FedLab: A Flexible Federated Learning Framework

Dun Zeng
School of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, China.
zengdun@std.uestc.edu.cn

Siqi Liang
School of Computer Science and Technology
Harbin Institute of Technology Shenzhen
Shenzhen, China.
zszxlsq@gmail.com

Xiangjing Hu
School of Computer Science and Technology
Harbin Institute of Technology Shenzhen
Shenzhen, China.
starryhu0110@gmail.com

Zenglin Xu*
School of Computer Science and Technology
Harbin Institute of Technology Shenzhen
Shenzhen, China.
xuzenglin@hit.edu.cn

October 18, 2021

Abstract

Federated learning (FL) is a machine learning field in which researchers try to facilitate model learning process among multiparty without violating privacy protection regulations. Considerable effort has been invested in FL optimization and communication related researches. In this work, we introduce `FedLab`, a lightweight open-source framework for FL simulation. The design of `FedLab` focuses on FL algorithm effectiveness and communication efficiency. Also, `FedLab` is scalable in different deployment scenario. We hope `FedLab` could provide flexible API as well as reliable baseline implementations, and relieve the burden of implementing novel approaches for researchers in FL community. The source code is available at https://github.com/SMILELab-FL/FedLab.

1 Introduction

Federated learning (FL), proposed by Google at the very beginning [1], is recently a burgeoning research area of machine learning, which aims to protect individual data privacy in distributed machine learning process, especially in finance [2], smart healthcare [3, 4] and edge computing [5, 6]. Different from traditional data-centered distributed machine learning, participants in FL setting utilize localized data to train local model, then leverages specific strategies with other participants to acquire the final model collaboratively, avoiding direct data sharing behavior.

Though it might differ in specific methodologies, current FL schemes can be summarized as repetition of training rounds, with each integrated by several basic steps: *i*) local update on client's model using their own localized data; *ii*) clients upload their local trained model parameters to server; *iii*) server performs aggregation strategy on collected clients' model parameters to obtain global model; *iv*) server selects a subset of clients and distributes the latest global model to them. Many FL researches try to improve algorithm effectiveness or efficiency on only one or more steps in this workflow with different scenarios: [7] suggests to add regularization term in step *i*) to achieve more robust convergence in heterogeneous settings; [8] applies gradient compression method in step *ii*) to reduce communication bandwidth; [9] tries to modify in step *i*), *ii*) and *iii*) for privacy-preserving purpose; [10] proposes better sample strategy in step *iv*) to address suboptimal result problem in Federated Multi-Task Learning. These indicate that the implementation of many FL algorithms only requires modification on several components of common workflow, without the necessity of repetitive implementation on basic FL workflow. The paradigm of FL and related research points are as depicted in figure 1.

*Corresponding author

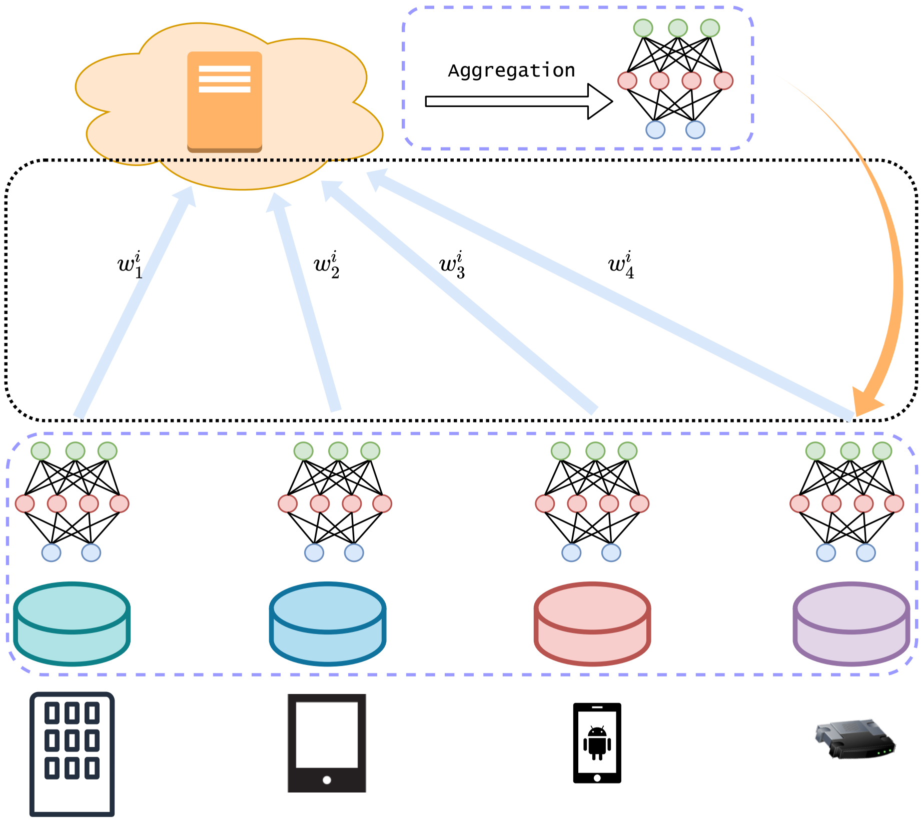


Figure 1: The paradigm of Ferderated Learning. The content in black dashed box indicates the communication strategy of FL system. The content in blue dashed box indicates the FL optimization, including global aggregation and local optimization.

However, though with several FL related frameworks or platforms available, researchers still prefer to implement FL algorithms using PyTorch [11] or TensorFlow [12] from scratch [13, 14]. This inefficient modus operandi in FL community can hamper researchers' enthusiasm in both procedures of reproducing previous work and fast verification of new ideas.

To relieve the burden of researchers in implementing FL algorithms and emancipate FL scientists from repetitive implementation of basic FL setting, we introduce highly customizable framework `FedLab` in this paper. `FedLab` provides the necessary modules for FL simulation, including communication, compression, model optimization, data partition and other functional modules. `FedLab` users can build FL simulation environment with custom modules like playing with LEGO bricks. In all, we make the following contributions to FL community:

- A flexible FL framework `FedLab` is proposed, in which the flexibility is given by highly customizable interfaces and scalability in FL system. `FedLab` allows users focus on interested components design while keeping other part default. What's more, `FedLab` also supports *standalone*, *cross machine* and *hierarchical* simulation paradigms.

- Various data partition tools for comprehensive data distribution scenarios in FL. `FedLab` provides a series of data partition functions as well as built-in data partition schemes for different data distributions over federation.

- Standardized FL implementation schemes are presented through `FedLab`. For instance, standard synchronous and asynchronous FL system are available. Besides, we also provides FL datasets benchmarks and functional modules for standard FL simulation.

- An open-source group is founded in GitHub repository for `FedLab`'s continuous maintenance. Elaborate document is published as well.

2 Background

Current FL community focuses mainly on two major challenges. Firstly, data heterogeneity across clients slows down model convergence [15] compared with that of data-center distributed learning [16]. The other major challenge is communication cost during both model uploading and downloading processes, which is also the bottleneck of distributed learning. There is an urgent need for improvement on communication, especially when it comes to cross-device scenario. A lot of works have been proposed to tackle these two challenges, which can be categorized into optimization algorithms and communication efficient strategies. In this section, these two popular research sub-fields of FL will be further illustrated, and the need of a convenient FL framework suitable for optimization effectiveness and communication efficiency research will be revealed.

2.1 Optimization Algorithms

Malicious attacker is able to steal private information by using gradient attack algorithms [17, 18, 19]. Therefore, clients can't transmit gradients but model parameters directly. FL server optimizes neural network by aggregating all parameters of clients (which is updated a few epochs locally) into global one. Typically, server aggregates model parameters collected from K clients at round i to update global weights w^{i+1} following FedAvg [1]:

$$w^{i+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^i$$

Under this setting, FL optimization still faces many challenges. In data-center distributed machine learning, each computation node get its dataset from parameter server, which makes data distribution independently identically distribution (I.I.D) across nodes. However, data in FL clients can be Non-I.I.D in many ways [20], which leads to inferior robustness and slow convergence.

Plenty of federated optimization algorithms are proposed to overcome data None-I.I.D problem. [21, 22, 23, 24] try to learn a better shared federated model based on different aggregation strategies. FL Pensonalization [25, 26] aims to learn personalized model for every client. The combination of FL and other deep learning techniques, such as meta learning [27], transfer learning [28], etc., are popular as well. To summarize, most optimization researches only relate to local training process on client and parameter aggregation process on FL server, which indicates that a flexible FL framework shall provide customizable interfaces for both local training design as well as server aggregation strategies.

2.2 Communication and Compression

Bandwidth problem is bottleneck of large-scale distributed training, and it becomes even worse when distributed training is performed in FL. Thus, deploying communication compression strategy is necessary, especially in cross-device setting. Two common-used and low resource-consumption compression methods as follows:

Quantization [29, 30, 31] replaces each tensor with a lower precision one (e.g., float16 instead of float32), accomplishing the trade-off between precision and compression ratio. **Sparsification** [32, 8, 33] selects a subset of tensors by appointed principle (e.g., Top-k selection) to transmit. It can achieve at least $100\times$ compression ratio. These two compression methods are model independent, which shows a flexible FL framework shall also provide model-independent compression module.

2.3 Related work

Several open-sources FL frameworks have been released. FATE[2] is a large federated secure computing framework. PaddleFL[3] and FedLearner[4] are proposed by Baidu and Bytedance that support applications and deployment of FL system in application scenario. Frameworks above are industrial-oriented, focusing on real-life applications but not suitable for laboratory FL simulation. Rosetta [34] and PySyft [35] mainly focus on secure multiparty computation of FL rather than algorithm and communication researches. TFF[5] supports the simulation of FL training but executes only on a single machine. FedML [36] is a comprehensive FL framework that includes most research fields in FL. And Flower [37] provides a FL communication framework supporting different deep learning framework (e.g., PyTorch, TensorFlow and MXNet). But they still hold varies of dependent libraries, which makes them heavy.

[2] `https://fate.fedai.org/`
[3] `https://github.com/PaddlePaddle/PaddleFL`
[4] `https://github.com/bytedance/fedlearner`
[5] `https://github.com/tensorflow/federated`

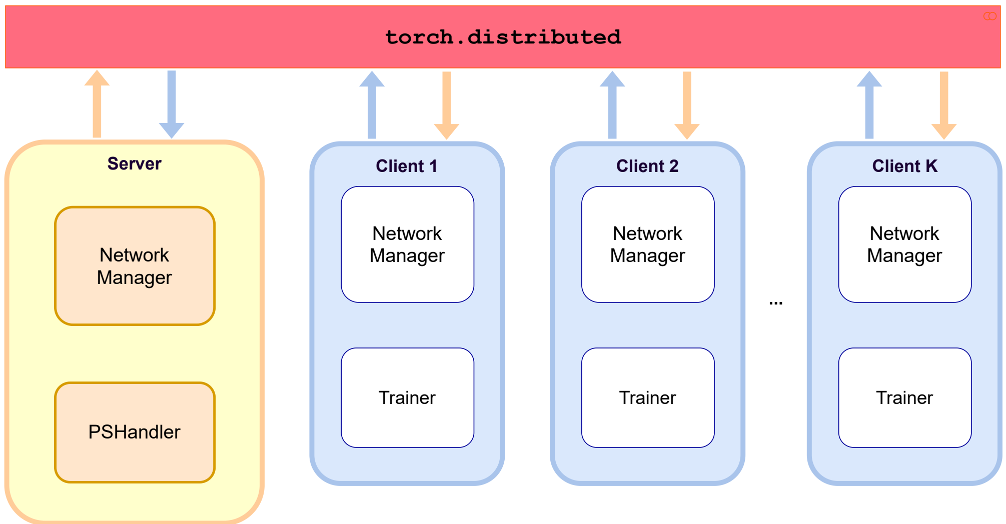


Figure 2: An overview of FedLab architecture. Two main roles in FedLab are define with two functional module: `NetworkManager` and `ParameterServerHandler/Trainer`. Communication backend is `torch.distributed` module.

Different from frameworks above, `FedLab` is designed to be lightweight. It focuses on optimization effectiveness and communication efficiency for FL system simulation. We encourage users to build FL system following standard program pipeline and providing custom interfaces at the same time. Features of `FedLab` are further illustrated in the next section.

3 Framework Overview

In this section, we mainly illustrate architectural designs and detailed features in both communication efficiency and optimization effectiveness aspects. `FedLab` provides two main roles in FL setting: Server and Client. Each Server/Client consists of two components called `NetworkManager` and `ParameterServerHandler/Trainer`. The overview of `FedLab`'s structure is shown in Figure 2.

`NetworkManager` module manages message process task, which provides interfaces to customize communication agreements and compression algorithms. In section 3.1, the details of communication module is demonstrated. `ParameterServerHandler/Trainer` takes charge of specific optimization algorithm design, and is illustrated in section 3.2. Finally, three deployment scenarios supported by `FedLab` are presented in section 3.3.

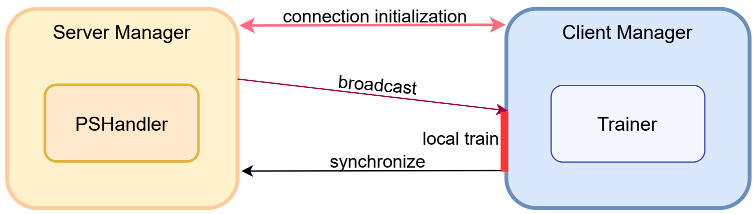
3.1 Communication Efficiency

In order to meet various requirements of FL network communication, `FedLab` implements `NetworkManager` to manage network topology, using `torch.distributed` as communication backend. `NetworkManager` is designed to be flexible in tensor agnostic, customization and scalability. Details of these features are stated below.

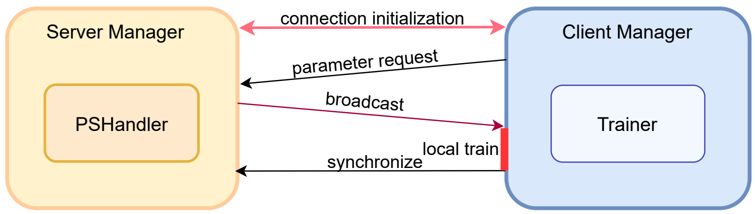
Everything is Tensor. Inspired by the structure of network message, the basic communication element in `FedLab` is called `Package`, which contains *header tensor* with necessary control information and *content tensor* with packed tensor list. What's more, `PackageProcessor` in `NetworkManager` provides useful functions for packing up tensor list and restoring content to tensor list. In this way, the details of `Package` are blocked from users. Besides, `Package` is represented by a one-dimension tenser (vector), which is compatible with interfaces of PyTorch precisely.

Compressor. Given feature of *Everything is Tensor*, compression algorithms can be naturally performed in pack and unpack process. Moreover, compression baseline algorithm Top-k compressor and SOTA DGC compressor [8] are available in `FedLab`. Customizable interfaces for compression algorithm are provided as well.

Communication Agreement Customizable. Communication agreements can be explained by following questions: What contents to send? How does client or server react after receiving message? Flexibility of communication module is given by `NetworkManager` module, which offers users the interfaces of customizing communication protocol. User can define additional information exchange, and control information flow for advanced algorithm development.



(a) Synchronous



(b) Asynchronous

Figure 3: NetworkManager in FedLab

Communication Pattern. Synchronous and Asynchronous communication patterns are implemented according to Federated Optimization algorithms. Specifically for figure 3(a), One round of synchronous communication flow can be describe as follows:

1) *Initialization*. Server and Clients initialize network connection.
2) *Sampling*. Server selects subset of clients to join current round of FL by broadcasting global model to them.
3) *Synchronization*. Client starts it local train process after receiving global model. Then, every Client sends needed information including local model to Server.
4) *Aggregation*. Finally, Server collects all information from Clients and performs aggregation.

Differently, in asynchronous communication, every client communicate with server asynchronously. A FL training round is begin with a parameter request from client. Besides, server update global model every time it receives a synchronization upload. Details are shown in figure 3(b).

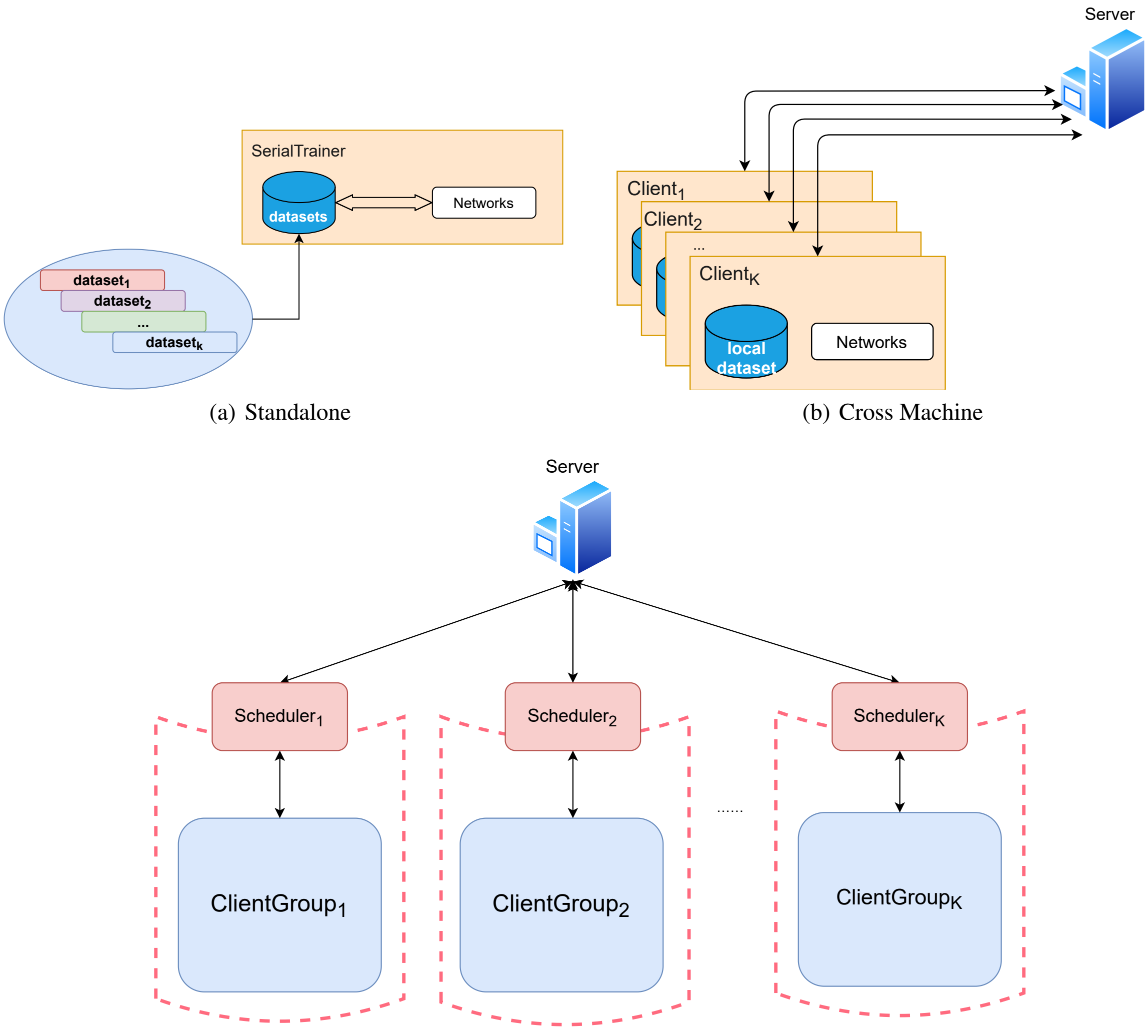
Scheduler. *Cross-silo* and *Cross-device* [38] are the common FL settings. Cross-silo FL system usually has 2 - 100 clients which with large bandwidth and powerful computing resources. In contrary, cross-device scenario indicates that more clients (up to 10^{10}) but less resources (power, bandwidth) with each client. Since there are needs of simulating more than 100 of clients, we designed message forward module `Scheduler` to extend the scalability of `FedLab`. Firstly, `Scheduler` is able to connect machines in different LAN(Local Area Network). What's more, users can overwrite the work flow of `Scheduler` to achieve hierarchical communication pattern. The usage of `Scheduler` will be further illustrated in section 3.3.

3.2 Optimization Effectiveness

Optimization module in `FedLab` achieves "high-cohesion and low-coupling", which means this module can be used independently just like LEGOs bricks. To be more specific, `ParameterServerHandler` and `Trainer` is executable without `NetworkManager`. Besides, `FedLab` does not provide high level APIs, but prepares the necessary implementation tools for developers, reflecting the flexibility of framework. In this section, some key features of `FedLab` for standard FL optimization are illustrated.

Aggregation. `Trainer/ParameterServerHandler` in `FedLab` is corresponding with Client/Server optimization process. We encourage standard optimization implementation paradigm for both Client and Server: `Trainer` manages local dataset and performs PyTorch training process. `ParameterServerHandler` is implementation of parameter aggregation. In `FedLab`, `ClientSGDTrainer` is a standard implementation of `Trainer` for users. Additionally, we provides standard demos of `ParameterServerHandler` with different aggregation algorithms, such as FedAvg [1] and FedAsgd [39].

Data Partition. In practice, Non-I.I.D datasets are not always accessible for researchers due to privacy restrictions. Thus, researchers tend to manually create Non-I.I.D data partition in experiment environment. For instance, FedAvg [1]



(a) Standalone

(b) Cross Machine

(c) Hierarchical

Figure 4: Supported Deployment Scenario in FedLab

sorts the MNIST dataset by digit label, and divides it into 2000 shards of size 300 to create pathological Non-I.I.D partition over clients. Also, current FL researches handling non-IID problems tend to design very specific non-IID scenarios rather than standard and systematic partition schemes [40]. Therefore, `FedLab` offers users a series of data partition functions, as well as built-in data partition schemes for some datasets based on design of NIID-bench [40] and [41]. What's more, `FedLab` provides PyTorch version of LEAF [42] (a Non-I.I.D partitioned FL datasets baseline).

3.3 Deployment Scenarios

`FedLab` encapsulates the network interface of `torch.distributed` module, providing stable end-to-end `tensor` transmission for FL simulation. Furthermore, we implement a scalable version of `NetworkManager`, called `Scheduler`, to ensure the flexibility of network topology and the scalability of the system. Different deployment scenarios of `FedLab` correspond to different experimental conditions, for scalability and flexibility.

Standalone. `FedLab` implements `SerialTrainer` for FL simulation in single process. `SerialTrainer` allows user to simulate a FL system with multiple clients, only with limited computation resources. However, it consumes more time to finish the whole FL experiment since the clients' real execution is one by one in serial. It is designed for simulation with limited computation resources. The paradigm of `SerialTrainer` is shown in figure 4(a).

Cross-Machine. FedLab also supports cross-machine FL simulation that's shown in figure 4(b). In practice, each role of FedLab is represented by single system process. FL system simulation can be executed on multiple machines with correct network topology configuration. More flexibly in parallel, SerialTrainer is able to replace the regular Trainer. In this way, machine with more computation resources can be assigned with more workload of simulating. The limitation of this scenario is that all machines must be in the same network (LAN or WAN).

Hierarchical. Users can break the limitation of **Cross-Machine** by using Scheduler to build client groups (a subset of clients sharing the same Scheduler), as depicted in figure 4(c). Server can communicate with client in LAN indirectly. A hierarchical FL system with K client groups as depicted in figure 4(c) can be easily formed using FedLab. More importantly, Scheduler is customizable for users. It can be applied for aggregating parameters from client group as a middle-server to share the communication and computation load of server. This design is for the scalability of framework in both computation and communication.

4 Pipeline and Examples

The pipeline of building a FL system with FedLab includes two parts. The first part is definition of communication agreements. The prototype of synchronous and asynchronous communication patterns have been implemented for users. With effortless modification on NetworkManager of client and server, users can fulfill the agreements as their will. The second part is ParameterServerHandler module of server and Trainer module of client, which represents FL optimization process. High level parameter aggregation algorithm and communication is available for server as well. In short, customizable interfaces and tools in FedLab support users to implement these two parts very quickly. We show the example implementation of FedAvg to demonstrate FedLab API's simplicity.

Core code of client is shown below:

```
1 model = ResNet()
2 optimizer = torch.optim.SGD(model.parameters(), lr=args.lr, momentum=0.9)
3 criterion = nn.CrossEntropyLoss()
4 trainloader, testloader = get_dataset(args)
5
6 handler = ClientSGDTrainer(model, trainloader, epoch=args.epoch, optimizer=optimizer, criterion=
      criterion, cuda=args.cuda)
7 network = DistNetwork(address=(args.server_ip, args.server_port),
                        world_size=args.world_size,
9                       rank=args.local_rank)
10
11 manager = ClientPassiveManager(handler=handler, network=network)
12 manager.run()
```

Code from line 1 to line 5 is the standard pipeline of training a neural network with PyTorch. From line 6 to the end is the usage of FedLab. In this example, FedLab provides high level API of network communication which allow users define network topology easily (line 7-11) and standard network training process (line 6).

FL server is also easily implemented in a couple lines of code:

```
1 model = ResNet()
2 ps = SyncParameterServerHandler(model, client_num_in_total=args.world_size-1)
3
4 network = DistNetwork(address=(args.server_ip, args.server_port),
5                       world_size=args.world_size,
6                       rank=0)
7 manager = ServerSynchronousManager(handler=ps, network=network)
8
9 manager.run()
```

Code in line 2 defines the ParameterServerHandler with FedAvg algorithm. Codes from line 4 to 7 define the NetworkManager of server.

5 Development

For continuous maintainence of FedLab, we establish a open-source group on GitHub. The framework will be further developed publicly through GitHub, in which we can track issues of bug reports, feature requests and usage questions. We use continuous integration (CI) to ensure robust of package. What's more, comprehensive and elaborate documentation is developed using popular Sphinx Python documentation generator and published on https://fedlab.readthedocs.io/en/latest/.

6 Summary and Future Work

In this paper, a flexible and lightweight FL framework `FedLab` is proposed. `FedLab` provides common-used FL communication patterns and optimization algorithms modules with both high-level API and open interfaces for standardized FL simulation. For easy usage and continuous maintainence, we build a open-source group to accept contributions and issues on GitHub with necessary configurations.

In the future, we will keep developing `FedLab`. Specifically, our plan includes but not limited to the following aspects:

- **Releasing research results**. We will use `FedLab` to explore our current and future ideas about optimization and communication. We will release those implementations on this framework in the future.
- **Providing more implementations**. Many excellent works are developed by different computation platform. Inconsistent implementations are not beneficial for the development of community. We plan to re-implement them with `FedLab` to provide more standard FL implementations.
- **Adding functional modules**. In the aspect of communication module, complicate network topology is under development. Besides, convenient network configuration script will be presented soon. Modules, which supporting other machine learning technique such as Unsupervised Learning, Semi-supervised Learning, Transfer Learning, etc,, are in schedule.

References

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[2] David Byrd and Antigoni Polychroniadou. Differentially private secure multi-party computation for federated learning in financial applications. *CoRR*, abs/2010.05867, 2020.

[3] Jie Xu, Benjamin S. Glicksberg, Chang Su, Peter B. Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *J. Heal. Informatics Res.*, 5(1):1–19, 2021.

[4] Theodora S. Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch. Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *Int. J. Medical Informatics*, 112:59–67, 2018.

[5] Yuang Jiang, Shiqiang Wang, Bong-Jun Ko, Wei-Han Lee, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *CoRR*, abs/1909.12326, 2019.

[6] Chuizheng Meng, Sirisha Rambhatla, and Yan Liu. Cross-node federated graph neural network for spatio-temporal data modeling. *CoRR*, abs/2106.05223, 2021.

[7] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *MLSys*. mlsys.org, 2020.

[8] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[9] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[10] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*, 2020.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[12] 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 *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[13] Wonyong Jeong, Jaehong Yoon, Eunho Yang, and Sung Ju Hwang. Fedmatch implementation in tensorflow. `https://github.com/wyjeong/FedMatch`, 2021.

[14] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Fedprox implementation in tensorflow. `https://github.com/litian96/FedProx`, 2021.

[15] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[16] Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, et al. Large scale distributed deep networks. 2012.

[17] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *NeurIPS*, pages 14747–14756, 2019.

[18] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *AISTATS*, volume 108 of *Proceedings of Machine Learning Research*, pages 2938–2948. PMLR, 2020.

[19] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *CCS*, pages 603–618. ACM, 2017.

[20] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*, pages 4387–4398. PMLR, 2020.

[21] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris S. Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *ICLR*. OpenReview.net, 2020.

[22] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[23] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *ICLR*. OpenReview.net, 2020.

[24] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for federated learning. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 5132–5143. PMLR, 2020.

[25] Michael Zhang, Karan Sapra, Sanja Fidler, Serena Yeung, and Jose M. Alvarez. Personalized federated learning with first order model optimization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[26] Yutao Huang, Lingyang Chu, Zirui Zhou, Lanjun Wang, Jiangchuan Liu, Jian Pei, and Yong Zhang. Personalized federated learning: An attentive collaboration approach. *CoRR*, abs/2007.03797, 2020.

[27] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*, 2019.

[28] Hyowoon Seo, Jihong Park, Seungeun Oh, Mehdi Bennis, and Seong-Lyun Kim. Federated knowledge distillation. *arXiv preprint arXiv:2011.02367*, 2020.

[29] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, 30:1709–1720, 2017.

[30] Tim Dettmers. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561*, 2015.

[31] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR, 2018.

[32] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *EMNLP*, pages 440–445. Association for Computational Linguistics, 2017.

[33] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. *arXiv preprint arXiv:1809.07599*, 2018.

[34] Yuanfeng Chen, Gaofeng Huang, Junjie Shi, Xiang Xie, and Yilin Yan. Rosetta: A Privacy-Preserving Framework Based on TensorFlow. `https://github.com/LatticeX-Foundation/Rosetta`, 2020.

[35] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning, 2018.

[36] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

[37] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[38] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[39] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *CoRR*, abs/1903.03934, 2019.

[40] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. *CoRR*, abs/2102.02079, 2021.

[41] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N. Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *ICLR*. OpenReview.net, 2021.

[42] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *CoRR*, abs/1812.01097, 2018.