Federated Learning of Deep Networks using Model Averaging

H. Brendan McMahan MCMAHAN@GOOGLE.COM
Eider Moore EIDERM@GOOGLE.COM
Daniel Ramage DRAMAGE@GOOGLE.COM
Blaise Agüera y Arcas BLAISEA@GOOGLE.COM
Google, Inc., 651 N 34th St., Seattle, WA 98103 USA

Abstract

Modern mobile devices have access to a wealth of data suitable for learning models, which in turn can greatly improve the user experience on the device. For example, language models can improve speech recognition and text entry, and image models can automatically select good photos. However, this rich data is often privacy sensitive, large in quantity, or both, which may preclude logging to the data-center and training there using conventional approaches. We advocate an alternative that leaves the training data distributed on the mobile devices, and learns a shared model by aggregating locally-computed updates. We term this decentralized approach *Federated Learning*.

We present a practical method for the federated learning of deep networks that proves robust to the unbalanced and non-IID data distributions that naturally arise. This method allows high-quality models to be trained in relatively few rounds of communication, the principal constraint for federated learning. The key insight is that despite the non-convex loss functions we optimize, parameter averaging over updates from multiple clients produces surprisingly good results, for example decreasing the communication needed to train an LSTM language model by two orders of magnitude.

1. Introduction

As datasets grow larger and models more complex, machine learning increasingly requires distributing the optimization of model parameters over multiple machines, e.g., (Dean et al., 2012). Many algorithms exist for distributed optimization, but these algorithms typically have communication requirements that realistically are only satisfied by a data-center-grade network fabric. Further, the theoretical justification and practical performance for these algorithms rests heavily on the assumption the data is IID (independently and identically distributed) over the compute nodes. Taken together, these requirements amount to an assumption that the full training dataset is controlled by the modeler and stored in a centralized location.

A parallel trend is the rise of phones and tablets as primary computing devices for many people. The powerful sensors present on these devices (including cameras, microphones, and GPS), combined with the fact these devices are frequently carried, means they have access to data of an unprecedentedly private nature. Models learned on such data hold the promise of greatly improving usability by powering more intelligent applications, but the sensitive nature of the data means there are risks and responsibilities to storing it in a centralized location.

We investigate a learning technique that allows users to collectively reap the benefits of shared models trained from this rich data, without the need to centrally store it. This approach also allows us to scale up learning by utilizing the cheap computation available at the edges of the network. We term our approach *Federated Learning*, since the learning task is solved by a loose federation of participating devices (which we refer to as *clients*) which are coordinated by a central *server*. Each client has a local training dataset which is never uploaded to the server. Instead, each client computes an update to the current global model maintained by the central server, and only this update is communicated. This is a direct application of the principle of *focused collection* or *data minimization* as outlined in the Consumer Privacy Bill of Rights (White House Report, 2013). Since these updates are specific to improving the current model, there is no reason to store them once they have been applied.

We introduce the `FederatedAveraging` algorithm, which combines local SGD training on each client with communication rounds where the central server performs model averaging. We perform extensive experiments on this algorithm, demonstrating it is robust to unbalanced and

non-IID data distributions, and can reduce the rounds of communication needed to train a deep network by one to two orders of magnitude.

1.1. Federated Learning

What tasks are best suited to federated learning? The ideal problems have the following properties:

- Training on real-world data from mobile devices provides a distinct advantage over training on proxy data that is generally available in the data-center.
- This data is privacy sensitive or large in size (compared to the size of the model), so it is preferable not to log it to the data-center purely for the purpose of model training (in service of the *focused collection* principle).
- For supervised tasks, labels on the data can be inferred naturally from a user's interaction with their device.

Many models that power intelligent behavior on mobile devices fit the above criteria. As two examples, we consider:

- Image classification, for example predicting which photos are most likely to be viewed multiple times in the future, or shared.
- Language models, which can be used to improve voice recognition and text entry on touch-screen keyboards by improving decoding, next-word-prediction, and even predicting whole replies (Corrado, 2015).

The potential training data for both these tasks (all the photos a user takes and everything they type on their mobile keyboard, including passwords, URLs, messages, etc) can be privacy sensitive. The distributions from which these examples are drawn are also likely to differ substantially from easily available proxy datasets: the use of language in chat and text messages is generally much different than standard language corpora, e.g., Wikipedia and other web documents, or public-domain books; the photos people take on their phone are likely quite different than typical Flickr photos. And finally, the labels for these problems are directly available: entered text is self-labeled for learning a language model, and photo labels can be defined by natural user interaction with their photo app (which photos are deleted, shared, or viewed).

Both of these tasks are well-suited to learning a neural network. For image classification feed-forward deep networks, and in particular convolutional networks, are well-known to provide state-of-the-art results (LeCun et al., 1998; Krizhevsky et al., 2012). For language modeling tasks recurrent neural networks, and in particular LSTMs, have achieved state-of-the-art results (Hochreiter & Schmidhuber, 1997; Bengio et al., 2003; Kim et al., 2015).

In the remainder of this section, we consider the privacy advantages of federated optimization, and the potential to decrease communication costs for large datasets.

Privacy for federated learning There are two main aspects to data privacy for federated learning. First, we must consider what an attacker might learn by inspecting the model parameters, which are shared with all clients participating in the optimization. Given this wide availability, we cannot rely on security to mitigate such attacks. However, because the model is the aggregate of updates from a large number of individual users, for many model classes such attacks are much more difficult.

For truly privacy sensitive learning tasks, techniques from differential privacy can provide rigorous worst-case privacy guarantees even when the adversary has arbitrary side-information; however, this comes at some cost in utility, as these techniques rely on adding some random noise to the model training process (Dwork & Roth, 2014). Additional steps may also be needed to address model inversion attacks (Wang et al., 2015; Fredrikson et al., 2015). We note that these same issues arise for a model trained on private data held in the data center, and then released for on-device inference; hence it is not specific to federated learning.

The next question is what can an adversary learn by gaining access to the update messages of an individual client. If one trusts the central server, then encryption and other standard security protocols are a primary line of defense for this type of attack. A stronger guarantee can be achieved by enforcing local differential privacy (Kasiviswanathan et al., 2008; Duchi et al., 2014), where rather than adding noise to the final model, we noise the individual updates, which precludes the central server from making any definitive inference about a client. It is also possible to use secure multiparty computation to perform aggregation over multiple client updates, allowing local differential privacy to be achieved using much less random noise (Goryczka et al., 2013).

Even unpaired with a differential privacy guarantee, federated learning has distinct privacy advantages compared to data-center training on persisted data. Holding even an "anonymized" dataset can still put user privacy at risk via joins with other data (Sweeney, 2000). In contrast, the information transmitted for federated learning is the minimal update necessary to improve a particular model.[1] The up-

[1] Naturally, the strength of the privacy benefit depends on the content of the updates. For example, if the update is the total gradient of the loss on all of the local data, and the features are a sparse bag-of-words, then the non-zero gradients reveal exactly which words the user has entered on the device. In contrast, the sum of many gradients for a dense model such as a CNN offers a harder target for attackers seeking information about individual training instances (though attacks are still possible).

dates themselves can (and should) be ephemeral. And the source of the updates is not needed by the aggregation algorithm, so updates can be transmitted without identifying meta-data over a mix network such as Tor (Chaum, 1981) or via a trusted third party. Thus, federated learning is strictly preferable to directly logging the raw data to a central server, and can be further enhanced using known techniques to provide even stronger privacy guarantees.

Advantages for large datasets Federated learning can also provide a distinct advantage when training on large volumes of data. The network traffic per-client necessary to train in the data-center is simply the size of a client's local dataset, which must be transmitted once; for federated learning, the per-client traffic is (#-communication-rounds) $\times$ (update-size). This latter quantity can be substantially smaller if the update-size (generally $\mathcal{O}$(#-model-parameters)) is relatively small compared to the volume of training data needed, as when training on high-resolution photos or videos.

1.2. Federated Optimization

We refer to the optimization problem implicit in federated learning as federated optimization, drawing a connection (and contrast) to distributed optimization. As hinted above, federated optimization has several key properties that differentiate it from the typical distributed optimization problem:

- **Non-IID** The training data on a given client is typically based on the usage of the mobile device by a particular user, and hence any particular user's local dataset will not be representative of the population distribution.
- **Unbalanced** Similarly, some users will make much heavier use of the service or app that produces training data, leading to some clients having large local training data sets, while others have little or no data.
- **Massively distributed** In realistic scenarios, we expect the number of clients participating in an optimization to be much larger than the average number of examples per client.

In this work, our emphasis will be on the Non-IID and Unbalanced properties, as dealing with these aspects potentially requires the most substantial algorithmic advances.

A deployed federated optimization system must address a myriad of practical issues: client datasets that change as data is added and deleted; client availability that correlates with the local data distribution in complex ways (e.g., phones from speakers of American English will likely be plugged in at different times than speakers of British English); and clients that never respond or send corrupted updates.

These issues are beyond the scope of the current work; instead, we use a controlled environment that is suitable for experiments, but still address the key issues of client availability and unbalanced and non-IID data. We assume a synchronous update scheme that proceeds in rounds of communication. There is a fixed set of K clients, each with a fixed local dataset. At the beginning of each round, a random fraction C of clients is selected, and the server sends the current global algorithm state to each of these clients (e.g., the current model parameters). Each client then performs local computation based on the global state and its local dataset, and sends an update to the server. The server then applies these updates to its global state, and the process repeats.

While we focus on non-convex neural network objectives, the algorithm we consider is applicable to any finite-sum objective of the form

$$\min_{w \in \mathbb{R}^d} f(w) \qquad \text{where} \qquad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w). \tag{1}$$

For a machine learning problem, we typically take $f_i(w) = \ell(x_i, y_i; w)$, that is, the loss of the prediction on example (x_i, y_i) made with model parameters w.

We assume there are K clients over which the data is partitioned, with $\mathcal{P}_k$ the set of indexes of data points on client k, with $n_k = |\mathcal{P}_k|$. Thus, we can re-write (1) via

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w).$$

If the partition $\mathcal{P}_k$ was formed by distributing the training examples over the clients uniformly at random, then we would have $\mathbb{E}_{\mathcal{P}_k}[F_k(w)] = f(w)$, where the expectation is over the set of examples assigned to a fixed client k. This is the IID assumption typically made by distributed optimization algorithms; we refer to the case where this does not hold (that is, F_k could be an arbitrarily bad approximation to f) as the Non-IID setting.

In data-center optimization, communication costs are relatively small, and computational costs dominate, with much of the recent emphasis being on using GPUs to lower these costs. In contrast, in federated optimization communication costs dominate: since the communication costs are symmetric, we will typically be limited by an upload bandwidth of 1 MB/s or less. Further, clients will typically only volunteer to participate in the optimization when they are charged, plugged-in, and on an unmetered wi-fi connection. Further, we expect each client will only participate in a small number of update rounds per day. On the other hand, since any single on-device dataset is small compared to the total dataset size, and modern smartphones have rel-

atively fast processors (including GPUs), computation becomes essentially free compared to communication costs for many model types. Thus, our goal is to use additional computation in order to decrease the number of rounds of communication needed to train a model. There are two primary ways we can add computation:

- **Increased parallelism** Use more clients working independently between each communication round.
- **Increased computation on each client** Rather than performing a simple computation like a gradient calculation, each client performs a more complex calculation between each communication round.

We investigate both of these approaches, but the speedups we achieve are due primarily to adding more computation on each client, once a minimum level of parallelism over clients is used.

1.3. Related Work

In the convex setting, the problem of distributed optimization and estimation has received significant attention (Balcan et al., 2012; Fercoq et al., 2014; Shamir & Srebro, 2014), and some algorithms do focus specifically on communication efficiency (Zhang et al., 2013; Shamir et al., 2013; Yang, 2013; Ma et al., 2015; Zhang & Xiao, 2015). In addition to assuming convexity, this existing work generally requires that the number of clients is much smaller than the number of examples per client, that the data is distributed across the clients in IID fashion, and that each node has an identical number of data points — all of these assumptions are violated in the federated optimization setting. Asynchronous distributed forms of SGD have also been applied to training neural networks, e.g., Dean et al. (2012), but these approaches require a prohibitive number of updates in the federated setting.

One endpoint of the (parameterized) algorithm family we consider is simple one-shot averaging, where each client solves for the model that minimizes (possibly regularized) loss on their local data, and these models are averaged to produce the final global model. This approach has been studied extensively in the convex case with IID data, and it is known that in the worst-case, the global model produced is no better than training a model on a single client (Zhang et al., 2012; Arjevani & Shamir, 2015). Zinkevich et al. (2011) studies an averaging algorithm very similar to ours in the convex, balanced, IID setting.

Perhaps the most relevant prior work is that of Shokri & Shmatikov (2015). They focus on training deep networks, emphasize the role of (global) differential privacy, and address communication costs by only sharing a subset of the parameters during each round of communication. However, they do not consider datasets that are unbalanced and Non-IID, properties that we believe are essential to the federated learning setting.

2. The `FederatedAveraging` Algorithm

The recent multitude of successful applications of deep learning have almost exclusively relied on variants of stochastic gradient descent (SGD) as the optimization algorithm; in fact, many advances can be understood as adapting the structure of the model (and hence the loss function) to be more amenable to optimization by simple gradient-based methods (Goodfellow et al., 2016). Thus, it is natural that we build algorithms for federated optimization by starting from SGD.

SGD can be applied naively to the federated optimization problem, where a single minibatch gradient calculation (say on a randomly selected client) is done per round of communication. This approach is computationally efficient, but requires very large numbers of rounds of training to produce good models (e.g., even using an advanced approach like batch normalization, Ioffe & Szegedy (2015) trained MNIST for 50000 steps on minibatches of size 60).

The algorithm family we study, which we term `FederatedAveraging` (or `FedAvg`), allows us to add computation along both axes outlined above, with the goal of decreasing communication. The amount of computation is controlled by three key parameters: C, the fraction of clients that perform computation on each round; E, then number of training passes each client performs over its local dataset on each round; and B, the minibatch size used for the client updates. We write $B = \infty$ to indicate that the full local dataset is treated as a single minibatch.

At one endpoint of this algorithm family, we can take $B = \infty$ and $E = 1$ to produce a form of SGD with a varying minibatch size. This algorithm selects a C-fraction of clients on each round, and computes the gradient of the loss over all the data held by these clients. Thus, in this algorithm C controls the *global* batch size, with $C = 1$ corresponding to full-batch (non-stochastic) gradient descent. Since we still select batches by using all the data on the chosen clients, we refer to this simple baseline algorithm as `FederatedSGD`. While the batch selection mechanism is different than selecting a batch by choosing individual examples uniformly at random, the batch gradients g computed by `FederatedSGD` still satisfy $\mathbb{E}[g] = \nabla f(w)$.

A typical implementation of distributed gradient descent with a fixed learning rate η has each client k compute $g_k = \nabla F_k(w_t)$, the average gradient on its local data at the current model w_t, and the central server aggregates these

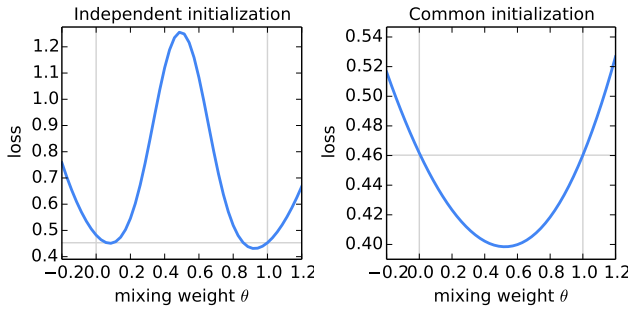

Figure 1. The loss on the full MNIST training set for models generated by averaging the parameters of two models w and w' using $\theta w + (1-\theta)w'$ for 50 evenly spaced values $\theta \in [-0.2, 1.2]$. The models w and w' were trained using SGD on different small datasets. For the left plot, w and w' were initialized using different random seeds; for the right plot, a shared seed was used. Note the different y-axis scales. The horizontal line gives the best loss achieved by w or w' (which were quite close, corresponding to the vertical lines at $\theta = 0$ and $\theta = 1$). When a common initialization is used, averaging the models produces a significant reduction in the loss on the total training set (much better than the loss of either parent model).

gradients and applies the update

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k,$$

since $\sum_{k=1}^{K} \frac{n_k}{n} g_k = \nabla f(w_t)$. However, it is easy to check that an equivalent update is given by

$$\forall k,\ w_{t+1}^k \leftarrow w_t - \eta g_k \qquad \text{and} \qquad w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k.$$

That is, each client locally takes one step of gradient descent on the current model using its local data, and the server then takes a weighted average of the resulting models. This is in fact how `FederatedSGD` is implemented as a special case of `FedAvg` in Algorithm 1. Once the algorithm is written this way, it is natural to ask what happens when the client iterates the local update $w^k \leftarrow w^k - \eta \nabla F_k(w^k)$ multiple times before the averaging step. For a client with n_k local examples, the number of local updates per round is given by $u_k = E\frac{n_k}{B}$; complete pseudocode is given in Algorithm 1.

Of course, for general non-convex objectives, averaging models in parameter space could produce an arbitrarily bad model. Following the approach of Goodfellow et al. (2015), we see exactly this bad behavior when we average two MNIST models[2] trained from different initial conditions (Figure 1, left). For this figure, the parent models w and w' were each trained on non-overlapping IID samples of 600 examples from the MNIST training set. Training

[2]We use the "2NN" model architecture described in Section 3.

Algorithm 1 `FederatedAveraging`

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $S_t = $ (random set of $\max(C \cdot K, 1)$ clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{t=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Executed on client k*
for each local epoch i from 1 to E **do**
  batches $\leftarrow$ (data $\mathcal{P}_k$ split into batches of size B)
  for batch b in batches **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

was via SGD with a fixed learning rate of 0.1 for 240 updates on minibatches of size 50 (or $E = 20$ passes over the mini-datasets of size 600). This is approximately the amount of training where the models begin to overfit their local datasets.

However, recent work indicates that in practice, the loss surfaces of sufficiently over-parameterized NNs are surprisingly well-behaved and in particular less prone to bad local minima than previously thought (Dauphin et al., 2014; Goodfellow et al., 2015; Choromanska et al., 2015). And indeed, when we start two models *from the same random initialization* and then again train each independently on a different subset of the data (as described above), we find that naive parameter averaging works surprisingly well (Figure 1, right): the average of these two models, $\frac{1}{2}w + \frac{1}{2}w'$, achieves significantly lower loss on the full MNIST training set than the best model achieved by training on either of the small datasets independently.

The success of dropout training also provides some intuition for the success of our model averaging scheme; dropout training can be interpreted as averaging models of *different* architectures which share parameters, and the inference-time scaling of the model parameters is analogous to the model averaging used in `FedAvg` (Srivastava et al., 2014).

3. Experimental Results

We are motivated by both image classification and language modeling tasks where good models can greatly enhance the usability of mobile devices. For each of these tasks we pick a proxy dataset of modest enough size that we can thoroughly investigate the hyper-parameters of the `FedAvg` algorithm. Thus, while each individual training run is relatively small, we trained over 2000 individual models for these experiments.

We study three model families on two datasets. The first two are for the MNIST digit recognition task (LeCun et al., 1998):

- A simple 2-hidden-layer model with 200 units per layer using ReLu activations (199,210 total parameters), which we refer to as the MNIST 2NN.
- A CNN for MNIST with two 5x5 convolution layers (the first with 32 channels, the second with 64, each followed with 2x2 max pooling), a fully connected layer with 512 units and ReLu activation, and a final softmax output layer (1,663,370 total parameters).

To study federated optimization, we also need to specify how the data is distributed over the clients. We study two ways of partitioning the MNIST data: **IID**, where the data is shuffled, and then partitioned into 100 clients each receiving 600 examples, and **Non-IID**, where we first *sort* the data by digit label, divide it into 200 shards of size 300, and assign each of 100 clients 2 shards. This is a pathological non-IID partition of the data, as most clients will only have examples from two digits. Thus, this lets us explore the degree to which our algorithms will break on highly non-IID data. Both of these partitions are balanced, however.

To study federated optimization for language models, we built a dataset from *The Complete Works of William Shakespeare*.[3] We construct a client dataset for each speaking role in each play with at least two lines. This produced a dataset with 1146 clients. For each client, we split the data into a set of training lines (the first 80% of lines for the role), and test lines (the last 20%, rounded up to at least one line). The resulting dataset has 3,564,579 characters in the training set, and 870,014 characters[4] in the test set. This data is substantially unbalanced, with many roles having only a few lines, and a few with a large number of lines. Further, observe the test set is not a random sample of lines, but is temporally separated by the chronology of each play. Using an identical train/test split, we also form a balanced and IID version of the dataset, also with 1146 clients.

On this data we train a stacked character-level LSTM language model, which after reading each character in a line, predicts the next character (Kim et al., 2015). The model takes a series of characters as input and embeds each of these into a learned 8 dimensional space. The embedded characters are then processed through 2 LSTM layers, each with 256 nodes. Finally the output of the second LSTM layer is sent to a softmax output layer with one node per character. The full model has 866,578 parameters, and we trained using an unroll length of 80 characters.

[3]Available as a single UTF-8 text file from `https://www.gutenberg.org/ebooks/100`

[4]We always use character to refer to a one byte string, and use role to refer to a part in the play.

Table 1. The effect of the client fraction C on the MNIST 2NN with $E = 1$ and CNN training with $E = 5$. Note $C = 0.0$ corresponds to one client per round; since we use 100 clients for the MNIST data, the rows correspond to 1, 10 20, 50, and 100 clients. Each table entry gives the number of rounds of communication necessary to achieve a test-set accuracy of 97% for the 2NN and 99% for the CNN, along with the speedup relative to the $C = 0$ baseline. Five runs with the large batch size did not reach the target accuracy in the allowed time.

2NN	IID		NON-IID	
C	$B = \infty$	$B = 10$	$B = \infty$	$B = 10$
0.0	1455	316	4278	3275
0.1	1474 (1.0×)	87 (3.6×)	1796 (2.4×)	664 (4.9×)
0.2	1658 (0.9×)	77 (4.1×)	1528 (2.8×)	619 (5.3×)
0.5	— (—)	75 (4.2×)	— (—)	443 (7.4×)
1.0	— (—)	70 (4.5×)	— (—)	380 (8.6×)
CNN, $E = 5$				
0.0	387	50	1181	956
0.1	339 (1.1×)	18 (2.8×)	1100 (1.1×)	206 (4.6×)
0.2	337 (1.1×)	18 (2.8×)	978 (1.2×)	200 (4.8×)
0.5	164 (2.4×)	18 (2.8×)	1067 (1.1×)	261 (3.7×)
1.0	246 (1.6×)	16 (3.1×)	— (—)	97 (9.9×)

SGD is sensitive to the tuning of the learning-rate parameter η. Thus, all of the results reported here are based on training over a sufficiently wide grid of learning rates (typically 11-13 values for η on a multiplicative grid of resolution $10^{\frac{1}{3}}$ or $10^{\frac{1}{6}}$). We checked to ensure the best learning rates were in the middle of our grids, and that there was not a significant difference between the best learning rates. Unless otherwise noted, we plot metrics for the best performing rate selected individually for each x-axis value. We find that the optimal learning rates do not vary too much as a function of the other parameters.

Increasing parallelism We first experiment with the effect of C, which controls the amount of multi-client parallelism. Table 1 shows the impact of varying C for both MNIST models. We report the number of communication rounds necessary to achieve a target test-set accuracy. To compute this, we construct a learning curve for each combination of parameter setting (optimizing η as described above), force the curve to be monotonically improving, and then compute the number of rounds where the curve reaches the target, using linear interpolation between the discrete points forming the curve. This is perhaps best understood by reference to Figure 2, where the light gray lines show the targets.

With $B = \infty$ (e.g., for MNIST processing all 600 client examples as a single batch per round), there is only a small advantage in increasing the client fraction. Using the smaller batch size $B = 10$ shows a significant improvement in using $C \geq 0.1$, especially in the Non-IID case. Based on these results, for most of the remainder of our experiments

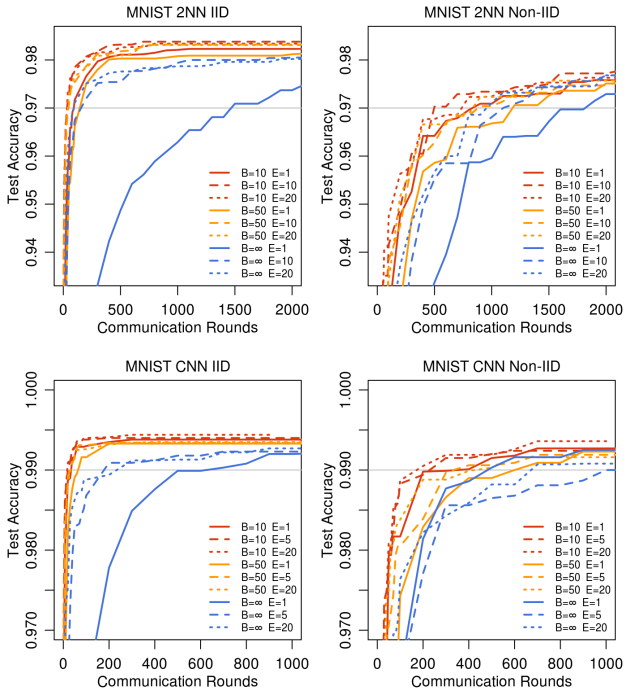

Figure 2. Test set accuracy vs. number of communication rounds for MNIST. The left column is the IID sharding of the data, and right is the pathological 2-digits-per-client non-IID distribution. The top row is the simple two-layer model, the second row is for the CNN. Note the rows use different x and y axis ranges. All runs in this figure are optimized over the fixed learning rate parameter and take $C = 0.1$.

we fix $C = 0.1$, which strikes a good balance between computational efficiency and convergence rate. While increasing C for a fixed B has a modest effect, comparing the number of rounds for $B = \infty$ and $B = 10$ shows a dramatic speedup. We investigate this in more detail in the next section.

Increasing computation per client In this section, we fix $C = 0.1$, and add more computation per client on each round, either decreasing B, increasing E, or both. The expected number of updates per client per round is given by $u = (\mathbb{E}[n_k]/B)E = En/(kB)$, where the expectation is over the draw of a random client k. We see that increasing u by varying both E and B is effective. As long as B is large enough to take full advantage of available parallelism on the client hardware, there is essentially no cost in computation time for lowering it, and so in practice this should be the first parameter tuned.

Figures 2 and 3 demonstrate that adding more local SGD updates per round and then averaging the resulting models can produce a dramatic speedup, and Tables 2 and 3 quantify these speedups. For the IID sharding of the MNIST data, using more computation per client decreases the number of rounds to reach the target accuracy by $35\times$ for the CNN and $46\times$ for the 2NN. The speedups for the pathologically sharded Non-IID data are smaller, but still substantial (2.8 – $3.7\times$). It is impressive that averaging provides *any* advantage (vs. actually diverging) when we naively average the parameters of models trained on entirely different pairs of digits. Thus, we view this as strong evidence for the robustness of this approach.

Table 2. Speedups in the number of communication rounds to reach a target accuracy (99% for the CNN, 97% for the 2NN) on the MNIST dataset. The u column gives $u = En/(kB)$, the expected number of updates per round.

CNN	E	B	u	IID	NON-IID
FEDSGD	1	∞	1	626	483
FEDAVG	5	∞	5	179 (3.5×)	1000 (0.5×)
FEDAVG	1	50	12	65 (9.6×)	600 (0.8×)
FEDAVG	20	∞	20	234 (2.7×)	672 (0.7×)
FEDAVG	1	10	60	34 (18.4×)	350 (1.4×)
FEDAVG	5	50	60	29 (21.6×)	334 (1.4×)
FEDAVG	20	50	240	32 (19.6×)	426 (1.1×)
FEDAVG	5	10	300	20 (31.3×)	229 (2.1×)
FEDAVG	20	10	1200	18 (34.8×)	173 (2.8×)
2NN	E	B	u	IID	NON-IID
FEDSGD	1	∞	1	1468	1817
FEDAVG	10	∞	10	156 (9.4×)	1100 (1.7×)
FEDAVG	1	50	12	144 (10.2×)	1183 (1.5×)
FEDAVG	20	∞	20	92 (16.0×)	957 (1.9×)
FEDAVG	1	10	60	92 (16.0×)	831 (2.2×)
FEDAVG	10	50	120	45 (32.6×)	881 (2.1×)
FEDAVG	20	50	240	39 (37.6×)	835 (2.2×)
FEDAVG	10	10	600	34 (43.2×)	497 (3.7×)
FEDAVG	20	10	1200	32 (45.9×)	738 (2.5×)

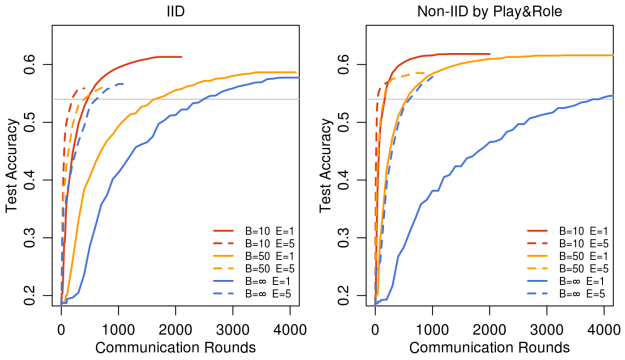

Figure 3. Learning curves for the Shakespeare LSTM. The gray lines shows the target accuracy of 54% used in Table 3.

The unbalanced and non-IID distribution of the Shakespeare data (by role in the play) is much more representative of the kind of data distribution we expect for real-world applications. Encouragingly, for this problem learning on the non-IID and unbalanced data is actually much easier (a $95\times$ speedup vs $13\times$ for the balanced IID data); we conjecture this is largely due to the fact some roles have relatively large local datasets, which makes increased local training particularly valuable.

Interestingly, for all three model classes, training runs

Table 3. Speedups in the number of communication rounds to reach a target test accuracy of 54% on the Shakespeare Char-LSTM problem.

LSTM	E	B	u	IID	NON-IID
FEDSGD	1	∞	1.0	2488	3906
FEDAVG	1	50	1.5	1635 (1.5×)	549 (7.1×)
FEDAVG	5	∞	5.0	613 (4.1×)	597 (6.5×)
FEDAVG	1	10	7.4	460 (5.4×)	164 (23.8×)
FEDAVG	5	50	7.4	401 (6.2×)	152 (25.7×)
FEDAVG	5	10	37.1	192 (13.0×)	41 (95.3×)

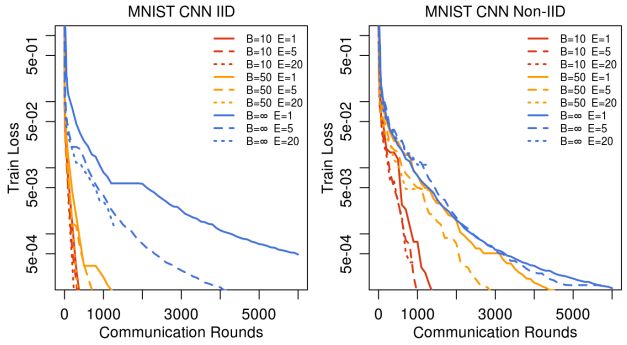

Figure 4. Training set convergence for the MNIST CNN. Note the y-axis is on a log scale, and the x-axis covers more training than Figure 2. These plots fix $C = 0.1$.

based on more local updates converge to a higher level of test-set accuracy than the baseline models. This trend continues even if the lines are extended beyond the plotted ranges. For example, for the CNN the $B = \infty, E = 1$ FedSGD model eventually reaches 99.22% accuracy after 1200 rounds (and had not improved further after 6000 rounds), while the $B = 10, E = 20$ FedAvg model reaches an accuracy of 99.44% after 300 rounds. We conjecture that in addition to lowering communication costs, model averaging produces a regularization benefit similar to that achieved by dropout (Srivastava et al., 2014).

We are primarily concerned with generalization performance, but FedAvg is effective at optimizing the training loss as well, even beyond the point where test-set accuracy plateaus. We observed similar behavior for all three model classes, and present plots for the MNIST CNN in Figure 4.

Can we over-optimize on the client datasets? The current model parameters only influence the optimization performed in each ClientUpdate via initialization. Thus, as $E \to \infty$, at least for a convex problem eventually the initial conditions should be irrelevant, and the global minimum would be reached regardless of initialization. Even for a non-convex problem, one might conjecture the algorithm would converge to the same local minimum as long as the initialization was in the same basin. That is, we would

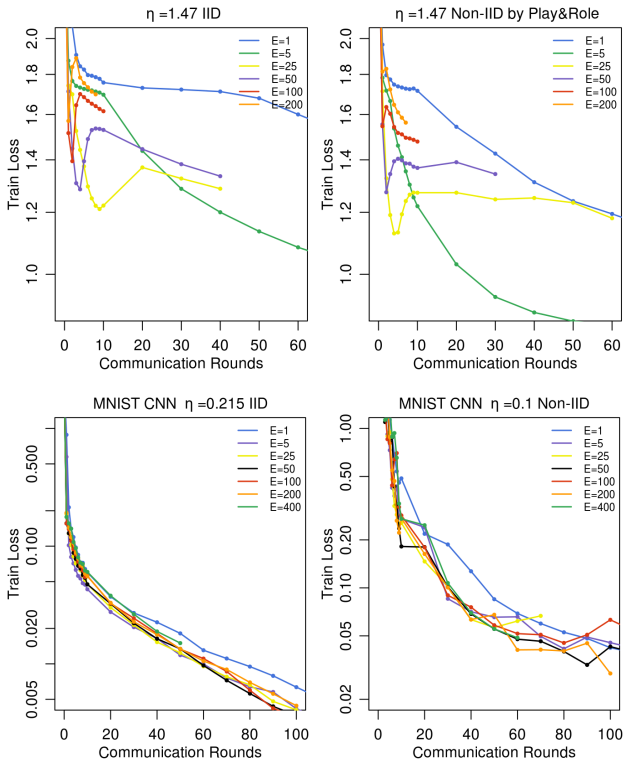

Figure 5. The effect of training for many local epochs (large E) between averaging steps, fixing $B = 10$ and $C = 0.1$. Top row: Training set loss for the Shakespeare LSTM with a fixed learning rate $\eta = 1.47$. Bottom row: Training loss for the MNIST CNN. Note different learning rates and y-axis scales are used due to the difficulty of our pathological Non-IID MNIST dataset.

expect that while one round of averaging might produce a reasonable model, additional rounds of communication (and averaging) would not produce further improvements.

Figure 5 (top row) shows the impact of large E during initial training on the Shakespeare LSTM problem. Indeed, for very large numbers of local epochs, FedAvg can plateau or diverge.[5] This result suggests that for some models, especially in the later stages of convergence, it may be useful to decay the amount of local computation per round (moving to smaller E or larger B) in the same way decaying learning rates can be useful. Figure 5 (bottom row) gives the analogous experiment for the MNIST CNN. Interestingly, for this model we see no significant degradation in the convergence rate for large values of E.

[5] Note that due to this behavior and because for large E not all experiments for all learning rates were run for the full number of rounds, we report results for a fixed learning rate (which perhaps surprisingly was near-optimal across the range of E parameters) and without forcing the lines to be monotonic.

4. Conclusions and Future Work

Our experiments show that federated learning has significant promise, as high-quality models can be trained using relatively few rounds of communication. Further empirical evaluation of the proposed approach on larger datasets that truly capture the massively distributed nature of real-world problems are an important next step. In order to keep the scope of algorithms explored tractable, we limited ourselves to building on vanilla SGD. Investigating the compatibility of our approach with other optimization algorithms such as AdaGrad (McMahan & Streeter, 2010; Duchi et al., 2011) and ADAM (Kingma & Ba, 2015), as well as with changes in model structure that can aid optimization, such as dropout (Srivastava et al., 2014) and batch-normalization (Ioffe & Szegedy, 2015), are another natural direction for future work.

References

Arjevani, Yossi and Shamir, Ohad. Communication complexity of distributed convex learning and optimization. In *Advances in Neural Information Processing Systems 28*. 2015.

Balcan, Maria-Florina, Blum, Avrim, Fine, Shai, and Mansour, Yishay. Distributed learning, communication complexity and privacy. *arXiv preprint arXiv:1204.3514*, 2012.

Bengio, Yoshua, Ducharme, Réjean, Vincent, Pascal, and Janvin, Christian. A neural probabilistic language model. *J. Mach. Learn. Res.*, 2003.

Chaum, David L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2), 1981.

Choromanska, Anna, Henaff, Mikael, Mathieu, Michaël, Arous, Gérard Ben, and LeCun, Yann. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

Corrado, Greg. Computer, respond to this email. `http://googleresearch.blogspot.com/2015/11/computer-respond-to-this-email.html`, November 2015.

Dauphin, Yann N., Pascanu, Razvan, Gülçehre, Çaglar, Cho, KyungHyun, Ganguli, Surya, and Bengio, Yoshua. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *uNIPS*, 2014.

Dean, Jeffrey, Corrado, Greg S., Monga, Rajat, Chen, Kai, Devin, Matthieu, Le, Quoc V., Mao, Mark Z., Ranzato, Marc'Aurelio, Senior, Andrew, Tucker, Paul, Yang, Ke, and Ng, Andrew Y. Large scale distributed deep networks. In *NIPS*, 2012.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2011.

Duchi, John, Jordan, Michael I., and Wainwright, Martin J. Privacy aware learning. *Journal of the Association for Computing Machinery*, 2014.

Dwork, Cynthia and Roth, Aaron. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2014.

Fercoq, Olivier, Qu, Zheng, Richtárik, Peter, and Takác, Martin. Fast distributed coordinate descent for non-strongly convex losses. In *Machine Learning for Signal Processing (MLSP), 2014 IEEE International Workshop on*, 2014.

Fredrikson, Matt, Jha, Somesh, and Ristenpart, Thomas. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM Conference on Computer and Communications Security*, 2015.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. Deep learning. Book in preparation for MIT Press, 2016.

Goodfellow, Ian J., Vinyals, Oriol, and Saxe, Andrew M. Qualitatively characterizing neural network optimization problems. In *ICLR*, 2015.

Goryczka, Slawomir, Xiong, Li, and Sunderam, Vaidy. Secure multiparty aggregation with differential privacy: A comparative study. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, 2013.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Computation*, 9(8), November 1997.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Kasiviswanathan, Shiva Prasad, Lee, Homin K., Nissim, Kobbi, Raskhodnikova, Sofya, and Smith, Adam. What can we learn privately? In *FOCS*, 2008.

Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. Character-aware neural language models. *CoRR*, abs/1508.06615, 2015.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 2012.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

Ma, Chenxin, Smith, Virginia, Jaggi, Martin, Jordan, Michael I, Richtárik, Peter, and Takáč, Martin. Adding vs. averaging in distributed primal-dual optimization. In *ICML*, 2015.

McMahan, H. Brendan and Streeter, Matthew. Adaptive bound optimization for online convex optimization. In *COLT*, 2010.

Shamir, Ohad and Srebro, Nathan. Distributed stochastic optimization and learning. In *Communication, Control, and Computing (Allerton)*, 2014.

Shamir, Ohad, Srebro, Nathan, and Zhang, Tong. Communication efficient distributed optimization using an approximate newton-type method. *arXiv preprint arXiv:1312.7853*, 2013.

Shokri, Reza and Shmatikov, Vitaly. Privacy-preserving deep learning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, 2015.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 2014.

Sweeney, Latanya. Simple demographics often identify people uniquely. 2000.

Wang, Yue, Si, Cheng, and Wu, Xintao. Regression model fitting under differential privacy and model inversion attack. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 2015.

White House Report. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. *Journal of Privacy and Confidentiality*, 2013.

Yang, Tianbao. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, 2013.

Zhang, Yuchen and Xiao, Lin. Communication-efficient distributed optimization of self-concordant empirical loss. *arXiv preprint arXiv:1501.00263*, 2015.

Zhang, Yuchen, Wainwright, Martin J, and Duchi, John C. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*, 2012.

Zhang, Yuchen, Duchi, John, Jordan, Michael I, and Wainwright, Martin J. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *Advances in Neural Information Processing Systems*, 2013.

Zinkevich, Martin, Weimer, Markus, Smola, Alexander J., and Li, Lihong. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23 (NIPS-10)*, 2011.