning rate decay. Notably, *Adam* [14] is the most popular design of this category with fine-tuned hyper-parameters. While *Adam* benefits from fast convergence from *EMA* strategy, it has difficulty prompt decelerating due to the loss of optimization history. What's worse, the problem is especially aggravated

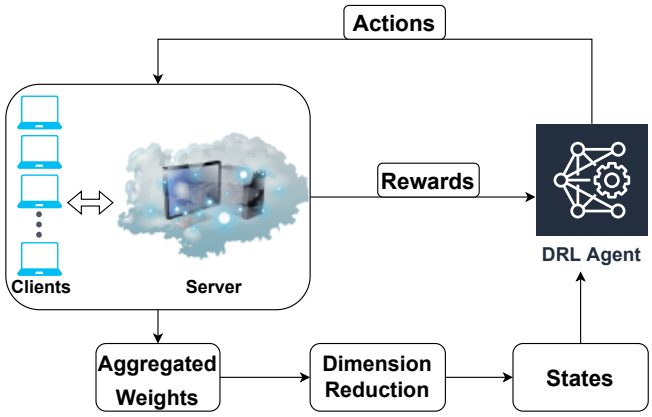


Fig. 2. **Structure of FedRL**

Algorithm 1 FedRL

Initialization: m_0, v_0, local iteration K, loss function F, state s_0, model weights ω_0, maximum round T
for $t = 0, \ldots, T-1$ **do**
  Select a subset $\mathcal{S}_t$ of clients at random
  for client $i \in \mathcal{S}_i$ **do**
    $\omega_t^i = \mathbf{SGD_K}(\omega_{t-1}, \eta_l, \nabla F_i(\omega_{t-1}))$
  end for
  $\Delta_t^i = \omega_t - \omega_t^i$
  $\Delta_t = \frac{\mathcal{S}_i}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \Delta_t^i$
  $\alpha_1, \alpha_2 = \mathbf{DRL\ agent}(s_t)$
  $m_t = \alpha_1 m_{t-1} + (1-\alpha_1)\Delta_t$
  $v_t = \alpha_2 v_{t-1} + (1-\alpha_2)\Delta_t^2$
  Server aggregation: $\omega_{t+1} = \omega_t - \eta_g \frac{m_t}{\sqrt{v_t}}$
end for

in high dimensional settings and when the variance of the gradients with respect to time is large, which is common in federated learning. That explains the failure of the above experiments.

In response, *AMSGrad* [5] that maintains past updates information, such as step sizes and moving directions, is put forward. Different from *Adam*, *AMSGrad* maintains the maximum of specific hyper-parameters until the present time step and uses these maximum values for normalization. While *AMSGrad* benefits from memory fragments and beats *Adam* in some cases, its performance highly depends on the order of incoming gradients, and has inadequate convergence in complicated non-convex settings like federated learning.

Open Challenges For federated learning, an adaptive aggregation mechanism that handles statistical heterogeneity is demanded. Essentially, for adaptive optimization in server aggregation, there is a gap in balancing the fast convergence behavior using *EMA* and the prompt halt at the optimum via leveraging past optimization trajectories.

III. FedRL: System Design

In this section, we propose the design of *FedRL* to address above challenge. We first formulate the convergence issue as a DRL problem and subsequently design the DRL agent. Further, with principal component analysis (*PCA*) [15], we optimize the state space by dimension reduction. Finally, we summarize the system in Fig. 2 and algorithm in Algorithm 1.

A. DRL problem formulation

The optimization process of federated learning can be modeled as a Markov Decision Process (MDP), where state s_t is represented by a function of aggregated gradients Δ_t computed from local model updates in communication round t. Given a state s_t, DRL agent takes an action a_t to adjust EMA hyper-parameters α_1 and α_2. Accordingly, the central server optimizes the global model and broadcasts it. After downloading the updated model, local clients run *SGD* on their local datasets for a number of iterations and communicate the model updates to the central server. When all updates from local clients are received, the central server updates the model based on aggregated gradients Δ_{t+1} and transits the state to s_{t+1}. Meanwhile, a reward signal r_t can be observed, which is a function of the test accuracy achieved by the global model so far on a held-out test set.

B. Design of DRL Agent

Suppose there is a federated learning job on N participating clients with a target accuracy Λ. At each communication round, the federated learning server receives models updates from remote clients and computes a state vector $\mathbf{s}$, in response to which the DRL agent acts and generates an action vector $\mathbf{a}$. Since the state vector $\mathbf{s}$ is computed from gradients of model weights, and the action vector is composed of variations of hyper-parameters, the state and action spaces are continuous. Therefore, we select proximal policy optimization (*PPO*) to serve as the DRL agent and design it follows.

State: We represent the state $\mathbf{s_t}$ at the time step $\mathbf{t}$ by a vector $\mathbf{s_t} = \mathbf{\Delta_t} \odot \mathbf{\Delta_t}$, where $\mathbf{\Delta_t}$ denotes the aggregated gradients computed by running averaging on the central server after communication round t. Note that the length of the state vector $\mathbf{\Delta_t^2}$ is exactly the same as the model weights $\omega_\mathbf{t}$, since it is calculated from the gradients of model weights $\omega_\mathbf{t}$. When the training model is complex, the state space could be unexpectedly large. To this end, we reduce the dimension of the state space to optimize the system and shall discuss it later.

Action: Given a state $\mathbf{s_t}$ at the time step $\mathbf{t}$, the DRL agent acts and generates an action vector $\mathbf{a_t}$, which is defined as $\mathbf{a_t} = [\mathbf{\Delta}\alpha_\mathbf{1}, \mathbf{\Delta}\alpha_\mathbf{2}]$, where $\mathbf{\Delta}\alpha_\mathbf{1}$ and $\mathbf{\Delta}\alpha_\mathbf{2}$ are variations of adaptive parameters. Notably, these variations can be utilized to compute the step sizes and moving directions for model weights at the current communication round, with which server aggregation could be performed in an optimized way. Afterwards, the updated model would be broadcasted by the central server, back to selected clients, completing the communication round..

Reward: Along with the state transition from $\mathbf{s_t}$ to $\mathbf{s_{t+1}}$, a reward signal r_t is generated. Balancing training accuracy and

convergence speed, we define the reward at time step t as $r_t = \Gamma^{\lambda-\Lambda} - \Omega,\ t = 1, \ldots, T$, where λ is the testing accuracy of the global models on the validation set after t rounds, Λ is the target accuracy, Γ and Ω are positive constants to encourage high test accuracy and punish slow convergence behaviors. The DRL agent is motivated to maximize the expectation of discounted accumulated reward, specifically

$$R = \sum_{t=1}^{T} \gamma^{t-1} r_t = \sum_{t=1}^{T} \gamma^{t-1} (\Gamma^{\lambda-\Lambda} - \Omega) \tag{1}$$

where $\gamma \in (0, 1]$ is the discount factor, which is selected to balance between current reward and future rewards.

Empirically, to stimulate the DRL agent to learn optimization policy that helps the FL server obtain the global model with high test accuracy, we set Γ as 64. What's more, to penalize futile or even disastrous actions, Ω is set to be 1. Consequently, $r_t \in (-1, 0]$, since $0 < \lambda \leq \Lambda$ and $\Gamma > 0$. To avoid short-sighting, we set the discount factor γ as 0.9 in our implementation.

C. Optimizing State Space

Since modern deep neural networks have millions of parameters, including the gradient of every parameter in the state space is infeasible. To reduce the dimensionality of the state space, we introduce a principal component analysis *PCA* module. Represented by the resulted vector, a lighter state vector is obtained. Empirically demonstrated, for classification tasks on the MNIST dataset, a clustering affecting local models according to their dominant labels can be still observed even if $431,078$ dimensions are dropped to 1 during the dimension reduction process [16]. No negative effect would be incurred with the dimension reduction module. In our system, for the sake of efficiency, we reduce state space to 1 dimension. The *PCA* model is trained at the first round of federated learning with the aggregated updates on the server side, and then fixed for each following communication round.

D. Overview

Now we conclude our design of *FedRL* in Fig. 2. At the beginning of each federated learning round, the FL server averages the received model updates and sends out the resulted vector to the dimension reduction module, where the state vector is computed. Afterwards, the DRL agent computes an action vector and sends it back to the FL server. With variations within the action vector, the FL server optimizes the global model and terminates current rounds.

Remarkably, no extra communication cost is introduced to federated learning clients in *FedRL*, since both the state vector and the action vector are communicated between the DRL agent and the FL server.

IV. Experimental results

To study the performance of *FedRL* on statistical heterogeneous setting, we conduct experiments on three real-world datasets: MNIST, FashionMNIST and CIFAR-10. As comparisons, we introduce the de facto algorithm *FedAvg*, and state-of-the-art aggregation algorithms, *FedAdam* and *FedAdagrad*. On each dataset, we train a CNN model with *FedRL*, *FedAvg*, *FedAdam* and *FedAdagrad* respectively, and evaluate the test accuracy of the trained model using the testing set. Additionally, we explore both static setting, where clients own the same data throughout the training, and dynamic setting where clients generate data as training processes.

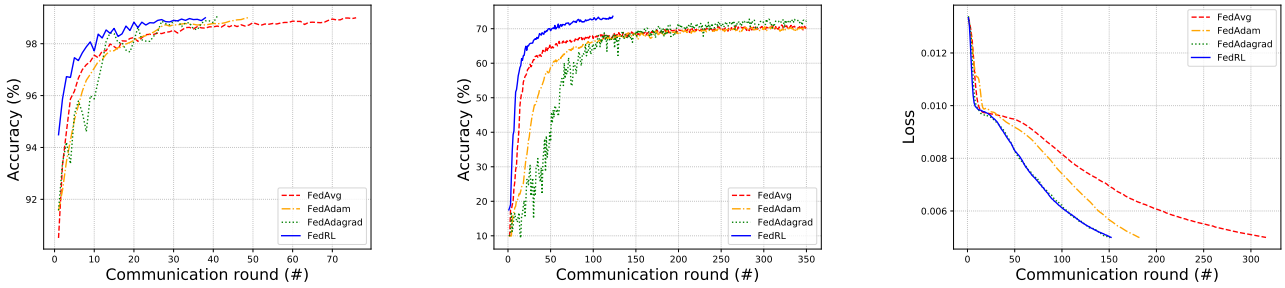
A. Datasets and Models

We explore different combinations of hyper-parameters for the CNN models on different datasets and choose the hyper-parameters leading to the best performance of *FedAvg* in IID settings.

- **MNIST:** We train a CNN model with two 5×5 convolution layers. The first layer has 20 output channels and the second has 50, with each layer followed by 2×2 max pooling. For each client, the batch size is set to be ten and the epoch number is five.
- **EMNIST:** We train a densely connected auto-encoder with layers of size 784 (28×28) – 128 – 128 and a symmetric decoder. For each client, the batch size is set to be 100 and the epoch number is five.
- **CIFAR-10:** We train a modified VGG-16 model with less convolution cores reducing requirement for memory and additional batch normalization accelerating training. Ten 3×3 convolution layers, three 1×1 convolution layers, and three full connected layers are included. The first two 3×3 convolution layers have 64 output channels, and a 2×2 max pooling follows. The second two 3×3 convolution layers have 128 output channels, and 2×2 max pooling follows. Next, two 3×3 convolution layers with 128 output channels are added, followed by one 1×1 convolution layers with the same output channels and 2×2 max pooling layer. Then, two similar units composed of two three convolution layers and a max pooling layers with 256 output channels and 512 output channels follow it. Further, two full connected layers with 1024 output channels are added, with each layer followed by 0.5 dropping out. Finally, a full connected layer with 10 output channels are included. For each client, the batch size is set to be 100 and the epoch number is 30.
- **Performance metrics:** To evaluate the convergence behavior, the amount of communication rounds required for convergence and the test accuracy (loss) is utilized as the performance metric.

B. Federated Learning Setting

Empirically, we simulate the federated learning setting with the Python threading library. We consider 100 clients in total, each of them running on a thread in parallel. At each federated learning round, we randomly select 10% of total clients to communicate their local updates to the central server. To evaluate performance with non-IID distributed data, we constructed three settings with different heterogeneity levels. In the low heterogeneous setting, 20% of the data belong to a dominant class, which is different across clients, and the



(a) Classification on MNIST data (b) Classification on CIFAR-10 data (c) Auto-encoder on EMNIST data

Fig. 3. **Evaluating the performance of *FedRL* with medium heterogeneity**

remaining 80% belong to other classes randomly. In contrast, in the medium (high) heterogeneous setting, 50% (80%) of the data belong to a dominant class. Besides, to simulate real-world data generation, we enable data to be successively generated as federated learning progresses in dynamic setting.

C. Training the DRL Agent

In the training phase, the server has limited access to local data. Therefore, we train the DRL agent with simulated IID data. Towards tasks on MNIST and EMNIST, we construct the training setting as that only ten clients are available in total and all of them communicate their local updates to the central server at each communication round. For task on CIFAR-10, we construct the training setting as that 100 clients are available and 10% of them can update to the central server at each communication round.

We set up the *PPO* agent with stable-baselines3[1] and train it on an instance with a Titan RTX GPU. Each iteration for MNIST and FashionMNIST training task takes only seconds, and each iteration for CIFAR-10 task takes $2 \sim 3$ minutes, due to the deep neural networks. An episode starts at the initialization of a federated learning job and ends when the job converges to the target accuracy Λ or the test accuracy plateaus. In image classification tasks, the target accuracy is set to 99% on MNIST and 74% on CIFAR-10. For auto-encoder training tasks on EMNIST, the target loss is set to 0.005.

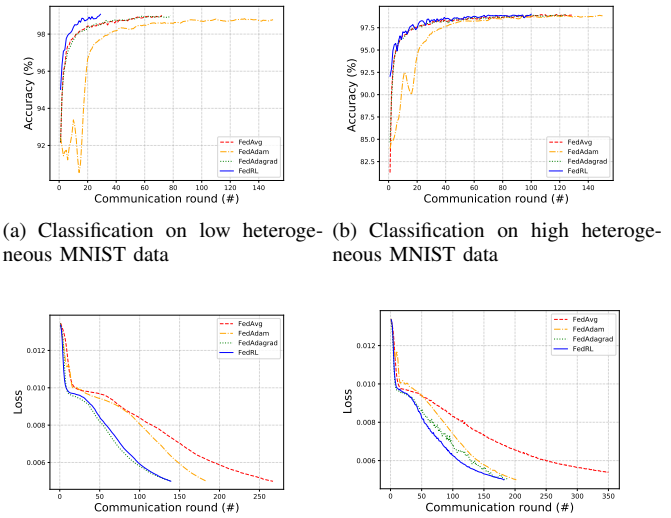
D. Evaluations

After finishing the training, we evaluate *FedRL* in settings where 100 clients with medium heterogeneous data exist. We plot the test accuracy v.s. the communication rounds on MNIST and CIFAR-10 in Fig. 3(a) and Fig. 3(b) respectively, and training loss v.s. the communication rounds on EMNIST in Fig. 3(c). The plotted data points are the average results of repeated experiments.

As shown in Fig. 3(a), for the image classification task on MNIST, *FedRL* converges to the target accuracy 99% after 38 rounds, outperforming *FedAvg*'s 76 rounds, *FedAdam*'s 49 rounds and *FedAdagrad*'s 43 rounds. Similarly, we plot the empirical result of the classification workload on CIFAR-10 in Fig. 3(b). While *FedRL* converges at the highest speed as we expected, it is the only one that successfully reaches the target accuracy. It indicates that by allowing more adaptivity, *FedRL* effectively handle the non-IID issue, preventing it from falling in the local optimum. Besides, we can refer the performance of *FedRL* on training an auto-encoder on EMNIST in Fig. 3(c). Here, *FedRL* is one of the top algorithms of converging to the target mean square error 0.005 with 154 rounds. For comparison, *FedAvg* and *FedAdam* take 328 and 180 rounds to converge.

In all, demonstrated by above results, *FedRL* shows its superiority over the comparison group when converging to a target in federated learning settings with medium heterogeneous data.

[1]https://github.com/DLR-RM/stable-baselines3



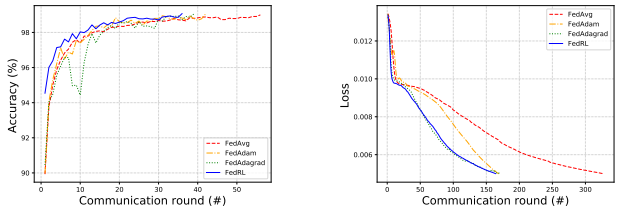
(a) Classification on low heterogeneous MNIST data (b) Classification on high heterogeneous MNIST data

(c) Autoencoder on low heterogeneous EMNIST data (d) Autoencoder on high heterogeneous EMNIST data

Fig. 4. **Generalization of *FedRL* on different heterogeneity levels**

E. Generalization and Robustness

In this section, we discuss the performance of *FedRL* facing unseen scenarios with different data heterogeneity levels and data generation dynamicity. To evaluate its robustness, we conducted experiments on several settings.



(a) Classification on dynamic MNIST data (b) Auto-encoder on dynamic EMNIST data

Fig. 5. **Robustness of *FedRL to dynamicity***

Generalization on heterogeneity To verify the generalization of *FedRL* across unseen heterogeneity levels, we conduct follow-up experiments. Without re-training, we run *FedRL* on settings where 100 clients with unseen low and high heterogeneous data exist. Also, for comparison, we run *FedAvg*, *FedAdam* and *FedAdagrad* on the same tasks. The experimental results on MNIST and EMNIST are plotted in Fig. 4.

Shown in Fig. 4(a) and Fig. 4(b), for classification tasks on MNIST dataset, while high heterogeneity level incurs slow convergence, *FedRL* is dominant over other aggregation algorithms on both levels, achieving the target accuracy with the fewest communication rounds. For the auto-encoder training task on EMNIST, demonstrated in Fig. 4(c) and Fig. 4(d), *FedRL* maintains an advantage in converging to target loss as before.

Robustness to dynamicity Previously, we evaluate *FedRL* on static setting where device data are only generated before the federated learning starts. To verify its robustness to dynamicity, we test it on a setting where data are successively generated as learning progresses. In this assessment, we set a medium heterogeneity level. As before, *FedRL* does not require re-training. The results on MNIST and EMNIST are plotted in Fig. 5, with Fig. 5(a) representing the results on MNIST and Fig. 5(b) as EMNIST.

Demonstrated by Fig. 5(a), for classification task on MNIST dataset, *FedRL* shows its advantage over the comparisons, converging to the target accuracy with the fewest communication rounds, even when new data are successively generated. For auto-encoder training task on EMNIST, we can refer similar superior of *FedRL* over *FedAvg*, *FedAdam* and *FedAdagrad* in dynamic setting.

V. Concluding Remarks

In this work, we explore the shortcomings of existing federated aggregation methods with optimization practice and recognize that the improper learning rate decay caused by memory loss in adaptive optimization methods is responsible for the failures of adaptive federated learning algorithm in non-IID settings.

To overcome the weakness, we present *FedRL*, a new aggregation algorithm that incorporates a DRL agent to guide optimization in federated aggregation. In particular, we introduce a policy-based deep reinforcement learning agent, namely, *PPO*, to adjust the hyper-parameters in adaptive optimization methods with EMA as the training processes. As the *PPO* agent can learn from past updates history, *FedRL* can adjust learning rate properly based on both fresh gradients and past experience, thus converging fast and well to the optimum.

Furthermore, to elaborate the effectiveness of *FedRL*, we conduct an extensive set of empirical evaluations on three real-world datasets and demonstrate superiority of *FedRL* in both convergence rate and model accuracy (training loss) in heterogeneous settings with non-IID data. Finally, we testify the robustness and generalization of *FedRL* on untrained data distributions and dynamic conditions where clients generate data as training progresses. Without the need for re-training, *FedRL* keeps the advantage over comparisons on all scenarios.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication efficient learning of deep networks from decentralized data," in *Proc. Artificial Intelligence and Statistics (AISTATS)*, 2017.

[2] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. International Conference on Machine Learning (ICML)*, 2020.

[3] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[4] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[5] S. Reddi, S. Kale, and S. Kumar, "On the convergence of ADAM and beyond." in *Proc. International Conference on Learning Representations (ICLR)*, 2018.

[6] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Conference on Machine Learning and Systems (MLSys)*, 2020.

[7] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smithy, "Feddane: A federated newton-type method," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2019, pp. 1227–1231.

[8] H. Yu, R. Jin, and S. Yang, "On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization," in *Proc. International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 7184–7193.

[9] H. Yuan and T. Ma, "Federated accelerated stochastic gradient descent," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[10] M. Yurochkin, M. Agarwal, S. Ghosh, K. Greenewald, N. Hoang, and Y. Khazaeni, "Bayesian nonparametric federated learning of neural networks," in *Proc. International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 7252–7261.

[11] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," in *Proc. International Conference on Learning Representations (ICLR)*, 2020.

[12] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *COURSERA: Neural Networks for Machine Learning*, vol. 14, no. 8, p. 2, 2012.

[13] T. Tieleman and G. Hinton, "Rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[14] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. International Conference on Learning Representations (ICLR)*, 2015.

[15] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[16] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *Proc. IEEE INFOCOM*, 2020.