A GRAPHICAL MODEL PERSPECTIVE ON FEDERATED LEARNING

Christos Louizos, Matthias Reisser, Joseph Soriaga, Max Welling
Qualcomm AI Research *
{clouizos,mreisser,jsoriaga,mwelling}@qti.qualcomm.com

ABSTRACT

Federated learning describes the distributed training of models across multiple clients while keeping the data private on-device. In this work, we formalize the server-orchestrated federated learning process as a *hierarchical latent variable model* where the server provides the parameters of a prior distribution over the client-specific model parameters. We then show that with simple Gaussian priors and a *hard* version of the well known Expectation-Maximization (EM) algorithm, learning in such a model corresponds to `FedAvg`, the most popular algorithm for this federated learning setting. This perspective on federated learning unifies several recent works in the field and opens up the possibility for extensions through different choices in the hierarchical model. Based on this view, we further propose a variant of the hierarchical model that employs prior distributions to promote sparsity. By using the hard-EM algorithm for learning, we obtain `FedSparse`, a procedure that can learn sparse neural networks in the federated learning setting. `FedSparse` reduces communication costs from client to server and vice-versa, as well as the computational costs for inference with the sparsified network – both of which are of great practical importance in federated learning.

1 INTRODUCTION

In this work we provide a novel perspective on server-orchestrated federated learning through the lens of a simple hierarchical latent variable model; the server provides the parameters of a prior distribution over the client specific neural network parameters which are used to "explain" the client specific dataset. A visualization of this graphical model can be seen at Figure 1. We show that when the server provides a Gaussian prior, learning with the *hard* version of the Expectation-Maximization (EM) algorithm will yield the `FedAvg` procedure. This novel view of `FedAvg` has several interesting consequences, as it connects several recent works in federated learning, bridges `FedAvg` with meta-learning and provides a good basis for extending the `FedAvg` algorithm. Through this perspective, we develop `FedSparse` by extending the graphical model to the one in Figure 2. `FedSparse` allows for learning sparse neural network models at the client and server via a careful choice of the priors within the hierarchical model. In this way, it provides models that simultaneously reduce the communication costs and computational requirements at the client devices, an aspect that is important for the "cross-device" setting (Kairouz et al., 2019) of federated learning.

2 A HIERARCHICAL MODEL INTERPRETATION OF FEDERATED LEARNING

The server-orchestrated variant of federated learning is mainly realized via the `FedAvg` (McMahan et al., 2016) algorithm, which is a simple iterative procedure consisting of four steps. At the beginning of each round t, the server communicates the model parameters, let them be $\mathbf{w}$, to a subset of the devices. The devices then proceed to optimize $\mathbf{w}$, *e.g.*, via stochastic gradient descent, on their respective dataset via a given loss function $\mathcal{L}_s(\mathcal{D}_s, \mathbf{w}) := \frac{1}{N_s} \sum_{i=1}^{N_s} L(\mathcal{D}_{si}, \mathbf{w})$ where s indexes the device, $\mathcal{D}_s$ corresponds to the dataset at device s and N_s corresponds to its size. After a specific amount of epochs of optimization on $\mathcal{L}_s$ is performed, the devices communicate the current state of

*Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc. and/or its subsidiaries.

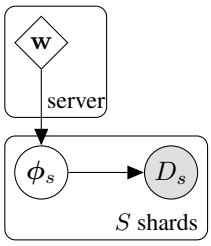


Figure 1: The simple hierarchical model for server-orchestrated federated learning. With a Gaussian prior for the local parameters $\boldsymbol{\phi}_s$ centered at the server parameters $\mathbf{w}$ and hard-EM for learning we obtain `FedAvg`.

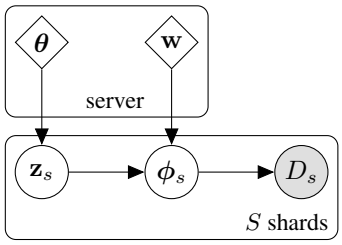


Figure 2: Modifying the hierarchical model to allow for sparsity in the local parameters $\boldsymbol{\phi}_s$ with the spike and slab distribution. The server parameter $\boldsymbol{\theta}$ governs a Bernoulli prior distribution over the local variables $\mathbf{z}_s$, which determine whether a parameter in $\boldsymbol{\phi}_s$ is active or not. When a parameter is active, the prior is a Gaussian centered at the server parameters $\mathbf{w}$. By using hard-EM on this hierarchical model, we obtain `FedSparse`.

their parameters, let it be $\boldsymbol{\phi}_s$, to the server. The server then performs an update to its own model by simply averaging the client specific parameters $\mathbf{w}_t = \frac{1}{S}\sum_s \boldsymbol{\phi}_s$.

2.1 FEDAVG AS LEARNING IN A HIERARCHICAL MODEL

Let us consider a likelihood per shard specific dataset $\mathcal{D}_s$, $p(\mathcal{D}_s|\boldsymbol{\phi}_s)$, where $\boldsymbol{\phi}_s$ are the parameters of the local model, and consider a prior over these parameters, $p(\boldsymbol{\phi}_s|\mathbf{w}) \propto \exp(-\frac{\lambda}{2}\|\boldsymbol{\phi}_s - \mathbf{w}\|^2)$, where $\mathbf{w}$ are the parameters of the prior and those are given by the server. This will yield an objective for the server parameters $\mathbf{w}$ of the form

$$\mathcal{F}(\mathbf{w}) := \sum_s \log p(\mathcal{D}_s|\mathbf{w}) = \sum_s \log \int p(\mathcal{D}_s|\boldsymbol{\phi}_s)\frac{\exp\left(-\frac{\lambda}{2}\|\boldsymbol{\phi}_s - \mathbf{w}\|^2\right)}{Z}\mathrm{d}\boldsymbol{\phi}_s, \tag{1}$$

and corresponds to the graphical model seen at Figure 1. The traditional way to optimize such objectives is through Expectation-Maximization (EM). EM consists of two steps, the E-step where we form the posterior distribution over these latent variables $p(\boldsymbol{\phi}_s|\mathcal{D}_s, \mathbf{w}) = \frac{p(\mathcal{D}_s|\boldsymbol{\phi}_s)p(\boldsymbol{\phi}_s|\mathbf{w})}{p(\mathcal{D}_s|\mathbf{w})}$, and the M-step where we maximize the probability of all $\mathcal{D}_s$ with respect to the parameters of the model $\mathbf{w}$ by marginalizing over this posterior

$$\arg\max_{\mathbf{w}} \sum_s \mathbb{E}_{p(\boldsymbol{\phi}_s|\mathcal{D}_s,\mathbf{w}_{\text{old}})}[\log p(\mathcal{D}_s, \boldsymbol{\phi}_s|\mathbf{w})] = \arg\max_{\mathbf{w}} \sum_s \mathbb{E}_{p(\boldsymbol{\phi}_s|\mathcal{D}_s,\mathbf{w}_{\text{old}})}[\log p(\boldsymbol{\phi}_s|\mathbf{w})]. \tag{2}$$

If we perform a single gradient step for $\mathbf{w}$ in the M-step, this procedure corresponds to doing gradient ascent on the original objective at Eq. 1, a fact we show in Appendix D. When posterior inference is intractable, hard-EM is usually employed as a simpler alternative. In this case we make "hard" assignment for the latent variables $\boldsymbol{\phi}_s$ in the E-step by approximating $p(\boldsymbol{\phi}_s|\mathcal{D}_s, \mathbf{w})$ with its most probable point, *i.e.* $\boldsymbol{\phi}_s^* = \arg\max_{\boldsymbol{\phi}_s} \log p(\mathcal{D}_s|\boldsymbol{\phi}_s) + \log p(\boldsymbol{\phi}_s|\mathbf{w})$. This is usually easier to do as we can use techniques such as stochastic gradient ascent. Given these hard assignments, the M-step then corresponds to another simple maximization $\arg\max_{\mathbf{w}} \frac{1}{S}\sum_s \log p(\boldsymbol{\phi}_s^*|\mathbf{w})$. As a result, performing hard-EM on the objective of Eq. 1 corresponds to a block coordinate ascent type of algorithm on the following objective function

$$\arg\max_{\boldsymbol{\phi}_{1:S},\mathbf{w}} \sum_s \log p(\mathcal{D}_s|\boldsymbol{\phi}_s) + \log p(\boldsymbol{\phi}_s|\mathbf{w}), \tag{3}$$

where we alternate between optimizing $\boldsymbol{\phi}_{1:S}$ and $\mathbf{w}$ while keeping the other fixed. How does this learning procedure correspond to `FedAvg`? By letting $\lambda \to 0$ in Eq. 1 it is clear that the hard assignments in the E-step mimic the process of optimizing a local model on the data of each shard. In fact, even by optimizing the model locally with stochastic gradient ascent for a fixed number of iterations with a given learning rate we implicitly assume a specific prior over the parameters (Grant et al., 2018). After obtaining $\boldsymbol{\phi}_s^*$ the M-step then corresponds to maximizing $\sum_s -\frac{\lambda}{2}\|\boldsymbol{\phi}_s^* - \mathbf{w}\|^2 + C$

w.r.t. $\mathbf{w}$ and we can easily find a closed form solution by setting the derivative of the objective w.r.t. $\mathbf{w}$ to zero and solving for $\mathbf{w}$:

$$\frac{\partial(\sum_s -\frac{\lambda}{2}\|\boldsymbol{\phi}_s^* - \mathbf{w}\|^2 + C)}{\partial \mathbf{w}} = 0 \Rightarrow \lambda \sum_s (\boldsymbol{\phi}_s^* - \mathbf{w}) = 0 \Rightarrow \mathbf{w} = \frac{1}{S}\sum_s \boldsymbol{\phi}_s^*. \tag{4}$$

It is easy to see that the optimal solution for $\mathbf{w}$ given $\boldsymbol{\phi}_{1:S}^*$ is the same as the one from `FedAvg`. Of course, `FedAvg` does not optimize the local parameters $\boldsymbol{\phi}_s$ to convergence at each round, so one might wonder whether this correspondence is still valid. It turns out that the alternating procedure of EM corresponds to block coordinate ascent on a single objective function, the variational lower bound of the marginal log-likelihood (Neal & Hinton, 1998) of a given model. More specifically for our setting, we can see that the EM iterations perform block coordinate ascent on:

$$\sum_s \mathbb{E}_{q_{\mathbf{w}_s}(\boldsymbol{\phi}_s)}\big[\log p(\mathcal{D}_s|\boldsymbol{\phi}_s) + \log p(\boldsymbol{\phi}_s|\mathbf{w})\big] + H[q_{\mathbf{w}_s}(\boldsymbol{\phi}_s)] \tag{5}$$

to optimize $\mathbf{w}_{1:S}$ and $\mathbf{w}$, where $\mathbf{w}_s$ are the parameters of the variational approximation to the posterior distribution $p(\boldsymbol{\phi}_s|\mathcal{D}_s, \mathbf{w})$ and $H[q]$ corresponds to the entropy of the q distribution. To obtain the hard-EM procedure, and thus `FedAvg`, we can use a (numerically) deterministic distribution for $\boldsymbol{\phi}_s$, $q_{\mathbf{w}_s}(\boldsymbol{\phi}_s) := \mathcal{N}(\mathbf{w}_s, \epsilon\mathbf{I})$. This leads us to the same objective as in Eq. 3, since the expectation concentrates on a single term and the entropy of $q_{\mathbf{w}_s}(\boldsymbol{\phi}_s)$ becomes a constant independent of the optimization. In this case, the optimized value for $\boldsymbol{\phi}_s$ after a fixed number of steps corresponds to the $\mathbf{w}_s$ of the variational approximation.

It is interesting to contrast recent literature under the lens of this hierarchical model interpretation. Optimizing the same model with hard-EM but with a non-trivial λ results into the same procedure that was proposed by Li et al. (2018). Furthermore, using the difference of the local parameters to the global parameters as a "gradient" (Reddi et al., 2020) is equivalent to hard-EM on the same model where in the M-step, instead of a closed form update, we take a single gradient step and absorb the scaling λ in the learning rate. In addition, this view makes precise the idea that `FedAvg` is a meta-learning algorithm (Jiang et al., 2019); the underlying hierarchical model it optimizes is similar to the ones used in meta-learning (Grant et al., 2018; Chen et al., 2019). Besides connecting recent work in FL, this novel view also serves as a good basis for new algorithms for federated learning by changing the prior in the hierarchical model. In this work we focus on tackling the communication and computational costs of FL, which is important and highly beneficial for practical applications of "cross-device" FL (Kairouz et al., 2019). For this reason, we replace the Gaussian prior with a sparsity inducing prior, namely the spike and slab (Mitchell & Beauchamp, 1988). We describe the resulting algorithm, `FedSparse`, in the next section. Details for alternative priors can be found at appendix G.

2.2 THE FEDSPARSE ALGORITHM: SPARSITY IN FEDERATED LEARNING

Encouraging sparsity in FL has two main advantages; the model becomes smaller and therefore less resource intensive to evaluate. Furthermore, it cuts down on communication costs as the pruned parameters do not need to be communicated. The golden standard for sparsity in probabilistic models is the spike and slab prior, which is a mixture of two components, a delta spike at zero, $\delta(0)$, and a continuous distribution over the real line, *i.e.* the slab. More specifically, by adopting a Gaussian slab for each local parameter $\boldsymbol{\phi}_{si}$ we have that

$$p(\mathbf{z}_{si}) = \text{Bern}(\boldsymbol{\theta}_i), \;\; p(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}=1, \mathbf{w}_i) = \mathcal{N}(\boldsymbol{\phi}_{si}|\mathbf{w}_i, 1/\lambda), \;\; p(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}=0) = \delta(0) \tag{6}$$

$$p(\boldsymbol{\phi}_{si}|\boldsymbol{\theta}_i, \mathbf{w}_i) = \sum_{\mathbf{z}_{si}} p(\mathbf{z}_{si}|\boldsymbol{\theta}_i) p(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}, \mathbf{w}_i), \tag{7}$$

where $\mathbf{z}_{si}$ plays the role of a "gating" variable that switches the parameter $\boldsymbol{\phi}_{si}$ on or off. We now modify the hierarchical model at Figure 1 to use this new prior. The resulting hierarchical model can be seen at Figure 2, where $\mathbf{w}$, $\boldsymbol{\theta}$ will be the server side model weights and probabilities of the binary gates. In order to stay close to the `FedAvg` paradigm of simple point estimation, we will use hard-EM in order to optimize $\mathbf{w}, \boldsymbol{\theta}$. By using approximate distributions $q_{\mathbf{w}_s}(\boldsymbol{\phi}_s|\mathbf{z}_s)$, $q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)$, the variational lower bound for this model becomes

$$\sum_s \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s) q_{\mathbf{w}_s}(\boldsymbol{\phi}_s|\mathbf{z}_s)}\big[\log p(\mathcal{D}_s|\boldsymbol{\phi}_s) + \log p(\boldsymbol{\phi}_s|\mathbf{w}, \mathbf{z}_s) + \\ + \log p(\mathbf{z}_s|\boldsymbol{\theta}) - \log q_{\mathbf{w}_s}(\boldsymbol{\phi}_s|\mathbf{z}_s)\big] + H[q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)], \tag{8}$$

which is to be optimized with respect to $\mathbf{w}_{1:S}, \mathbf{w}, \boldsymbol{\pi}_{1:S}, \boldsymbol{\theta}$. For the shard specific weight distributions, as they are continuous, we will use $q_{\mathbf{w}_s}(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}=1) := \mathcal{N}(\mathbf{w}_{si}, \epsilon), q(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}=1) := \mathcal{N}(0, \epsilon)$ with $\epsilon \approx 0$ which will be, numerically speaking, deterministic. For the gating variables, as they are binary, we will use $q_{\boldsymbol{\pi}_{si}}(\mathbf{z}_{si}) := \text{Bern}(\boldsymbol{\pi}_{si})$ with $\boldsymbol{\pi}_{si}$ being the probability of activating local gate $\mathbf{z}_{si}$. In order to do hard-EM for the binary variables, we will remove the entropy term for the $q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)$ from the aforementioned bound as this will encourage the approximate distribution to move towards the most probable value for $\mathbf{z}_s$. After some further simplifications, we arrive at the final objective

$$\sum_s \mathcal{L}_s(\mathcal{D}_s, \mathbf{w}, \boldsymbol{\theta}, \mathbf{w}_s, \boldsymbol{\pi}_s) := \sum_s \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)} \left[\sum_i^{N_s} L(\mathcal{D}_{si}, \mathbf{w}_s \odot \mathbf{z}_s) \right] - \frac{\lambda}{2} \sum_j \boldsymbol{\pi}_{sj} (\mathbf{w}_{sj} - \mathbf{w}_j)^2 - \lambda_0 \sum_j \boldsymbol{\pi}_{sj} + \sum_j \left(\boldsymbol{\pi}_{sj} \log \boldsymbol{\theta}_j + (1 - \boldsymbol{\pi}_{sj}) \log(1 - \boldsymbol{\theta}_j) \right). \tag{9}$$

The derivation can be found at Appendix E. It is interesting to see that the final objective at each shard intuitively tries to find a trade-off between four things: 1) explaining the local dataset $\mathcal{D}_s$, 2) having the local weights close to the server weights (regulated by λ), 3) having the local gate probabilities close to the server probabilities and 4) reducing the local gate activation probabilities so as to prune away a parameter (regulated by λ_0). The latter is an L_0 regularization term, akin to the one proposed by Louizos et al. (2017). Now let us consider what happens at the server after the local shard, through some procedure, optimized $\mathbf{w}_s$ and $\boldsymbol{\pi}_s$. Since the server loss for $\mathbf{w}, \boldsymbol{\theta}$ is the sum of all local losses, the gradient and stationary points for each of the parameters will be

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_s \lambda \boldsymbol{\pi}_s (\mathbf{w}_s - \mathbf{w}), \quad \mathbf{w} = \frac{1}{\sum_j \boldsymbol{\pi}_j} \sum_s \boldsymbol{\pi}_s \mathbf{w}_s \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \sum_s \left(\frac{\boldsymbol{\pi}_s}{\boldsymbol{\theta}} - \frac{1 - \boldsymbol{\pi}_s}{1 - \boldsymbol{\theta}} \right), \quad \boldsymbol{\theta} = \frac{1}{S} \sum_s \boldsymbol{\pi}_s, \tag{11}$$

i.e., the stationary points are a weighted average of the local weights and an average of the local probabilities of keeping these weights. Therefore, since the $\boldsymbol{\pi}_s$ are being optimized to be sparse through the L_0 penalty, the server probabilities $\boldsymbol{\theta}$ will also become small for the weights that are used by only a small fraction of the shards. As a result, to obtain the final sparse architecture, we can prune the weights whose corresponding server inclusion probabilities $\boldsymbol{\theta}$ are less than a threshold, *e.g.* 0.1. It should be mentioned that the server needs to communicate to the clients the updated distributions at each round. Unfortunately, for simple unstructured pruning, this doubles the communication cost as for each weight $\mathbf{w}_i$ there is an associated $\boldsymbol{\theta}_i$ that needs to be sent to the client. To mitigate this effect we will employ structured pruning, which introduces a single additional parameter for each group of weights. For groups of moderate sizes, *e.g.*, the set of weights of a given convolutional filter, the extra overhead is small. In the appendix, we provide further practical details for `FedSparse` along with how it can lead to reductions in communication costs.

3 Experiments

To evaluate `FedSparse` we considered the non-i.i.d. Femnist classification with a LeNet-5 convolutional architecture (LeCun et al., 1998) which was optimized for 6k rounds. As a baseline that also reduces communication costs by sparsifying the model, we consider the federated dropout procedure from Caldas et al. (2018), which we refer to as `FedDrop`. We present the results for `FedSparse` with regularization strengths that target three sparsity levels: low, mid and high. The results can be seen at Fig. 3, where the x-axis corresponds to the total GB communicated and the y-axis to the server model accuracy on the union of the shard specific test sets. More details along with additional results on other datasets can be found at Appendix C.

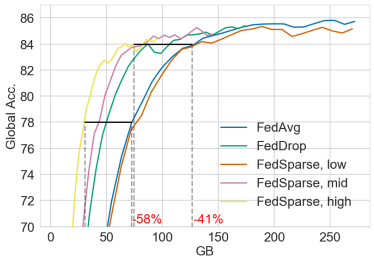


Figure 3: Femnist results.

We see that `FedSparse` reaches comparable accuracy to `FedDrop` and `FedAvg` while requiring in general less communication. More specifically, in the high sparsification setting, it can reach $\sim$84% accuracy while requiring $\sim$41% less communication compared to `FedAvg`. We can also see that the sparsity in the model depends on the regularization strength, and it can reach up to $\sim$60%. The figures can be found in the appendix.

REFERENCES

Mohammad Mohammadi Amiri, Deniz Gunduz, Sanjeev R Kulkarni, and H Vincent Poor. Federated learning with quantized global model updates. *arXiv preprint arXiv:2006.10672*, 2020.

Kambiz Azarian, Yash Bhalgat, Jinwon Lee, and Tijmen Blankevoort. Learned threshold pruning. *arXiv preprint arXiv:2003.00075*, 2020.

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

Yutian Chen, Abram L Friesen, Feryal Behbahani, David Budden, Matthew W Hoffman, Arnaud Doucet, and Nando de Freitas. Modular meta-learning with shrinkage. *arXiv preprint arXiv:1909.05557*, 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

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

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 preprint arXiv:1912.04977*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

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

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

Toby J Mitchell and John J Beauchamp. Bayesian variable selection in linear regression. *Journal of the american statistical association*, 83(404):1023–1032, 1988.

Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pp. 355–368. Springer, 1998.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

APPENDIX

A REDUCING THE COMMUNICATION COST

The framework described in the main text allow us to learn a more efficient model. We now discuss how we can use it in order to cut down both download and upload communication costs during training.

Reducing client to server communication cost In order to reduce the client to server cost we will communicate sparse samples from the local distributions instead of the distributions themselves; in this way we do not have to communicate the zero values of the parameter vector which leads to large savings. More specifically, we can express the gradients and stationary points for the server weights and probabilities as follows

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_s \lambda \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)} \left[\mathbf{z}_s (\mathbf{w}_s - \mathbf{w}) \right], \tag{12}$$

$$\mathbf{w} = \mathbb{E}_{q_{\boldsymbol{\pi}_{1:S}}(\mathbf{z}_{1:S})} \left[\frac{1}{\sum_j \mathbf{z}_j} \sum_s \mathbf{z}_s \mathbf{w}_s \right], \tag{13}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \sum_s \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)} \left[\frac{\mathbf{z}_s}{\boldsymbol{\theta}} - \frac{1 - \mathbf{z}_s}{1 - \boldsymbol{\theta}} \right], \tag{14}$$

$$\boldsymbol{\theta} = \frac{1}{S} \sum_s \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)} \left[\mathbf{z}_s \right]. \tag{15}$$

As a result, we can then communicate from the client only the subset of the local weights $\hat{\mathbf{w}}_s$ that are non-zero in $\mathbf{z}_s \sim q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)$, $\hat{\mathbf{w}}_s = \mathbf{w}_s \odot \mathbf{z}_s$, along with the $\mathbf{z}_s$. Having access to those samples, the server can then form 1-sample stochastic estimates of either the gradients or the stationary points for $\mathbf{w}, \boldsymbol{\theta}$. Notice that this is a way to reduce communication without adding bias in the gradients of the original objective. In case that we are willing to incur extra bias, we can further use techniques such as quantization (Amiri et al., 2020) and top-k gradient selection (Lin et al., 2017) to reduce communication even further. Such approaches are left for future work.

Reducing the server to client communication cost The server needs to communicate to the clients the updated distributions at each round. Unfortunately, for simple unstructured pruning, this doubles the communication cost as for each weight $\mathbf{w}_i$ there is an associated $\boldsymbol{\theta}_i$ that needs to be sent to the client. To mitigate this effect we will employ structured pruning, which introduces a single additional parameter for each group of weights. For groups of moderate sizes, *e.g.*, the set of weights of a given convolutional filter, the extra overhead is small. We can also take the communication cost reductions one step further if we allow for some bias in the optimization procedure; we can prune the global model during training after every round and thus send to each of the clients only the subset of the model that has survived. Notice that this is easy to do and does not require any data at the server. The inclusion probabilities $\boldsymbol{\theta}$ are available at the server, so we can remove the parameters that have $\boldsymbol{\theta}$ less than a threshold, *e.g.* 0.1. This can lead to large reductions in communication costs, especially once the model becomes sufficiently sparse.

B FEDSPARSE IN PRACTICE

Local optimization While optimizing for $\mathbf{w}_s$ locally is straightforward to do with gradient based optimizers, $\boldsymbol{\pi}_s$ is more tricky, as the expectation over the binary variables $\mathbf{z}_s$ in Eq. 9 is intractable to compute in closed form and using Monte-Carlo integration does not yield reparametrizable samples. To circumvent these issues, we rewrite the objective in an equivalent form and use the hard-concrete relaxation from (Louizos et al., 2017), which can allow for the straightforward application of gradient ascent. We provide the details in Appendix F. When the client has to communicate to the server, we propose to form $\hat{\mathbf{w}}_s$ by sampling from the zero-temperature relaxation, which yields exact binary samples. Furthermore, at the beginning of each round, following the practice of `FedAvg`, the participating clients initialize their approximate posteriors to be equal to the priors that were

Algorithm 1 The server side algorithm for `FedSparse` (assuming weight sparsity for simplicity). $\sigma(\cdot)$ is the sigmoid function, ϵ is the threshold for pruning.

Initialize $\mathbf{v}$ and $\mathbf{w}$
for round t in $1, \dots T$ **do**
  $\boldsymbol{\tau} \leftarrow \log(1 + \exp(\mathbf{v}))$
  $\boldsymbol{\theta} \leftarrow \sigma\left((|\mathbf{w}| - \boldsymbol{\tau})/T\right)$
  $\mathbf{w} \leftarrow \mathbb{I}[\boldsymbol{\theta} > \epsilon]\mathbf{w}$ ▷ prune global model
  Initialize $\nabla^t_{\mathbf{w}} = \mathbf{0}, \nabla^t_{\mathbf{v}} = \mathbf{0}$
  for s in random subset of the clients **do**
    $\mathbf{z}_s, \hat{\mathbf{w}}^t_s \leftarrow \textsc{Client}(s, \mathbf{w}, \mathbf{v})$
    $\nabla^t_{\mathbf{w}} += \mathbf{z}_s(\hat{\mathbf{w}}^t_s - \mathbf{w})$
    $\nabla^t_{\mathbf{v}} += -\left(\mathbf{z}_s(1 - \boldsymbol{\theta}) - (1 - \mathbf{z}_s)\boldsymbol{\theta}\right)\sigma(\mathbf{v})/T$
  end for
  $\mathbf{w}^{t+1}, \mathbf{v}^{t+1} \leftarrow \textsc{Adam}(\nabla^t_{\mathbf{w}}), \textsc{AdaMax}(\nabla^t_{\mathbf{v}})$
end for

Algorithm 2 The client side algorithm for `FedSparse`.

Get $\mathbf{w}, \mathbf{v}$ from the server
$\boldsymbol{\theta} \leftarrow \sigma\left((|\mathbf{w}| - \boldsymbol{\tau})/T\right)$
$\mathbf{w}_s, \mathbf{v}_s \leftarrow \mathbf{w}, \mathbf{v}$
for epoch e in $1, \dots, E$ **do**
  for batch $b \in B$ **do**
    $\boldsymbol{\tau}_s \leftarrow \log(1 + \exp(\mathbf{v}_s))$
    $\boldsymbol{\pi}_s \leftarrow \sigma\left((|\mathbf{w}_s| - \boldsymbol{\tau}_s)/T\right)$
    $L_s \leftarrow \mathcal{L}_s(b, \mathbf{w}, \boldsymbol{\theta}, \mathbf{w}_s, \boldsymbol{\pi}_s)$
    $\mathbf{w}_s \leftarrow \text{SGD}(\nabla_{\mathbf{w}_s} L_s)$
    $\mathbf{v}_s \leftarrow \textsc{AdaMax}(\nabla_{\mathbf{v}_s} L_s)$
  end for
end for
$\boldsymbol{\pi}_s \leftarrow \sigma\left((|\mathbf{w}_s| - \boldsymbol{\tau}_s)/T\right)$
$\mathbf{z}_s \sim q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)$
return $\mathbf{z}_s$, $\mathbf{z}_s \odot \mathbf{w}_s$

communicated from the server. Empirically, we found that this resulted in better global model accuracy.

Parameterization of the probabilities There is evidence that such optimization based pruning can be inferior to simple magnitude based pruning (Gale et al., 2019). We therefore take an approach that combines the two and reminisces the recent work of Azarian et al. (2020). We parameterize the probabilities $\boldsymbol{\theta}, \boldsymbol{\pi}_s$ as a function of the model weights and magnitude based thresholds that regulate how active a parameter can be. More specifically, we use the following parameterization

$$\theta_g := \sigma\left(\frac{\|\mathbf{w}_g\|_2 - \tau_g}{T}\right), \;\; \pi_{sg} := \sigma\left(\frac{\|\mathbf{w}_{sg}\|_2 - \tau_{sg}}{T}\right), \tag{16}$$

where the subscript g denotes the group, $\sigma(\cdot)$ is the sigmoid function, τ_g, τ_{sg} are the global and client specific thresholds for a given group g and T is a temperature hyperparameter. Following Azarian et al. (2020) we also "detach" the gradient of the weights through $\boldsymbol{\theta}, \boldsymbol{\pi}_s$, to avoid decreasing the probabilities by just shrinking the weights. With this parametrization we lose the ability to get a closed form solution for the server thresholds, but nonetheless we can still perform gradient based optimization at the server by using the chain rule. For a positive threshold, we use a parametrization in terms of a softplus function, *i.e.*, $\boldsymbol{\tau} = \log(1 + \exp(\mathbf{v}))$ where $\mathbf{v}$ is the learnable parameter. The `FedSparse` algorithm is described in Alg. 1, 2.

C EXPERIMENTAL DETAILS AND ADDITIONAL RESULTS

Besides the experiment in the main text, we also performed additional experiments on federated versions of CIFAR 10 and CIFAR 100. For the federated version of CIFAR10 classification we partition the data among 100 shards in a non-i.i.d. way by following Hsu et al. (2019). For the model we employ a LeNet-5 convolutional architecture (LeCun et al., 1998) with the addition of dropout(0.1) for the second convolutional layer and dropout(0.3) for the first fully connected layer in order to prevent overfitting locally at the shard. We optimize the model for 1k communication rounds. For CIFAR 100 classification, we consider the 500 shard federated version from Reddi et al. (2020). For the model we use a ResNet20, where we replace the batch normalization layers with group normalization, following Reddi et al. (2020), and we optimize for 6k rounds.

During training we randomly select 10 clients without replacement in a given round but with replacement across rounds. For the local optimizer of the weights we use stochastic gradient descent with a learning rate of 0.05, whereas for the global optimizer we use Adam (Kingma & Ba, 2014) with the default hyperparameters provided in (Kingma & Ba, 2014). For the pruning thresholds in `FedSparse` we used the Adamax (Kingma & Ba, 2014) optimizer with $1e-3$ learning rate at the shard level and the Adamax optimizer with $1e-2$ learning rate at the server. For all three of the tasks we used $E = 1$ with a batch size of 64 for CIFAR10 and 20 for CIFAR100 and Femnist. It should be noted that for all the methods we performed gradient based optimization using the difference gradient for the weights (Reddi et al., 2020) instead of averaging.

For the `FedDrop` baseline, we used a very small dropout rate of 0.01 for the input and output layer and tuned the dropout rates for convolutional and fully connected layers separately in order to optimize the accuracy / communication tradeoff. For convolutional layers we considered rates in $\{0.1, 0.2, 0.3\}$ whereas for the fully connected layers we considered rates in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. For CIFAR10 we did not employ the additional dropout noise at the shard level, since we found that it was detrimental for the `FedDrop` performance. Furthermore, for Resnet20 on CIFAR100 we did not apply federated dropout at the output layer. For CIFAR10 the best performing dropout rates were 0.1 for the convolutional and 0.5 for the fully connected, whereas for CIFAR100 it was 0.1 for the convolutional. For Femnist, we saw that a rate of 0.2 for the convolutional and a rate of 0.4 for the fully connected performed better.

For `FedSparse`, we initialized $\mathbf{v}$ such that the thresholds $\boldsymbol{\tau}$ lead to $\boldsymbol{\theta} = 0.99$ initially, *i.e.* we started from a dense model. The temperature for the sigmoid in the parameterization of the probabilities was set to $T = 0.001$. Furthermore, we downscaled the cross-entropy term between the client side probabilities, $\boldsymbol{\pi}_s$, and the server side probabilities, $\boldsymbol{\theta}$ by mutltiplying it with $1e-4$. Since at the beginning of each round we were always initializing $\boldsymbol{\pi}_S = \boldsymbol{\theta}$ and we were only optimizing for a small number of steps before synchronizing, we found that the full strength of the cross-entropy was not necessary. Furthermore, for similar reasons, *i.e.* we set $\mathbf{w}_s = \mathbf{w}$ at the beginning of each round, we also used $\lambda = 0$ for the drift term $\frac{\lambda}{2}\boldsymbol{\pi}_{sj}(\mathbf{w}_s - \mathbf{w})^2$. The remaining hyperparameter λ_0 dictates how sparse the final model will be. For the LeNet-5 model the λ_0's we report are $\{5e-7, 5e-6, 5e-5\}$ for the "low", "mid" and "high" settings respectively, which were optimized for CIFAR10 and used as-is for Femnist. For CIFAR100 and Resnet20, we did not perform any pruning for the output layer and the λ_0's for the "low", "mid" and "high" settings were $\{5e-7, 5e-6, 5e-5\}$ respectively. These were chosen so that we obtain models with comparable sparsity ratios as the one on CIFAR10.

C.1 ADDITIONAL RESULTS

Besides evaluating `FedSparse` and the baselines on the server model accuracy as a function of the communication costs, we also considered an additional metric which corresponds to the average accuracy of the shard specific "local models" on the shard specific test sets. The "local model" on each shard is the model configuration that the shard last communicated to the server, and serves as a proxy for the personalized model performance on each shard. The later metric is motivated from the meta-learning (Jiang et al., 2019) and hierarchical model view of federated learning, and corresponds to using the local posteriors for prediction on the local test set instead of the server side priors. The assumption we make here, is that each client has the same data generating mechanism for its training and test sets, which is not an unrealistic assumption to make in practice.

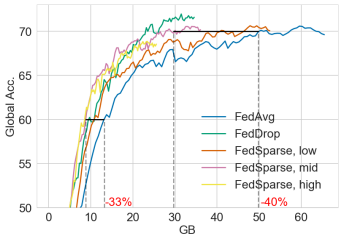
Accuracy vs. communication costs results on all datasets Here we show the table and figures from the results on all three tasks, by considering either the server model performance or the local models performance.

We can see that for CIFAR 10, the `FedSparse` models with medium (∼45%) and high (∼62%) sparsity outperform all other methods for small communications budgets on the global accuracy front, but are eventually surpassed by `FedDrop` on higher budgets. However, on the local accuracy front, we see that the `FedSparse` models Pareto dominate both baselines, achieving, *e.g.*, 87% local accuracy with 43% less communication compared to `FedAvg`. Overall, judging the final performance only, we see that `FedDrop` reaches the best accuracy on the global model, but `FedSparse` reaches the best accuracy in the local models.

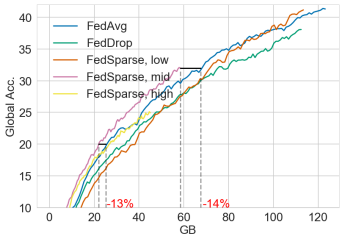
On CIFAR 100, the differences are less pronounced, as the models did not fully converge for the maximum number of rounds we use. Nevertheless, we still observe similar patterns; for small communication budgets, the sparser models are better for both the global and local accuracy as, *e.g.*, they can reach 32% global accuracy while requiring 13% less communication than `FedAvg`.

Table 1: Average global, local test-set accuracies across clients in %, along with total communications costs in GB and sparsity of the final model for Cifar10, Cifar100 and Femnist. We report the average over the last 10 evaluations.

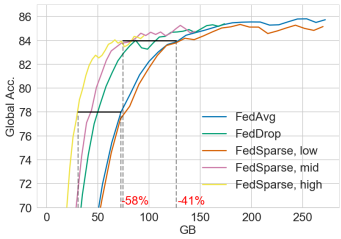
Method	Cifar 10				Cifar 100				Femnist			
	G.Acc.	L.Acc.	Comm.	Spars.	G.Acc.	L.Acc.	Comm.	Spars.	G.Acc.	L.Acc.	Comm.	Spars.
`FedAvg`	69.97	86.71	65	-	41.11	61.61	123	-	85.62	90.81	272	-
`FedDrop`	71.54	86.86	35	-	37.65	58.02	112	-	85.23	89.37	174	-
`FedSparse`, low	70.30	87.65	52	20.1	40.69	61.31	113	2.2	85.03	90.82	270	1.0
`FedSparse`, mid	70.30	87.54	36	45.7	31.94	54.83	59	51.0	84.83	90.61	145	47.0
`FedSparse`, high	68.46	87.17	26	62.4	24.68	49.58	45	62.9	84.13	89.68	95	65.5



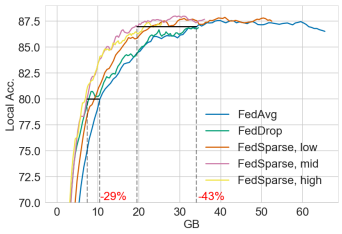
(a) CIFAR 10 global acc.



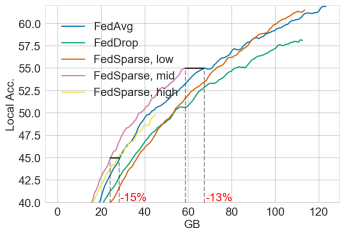
(b) CIFAR100 global acc.



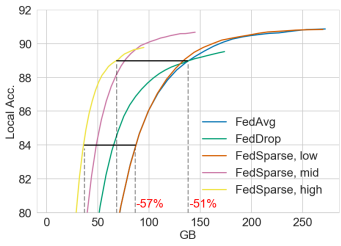
(c) Femnist global acc.



(a) CIFAR 10 local acc.



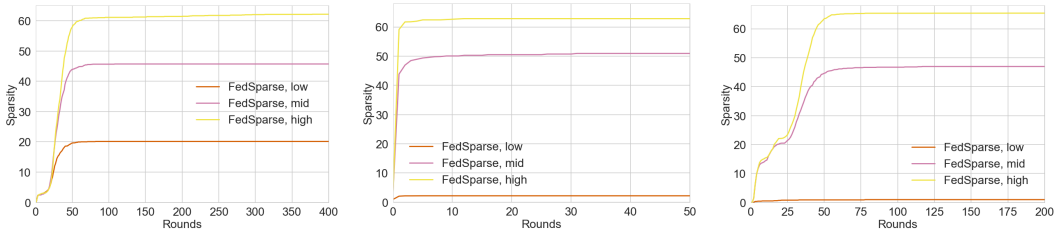
(b) CIFAR100 local acc.



(c) Femnist local acc.

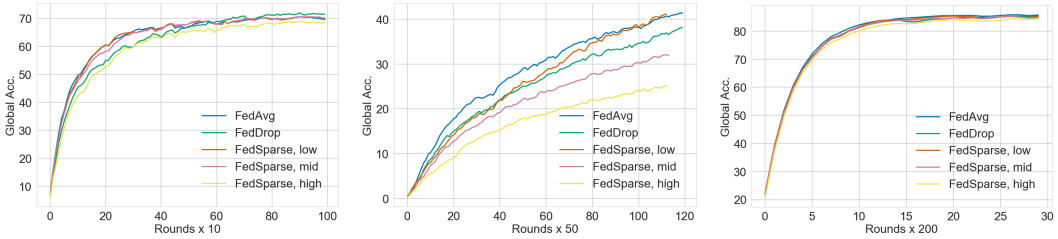
Evolution of sparsity We show the evolution of the sparsity ratios for all tasks and configurations in the following plot. We can see that in all settings the model attains its final sparsity quite early in training.

Convergence plots in terms of communication rounds. In order to understand whether the extra noise is detrimental to the convergence speed of `FedSparse`, we plot the validation accuracy in terms of communication rounds for all tasks and baselines. As it can be seen, there is no inherent



(a) CIFAR10 sparsity (b) CIFAR100 sparsity (c) Femnist sparsity

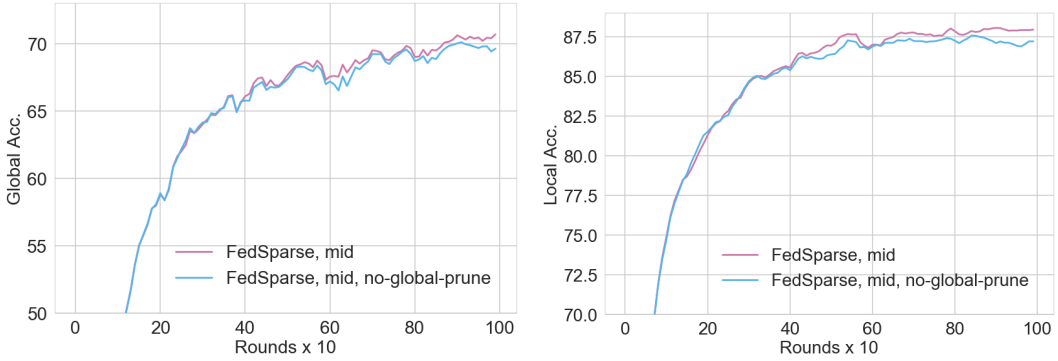
difference before `FedSparse` starts pruning. This happens quite early in training for CIFAR 100 thus it is there where we observe the most differences.



(a) CIFAR10 (b) CIFAR100 (c) Femnist

Figure 7: Evolution of the validation accuracy in terms of communication rounds.

Impact of server side pruning. In order to understand whether server side pruning is harmful for convergence, we plot both the global and average local validation accuracy on CIFAR 10 for the "mid" setting of `FedSparse` with and without server side pruning enabled. As we can see, there are no noticeable differences and in fact, pruning results into a slightly better overall performance.



(a) CIFAR10 global val. acc. (b) CIFAR100 local val. acc.

Figure 8: Evolution of the validation accuracy in terms of communication rounds with and without server side pruning.

D CORRESPONDENCE BETWEEN SINGLE STEP EM AND GRADIENT ASCENT

With the addition of the auxiliary variables ϕ_s we have that the overall objective for the server becomes

$$\arg\max_{\mathbf{w}} \frac{1}{S} \sum_{s=1}^{S} \log \int p(\mathcal{D}_s|\phi_s) p(\phi_s|\mathbf{w}) \mathrm{d}\phi_s. \tag{17}$$

By performing EM with a single gradient step for $\mathbf{w}$ in the M-step (instead of full maximization), we are essentially doing gradient ascent on the original objective at 17. To see this, we can take the gradient of Eq. 17 w.r.t. $\mathbf{w}$ where $Z_s = \int p(\mathcal{D}_s|\boldsymbol{\phi}_s)p(\boldsymbol{\phi}_s|\mathbf{w})\mathrm{d}\boldsymbol{\phi}_s$

$$\frac{1}{S}\sum_s \frac{1}{Z_s}\int p(\mathcal{D}_s|\boldsymbol{\phi}_s)\frac{\partial p(\boldsymbol{\phi}_s|\mathbf{w})}{\partial \mathbf{w}}\mathrm{d}\boldsymbol{\phi}_s = \tag{18}$$

$$\frac{1}{S}\sum_s \int \frac{p(\mathcal{D}_s|\boldsymbol{\phi}_s)p(\boldsymbol{\phi}_s|\mathbf{w})}{Z_s}\frac{\partial \log p(\boldsymbol{\phi}_s|\mathbf{w})}{\partial \mathbf{w}}\mathrm{d}\boldsymbol{\phi}_s = \tag{19}$$

$$\frac{1}{S}\sum_s \int p(\boldsymbol{\phi}_s|\mathcal{D}_s,\mathbf{w})\frac{\partial \log p(\boldsymbol{\phi}_s|\mathbf{w})}{\partial \mathbf{w}}\mathrm{d}\boldsymbol{\phi}_s \tag{20}$$

where to compute Eq. 20 we see that we first have to obtain the posterior distribution of the local variables $\boldsymbol{\phi}_s$ and then estimate the gradient for $\mathbf{w}$ by marginalizing over this posterior.

E Derivation of the local loss for FedSparse

Let $p(\boldsymbol{\phi}_{si}|\mathbf{w}_i, \mathbf{z}_{si}=1) = \mathcal{N}(\mathbf{w}_i, 1/\lambda), p(\boldsymbol{\phi}_{si}|\mathbf{w}_i, \mathbf{z}_{si}=0) = \mathcal{N}(0, 1/\lambda_2)$ and $q(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}=1) = \mathcal{N}(\mathbf{w}_{si}, \epsilon^2), q(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}=0) = \mathcal{N}(0, \epsilon^2)$. Furthermore, let $q(\mathbf{z}_{si}) = \text{Bern}(\boldsymbol{\pi}_{si})$. The local objective that stems from 8 can be rewritten as:

$$\begin{aligned}\underset{\mathbf{w}_{1:S},\boldsymbol{\pi}_{1:S}}{\arg\max}\, \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)q_{\mathbf{w}_s}(\boldsymbol{\phi}_s|\mathbf{z}_s)}\big[\log p(\mathcal{D}_s|\boldsymbol{\phi}_s)\big] - \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)}\big[KL(q_{\mathbf{w}_s}(\boldsymbol{\phi}_s|\mathbf{z}_s)||p(\boldsymbol{\phi}_s|\mathbf{w},\mathbf{z}_s))\big] \\ + \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)}[\log p(\mathbf{z}_s|\boldsymbol{\theta})],\end{aligned} \tag{21}$$

where we omitted from the objective the entropy of the distribution over the local gates.

One of the quantities that we are after is

$$\begin{aligned}&\mathbb{E}_{q(\mathbf{z}_{si})}[KL(q(\boldsymbol{\phi}_{si}|\mathbf{z}_{si})||p(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}))] = \\ &\quad \boldsymbol{\pi}_{si}KL(\mathcal{N}(\mathbf{w}_{si},\epsilon^2)||\mathcal{N}(\mathbf{w}_i,1/\lambda)) + (1-\boldsymbol{\pi}_{si})KL(\mathcal{N}(0,\epsilon^2)||\mathcal{N}(0,1/\lambda_2)).\end{aligned} \tag{22}$$

The KL term for when $\mathbf{z}_{si}=1$ can be written as

$$KL(\mathcal{N}(\mathbf{w}_{si},\epsilon^2)||\mathcal{N}(\mathbf{w}_i,1/\lambda)) = -\frac{1}{2}\log\lambda - \log\epsilon + \frac{\lambda\epsilon^2}{2} - \frac{1}{2} + \frac{\lambda}{2}(\mathbf{w}_{si}-\mathbf{w}_i)^2. \tag{23}$$

The KL term for when $\mathbf{z}_{si}=0$ can be written as

$$KL(\mathcal{N}(0,\epsilon^2)||\mathcal{N}(0,1/\lambda_2)) = -\frac{1}{2}\log\lambda_2 - \log\epsilon + \frac{\lambda_2\epsilon^2}{2} - \frac{1}{2}. \tag{24}$$

Taking everything together we thus have

$$\begin{aligned}\mathbb{E}_{q(\mathbf{z}_{si})}[KL(q(\boldsymbol{\phi}_{si}|\mathbf{z}_{si})||p(\boldsymbol{\phi}_{si}|\mathbf{z}_{si}))] &= \frac{\lambda\boldsymbol{\pi}_{si}}{2}(\mathbf{w}_{si}-\mathbf{w}_i)^2 + \boldsymbol{\pi}_{si}(-\frac{1}{2}\log\lambda - \log\epsilon + \frac{\lambda\epsilon^2}{2} - \frac{1}{2}) + \\ &\quad (1-\boldsymbol{\pi}_{si})(-\frac{1}{2}\log\lambda_2 - \log\epsilon + \frac{\lambda_2\epsilon^2}{2} - \frac{1}{2}) \qquad (25)\\ &= \frac{\lambda\boldsymbol{\pi}_{si}}{2}(\mathbf{w}_{si}-\mathbf{w}_i)^2 + \boldsymbol{\pi}_{si}\left(\frac{1}{2}\log\frac{\lambda_2}{\lambda} + \frac{\epsilon^2}{2}(\lambda-\lambda_2)\right) + C \qquad (26)\\ &\approx \frac{\lambda\boldsymbol{\pi}_{si}}{2}(\mathbf{w}_{si}-\mathbf{w}_i)^2 + \lambda_0\boldsymbol{\pi}_{si} + C, \qquad (27)\end{aligned}$$

where $\lambda_0 = \frac{1}{2}\log\frac{\lambda_2}{\lambda}$ and $\frac{\epsilon^2}{2}(\lambda-\lambda_2)$ was omitted due to $\epsilon^2 \approx 0$. In the appendix of Louizos et al. (2017), the authors argue about a hypothetical prior that results into needing λ nats to transform that prior to the approximate posterior. Here we make this claim more precise and show that this prior is approximately equivalent to a mixture of Gaussians prior where the precision of the non-zero prior component $\lambda \to \epsilon$ (in order to avoid the L_2 regularization term) and the precision of the zeroth component λ_2 is equivalent to $\lambda\exp(2\lambda_0)$, where λ_0 is the desired L_0 regularization strength.

Furthermore, the cross-entropy from $q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)$ to $p(\mathbf{z}_s|\boldsymbol{\theta})$ is straightforward to compute as

$$\mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)}[\log p(\mathbf{z}_s|\boldsymbol{\theta})] = \sum_j \left(\boldsymbol{\pi}_{sj} \log \boldsymbol{\theta}_j + (1 - \boldsymbol{\pi}_{sj}) \log(1 - \boldsymbol{\theta}_j)\right). \tag{28}$$

By putting everything together we have that the local objective becomes

$$\begin{aligned}\arg\max_{\mathbf{w}_s, \boldsymbol{\pi}_s} \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)} \left[\sum_i^{N_s} L(\mathcal{D}_{si}, \mathbf{w}_s \odot \mathbf{z}_s)\right] - \frac{\lambda}{2} \sum_j \boldsymbol{\pi}_{sj} (\mathbf{w}_{sj} - \mathbf{w}_j)^2 - \lambda_0 \sum_j \boldsymbol{\pi}_{sj} \\ + \sum_j \left(\boldsymbol{\pi}_{sj} \log \boldsymbol{\theta}_j + (1 - \boldsymbol{\pi}_{sj}) \log(1 - \boldsymbol{\theta}_j)\right) + C. \end{aligned} \tag{29}$$

F LOCAL OPTIMIZATION OF THE BINARY GATES

We propose to rewrite the local loss in Eq. 8 to

$$\begin{aligned}\mathcal{L}_s(\mathcal{D}_s, \mathbf{w}, \boldsymbol{\theta}, \boldsymbol{\phi}_s, \boldsymbol{\pi}_s) := \mathbb{E}_{q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)} \Bigg[\sum_i^{N_s} L(\mathcal{D}_{si}, \mathbf{w}_s \odot \mathbf{z}_s) - \lambda \sum_j \mathbb{I}[\mathbf{z}_{sj} \neq 0] (\mathbf{w}_{sj} - \mathbf{w})^2 \\ - \lambda_0 \sum_j \mathbb{I}[\mathbf{z}_{sj} \neq 0] + \sum_j \left(\mathbb{I}[\mathbf{z}_{sj} \neq 0] \log \frac{\boldsymbol{\theta}_j}{1 - \boldsymbol{\theta}_j} + \log(1 - \boldsymbol{\theta}_j) \right) \Bigg], \end{aligned} \tag{30}$$

and then replace the Bernoulli distribution $q_{\boldsymbol{\pi}_s}(\mathbf{z}_s)$ with a continuous relaxation, the hard-concrete distribution (Louizos et al., 2017). Let the continuous relaxation be $r_{\mathbf{u}_s}(\mathbf{z}_s)$, where $\mathbf{u}_s$ are the parameters of the surrogate distribution. In this case the local objective becomes

$$\begin{aligned}\mathcal{L}_s(\mathcal{D}_s, \mathbf{w}, \boldsymbol{\theta}, \boldsymbol{\phi}_s, \mathbf{u}_s) := \mathbb{E}_{r_{\mathbf{u}_s}(\mathbf{z}_s)} \left[\sum_i^{N_s} L(\mathcal{D}_{si}, \mathbf{w}_s \odot \mathbf{z}_s) \right] - \lambda \sum_j R_{\mathbf{u}_{sj}}(\mathbf{z}_{sj} > 0) (\mathbf{w}_{sj} - \mathbf{w})^2 \\ - \lambda_0 \sum_j R_{\mathbf{u}_{sj}}(\mathbf{z}_{sj} > 0) + \sum_j \left(R_{\mathbf{u}_{sj}}(\mathbf{z}_{sj} > 0) \log \frac{\boldsymbol{\theta}_j}{1 - \boldsymbol{\theta}_j} + \log(1 - \boldsymbol{\theta}_j) \right), \end{aligned} \tag{31}$$

where $R_{\mathbf{u}_s}(\cdot)$ is the cumulative distribution function (CDF) of the continuous relaxation $r_{\mathbf{u}_s}(\cdot)$. We can now straightforwardly optimize the surrogate objective with gradient ascent.

G ALTERNATIVE PRIORS FOR THE HIERARCHICAL MODEL

In the main text we argued that the hierarchical model interpretation is highly flexible and allows for straightforward extensions. In this section we will demonstrate two variants that use either a Laplace or a mixture of Gaussians prior, coupled with hard-EM for learning the parameters of the hierarchical model.

G.1 FEDERATED LEARNING WITH LAPLACE PRIORS

Lets start with the Laplace variant; the Laplace density for a specific local parameter $\boldsymbol{\phi}_{si}$ will be:

$$p(\boldsymbol{\phi}_{si}|\mathbf{w}_i) = \frac{\lambda}{2} \exp(-\lambda|\boldsymbol{\phi}_{si} - \mathbf{w}_i|). \tag{32}$$

Therefore the local objective of each shard and the global objective will be

$$\mathcal{L}_s(\mathcal{D}_s, \mathbf{w}, \mathbf{w}_s) := \sum_i^{N_s} L(\mathcal{D}_{si}, \mathbf{w}_s) - \lambda \sum_j |\mathbf{w}_{sj} - \mathbf{w}_j| + C, \tag{33}$$

$$\mathcal{L}(\mathbf{w}) := \sum_s \mathcal{L}_s(\mathcal{D}_s, \mathbf{w}, \mathbf{w}_s) \tag{34}$$

where again the global objective is a sum of all the local objectives and C is a constant independent of the optimization. We can then proceed, in a similar fashion to traditional "cross-device" FL,

by selecting a subset of shards B to approximate the global objective. On these specific shards, we will then optimize $\mathcal{L}_s$ with respect to $\mathbf{w}_s$ while keeping $\mathbf{w}$ fixed. Interestingly, due to the L_1 regularization term that appears in the local objective, we will have, depending on the regularization strength λ, several local parameters $\mathbf{w}_s$ that will be exactly equal to the server parameters $\mathbf{w}$ even after optimization. Now given the optimized parameters from these shards, $\mathbf{w}_s^*$, we will update the server parameters $\mathbf{w}$ for the M-step by either a gradient update or a closed form solution. Taking the derivative of the global objective with respect to a $\mathbf{w}_j$ we see that it has the following simple form

$$\frac{\partial L}{\partial \mathbf{w_j}} = \lambda \sum_{s \in B} \text{sign}(\mathbf{w}_{sj} - \mathbf{w}_j). \tag{35}$$

By setting it to zero, we see that the closed form solution is again easy to obtain

$$\frac{\partial L}{\partial \mathbf{w}_j} = 0 \Rightarrow \lambda \sum_{s \in B} \text{sign}(\mathbf{w}_{sj} - \mathbf{w}_j) = 0 \Rightarrow \mathbf{w}_j = \text{median}(\mathbf{w}_{1j}, \ldots, \mathbf{w}_{Bj}) \tag{36}$$

since the median produces an equal number of positive and negative signs. Using the median in the server for updating its parameters is interesting, as it is more robust to "outlier" updates from the clients.

G.2 Federated learning with mixture of Gaussian priors

The mixture of Gaussians prior will allow us to learn an ensemble of models at the server. The density for the entire vector of local parameters $\mathbf{w}_s$ in the case of K equiprobable components in the mixture will will be

$$p(\boldsymbol{\phi}_s | \mathbf{w}_{1:K}) = \frac{1}{K} \sum_k \mathcal{N}(\mathbf{w}_k, (1/\lambda)\mathbf{I}). \tag{37}$$

This will lead to the following local and global objectives

$$\mathcal{L}_s(\mathcal{D}_s, \mathbf{w}_{1:K}, \mathbf{w}_s) := \sum_i^{N_s} L(\mathcal{D}_{si}, \mathbf{w}_s) + \log \sum_k \frac{Z_k}{K} \exp\left(-\frac{\lambda}{2}\|\mathbf{w}_s - \mathbf{w}_k\|^2\right), \tag{38}$$

$$\mathcal{L}(\mathbf{w}_{1:K}) := \sum_s \mathcal{L}_s(\mathcal{D}_s, \mathbf{w}_{1:K}, \mathbf{w}_s) \tag{39}$$

where Z_k is the normalizing constant of component k. Now we can proceed in a similar fashion and select a subset of shards B to obtain the $\boldsymbol{\phi}_s^*$ while keeping the parameters of the prior $\mathbf{w}_{1:K}$ fixed. Now by taking the gradient with respect to one of the members of the ensemble $\mathbf{w}_k$, given the optimized local parameters $\boldsymbol{\phi}_s^*$, we see that

$$\frac{\partial L}{\partial \mathbf{w}_k} = \lambda \sum_s \frac{\frac{Z_k}{K} \exp(-\frac{\lambda}{2}\|\mathbf{w}_s - \mathbf{w}_k\|^2)}{\sum_j \frac{Z_j}{K} \exp(-\frac{\lambda}{2}\|\mathbf{w}_s - \mathbf{w}_j\|^2)} (\mathbf{w}_s - \mathbf{w}_k). \tag{40}$$

Notice that this gradient estimate can also be interpreted in terms of a posterior distribution over the index z (taking values in $\{1, \ldots, K\}$) given the "observed" variables $\mathbf{w}_s$; we can treat $1/K$ as the prior probability of selecting component $z = k$, i.e., $p(z = k)$ and $Z_k \exp(-\frac{\lambda}{2}\|\mathbf{w}_s - \mathbf{w}_k\|)$ as the probability of $\mathbf{w}_s$ under Gaussian component k. In this way, the weighting term can be written as

$$\frac{\frac{Z_k}{K} \exp(-\frac{\lambda}{2}\|\mathbf{w}_s - \mathbf{w}_k\|^2)}{\sum_j \frac{Z_j}{K} \exp(-\frac{\lambda}{2}\|\mathbf{w}_s - \mathbf{w}_j\|^2)} = p(z = k | \mathbf{w}_s, \mathbf{w}_{1:K}, \lambda). \tag{41}$$

This make apparent the connection to Gaussian mixture models, since $p(z = k | \mathbf{w}_s, \mathbf{w}_{1:K}, \lambda)$ is equivalent to the responsibility of component k generating $\mathbf{w}_s$. Now we can again find a closed form solution for $\mathbf{w}_k$ by setting the derivative to zero

$$\frac{\partial L}{\partial \mathbf{w}_k} = 0 \Rightarrow \lambda \sum_s p(z = k | \mathbf{w}_s, \mathbf{w}_{1:K}, \lambda)(\mathbf{w}_s - \mathbf{w}_k) \Rightarrow \mathbf{w}_k = \sum_s \frac{p(z = k | \mathbf{w}_s, \mathbf{w}_{1:K}, \lambda)}{\sum_j p(z = k | \mathbf{w}_j, \mathbf{w}_{1:K}, \lambda)} \mathbf{w}_s, \tag{42}$$

which is again similar to the closed form update for the centroids in a Gaussian mixture model when trained with EM.