FEDJAX: Federated learning simulation with JAX

Jae Hun Ro
Google Research
jaero@google.com

Ananda Theertha Suresh
Google Research
theertha@google.com

Ke Wu
Google Research
wuke@google.com

Abstract

Federated learning is a machine learning technique that enables training across decentralized data. Recently, federated learning has become an active area of research due to an increased focus on privacy and security. In light of this, a variety of open source federated learning libraries have been developed and released. We introduce FEDJAX, a JAX-based open source library for federated learning simulations that emphasizes ease-of-use in research. With its simple primitives for implementing federated learning algorithms, prepackaged datasets, models and algorithms, and fast simulation speed, FEDJAX aims to make developing and evaluating federated algorithms faster and easier for researchers. Our benchmark results show that FEDJAX can be used to train models with federated averaging on the EMNIST dataset in a few minutes and the Stack Overflow dataset in roughly an hour with standard hyperparameters using TPUs.

1 Introduction

Federated learning is a machine learning setting where many clients collaboratively train a model under the orchestration of a central server, while keeping the training data decentralized. Clients can be either mobile devices or whole organizations depending on the task at hand [Konečný et al., 2016b,a, McMahan et al., 2017, Yang et al., 2019]. Federated learning is typically studied in two scenarios: *cross-silo* and *cross-device*. In cross-silo federated learning, the number of clients is small, where as in cross-device, the number of clients is very large and can be in the order of millions. Figure 1 highlights key characteristics of most federated learning algorithms in the cross-device settings. Typically, federated learning algorithms first initialize the model at the server and then complete three key steps for each round of training:

1. The server selects a subset of clients to participate in training and sends the model to these clients.
2. Each selected client completes some steps of training on their local data.
3. After training, the clients send their updated models to the server and the server aggregates them together.

For example, Algorithm 1 illustrates the popular *federated averaging* algorithm [McMahan et al., 2017], which follows the above three steps. Federated learning has demonstrated usefulness in a variety of contexts, including next word prediction [Hard et al., 2018, Yang et al., 2018] and healthcare applications [Brisimi et al., 2018]. We refer to [Li et al., 2019a, Kairouz et al., 2021] for a more detailed survey of federated learning.

Federated learning poses several interesting challenges. For example, training typically occurs mostly on small devices, limiting the size of models that can be trained. Furthermore, the devices may have low communication bandwidth and require model updates to be compressed. Data is also distributed in a non-i.i.d. fashion across devices which raises several optimization questions. Finally, privacy

1st NeurIPS Workshop on New Frontiers in Federated Learning (NFFL 2021), Virtual Meeting.

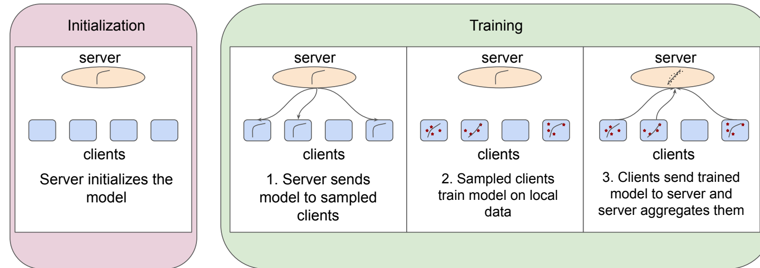


Figure 1: An example federated learning algorithm with four clients.

and security are of utmost importance in federated learning and addressing them requires techniques ranging from differential privacy to cryptography.

Given these challenges, federated learning has become an increasingly active area of research. This includes new learning scenarios [Mohri et al., 2019, Abay et al., 2020], optimization algorithms [Li et al., 2018, Yu et al., 2019, Li et al., 2019b, Haddadpour and Mahdavi, 2019, Khaled et al., 2020, Karimireddy et al., 2019, Ro et al., 2021], compression algorithms [Suresh et al., 2017, Caldas et al., 2018a, Xu et al., 2020], differentially private algorithms [Agarwal et al., 2018, Peterson et al., 2019, Sattler et al., 2019], cryptography techniques [Bonawitz et al., 2017], and algorithms that incorporate fairness [Li et al., 2020, Du et al., 2020, Huang et al., 2020]. Motivated by this, there are several libraries for federated learning, including, TensorFlow Federated [TFF, 2018], PySyft [Ryffel et al., 2018], FedML [He et al., 2020], FedTorch [Ludwig et al., 2020], and Flower [Ludwig et al., 2020].

Recently, JAX [Bradbury et al., 2018] was introduced to provide utilities to convert Python functions into Accelerated Linear Algebra (XLA) optimized kernels, where compilation and automatic differentiation can be composed arbitrarily. This enables expressiveness for sophisticated algorithms and efficient performance without leaving Python. Given its ease-of-use, several libraries to support machine learning have been built on top of JAX, including, but not limited to, Flax [Heek et al., 2020], Objax [Objax Developers, 2020], Jraph [Godwin* et al., 2020], and Haiku [Hennigan et al., 2020] for neural network architectures and training, Optax [Hessel et al., 2020] for optimizers, Chex [Budden et al., 2020a] for testing, and RLax [Budden et al., 2020b] for reinforcement learning.

We present FedJAX, a JAX and Python based library for federated learning simulation for research. FedJAX is designed for ease-of-use for research and is not intended to be deployed over distributed devices. Focusing on ease-of-use, the FedJAX API is structured to reduce the amount of new concepts that users have to learn to get started and comes packaged with several standard datasets, models, and algorithms that can be used straight out of the box. Additionally, since it is based on JAX, FedJAX can run on accelerators (GPU and TPU) with minimal additional effort.

The rest of the paper is organized as follows. In Section 2, we overview the system design, in Section 3, we demonstrate a sample federated learning algorithm with FedJAX, and in Section 4, we benchmark training with FedJAX on two datasets.

2 System design

A typical federated learning experiment consists of a federated dataset, model and optimizer, local client training strategy, and server aggregation strategy, so we structure FedJAX accordingly. The design was primarily driven by ease-of-use and performance when addressing the challenges uniquely attributed to using JAX for federated learning.

2.1 Datasets and models

Federated dataset In the context of federated learning, data is decentralized and distributed across clients, with each client having their own local set of examples. We refer to two levels of organization for datasets:

- Federated dataset: A collection of clients, each with their own local dataset and metadata.
- Client dataset: The set of local examples for a particular client.

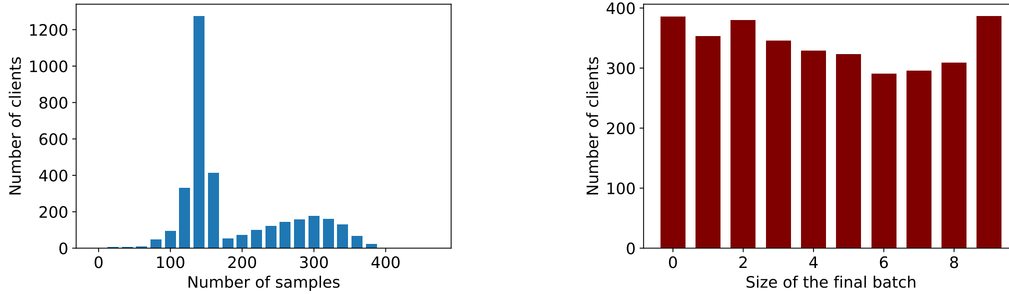
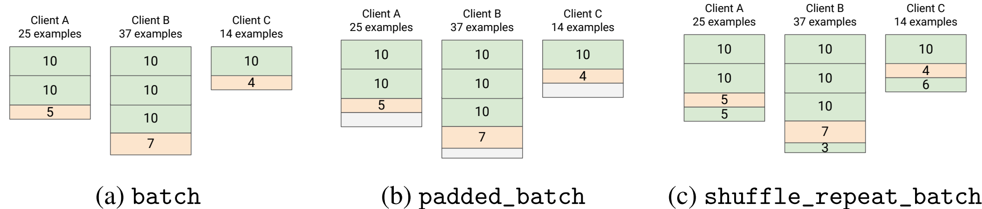


Figure 2: EMNIST dataset characteristics. Left: Histogram of number of clients as a function of number of samples. Right: Histogram of number of clients as a function of the size of the final batch, with batch size 10.



(a) `batch` (b) `padded_batch` (c) `shuffle_repeat_batch`

Figure 3: Resulting batches for three clients of various sizes with different batching strategies applied.

In its simplest form, federated datasets are just mappings from clients to their local examples. Specifically, clients have a unique identifier for querying their local dataset, which is essentially treated as a list of examples. Furthermore, these local client datasets are typically small as seen in the EMNIST example in Figure 2. This is in contrast to standard centralized machine learning which requires iterating over a single dataset in a large number of batches. With this difference in mind, we designed `fedjax.FederatedData` and `fedjax.ClientDataset` to rely mostly on NumPy and Python, making it easy to use and troubleshoot. Finally, in order to take full advantage of JAX, we also provide several helpful functions for accessing and iterating over federated and client datasets. We refer readers to the tutorial[1] for an overview of these functionalities.

JIT efficient batching Despite the small size of client datasets, there is no guarantee that their sizes will be the same. As a result, when we run batch evaluation on client datasets, the sizes of the final batch can vary greatly depending on the client, as shown in Figure 2. This poses a practical challenge when using JAX. For best performance, `jax.jit`[2] is used to perform Just In Time (JIT) compilation of a JAX Python function into XLA compiled machine code. `jax.jit` invokes the XLA compiler for each unique combination of input shapes. Left alone, the large number of possible final batch sizes results in excessive JIT recompilations, significantly slowing execution time. In response to this, we implemented three different batching strategies:

- `batch` produces batches in a fixed sequential order without padding the final batch. This is included mostly for illustration purposes.
- `padded_batch` produces padded batches in a fixed sequential order for evaluation. By padding the final batch to a small number of fixed sizes, we can set a limit on the maximum number of possible JIT recompilations. This is typically used in evaluation.
- `shuffle_repeat_batch` produces batches in a shuffled and repeated order for training where shuffling is done without replacement and batches are always the same size. This is typically used in training.

Figure 3 showcases the resulting batches using each of these batching strategies for different client dataset sizes.

[1]`https://fedjax.readthedocs.io/en/latest/notebooks/dataset_tutorial.html`
[2]`https://jax.readthedocs.io/en/latest/jax-101/02-jitting.html`

Model and optimizer The model and optimizer described in this section are unchanged between the standard centralized learning setting and the federated learning setting. There are already numerous JAX based libraries for neural networks and optimizers, such as Flax, Haiku, Objax, and Optax. Thus, for convenience, FEDJAX provides an implementation-agnostic wrapper to make porting existing models and optimizers into FEDJAX as simple as possible. For example, using an existing Haiku model in FEDJAX is as easy as wrapping the module in a `fedjax.Model`. We refer readers to the tutorial[3] for an overview of these functionalities.

Included datasets and models Currently in federated learning research, there are a variety of commonly used datasets and models, such as image recognition, language modeling, and more. A growing set of these datasets and models can be used straight out of the box in FEDJAX. This not only encourages valid comparisons between various federated algorithms but also accelerates the development of new algorithms since the preprocessed datasets and models are readily available for use and do not have to be written from scratch.

At present, FEDJAX comes packaged with the following datasets and sample models:

- EMNIST-62 [Caldas et al., 2018b], a character recognition task
- Shakespeare [McMahan et al., 2017], a next character prediction task
- Stack Overflow [Reddi et al., 2020], a next word prediction task

FEDJAX also provides tools to create new datasets and models that can be used with the rest of the library along with implementations of federated averaging [McMahan et al., 2017] and other algorithms, such as adaptive federated optimizers [Reddi et al., 2020], agnostic federated averaging [Ro et al., 2021], and Mime [Karimireddy et al., 2020].

Metrics When evaluating accuracy on a dataset, we wish to know the proportion of examples that are correctly predicted by the model. In federated learning specifically, we typically want to know the accuracy for each client and over all clients. Thus, we need to divide the work across clients, evaluate each separately, and finally somehow combine the results.

Assuming the metric evaluation logic is JIT compiled, this faces the same issue of excessive recompilations due to different subset sizes resulting in shape differences. While `padded_batch` can be used to address this issue, it will also result in batches padded with empty examples which should not be counted in the metric calculation. In an effort to improve ease-of-use while maintaining performance, we designed `fedjax.Metric` to be defined on a single example rather than a batch of examples. This way, the metric calculation can be freely vectorized with `jax.vmap`[4] and empty examples in padded batches ignored without users having to explicitly account for this themselves.

2.2 Client training and aggregators

Client training As mentioned previously, federated learning experiments typically include a step of training across decentralized and distributed clients. FEDJAX provides core functions for expressing this step. Because simulation is the focus, there is no need to introduce any design or technical overhead for distributed machine communication. Instead, the entire per-client work can run on a single machine, making the API simpler and the execution faster in most cases.

In its simplest form, a basic for-loop can be used for conducting training across multiple clients in FEDJAX. For even faster training speeds, FEDJAX provides the `fedjax.for_each_client` primitive backed by `jax.jit` and `jax.pmap`[5], which enables the simulation work to easily run on one or more accelerators such as GPU and TPU. We refer readers to the tutorial[6] for an overview of these functionalities. By defining client work in terms of `fedjax.for_each_client`, we are able to arbitrarily group cohorts of clients to be executed in parallel for greater performance without additional burden on the user.

[3] https://fedjax.readthedocs.io/en/latest/notebooks/model_tutorial.html
[4] https://jax.readthedocs.io/en/latest/jax.html?highlight=vmap#jax.vmap
[5] https://jax.readthedocs.io/en/latest/jax.html?highlight=pmap#jax.pmap
[6] https://fedjax.readthedocs.io/en/latest/notebooks/algorithms_tutorial.html

Algorithm 1 Federated averaging algorithm [McMahan et al., 2017]

procedure FEDERATEDAVERAGING
  T: total number of rounds, c: number of clients per round, η_s: server learning rate.
  Initialize parameters: w_0
  for round $t = 1$ to T **do**
    $C_t \leftarrow$ (random set of c clients)
    for client $k \in C_t$ **do**

$$\Delta_k, n_k \leftarrow \text{CLIENTUPDATE}(k, w_{t-1})$$

    end for
    $w_t \leftarrow w_{t-1} - \eta_{\text{s}} \frac{\sum_{k \in C_t} n_k \Delta_k}{\sum_{k' \in C_t} n_{k'}}$
  end for
end procedure

procedure CLIENTUPDATE(k, w)
  S_k: dataset of client k, B: batch size, E: Number of epochs, η_c: client learning rate.
  $w' \leftarrow w$
  $\mathcal{B} \leftarrow$ (split S_k into batches of size B)
  for epoch $e = 1$ to E **do**
    for batch $b \in \mathcal{B}$ **do**

$$w' \leftarrow w' - \eta_{\text{c}} \nabla \sum_{(x_i, y_i) \in b} L(w', x_i, y_i)$$

    end for
  end for
  return $w - w', |S_k|$
end procedure

Server aggregation The final step of server aggregation is often what differs the most significantly between federated learning algorithms. Thus, making this step as easily expressible and interpretable as possible is a core design goal of FEDJAX. This is achieved by providing basic underlying functions for working with model parameters, which are usually structured as pytrees[7] in JAX, as well as high-level pre-defined common aggregators in `fedjax.aggregators`. These aggregators can also be used to implement compression and differentially private federated learning algorithms.

3 Example

In this section, we demonstrate how to implement federated averaging with FEDJAX. Because FEDJAX only introduces a few core concepts and is clear and straightforward, code written in FEDJAX tends to resemble the pseudo-code used to describe novel algorithms in research papers, making it easy to get started with. While FEDJAX provides primitives for federated learning, they are not required and can be replaced with just NumPy and JAX. The advantage of building on top of JAX is that even the most basic implementations can still be reasonably fast.

Below, we walk through a simple example of federated averaging (Algorithm 1) for linear regression implemented in FEDJAX. The first steps are to set up the experiment by loading the federated dataset, initializing the model parameters, and defining the loss and gradient functions. The code for these steps is given in Figure 3.

```
import jax
import jax.numpy as jnp
import fedjax

# {"client_id": client_dataset}
federated_data = fedjax.FederatedData()
# Initialize model parameters
server_params = jnp.array(0.5)
# Mean squared error
mse_loss = lambda params, batch: jnp.mean(
        (jnp.dot(batch["x"], params) - batch["y"])**2)
# jax.jit for XLA and jax.grad for autograd
grad_fn = jax.jit(jax.grad(mse_loss))
```

Figure 4: Dataset and model initialization.

Next, we use `fedjax.for_each_client` to coordinate the training that occurs across multiple clients. For federated averaging, `client_init` initializes the client model using the server

[7] `https://jax.readthedocs.io/en/latest/pytrees.html`

model, `client_step` completes one step of local mini-batch SGD, and `client_final` returns the difference between the initial server model and the trained client model. By using `fedjax.for_each_client`, this work will run on any available accelerators and possibly in parallel because it is backed by `jax.jit` and `jax.pmap`. While this is already simple to implement, the same could also be written out as a basic for-loop over clients if desired. The code for these steps is given in Figure 5. This code implements the CLIENTUPDATE procedure of federated averaging from Algorithm 1.

```
# For-loop over clients with client learning rate 0.1
for_each_client = fedjax.for_each_client(
  client_init=lambda server_params, _: server_params,
  client_step=(
    lambda params, batch: params - grad_fn(params, batch) * 0.1),
  client_final=lambda server_params, params: server_params - params)
```

Figure 5: Client update.

Finally, we run federated averaging for 100 training rounds by sampling clients from the federated dataset, training across these clients using `fedjax.for_each_client`, and aggregating the client updates using weighted averaging to update the server model in Figure 6. This code implements the FEDERATEDAVERAGING procedure of federated averaging from Algorithm 1.

```
# 100 rounds of federated training
for _ in range(100):
  clients = federated_data.clients()
  client_updates = []
  client_weights = []
  for client_id, client_update in for_each_client(server_params, clients):
    client_updates.append(client_update)
    client_weights.append(federated_data.client_size(client_id))
  # Weighted average of client updates
  server_update = (
    jnp.sum(client_updates * client_weights) /
    jnp.sum(client_weights))
  # Server learning rate of 0.01
  server_params = server_params - server_update * 0.01
```

Figure 6: Server update and the federated learning algorithm.

4 Benchmarks

We benchmark the FEDJAX federated averaging implementation on the image recognition task for the federated EMNIST-62 dataset [Caldas et al., 2018b] and the next word prediction task for Stack Overflow [Reddi et al., 2020].

The EMNIST-62 dataset consists of 3400 writers and their writing samples, which are one of 62 classes (alphanumeric). Following Reddi et al. [2020], we train a convolutional neural network for 1500 rounds with 10 clients per round using federated averaging. We run experiments on GPU (a single NVIDIA V100) and TPU (a single TensorCore on a Google TPU v2) and report the final test accuracy, overall execution time, average training round duration, and full evaluation time in Table 1. We note that with a singleTensorCore, training takes under five minutes.

The Stack Overflow dataset consists of questions and answers from the Stack Overflow forum, grouped by username. This dataset consists of roughly 342K users in the train split and 204K in the test split. Following Reddi et al. [2020], we train a single layer LSTM for 1500 rounds with 50 clients per round using federated averaging. We run experiments on GPU (a single NVIDIA V100), TPU (a single TensorCore on a Google TPU v2) using only `jax.jit`, and multi-core TPU (eight TensorCores on a Google TPU v2) using `jax.pmap` and report results in Table 2. Benchmarks show

Table 1: Benchmark results on EMNIST with federated averaging.

Hardware	Test accuracy	Overall (s)	Average training round (s)	Full evaluation (s)
GPU	85.92%	418	0.26	7.21
TPU	85.85%	258	0.16	4.06

Table 2: Benchmark results on Stack Overflow with federated averaging.

Hardware	Test accuracy	Overall (m)	Average training round (s)	Full evaluation (m)
GPU	24.74%	127.2	4.33	17.32
TPU (`jax.jit`)	24.44%	106.8	3.73	11.97
TPU (`jax.pmap`)	24.67%	48.0	1.26	11.97

that with multiple TensorCores, recurrent language models can be trained in under an hour. Figure 7 also shows the average training round duration as the number of clients per round increases. We note that training with multiple TensorCores is substantially faster as the number of clients per round increases.

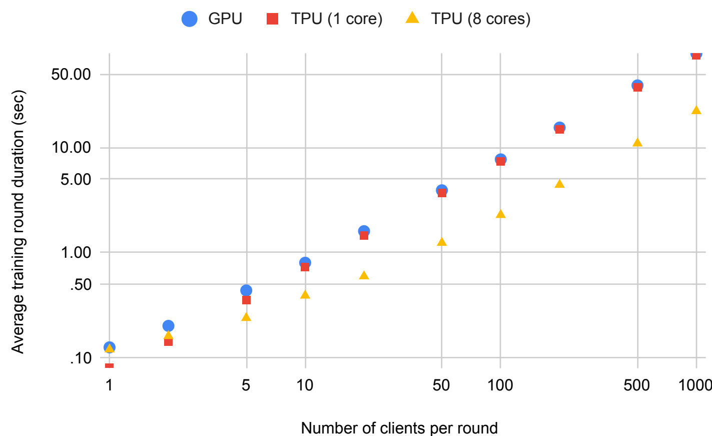


Figure 7: Stack Overflow average training round duration as the number of clients per round increases.

5 Conclusion

We introduced FEDJAX, a JAX-based open source library for federated learning simulations that emphasizes ease-of-use in research. FEDJAX provides simple primitives for federated learning along with a collection of canonical datasets, models, and algorithms to make developing and evaluating federated algorithms easier and faster. Implementing additional algorithms, datasets, and models remains an on-going effort.

Acknowledgments and Disclosure of Funding

Authors thank Sai Praneeth Kamireddy for contributing to the library and various discussions early on in development.

Authors also thank Ehsan Amid, Theresa Breiner, Mingqing Chen, Fabio Costa, Roy Frostig, Zachary Garrett, Satyen Kale, Rajiv Mathews, Lara Mcconnaughey, Brendan McMahan, Mehryar Mohri, Krzysztof Ostrowski, Max Rabinovich, Michael Riley, Gary Sivek, Jane Shapiro, Luciana Toledo-Lopez, and Michael Wunder for helpful comments and contributions.

References

Annie Abay, Yi Zhou, Nathalie Baracaldo, Shashank Rajamoni, Ebube Chuba, and Heiko Ludwig. Mitigating bias in federated learning, 2020.

Naman Agarwal, Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, and Brendan McMahan. cpSGD: Communication-efficient and differentially-private distributed SGD. In *Proceedings of NeurIPS*, pages 7575–7586, 2018.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

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. *International journal of medical informatics*, 112:59–67, 2018.

David Budden, Matteo Hessel, Iurii Kemaev, Stephen Spencer, and Fabio Viola. Chex: Testing made fun, in jax!, 2020a. URL `http://github.com/deepmind/chex`.

David Budden, Matteo Hessel, John Quan, Steven Kapturowski, Kate Baumli, Surya Bhupatiraju, Aurelia Guy, and Michael King. RLax: Reinforcement Learning in JAX, 2020b. URL `http://github.com/deepmind/rlax`.

Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018a.

Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018b.

Wei Du, Depeng Xu, Xintao Wu, and Hanghang Tong. Fairness-aware agnostic federated learning, 2020.

Jonathan Godwin*, Thomas Keck*, Peter Battaglia, Victor Bapst, Thomas Kipf, Yujia Li, Kimberly Stachenfeld, Petar Veličković, and Alvaro Sanchez-Gonzalez. Jraph: A library for graph neural networks in jax., 2020. URL `http://github.com/deepmind/jraph`.

Farzin Haddadpour and Mehrdad Mahdavi. On the convergence of local descent methods in federated learning. *arXiv preprint arXiv:1910.14425*, 2019.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

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.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL `http://github.com/google/flax`.

Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL `http://github.com/deepmind/dm-haiku`.

Matteo Hessel, David Budden, Fabio Viola, Mihaela Rosca, Eren Sezener, and Tom Hennigan. Optax: composable gradient transformation and optimisation, in jax!, 2020. URL `http://github.com/deepmind/optax`.

Wei Huang, Tianrui Li, Dexian Wang, Shengdong Du, and Junbo Zhang. Fairness and accuracy in federated learning, 2020.

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2021.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for on-device federated learning. *arXiv preprint arXiv:1910.06378*, 2019.

Sai Praneeth Karimireddy, Martin Jaggi, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Mime: Mimicking centralized stochastic algorithms in federated learning. *arXiv preprint arXiv:2008.03606*, 2020.

Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. Tighter theory for local SGD on indentical and heterogeneous data. In *Proceedings of AISTATS*, 2020.

Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016a.

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*, 2016b.

Tian Li, Anit Kumar Sahu, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873*, 2019a.

Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ByexElSYDr`.

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*, 2019b.

Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, et al. Ibm federated learning: an enterprise framework white paper v0. 1. *arXiv preprint arXiv:2007.10987*, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of AISTATS*, pages 1273–1282, 2017.

Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 4615–4625. PMLR, 2019.

Objax Developers. Objax, 2020. URL `https://github.com/google/objax`.

Daniel Peterson, Pallika Kanani, and Virendra J Marathe. Private federated learning with domain adaptation. *arXiv preprint arXiv:1912.06733*, 2019.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization, 2020.

Jae Ro, Mingqing Chen, Rajiv Mathews, Mehryar Mohri, and Ananda Theertha Suresh. Communication-efficient agnostic federated averaging. *arXiv preprint arXiv:2104.02748*, 2021.

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.

Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multi-task optimization under privacy constraints. *arXiv preprint arXiv:1910.01991*, 2019.

Ananda Theertha Suresh, Felix X Yu, Sanjiv Kumar, and H Brendan McMahan. Distributed mean estimation with limited communication. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3329–3337. JMLR. org, 2017.

TFF. Tensorflow federated, 2018. URL `https://www.tensorflow.org/federated`.

Jinjin Xu, Wenli Du, Yaochu Jin, Wangli He, and Ran Cheng. Ternary compression for communication-efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5693–5700, 2019.