RayLEAF: Benchmarking Compressed Federated Models

Ryan Panwar

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-204
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-204.html

August 18, 2021

RayLEAF: Benchmarking Compressed Federated Models

RayLEAF: Benchmarking Compressed Federated Models

by

Ryan Panwar

A thesis submitted in partial satisfaction of the

requirements for the degree of

Masters of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor James Demmel, Chair
Professor Martin Wainwright

Spring 2021

RayLEAF: Benchmarking Compressed Federated Models

by Ryan Panwar

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

James Demmel

Professor James Demmel
Research Advisor

12 August 2021

(Date)

* * * * * * *

[illegible]

Professor Martin Wainwright
Second Reader

08/16/21

(Date)

Abstract

RayLEAF: Benchmarking Compressed Federated Models

by

Ryan Panwar

Masters of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor James Demmel, Chair

Privacy and communication cost concerns have led to interest in federated learning (FL) for edge machine learning applications. While standard machine learning has a rich ecosystem of distributed training libraries, federated learning's novelty means researchers lack the necessary frameworks to efficiently explore the large design space unique to FL. In this work we propose, build, and evaluate a benchmarking framework for FL algorithms. Our system, RayLEAF, allows users to train FL algorithms in parallel while testing model compression approaches in a simulated federated setting.

Contents

List of Figures

List of Tables

Acknowledgments

This work would not be possible without the learning, mentorship, and support I have received throughout my four years at Berkeley from my professors, peers, friends, and family. First and foremost I would like to thank my advisor Professor Jim Demmel for his invaluable help the past two years. This work was also done in collaboration with Robert Zangnan Yu, Arissa Wongpanich, and Vivek Bharadwaj, who have helped build RayLEAF and test its capabilities. I would also like to thank the other members of our research group who have assisted in this work, especially Professor Martin Wainwright for being my second reader.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Finally, I would like to thank my parents for enabling me to embark on this project and supporting me through its completion.

Chapter 1

Introduction

1.1 Learning at the Edge

Every year more and more problems are attacked with machine learning. Advances in computer vision and natural language processing, among others, have yielded many interesting applications [27] [7]. However, increasingly the problems we are interested in applying machine learning to reside on the edge. Devices like smartphones, smartwatches, cars, and sensors take in large volumes of data that we would like to analyze. The standard approach for such edge learning applications is to upload the data to a centralized server and perform machine learning in the cloud. However, this solution runs into two key challenges.

First, the large volumes of data at the edge can present a challenge for network constraints. Computer vision applications may require recording HD video that is infeasible to upload to a central server. Applications like self driving cars may record many hours of video a day across several cameras, all while on a wireless connection [3]. As the volume of data collection grows, this will increasingly become a bottleneck for applying machine learning at the edge.

However, even for applications where the volume of data is small, like text-based tasks, privacy concerns may arise due to the sensitive nature of much data collected on edge devices. Consumers are increasingly aware of the privacy risks of uploading their pictures, videos, texts, and more to the cloud. Being able to deliver intelligent user experiences without compromising privacy is a major challenge for edge software providers.

To deal with these issues, researchers have proposed a new model of machine learning, termed federated learning.

1.2 Federated Learning

Overview of Federated Learning

To avoid uploading data to central servers, federated learning works by performing all training locally on-device, and only sending model updates to the server. Thus all user data is kept local to the device, and the only communication between client and server involves the model parameters. Ideally, model parameters are both less voluminous than the data they are trained on, and contain less private information.

More formally, let $W_t \in \mathbb{R}^{d_1 \times d_2}$ be a weight matrix encoding the global model parameters after t iterations. Then each local model i computes an updated weight matrix W_t^i based on its local dataset S_i. To compute the next global model, we average across the local updates weighting by the size of their local datasets $n_i = |S_i|$ as follows:

$$W_{t+1} = W_t + \sum_i \frac{n_i}{n}(W_t^i - W_t) \tag{1.1}$$

Here $n = \sum_i n_i$, the total number of data points across all clients. Typically W_t^i is computed by performing E epochs of stochastic gradient descent on the data S_i, with mini-batches of size B. This processes is known as federated averaging or FedAvg [19]. In the case where a single gradient update is performed encompassing all the data in S_i, which is effectively $E = 1$ and $B = \infty$, this is called Federated Stochastic Gradient Descent, or FedSGD. Other techniques for updating W_t and W_t^i have been explored, such as FedProx [16] and FedSplit [23].

Model Compression

Although federated learning removes the necessity to upload large amounts of training data, it does not completely eliminate network constraints as a concern. As machine learning models grow larger and larger, model sizes themselves can become a potential issue for both network and device memory capacity constraints. With state of the art NLP models like GPT-3 requiring hundreds of billions of parameters [4], finding ways to compress models is key to reducing the amount of data communicated between sever and client.

Several approaches have been tried for model compression, as catalogued by Konecny et. al [14]. These techniques include low rank learning, random masks, subsampling, and quantization. Other approaches like ternary encoding [26] and deep gradient compression [17] offer additional directions for compression algorithms. These algorithms all attempt to reduce the communication overhead while preserving as much model accuracy as possible. Since reducing the size of a model update invariably drops important information, ensuring the model accuracy remains high while compressing the model update is a challenging problem.

1.3 Contribution

There is a large design space for federated learning algorithms. From choosing the averaging algorithm, to the model architecture and hyperparameters, to the compression algorithm, to privacy algorithms [1] [2], federated learning research and design is a many-step process that requires researchers and developers to run many federated learning experiments to test their hypotheses and identify optimal configurations. Unfortunately, the existing frameworks for distributed machine learning are poor fits for federated learning. Existing frameworks such as Tensorflow [18] or Pytorch's [22] native distributed libraries, or external libraries such as Horovod [24] assume a single homogeneous dataset. Since a key challenge of federated learning involves learning on heterogeneous data across clients and averaging the results, a framework that does not allow users to simulate this does not expose users to the full complexity of the federated setting.

Thus, federated learning-specific training frameworks are essential for proper benchmarking and evaluation of federated learning models. Recent interest in such frameworks has led to the development of frameworks like FedML [11] and LEAF [5], allowing users to train a variety of models on federated datasets with different hyperparameter settings. LEAF is an important advance for federated learning research, but it lacks scalability and compression support. While FedML allows for distributed training, it too lacks compression support.

We set out to build an extended version of LEAF, termed RayLEAF, that scales the basic LEAF framework by parallelizing it using the Ray distributed computing library [21]. As part of RayLEAF, we also built support for testing multiple compression algorithms, with an easy interface to add new compression algorithms for benchmarking. RayLEAF thus takes all the great features of LEAF and adds on scalability and compression support for a powerful federated learning research tool.

The rest of this thesis is organized as follows: in Chapter 2 we first present an overview of the architecture of LEAF and RayLEAF. This is followed with experimental data benchmarking the performance and scaling of RayLEAF. In Chapter 3 we take a look at various compression approaches used in federated learning and show data from testing them on RayLEAF. Finally we end with a summary and overview of some future directions for RayLEAF and federated learning algorithms built on it.

Chapter 2

Parallel Training System Design

2.1 The LEAF Architecture

Our work builds on top of the LEAF system and its client-server architecture that simulates a federated learning system. LEAF simulates the training of federated models by training clients separately and aggregating them together at the server. LEAF also computes statistics about the distribution of test and train performance across clients, as well as the communication usage. This makes LEAF a useful benchmarking tool for understanding how accuracy trades off against communication across clients. Because LEAF already has these major features necessary for measuring the performance of federated learning algorithms, we decided that the best approach to build a scalable benchmarking framework was to build on top of LEAF's flexible system.

As shown in Fig. 2.1, in LEAF the federated learning problem is split into one server and m clients, as determined by the dataset configuration. Each client i has its own subset of the data S_i. At each round, a certain percentage of the clients are randomly sampled to participate in training. These clients sequentially train a single Tensorflow model on their datasets S_i, and each report their updated model back to the server which averages across the client updates to update the global model. Thus, the single Tensorflow model forms a bottleneck for training, significantly slowing down large-scale training of many clients.

2.2 Adding Ray

We extended LEAF using Ray to make RayLEAF, allowing for parallel training of many clients at once. Ray is a distributed computing framework that uses an actor model for programming stateful distributed systems. Ray's backend handles cluster management, allowing us to run seamlessly on either a cloud environment or an HPC system. Because of its scalability and actor programming model Ray is a natural fit for distributed learning systems such as LEAF.

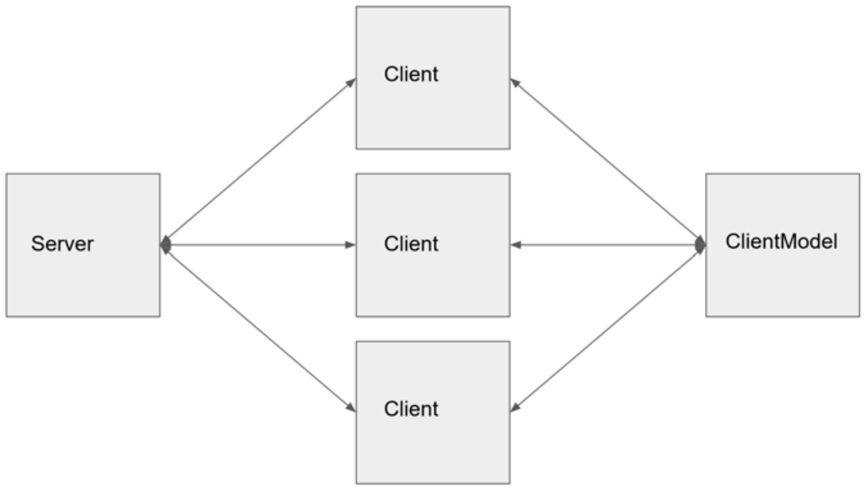

Figure 2.1: The client-server architecture of LEAF

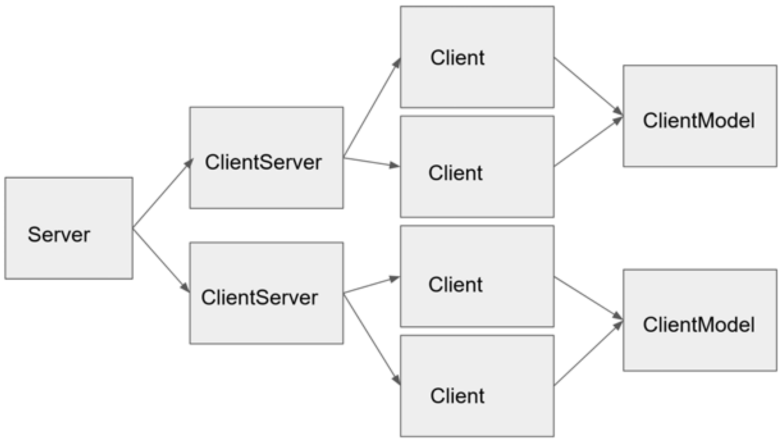

Figure 2.2: Updated architecture of RayLEAF.

To allow users to scale the amount of parallelism based on their needs and resources, we created a ClientServer abstraction that serves as the Ray remote actor and contains many Clients with their training data. Each ClientServer has its own TensorFlow model, allowing us to train clients in parallel by spreading the computation across many ClientServers. By increasing or decreasing the number of ClientServers the user can adjust the parallelism to their needs and resources. This architecture is shown in Fig. 2.2, where the server sends training commands to multiple ClientServers, which in turn select from their subset of clients to train. Since we do not require each client to be its own actor, this hierarchical approach allows for training on datasets involving large numbers of clients even with limited resources.

2.3 Scalability Results

Experiment Setup

We tested RayLEAF on the NERSC Cori KNL (Intel Knights Landing) cluster [6]. Cori is a Cray XC40 supercomputer with 9,688 KNL nodes and 68 physical cores and 112 GB memory per node. Our dataset was the standard Federated Extended MNIST (FEMNIST) dataset [8] used by LEAF and other popular benchmarks. We used a simple convolutional neural network similar to the ImageNet model [15]. This model consists of two convolutional layers with pooling, two fully connected layers, and a softmax output layer.

Results

Varying the number of clients involved in parallel training in 2.3, we found that RayLEAF scales imperfectly. However, a large part of this is due to the dataset load imbalance, as seen in Fig. 2.5. Comparing against the largest ClientServer, we find that RayLEAF has a relatively small overhead. We are still able to achieve substantial overall speedups in training time despite this load imbalance.

If we expand the number of clients being trained per round while we increase the number of ClientServers to benchmark the system's weak scaling, we find that it scales to run on 32x the data with only a 3x slowdown, as shown in Fig. 2.4. Again, we see very low overhead with most of the slowdown caused by the uneven data distribution - as seen in Fig. 2.5 some clients have significantly more data to train, and thus the max training time per client increases when we sample more clients. This load imbalance is an inherent feature of the federated learning problem, as real-world federated systems train on the data of devices that may have very different size datasets.

We also tracked the model training and test loss across different ClientServer setups to verify that the amount of parallelism used has no effect on overall model accuracy. Fig. 2.6 confirms that, excluding minor differences due to randomization in the client sampling, the model performs consistently regardless of the number of ClientServers used. Overall,

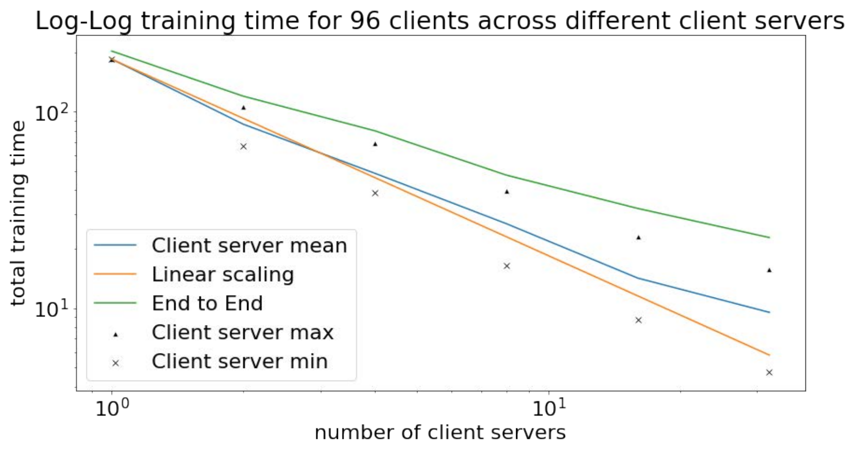

Figure 2.3: Strong Scaling

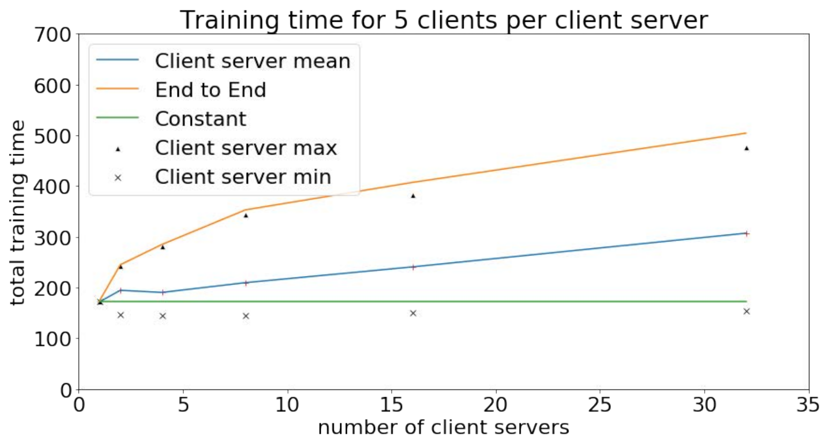

Figure 2.4: Weak Scaling

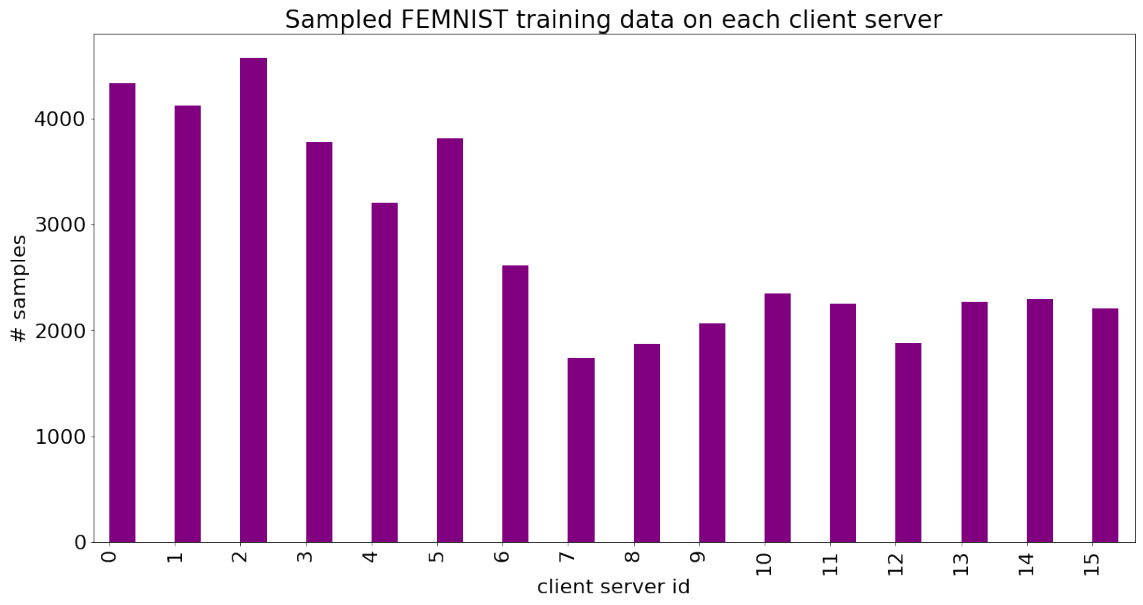

Figure 2.5: Number of samples varies significantly across clients

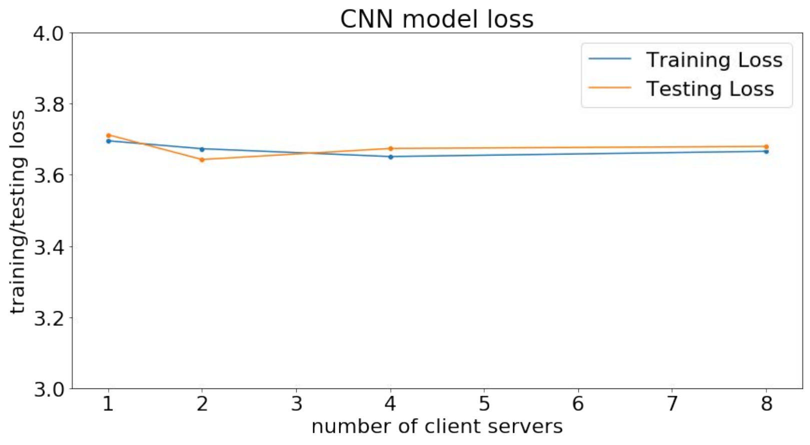

Figure 2.6: Test and training accuracy are consistent as we scale parallelism

our experimental results demonstrate that we are able to maintain model correctness while achieving substantial speedups to training.

Chapter 3

Compression Experiment Results

3.1 Introduction

The RayLEAF API allows for users to easily add their own compression plugins in between the client and server. When each client finishes training, it computes the delta between the new weights and the server's weights, and compresses that delta with the user-specified compression algorithm. The compressed weights are then sent to the server, where they are decompressed, averaged, and finally added to the server weights to update the global model. Formally, if c is the compression function and d is the corresponding decompression function then we update the global model as follows:

$$W_{t+1} = W_t + \sum_i \frac{n_i}{n} d(c(W_t^i - W_t)) \tag{3.1}$$

We perform compression on the delta $W_t^i - W_t$ in order to better take advantage of sparsity in the training process, as not all values in W_t^i will be updated from W_t. RayLEAF measures the overall amount of communication between client and server, the number of bytes $f(W_t^i - W_t)$ takes up, in order to benchmark compression performance. RayLEAF thus enables compression algorithm developers to minimize communication costs while maintaining model accuracy.

3.2 Models and Datasets

We evaluated compression algorithms on two very different datasets to demonstrate the ability of RayLEAF to test compression approaches across different domains. One dataset is the FEMNIST dataset used for general scaling experiments, with the same ImageNet model. Since this is a vision dataset, we also tested on the Reddit natural language dataset [5] with a stacked LSTM model [12]. The Reddit dataset involves next-work prediction on up to 50 million Reddit comments. This large scale provides a good example of the problems RayLEAF is well-suited for.

Layer	**Dimensions**	**Memory**
Conv. 1	25×32	6.4 KB
Conv. 2	800×64	409.6 KB
Dense 1	3136×2048	51.4 MB
Dense 2	2048×62	1.01 MB

Table 3.1: The FEMNIST CNN Model

Layer	**Dimensions**	**Memory**
Embedding	10000×256	20.48 MB
LSTM 1	512×1024	4.2 MB
LSTM 2	512×2024	4.2 MB
Dense	10000×256	20.48 MB

Table 3.2: The Reddit LSTM Model

For the FEMNIST CNN the largest layer is the first dense layer, which involves a 3136×2048 weight matrix taking a total of 51.4 MB assuming double precision, which is 97% of the overall model memory footprint. Since this layer is the clear communication bottleneck, we focus on just compressing it instead of the entire model.

The stacked LSTM model embeds from a 10000 word vocabulary to a 256 dimensional space, passes through two LSTM cells, and then goes through a fully connected layer to project back to the vocabulary size. Due to the large vocabulary size the LSTM cells take up less than 20% of the overall model size, and as such much of the memory cost of the stacked LSTM model is roughly evenly split between the embedding layer and the dense layer, each involving a 10000×256 matrix taking up 20 MB. For simplicity we mainly look at compressing the final dense layer.

In all experiments we use the federated averaging algorithm [19].

3.3 Subsampling

One compression approach we test is subsampling. Subsampling involves randomly sampling values from the update matrix $W_t^i - W_t$ to send back. This can yield substantial communication savings by zeroing out many values from the update matrix. The values we do send back are corrected so that we send the correct value for all matrix entries in expectation. Thus by averaging across a large number of clients we expect to recover accurate approximations of the updates we are learning. In other words, if we are compressing by a factor of r, such that the compressed data is $\frac{1}{r}$ times as large as the original where $r \geq 1$, for matrix entry w_{jk} and a randomly generated float $q \in [0, 1)$ we send:

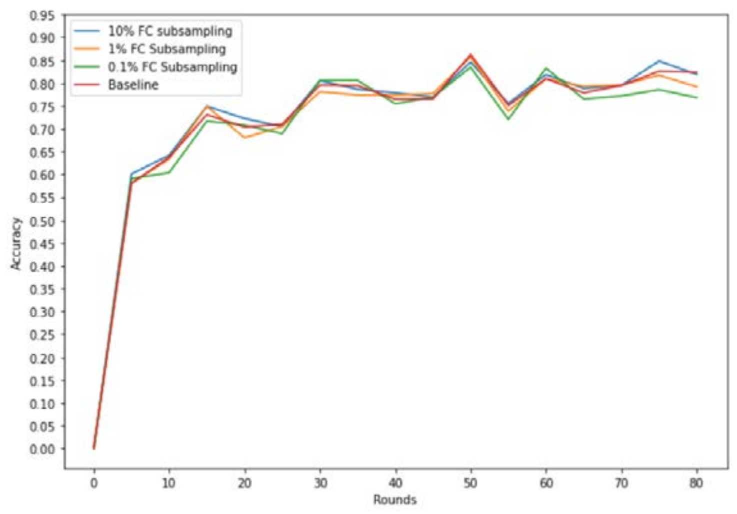

Figure 3.1: Subsampling test accuracy for FEMNIST

$$c(w_{jk}) = \begin{cases} rw_{jk} & q \le \frac{1}{r} \\ 0 & q > \frac{1}{r} \end{cases} \tag{3.2}$$

Thus $\mathbb{E}[d(c(w_{jk}))] = w_{jk}$ if the decompression function d is simply the identity function, meaning there is no need for additional decompression.

We find that this approach allows us to compress matrices significantly, while still retaining most of the model performance. Fig. 3.1 demonstrates this, as it plots the test accuracy for various compression levels and shows that 10x compression on FEMNIST is possible with essentially no accuracy loss. In fact, even 100x compression is possible with accuracy degradation within a few percent. However, we find that this does not hold when transferred to Reddit. Fig. 3.2 shows a model applying 10% subsampling to the final dense layer of the stacked LSTM has a difficult time learning anything at all, with its loss declining only slightly from the start and showing significant variability round to round. Clearly, subsampling as a compression technique does not offer universal value, and thus RayLEAF's ability to easily benchmark compression algorithms across datasets and models allows for developers to match the right compression with the right problem.

3.4 Singular Value Decomposition

Another compression approach we evaluated using RayLEAF was the Singular Value Decomposition (SVD). The SVD is a standard dimensionality reduction technique in linear algebra.

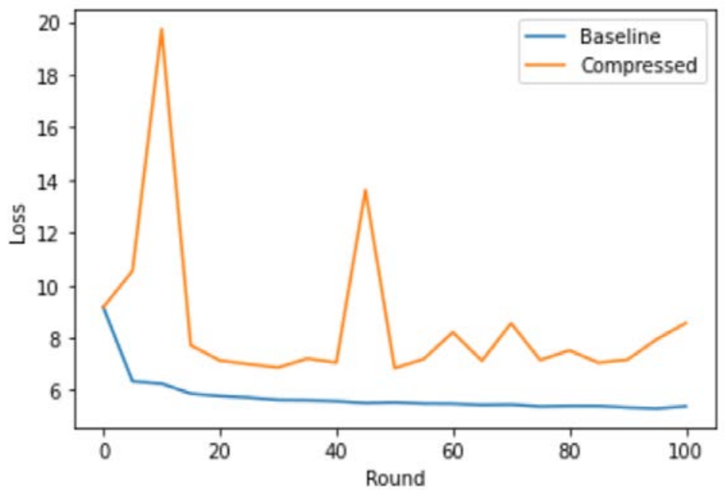

Figure 3.2: Subsampling test loss for Reddit

The SVD works by decomposing the matrix W_t^i into a matrix product $U\Sigma V^T$ where Σ is diagonal. By truncating the bottom singular values and leaving only the k largest, we get:

$$W_t^i \approx U_k \Sigma_k V_k^T \tag{3.3}$$

Here U_k is the k leftmost columns of U, Σ_k is the upper left $k \times k$ submatrix of Σ, and V_k is the upper k rows of V. Effectively we are dropping the lowest information parts of W_t^i. Thus by sending U_k, Σ_k, and V_k^T we are able to achieve substantial communication savings versus sending the full size W_t^i.

Fig. 3.3 shows the trade off between accuracy and compression as we increase the rank used by the SVD algorithm. It contains data for two SVD algorithms we have tested on RayLEAF, both a standard deterministic algorithm and a faster randomized approximate algorithm [10]. Due to the long-tale nature of the singular values, much of the information from the weight update is associated with the first few singular values, and we can truncate later singular values for large compression savings with relatively little impact on the model accuracy. For example, truncating after rank 64 yields a $\frac{2056}{64} = 16$x communication saving while maintaining accuracy within 2 percent of the 80% baseline.

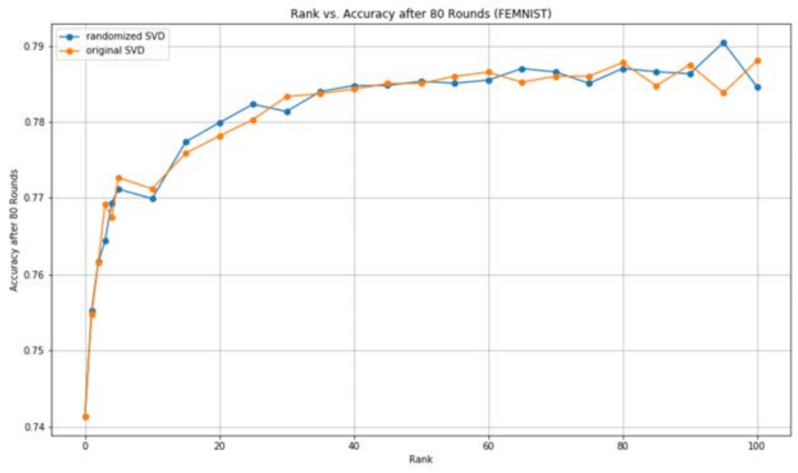

Figure 3.3: Accuracy attained for different SVD rank values for FEMNIST

Chapter 4

Future Work and Conclusion

Additional support for other areas of the federated learning design space are natural next steps for RayLEAF. Privacy in particular is a major area, as researchers applying differential privacy [9] to federated learning seek ways to add noise [25] [1] [2] to create privacy guarantees. Another area RayLEAF currently lacks support is for multiple precision types. Experiments with integer precision [28], mixed precision [20], and even newly proposed precision types such as bfloat16 [13] show promise for training ML models with lower compute and communication costs while maintaining performance. Expanding beyond FedAvg and FedSGD to other optimization algorithms such as FedProx [16] and FedSplit [23] are also good future directions.

Overall, RayLEAF provides a scalable framework for training and benchmarking federated learning algorithms and compression approaches. RayLEAF combines the federated design of LEAF with Ray's scalability to enable federated learning researchers to explore the large federated design space and profile the performance of different approaches in terms of both accuracy and communication. As an increasing number of machine learning models run at the edge, understanding the accuracy and communication tradeoffs are essential to developing the right architectures. RayLEAF's compression framework, as demonstrated by our experiments with subsampling and singular value decompositions, helps researchers understand how different compression algorithms affect model performance. We hope this system will be a valuable tool in making edge machine learning smarter and less bandwidth-hungry while protecting user privacy.

Bibliography

[1] Martin Abadi et al. “Deep Learning with Differential Privacy”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Oct. 2016). DOI: `10.1145/2976749.2978318`. URL: `http://dx.doi.org/10.1145/2976749.2978318`.

[2] Abhishek Bhowmick et al. *Protection Against Reconstruction and Its Applications in Private Federated Learning*. 2019. arXiv: `1812.00984 [stat.ML]`.

[3] Mariusz Bojarski et al. *End to End Learning for Self-Driving Cars*. 2016. arXiv: `1604.07316 [cs.CV]`.

[4] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: `2005.14165 [cs.CL]`.

[5] S. Caldas et al. “LEAF: A Benchmark for Federated Settings”. In: *ArXiv* abs/1812.01097 (2018).

[6] National Energy Research Scientific Computing Center. *Cori*. URL: `https://docs.nersc.gov/systems/cori/`. (accessed: 8.11.2021).

[7] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: `2107.03374 [cs.LG]`.

[8] Gregory Cohen et al. *EMNIST: an extension of MNIST to handwritten letters*. 2017. arXiv: `1702.05373 [cs.CV]`.

[9] C. Dwork. “Differential Privacy”. In: *ICALP*. 2006.

[10] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*. 2010. arXiv: `0909.4061 [math.NA]`.

[11] Chaoyang He et al. *FedML: A Research Library and Benchmark for Federated Machine Learning*. 2020. arXiv: `2007.13518 [cs.LG]`.

[12] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: `10.1162/neco.1997.9.8.1735`.

[13] Dhiraj Kalamkar et al. *A Study of BFLOAT16 for Deep Learning Training*. 2019. arXiv: `1905.12322 [cs.LG]`.

[14] Jakub Konecný et al. *Federated Learning: Strategies for Improving Communication Efficiency*. 2017. arXiv: `1610.05492 [cs.LG]`.

[15] A. Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60 (2012), pp. 84–90.

[16] Tian Li et al. *Federated Optimization in Heterogeneous Networks*. 2020. arXiv: `1812.06127 [cs.LG]`.

[17] Yujun Lin et al. *Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training*. 2020. arXiv: `1712.01887 [cs.CV]`.

[18] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[19] H. Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017. JMLR: W&CP volume 54* (2017). arXiv: `1602.05629v3 [cs.LG]`.

[20] Paulius Micikevicius et al. *Mixed Precision Training*. 2018. arXiv: `1710.03740 [cs.AI]`.

[21] Philipp Moritz et al. "Ray: A Distributed Framework for Emerging AI Applications". In: *OSDI*. 2018.

[22] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *NeurIPS*. 2019.

[23] Reese Pathak and Martin J. Wainwright. *FedSplit: An algorithmic framework for fast federated optimization*. 2020. arXiv: `2005.05238 [cs.LG]`.

[24] Alexander Sergeev and Mike Del Balso. "Horovod: fast and easy distributed deep learning in TensorFlow". In: *ArXiv* abs/1802.05799 (2018).

[25] Kang Wei et al. "Federated Learning With Differential Privacy: Algorithms and Performance Analysis". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3454–3469. DOI: `10.1109/TIFS.2020.2988575`.

[26] Wei Wen et al. *TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning*. 2017. arXiv: `1705.07878 [cs.LG]`.

[27] Chong Yu and Jeff Pool. *Self-Supervised GAN Compression*. 2020. arXiv: `2007.01491 [cs.LG]`.

[28] Feng Zhu et al. *Towards Unified INT8 Training for Convolutional Neural Network*. 2019. arXiv: `1912.12607 [cs.LG]`.