Does Federated Dropout actually work?

Gary Cheng
Stanford University
chenggar@stanford.edu

Zachary Charles
Google Research
zachcharles@google.com

Zachary Garrett
Google Research
zachgarrett@google.com

Keith Rush
Google Research
krush@google.com

Abstract

Model sizes are limited in Federated Learning due to network bandwidth and on-device memory constraints. The success of increasing model sizes in other machine learning domains motivates the development of methods for training large-scale models in Federated Learning. To this end, [3] draws inspiration from dropout and proposes Federated Dropout: an algorithm where clients train randomly selected subsets of a larger server model. Despite the promising empirical results and the many other works that build on it [1, 8, 13], we argue in this paper that the metrics used to measure the performance of Federated Dropout and its variants are misleading. We propose and perform new experiments which suggest that Federated Dropout is actually detrimental to scaling efforts. We show how a simple ensembling technique outperforms Federated Dropout and other baselines. We perform ablations that suggest that the best performing variations of Federated Dropout approximate ensembling. The simplicity of ensembling allows for easy, practical implementations. Furthermore, ensembling naturally leverages the parallelizable nature of Federated Learning—recall that it is easy to train several models independently because there are a lot of clients and server-compute is not the bottleneck. Ensembling's strong performance against our baselines suggests that Federated Learning models may be more easily scaled than previously thought with more sophisticated ensembling strategies e.g., via boosting.

1. Introduction

Federated Learning is the distributed learning setting where a central server coordinates with client devices to train machine learning models in a communication efficient manner. This learning regime has popularized local-SGD-type training algorithms [27] where the server periodically averages local client updates, the most notable of these algorithms being Federated Averaging (FedAvg) [18]. Already, Federated Learning is successfully being used as a means of training machine learning models on mobile devices [12] and has promising applications in hospitals and banking [14]. However, its distributed nature inherently limits the size of the models trained. Training larger models require larger messages between server and client devices. Moreover, client devices are often memory and compute-constrained (e.g., cell phones), so it may not be possible to backpropagate or even fit a large model on client devices.

In contrast, the current trend in centralized machine learning is that bigger is better. Larger models perform better than smaller models, even with distribution shift [25], and they are (nearly) able to one-shot learn new tasks [2]. Recent work suggests that model performance will continue to improve as model sizes increase, with no apparent end in sight [15]. The success of these large models at handling new tasks and distributions in centralized settings leaves a lot to be desired in Federated Learning. Many pressing problems in Federated Learning, such as handling heterogeneous users, could be mitigated just by simply scaling up models.

Motivated by the centralized setting, there have been some efforts in scaling up models in Federated Learning. One notable line of work is Federated Dropout [3]. The idea draws inspiration from the popular neural net training technique dropout [24], and it works as follows: at every iteration of a local-SGD-type algorithm, each client trains a random subset (dropped out version) of the server model. The updates are aggregated after local training and the corresponding parameters of the server model are updated. The algorithm is intuitive and has led to many follow-up works [1, 13], including its use in training speech recognition models [8]. However, despite its promise, we argue in this paper that Federated Dropout may actually not be that good, especially in comparison to simpler alternatives. In particular, we make the following contributions:

Contributions

1. We propose a new way of baselining Federated-Dropout-type methods, and we justify why this baseline does a better job of respecting real-world constraints than the metrics used in [3] (*cf*. Section 2).

2. We show that Federated Dropout does not beat our proposed baseline, while a very simple ensembling method does (*cf*. Section 4). The ensembling method we consider, which we term Simple Ensemble Averaging (*cf*. Section 3), independently trains R copies of the client model and averages the outputs of these models at evaluation time. We perform additional ablations that suggest the best performing configuration of Federated Dropout is one that approximately performs Simple Ensemble Averaging (*cf*. Section 4).[1]

We acknowledge that Simple Ensemble Averaging is not novel; we give it this name for referential clarity and to emphasize that the algorithm is extremely simple. The idea of averaging models has been around far longer than Federated Learning [20], and it has even been studied in the Federated Learning literature e.g., in [21]. Having said that, it is surprising that in the federated regime, dropout and ensembling seem to behave in fundamentally different ways, as the two are often understood to be intimately related in the centralized setting [7, 11]. We hypothesize that this distinction is fundamental to the federated setting, particularly the infrequent synchronization between the portions of models partitioned to client devices. Our results highlight a gap in the literature on cross-device federated learning: 1. there are no extant algorithms that explicitly leverage the parallelizable nature of federated learning to train more accurate machine learning models, and 2. for such algorithms to be effective, they will need to be designed with the federated case in mind.

On a more pragmatic level, Simple Ensemble Averaging is (relatively) easy to deploy—just train R models in parallel using existing infrastructure and algorithms. Simple Ensemble Averaging's practical effectiveness suggests that 1. it should serve as a baseline for any method attempting to scale up FL model sizes and 2. more advanced strategies of ensembling (e.g., boosting, training different sub-models for different subpopulations) could perform even better.

1.1. Related Work

There has been a lot of work focused on designing new methods of scaling up Federated Learning models. Some works focus on reducing communication costs between the server and clients by compressing communication in some way [3, 16]. Other works propose ways of reducing training times by mitigating the effect of stragglers [5], improving convergence rates [19], or reducing the number of trainable parameters [22].

Another vein of research, which is the primary focus of this paper, has been on exploiting user-server model asymmetry. Vanilla FedAvg [18], for example, requires that the trainable parameters on the server are the same as the ones client-side, but nothing fundamental about Federated Learning requires this congruence. [6, 17] propose to have clients train local representations of their data that the server then uses to train a global model. [23] proposes to have the server only keep track of a set of global parameters which the clients use to reconstruct their local parameters. However, this concept gives clients the additional computational burden of reconstructing local parameters, making it difficult to deploy in practice. Split Learning [9, 26] splits the model being trained into two portions, with one trained on the server and the other trained on the client. However, this training procedure requires a good deal of synchronization between server and clients, making it difficult to implement in practice.

Federated Dropout Federated Dropout is another algorithm that exploits user-server model asymmetry. [3] proposed Federated Dropout as a means of leveraging compute and communication constrained clients to train larger models. In empirical experiments, [3] fixes the server model size and measures how much increasing the dropout rate (i.e., shrinking the client model size) degrades performance. Their results look promising: the accuracy degrades slowly. However, these plots are misleading. Given that the client model size is dictated a priori by client limitations, fixing the client model size and varying the server model size is a more realistic comparison. In this alternative experimental setup, we can compare the performance of a larger model trained via Federated Dropout against a smaller model trained via FedAvg. The current ubiquity of FedAvg-variants (local-SGD-type algorithms) in current applications suggests that any method worth using should beat FedAvg (along some metric). Since [3] does not do this comparison, the question of whether Federated Dropout actually leads to performance improvements in practice remains open.

Many more works have proposed follow-ups or are related to Federated Dropout. [1] propose an adaptive version of Federated Dropout where the dropout rate is tuned for each activation based on feedback from the training loss. [13] propose a new method called "Ordered Dropout" where client sub-models are selected in a nested fashion from the server model; this design was an effort to adapt to user heterogeneity. With this approach, clients can train a model of size in proportion to their computational resources. [16] proposes to have client model updates be restricted to a low-rank subspace. None of the works cited here (to our

[1]Code for our experiments can be found at https://github.com/google-research/federated/tree/master/shrink_unshrink

knowledge) perform experiments with a fixed client model size as we propose above.

Ensemble Methods Generally speaking, ensemble methods are techniques that combine several weak learners into one larger, better-performing model. Simple Ensemble Averaging may be arguably the most naive ensembling method. More sophisticated methods have been proposed and studied for a much longer time. Some examples include boosting, bagging, and stacking. See [20] for a more in-depth outline of the various ensembling methods. Simple Ensemble Averaging has been studied in Federated Learning before [21]; however, the focus of [21] is more on its convergence properties. Other variations inspired by boosting have been studied as well [10].

2. Problem Statement

We will be working in the Federated Learning setting with m clients each with datasets $\mathcal{D}_i$ comprised of datapoints $(x, y) \in \mathbb{R}^d \times \mathbb{R}$. We will assume that each client has limited memory and computational resources and so is only able to train (i.e., backpropagate) a small model denoted by $f_c : \Theta \times \mathbb{R}^d \to \mathbb{R}^k$ parameterized with weights $\theta \in \Theta$. However, we will assume that the client can do inference (i.e., do a forward pass) on a larger model $f_s : \Theta^R \times \mathbb{R}^d \to \mathbb{R}^k$ with weights $\theta \in \Theta^R$ which has R times the dimension of the smaller model parameters, as these operations are not as memory intensive. This assumption is realistic, for example, for convolutional neural networks, where the memory consumption of storing the gradients dominates the memory consumption of storing the weights. For some sample loss $\ell : \mathbb{R}^k \times \mathbb{R} \to \mathbb{R}$, let the empirical risk with respect to a dataset $\mathcal{D}$ and an estimator $g : \mathbb{R}^d \to \mathbb{R}^k$ be defined as

$$L_{\mathcal{D}}(g) \coloneqq \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \ell(g(x), y). \tag{1}$$

In our model, we can think of $g(x) = f_c(\theta, x)$ or $g(x) = f_s(\theta, x)$. To aid understanding, the output space $\mathbb{R}^k$ can be thought of as the space of unnormalized logits, y as discrete labels in $[k]$, and ℓ being a classification-type loss (e.g., cross-entropy). We will further assume that the server's resources are effectively unbounded; it will be able to store whatever it needs in memory and communicate with as many clients as it wants. These assumptions naturally model what happens in practical Federated Learning applications.

The status quo training procedure in our setting is Federated Averaging, which prescribes that the trainable parameters on the server are the same trainable parameters on the client. In this paper, we explore (existing and new) methods which leverage server resources to break this model symmetry. In particular, we experiment with methods of training a large model f_s on the server by utilizing model updates on the smaller model f_c on client devices. Since forward passes of f_s are possible on client devices, this larger model only needs to be communicated once to clients at the end of training.

To baseline the methods in this paper, we will compare the test performance of f_s trained using Federated Dropout or Simple Ensemble Averaging (by using f_c model updates client-side) against the test performance of f_c trained using Federated Averaging. We stress that any method worth implementing needs to beat this baseline, as the baseline method satisfies the constraints we laid out and is, by and large, how models are trained in practice. We note that this baseline differs from the baseline proposed by the Federated Dropout paper [3]; they compare their method against the performance of the larger f_s trained via Federated Averaging. However, given the constraints we laid out and motivated above, f_s is not trainable using standard federated methods and thus, does not provide an implementable, alternative procedure to Federated Dropout and Simple Ensemble Averaging.

3. Federated Dropout and Simple Ensemble Averaging

We explain how Federated Dropout works using a two-layer neural net example in Algorithm 1; to apply Federated Dropout on deeper neural networks, just repeatedly apply the algorithm described on each feed-forward layer. For more details about the algorithm, check out the Federated Dropout paper [3]. We will call the net being optimized on the server f_s. It maps inputs from $\mathbb{R}^d$ to outputs in $\mathbb{R}^k$ and is of the form $f_s(U_s, V_s; x) = U_s\sigma(V_s x)$. The weight matrices $U_s \in \mathbb{R}^{k,l_s}, V_s \in \mathbb{R}^{l_s,d}$ are the trainable parameters and the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is applied element-wise. We assume that the client is unable to train (i.e., backpropagate) f_s directly. Instead, it only is able to train a smaller network $f_c(U_c, V_c; x) = \frac{l_s}{l_c} U_c\sigma(V_c x)$ where $U_c \in \mathbb{R}^{k,l_c}, V_c \in \mathbb{R}^{l_c,d}$ for $l_c < l_s$ (i.e., number of client parameters is less than the number of server parameters). The l_s/l_c factor is to ensure that the scaling of the intermediate layers remains constant.

At a high level, for every communication round, Federated Dropout randomly selects a subset of activations (or convolutional filters in the context of CNNs) of each layer to keep for clients to train. On a more technical level, Federated Dropout is a special case of the algorithm described in Algorithm 1. In particular, the columns of P_i are selected uniformly at random without replacement from the set of standard basis vectors of dimension l_s, $\{e_i\}_{i=1}^{l_s}$. By generating projection matrices in this way, Federated Dropout is keeping the activations with indices matching the indices of the standard basis vectors selected. This is analogous to how dropout is normally implemented where some fraction of the activations are zeroed out, with the exception that in our setting, activations are removed altogether in the client

Algorithm 1: Generalized Federated Dropout on a Two-Layer Neural Network

Data: m: number of clients; T: epochs; K: local iterations; α: server step size; $L_{\mathcal{D}_i}$: loss function for client i; $U_s \in \mathbb{R}^{k,l_s}, V_s \in \mathbb{R}^{l_s,d}$: initialized server weights

Result: U_s, V_s: Server weights

Let $f_s(U_s, V_s; x) = U_s\sigma(V_s x)$ and $f_c(U_c, V_c; x) = \frac{l_s}{l_c} U_c\sigma(V_c x)$.

for $t \leftarrow 1$ **to** T **do**

  for $i \leftarrow 1$ **to** m *in parallel* **do**

    1. The server generates a projection matrix $P_i \in \mathbb{R}^{l_s,l_c}$.

    2. The server shrinks server weights U_s, V_s to client sized weights $U_c^{(i)} \in \mathbb{R}^{k,l_c}, V_c^{(i)} \in \mathbb{R}^{l_c,d}$ by setting $U_c^{(i)} \leftarrow U_s P_i$ and $V_c^{(i)} \leftarrow P_i^T V_s$. The server sends $U_c^{(i)}$ and $V_c^{(i)}$ to client i.

    3. Client i performs local trains f_c with respect to the loss $L_{\mathcal{D}_i}$ using $U_c^{(i)}, V_c^{(i)}$ as initialization (e.g., run K steps of SGD initialized at $U_c^{(i)}, V_c^{(i)}$) and sends the difference between the output of the local training and the input from the server, $\Delta_{U_c}^{(i)}$ and $\Delta_{V_c}^{(i)}$, back to the server.

    4. The server unshrinks the update by executing $\Delta_{U_s}^{(i)} \leftarrow \Delta_{U_c}^{(i)} P_i^T$ and $\Delta_{V_s}^{(i)} \leftarrow P_i \Delta_{V_c}^{(i)}$.

  end

  The server updates the server weights $U_s \leftarrow U_s + \frac{\alpha}{m}\sum_{j=1}^{m} \Delta_{U_s}^{(i)}$ and $V_s \leftarrow V_s + \frac{\alpha}{m}\sum_{j=1}^{m} \Delta_{V_s}^{(i)}$.

end

return U_s, V_s

model. There any many variations of the algorithm that can be explored. For example, we can use the same P_i for all clients, or we can alter how frequently P_i is changed. We explore how these design decisions affect performance in Section 4.

Simple Ensemble Averaging (Algorithm 2) is a very simple algorithm to implement. Just independently train R copies of a specified model architecture; at evaluation time, just average the outputs of each of the R models. While this is certainly what one would do in practice, for the sake of implementation simplicity, our experiments will execute Simple Ensemble Averaging using the algorithmic framework specified in Algorithm 1. For the sake of the experiments later, we outline how that is done now. Assume that $R = l_s/l_c$ is a positive integer. R projection matrices are constructed by partitioning the columns of identity matrix $I \in \mathbb{R}^{l_s,l_s}$ into R contiguous but disjoint sets of size l_c; i.e., the rth projection matrix consists of the rl_c+1 to $(r+1)l_c$ elements of the l_s-dimensional standard basis vectors, $\{e_i\}_{i=1}^{l_s}$. For every iteration of the algorithm described above, P_i is selected uniformly at random from the set of R matrices constructed. By generating projection matrices in this way, this algorithm is training R different versions of f_c inside the model f_s (assuming we delete the l_s/l_s rescaling term in intermediate layers of f_c), with each of the R projection matrices corresponding to one of the versions. Forward passes through f_s in this setting correspond to averaging the output of R independently trained copies of f_c.

Algorithm 2: Simple Ensemble Averaging

Data: m: number of clients; $\mathcal{A}$: federated training algorithm; R: size of the ensemble

Result: f_s: Server model

Partition m clients into R disjoint sets $\{\mathcal{C}_r\}_{r=1}^{R}$

for $r \leftarrow 1$ **to** R *in parallel* **do**

  Train a client-sized model $f_c^{(r)}$ using $\mathcal{A}$ and clients in $\mathcal{C}_r$

end

Let $f_s = \frac{1}{R}\sum_{r=1}^{R} f_c^{(r)}$ **return** f_s

3.1. Intuition why ensembled models outperform a single model

We will see in Section 4 that training a model f_s using Simple Ensemble Averaging outperforms training a single model f_c. The intuition why this is true can be explained using Jensen's inequality.

Suppose we have a multi-class classification task with datapoints $x \in \mathbb{R}^d$ and $y \in [k]$. Let $f_c : \Theta \times \mathbb{R}^d \to \mathbb{R}^k$ be a a neural network with parameters $\theta \in \Theta$ that maps datapoints $x \in \mathbb{R}^d$ to outputs $\mathbb{R}^k$. Let $f_s : \Theta^R \times \mathbb{R}^d \to \mathbb{R}^k$ be a model which averages R client-sized models f_c, $f_s(\theta^{(1)}, \ldots, \theta^{(R)}, x) := \frac{1}{R}\sum_{j=1}^{R} f_c(\theta^{(j)}, x)$. Further, suppose that we evaluate our models using the multi-class cross-entropy loss; in particular, we let

$$\ell(f_c(\theta, x); y) := -\log\left(\frac{\exp(f_c(\theta, x)_y)}{\sum_{i=1}^{k}\exp(f_c(\theta, x)_i)}\right). \quad (2)$$

We let θ be the result of a randomized training algorithm (such as SGD). Now, we examine the population loss

$$L(f_c) := \mathbb{E}_{\theta,(x,y)}[\ell(f_c(\theta, x); y)], \quad (3)$$

where the expectation is taken over the randomness of the training algorithm used to generate θ and the data (x, y). If

we let $\{\theta^{(j)}\}_{j=1}^{R}$ denote R independently trained parameters, by Jensen's inequality, we have

$$L(f_c) = \frac{1}{R}\sum_{j=1}^{R}\mathbb{E}[\ell(f_c(\theta^{(j)}, x); y)] \quad (4)$$

$$\geq \mathbb{E}\left[\ell\left(\frac{1}{R}\sum_{j=1}^{R} f_c(\theta^{(j)}, x); y\right)\right] = L(f_s). \quad (5)$$

The inequality results from the convexity of ℓ in its first argument. By observing that training f_s via Simple Ensemble Averaging is training R independently trained models, we see that Simple Ensemble Averaging should perform no worse than a single client model, at least for a convex test loss.

4. Experiments

We empirically compare Federated Dropout and Simple Ensemble Averaging on the Federated EMNIST dataset [4] which consists of 341,873 training examples of 28 by 28 images split over 3383 users and 62 label classes. Throughout the experiment, we train a two-layer convolutional neural networks of varying size for $T = 2000$ server communication rounds. We sweep over client learning rates$\in \{0.01, 0.1\}$ and server learning rates$\in \{1, 3, 5, 10\}$. We found that these learning rate ranges were most effective. We will sweep over two convolutional neural net architectures with ReLu activations

- CNN w/ dropout: two convolutional layers both using 3 by 3 kernels with no padding, followed by a max pooling layer, dropout with rate 0.25, a dense layer, dropout with rate 0.5, and a final dense layer.
- CNN from [18]: two convolutional layers both using 5 by 5 kernels with padding and max pool layer between, followed by a max pooling layer, and two dense layers.

In our experiments, we will also consider three different architecture sizes:

- **S**: 8 filters per convolutional layer and 16 dimensional output on the first dense layer.
- **M**: 32 filters per convolutional layer and 64 dimensional output on the first dense layer.
- **L**: 64 filters per convolutional layer and 128 dimensional output on the first dense layer.

We will set the client model size to be **S** and vary the server model size. This models the fundamental assumption throughout the paper that the client model size is no larger than the server model size. We note that if the server model size and the client model size are the same, both methods are equivalent to standard Federated Averaging. Let's give some perspective on the memory footprint of each of the models. Recall that for convolutional neural networks, the memory consumption of computing gradients dominates the memory consumption of storing the weights. **M** sized architectures need ~ 16 times the memory to store its gradients relative to **S** sized architectures, and **L** sized architectures need ~ 64 times the memory to store its gradients relative to **S** sized architectures.

Federated Dropout is implemented in the way described in Section 3. We note that on feedforward layers, Federated Dropout will select a random subset of the activations to keep; this procedure exactly mirrors the procedure described in Section 3. On convolutional layers, Federated Dropout will select a random subset of the convolutional filters to keep. Dropping out convolutional filters is important because these filters form the bulk of the memory footprint during training (i.e., during backpropagation). For both feedforward layers and convolutional layers, an appropriate rescaling is applied as described in Section 3.

Simple Ensemble Averaging is implemented in the way described in Section 3. Readers can think of Simple Ensemble Averaging as training R copies of the client model, where R is the ratio of the number of server activations/filters to the number of client activations/filters. For example, if we choose the server model size to be **L** and the client model size is **S**, then $R = 8$.

Remark In our implementation of Simple Ensemble Averaging, described in Section 3, the weight matrices of the dense layer in the server model are randomly initialized. Thus, there exist parameters in the dense layer which have an effect on the output but are not a part of any submodel's weight matrices. Moreover, at evaluation time, the intermediate outputs of client submodels (albeit after the convolutional and max-pooling layers) may have some effect on one another. While this technically means the algorithm we implemented is not exactly the algorithm described in Algorithm 2, if anything, given that the interactions are random and not accounted for during any part of the training, we believe this should only hurt the performance of the final server model. However, even if it does somehow help the performance, we believe the message of the paper is unchanged: Federated Dropout is easily outperformed by a simple ensembling-type alternative.

In our implementation, every client is assigned a seed from a set $\mathcal{S} \subset \mathbb{Z}$. During Federated Dropout, when a client is selected for training, said client will use their assigned seed to generate a submodel used for local training (two clients with the same seed will produce the same submodel). Depending on the experiment, the assigned seed may change from iteration to iteration, or it may stay the same throughout; we will provide the details of how seeds are generated in each

Server model size	Federated Dropout test accuracy	Simple Ensemble Averaging test accuracy	Federated Averaging test accuracy
S	-	-	73.16%
M	64.14%	75.86%	-
L	56.53%	74.79%	-

Table 1. Server model size against (hyperparameter-tuned) test accuracy for Federated Dropout, Simple Ensemble Averaging, and Federated Averaging.

subsection. In the context of Simple Ensemble Averaging, the seed specifies which of the models in the ensemble the client should train.

4.1. Effect of Server Model Size

In the first subsection, we report the test accuracy (after hyperparameter tuning) for different server model sizes for both Federated Dropout and Simple Ensemble Averaging. In this experiment, each client is assigned a different unique seed to use to generate their submodel which they use for the entire training procedure. The standard machine learning intuition is that larger models perform better and that dropout should aid training. However, surprisingly, increasing server model size actually hurts the performance of Federated Dropout (see Table 1). Ignoring Simple Ensemble Averaging for a moment, this result suggests that, for a given fixed client model size, one should just train the client model with Federated Averaging (without Federated Dropout).

For the **L** server model size, we also plot the performance of the five best hyperparameters we swept over for Federated Dropout and Simple Ensemble Averaging in Figure 1 and compare them against the FedAvg baseline. In both cases, the fifth-best hyperparameter choice for Simple Ensemble Averaging still performs better than the best hyperparameter choice for Federated Dropout (i.e., when server and client models are size **S**). We see that Simple Ensemble provides a couple of percentage points of improvement over FedAvg, and we see that Federated Dropout often does much worse than FedAvg.

These results are quite striking. At the very least, these results indicate that Federated Dropout is not able to reduce the number of parameters being trained by a factor of 16. This suggests that Federated Dropout will not lead to a significant scale-up in model sizes. While this does technically leave the door open for reductions of a smaller magnitude, this untested, hypothetical benefit of Federated Dropout is further reduced by the fact that a very straightforward idea like Simple Ensemble Averaging performs so well in comparison.

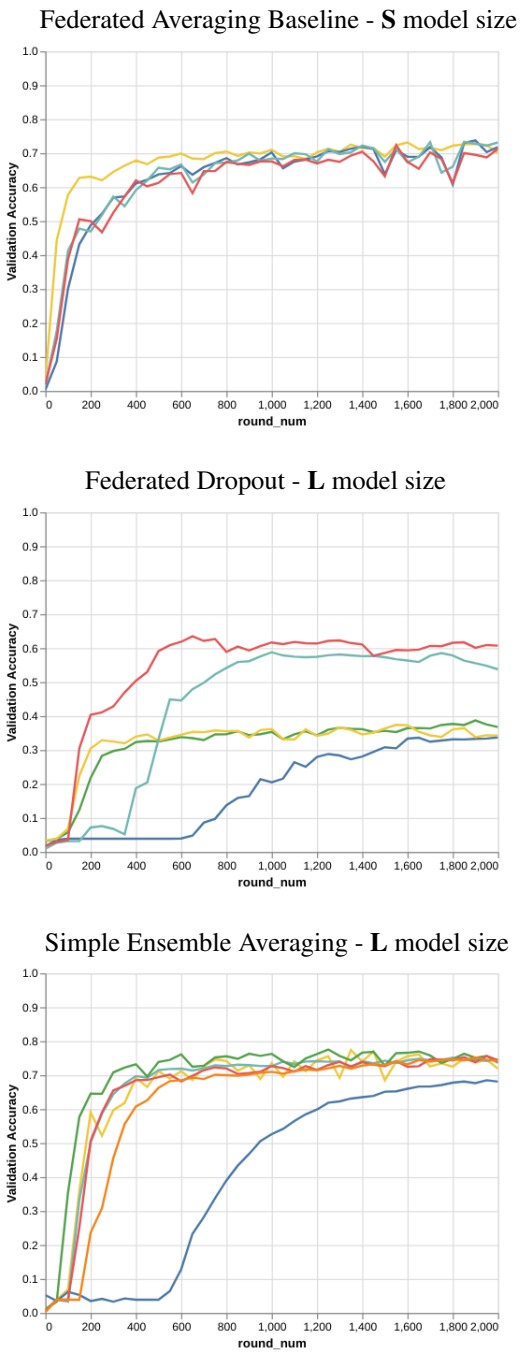

Figure 1. Test accuracy of top five hyperparameter configurations against communication rounds for Federated Averaging on **S**-sized models, Federated Dropout on **L**-sized models, and Simple Ensemble Averaging on **L**-sized models.

Simple ensembling techniques like Simple Ensemble Averaging can leverage the extra capacity of a larger server model. In and of itself, this is not surprising; ensembled models are widely known to outperform their constituent elements. However, as mentioned earlier in the introduction, it *is* surprising that in the federated regime, dropout and en-

sembling seem to behave in fundamentally different ways, as the two are often understood to be intimately related in the centralized setting [7, 11]. We believe this suggests that intuitions from the centralized setting are not plug-and-play in the federated setting. Instead, maybe somewhat obviously, we believe the design of learning algorithms that leverage the parallelizable nature of federated learning must explicitly keep the quirks of federated learning (e.g., infrequent synchronization) in mind.

While the results presented in Table 1 and Figure 1 are essentially the main punchline of the paper, we will perform more ablations that drive this point further. Recall that the main difference between Federated Dropout and Simple Ensemble Averaging is that Federated Dropout is attempting to train several smaller models that share parameters, whereas Simple Ensemble Averaging is training several disjoint models. The ablations we perform now suggest that the best versions of Federated Dropout are simply ones that attempt to train models which are "more" disjoint. Said differently, our experiments suggest that Federated Dropout performs best when it approximates Simple Ensemble Averaging.

4.2. Effect of Submodel Variety

In Table 2, we vary the number of unique submodels that Federated Dropout can select from during training; i.e., we vary the size of $|\mathcal{S}|$. Recall that each client is assigned a seed from $\mathcal{S}$ that they use for the entire training procedure. In Table 2, the test accuracy decreases as the number of unique seeds increases. For this ablation, the best version of Federated Dropout is to use two unique seeds. A potential reason why this performs better than the baseline is that, with high probability, the overlap between the two selected submodels is so small that it is essentially equivalent to Simple Ensemble Averaging with two models ($R = 2$). Indeed, for $|\mathcal{S}| = 2$, we see that the probability that any given parameter is selected for both of the random seeds in $\mathcal{S}$ is $1/64$. For example, this means that we expect to see only 1 out of 64 convolutional filters per layer being shared between the two submodels. When we perform the same ablation for **M** sized server models in Table 3, we see that even selecting two submodels in Federated Dropout hurts performance. We conjecture that this is because the overlap between submodels becomes too large. With **M**-sized server models, we expect to see 2 out of 32 convolutional filters being shared between the two submodels.

4.3. Effect of Submodel Changing Frequency

Before getting to the results, we provide some additional setup. In this subsection, all the clients use the same seed (i.e., they all train the same submodel at any given communication round), but the value of the seed changes over time. In this ablation, we vary how frequently the seed changes. We let E denote the number of communication rounds run before the seed is changed. Recall that $T = 2000$ is the total number of server communication rounds we run our algorithms for; for example, $E = 1000$ corresponds to changing the seed once through training. We denote the number of times the seed has changed throughout training by T/E.

In Table 4 and Table 5, we report how test accuracy changes as a function of E. Similar to the message of the previous subsections, our experiments in this setting find that increasing the frequency of changing the submodel trained hurts performance. Again, following the intuition presented in the previous subsection, training two (nearly) disjoint models one after the other proves to be the best option for **L** sized server models, and training only one client model proves to be the best for **M** sized server models.

$\lvert\mathcal{S}\rvert$	Test accuracy
1	69.48%
2	74.34%
3	72.97%
5	67.55%
10	62.26%
700000	05.09%

Table 2. $|\mathcal{S}|$ against (hyperparameter-tuned) test accuracy for Federated Dropout for **L**-sized server models.

$\lvert\mathcal{S}\rvert$	Test accuracy
1	72.37%
2	71.61%
3	70.94%
5	69.80%
10	68.00%
700000	41.63%

Table 3. $|\mathcal{S}|$ against (hyperparameter-tuned) test accuracy for Federated Dropout for **M**-sized server models.

T/E	E	Test accuracy
2000	1	04.70%
1000	2	04.70%
400	5	04.70%
200	10	14.60%
20	100	58.27%
2	1000	72.75%
1	2000	56.46%

Table 4. Number of rounds before seed change E against (hyperparameter-tuned) test accuracy for Federated Dropout for **L**-sized server models. T/E can be understood as the number of unique submodels trained.

T/E	E	Test accuracy
2000	1	05.57%
1000	2	05.57%
400	5	05.57%
200	10	05.57%
20	100	05.57%
2	1000	05.34%
1	2000	62.70%

Table 5. Number of rounds before seed change E against (hyperparameter-tuned) test accuracy for Federated Dropout for **M**-sized server models. T/E can be understood as the number of unique submodels trained.

5. Takeaways

Based on our experimentation, Federated Dropout does not seem like an effective way of scaling up model sizes in Federated Learning. Far simpler ensembling ideas seem to do much better. Our experiments suggest that Federated Dropout performs the best when it approximates Simple Ensemble Averaging, and even then, it still does not perform as well as Simple Ensemble Averaging. More sophisticated methods of ensembling (e.g., boosting) may prove to do even better. Our results also highlight the differences between dropout and ensembling in the federated setting, contrasting with traditional wisdom in the centralized setting, suggesting that more work needs to be done to better understand how infrequent synchronization affects learning.

References

[1] Nader Bouacida, Jiahui Hou, Hui Zang, and Xin Liu. Adaptive federated dropout: Improving communication efficiency and generalization for federated learning. *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2021. 1, 2

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020. 1

[3] Sebastian Caldas, Jakub Konecný, H. Brendan McMahan, and Ameet S. Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *ArXiv*, abs/1812.07210, 2018. 1, 2, 3

[4] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. B. McMahan, Virginia Smith, and Ameet S. Talwalkar. Leaf: A benchmark for federated settings. *ArXiv*, abs/1812.01097, 2018. 5

[5] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020. 2

[6] Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. Exploiting shared representations for personalized federated learning. *ArXiv*, abs/2102.07078, 2021. 2

[7] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016. 2, 7

[8] Dhruv Guliani, Lillian Zhou, Changwan Ryu, Tien-Ju Yang, Harry Zhang, Yonghui Xiao, Françoise Beaufays, and Giovanni Motta. Enabling on-device training of speech recognition models with federated dropout. *ArXiv*, abs/2110.03634, 2021. 1

[9] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *J. Netw. Comput. Appl.*, 116:1–8, 2018. 2

[10] Jenny Hamer, Mehryar Mohri, and Ananda Theertha Suresh. Fedboost: A communication-efficient algorithm for federated learning. In *ICML*, 2020. 3

[11] Kazuyuki Hara, Daisuke Saitoh, and Hayaru Shouno. Analysis of dropout learning regarded as ensemble learning. *Lecture Notes in Computer Science*, page 72–79, 2016. 2, 7

[12] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *ArXiv*, abs/1811.03604, 2018. 1

[13] Samuel Horvath, Stefanos Laskaridis, Mário Almeida, Ilias Leondiadis, Stylianos I. Venieris, and Nicholas D. Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *ArXiv*, abs/2102.13451, 2021. 1, 2

[14] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv:1912.04977 [cs.LG]*, 2019. 1

[15] Jared Kaplan, Sam McCandlish, T. J. Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeff Wu, and Dario Amodei. Scaling laws for neural language models. *ArXiv*, abs/2001.08361, 2020. 1

[16] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *ArXiv*, abs/1610.05492, 2016. 2

[17] Paul Pu Liang, Terrance Liu, Liu Ziyin, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *ArXiv*, abs/2001.01523, 2020. 2

[18] H. 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 the 20th International Conference on Artificial Intelligence and Statistics*, 2017. 1, 2, 5

[19] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *Proceedings of the Ninth International Conference on Learning Representations*, 2021. 2

[20] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8, 2018. 2, 3

[21] Naichen Shi, Fan Lai, Raed Al Kontar, and Mosharaf Chowdhury. Fed-ensemble: Improving generalization through model ensembling in federated learning. *ArXiv*, abs/2107.10663, 2021. 2, 3

[22] Hakim Sidahmed, Zheng Xu, Ankush Garg, Yuan Cao, and Mingqing Chen. Efficient and private federated learning with partially trainable networks. *ArXiv*, abs/2110.03450, 2021. 2

[23] K. Singhal, Hakim Sidahmed, Zachary Garrett, Shanshan Wu, Keith Rush, and Sushant Prakash. Federated reconstruction: Partially local federated learning. *ArXiv*, abs/2102.03448, 2021. 2

[24] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014. 1

[25] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. *ArXiv*, abs/2007.00644, 2020. 1

[26] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, and Seyit Ahmet Çamtepe. Splitfed: When federated learning meets split learning. *ArXiv*, abs/2004.12088, 2020. 2

[27] Blake Woodworth, Kumar Kshitij Patel, Sebastian Stich, Zhen Dai, Brian Bullins, Brendan Mcmahan, Ohad Shamir, and Nathan Srebro. Is local SGD better than minibatch SGD? In *Proceedings of the 37th International Conference on Machine Learning*, 2020. 1