Supplemental information

A. A Generalization Bound

Proof. The main idea of the proof is to first bound the error of all (labeled) training clients, using results from multitask learning. Then, treat the novel client as a target domain in a domain adaptation problem. This allow us to use results from domain adaptation to bound the novel client error.

The error of a hypothesis h over a distribution P is defined by $err_P(h) = \int_{X\times Y} l(h(x), y)dP(x, y)$. The error of the novel client for a given hypothesis **space** H is defined by

$$err_{P_{new}}(H) := \inf_{h\in H} err_{P_{new}}(h) \tag{2}$$

Since P_{new} is independent of Q, we can integrate over all $P \sim Q$ and obtain

$$err_{P_{new}}(H) = \int_P \inf_{h\in H} err_{P_{new}}(h)dQ(P). \tag{3}$$

Using Theorem 2 from [3] with P_{new} treated as the target domain and P as the source domain, gives that $\forall h, \forall P$: $err_{P_{new}}(h) \le err_P(h) + \frac{1}{2}\hat{d}_{H\Delta H}(P, P_{new})$. Plugging into Eq. (3) gives

$$err_{P_{new}}(H) \le \tag{4}$$
$$\int_P \inf_{h\in H}\left[err_P(h) + \tfrac{1}{2}\hat{d}_{H\Delta H}(P, P_{new})\right] dQ(P) \tag{5}$$
$$= err_Q(H) + \tfrac{1}{2}\int_P \inf_{h\in H} \hat{d}_{H\Delta H}(P, P_{new})dQ(P). \tag{6}$$

Since $err_Q(H)$ is unknown, we use Theorem 2 from [2] to bound the error of the novel client. That yields the bound in the theorem. □

Now we explain this in detail. If we have only one task and one domain, the most common solution is to find $h \in \mathcal{H}$ that minimizes the loss function on a training set sampled from probability distribution P. In general, $\mathcal{H}$ is a hyperparameter defined by the network architecture. [5] shows that in this simple case the generalization error is bounded. The bound depends on the "richness" of H. Choosing a "rich" H (with large VC-dimension), increase the generalization error.

In OD-PFL, each client may use a different hypothesis. Instead of bounding the error of one chosen hypothesis, we bound the error of the chosen hypotheses space that each client chooses from. In this way, we can bound the error of a novel client, without assuming anything about the way it chose from the hypothesis space.

First, we find H that is "rich" enough to contain hypotheses that can fit all data of the clients. Second, for each client, we select the best hypothesis $h \in H$ according to the client data. We define Q as a distribution over P, so, each client sample from Q a distribution P_i. We further define $\mathbb{H}$ as a hypothesis space family, where each $H \in \mathbb{H}$ is a set of functions $h : X \to Y$.

The first goal is to find a hypothesis space $H \in \mathbb{H}$ that minimizes the weighted error of all clients, assuming each client uses the best hypothesis $h \in H$. We define this error using the following loss:

$$err_Q(H) := \int_P \inf_{h\in H} err_P(h)dQ(P) \tag{7}$$

while $err_P(h) := \int_{X\times Y} l(h(x), y)dP(x, y)$. In practice, Q is unknown, so we can only estimate $err_Q(H)$ using the sampled clients and their data.

For each client $i = 1..n$, we sample the training data of the client from $X \times Y \sim P_i$. We denote the sampled training set by $z_i := (x_1, y_1), ..., (x_m, y_m)$, and $z = z_1, ...z_n$. The empirical error of a specific hypothesis is defined by $\hat{er}_z(h) := \frac{1}{m}\sum_{i=1}^m l(h(x_i), y_i)$. In training we minimize the empirical loss

$$\hat{er}_z(H) := \frac{1}{n}\sum_{i=1}^n \inf_{h\in H} \hat{er}_{z_i}(h) \tag{8}$$

[2] shows that if the number of clients n satisfies $n \ge max\{\frac{256}{\epsilon^2}log(\frac{8C(\frac{32}{\epsilon},H^*)}{\delta}), \frac{64}{\epsilon^2}\}$, and the number of samples per client m satisfies $m \ge max\{\frac{256}{n\epsilon^2}log(\frac{8C(\frac{32}{\epsilon},H_l^n)}{\delta}), \frac{64}{\epsilon^2}\}$, then with probability $1 - \delta$ all $H \in \mathcal{H}$ satisfies

$$err_Q(H) \le \hat{er}_z(H) + \epsilon \tag{9}$$

were $C(\frac{32}{\epsilon}, H_l^n)$ and $C(\frac{32}{\epsilon}, H_l^n)$ are the covering numbers defined in [2], and can be referred to as a way to measure the complexity of H. Note that a very "rich" H makes $\hat{er}_z(H)$ small, but increases the covering number, so for the same amount of data, ϵ increases.

For the PFL setup, this is enough, since we can ensure that for a client that sampled from Q and was a part in the federation, the chosen hypothesis $h \in H$has an error close to the empirical one $\hat{er}_z(H)$. For a novel client, this may not be the case. The novel client may sample from a different distribution over P. In the general case, the novel client may even have a different distribution over $X \times Y$. In the most general case, the error on the novel client cannot be bound. In DA, a common distribution shift is a covariate shift, where $P(x)$ may change but $P(y|x)$ remains constant. This assumption lets us bound the error of the novel client.

[3] shows that for a given $H \in \mathcal{H}$, if S and T are two datasets with m samples, then with probability $1 - \delta$, for every hypothesis $h \in H$:

$$err_T(h) \le \quad err_S(h) + \tfrac{1}{2}\hat{d}_{H\Delta H}(S, T) \tag{10}$$
$$+4\sqrt{\frac{2d\ log(2m) + log(\frac{2}{\delta})}{m}} + \lambda \tag{11}$$

where $err_D(h) = E_{(x,y)\ D}[|h(x) - y|]$ is the error of the hypothesis on the probability distribution of the domain D. $\hat{d}_{H\Delta H}(S,T)$ is a distance measure between the domains S and T, and $\lambda = \arg\max_{h\in H} err_T(h) + err_S(h)$. Note that for over-parametrized models like deep neural networks λ should be very small. To keep the analysis shorter we assume this is the case. We also assume that m is large enough to neglect $4\sqrt{\frac{2d\ log(2m)+log(\frac{2}{\delta})}{m}}$. These assumptions are not mandatory, and the following analysis can be performed without them. This allows us to treat P_{new} as the target domain and P as the source domain in Eq. (4)

B. Additional Details about Training

B.1. Encoding batches of a dataset

We encode the whole dataset by feeding a large batch to the encoder. We also tested an alternative approach that can be applied to large datasets that do not fit in a single batch in memory. In these cases, we randomly split the data into smaller batches, encoded each batch, and used the average over batch descriptors as the final descriptor. When we tested this approach for several batch sizes, we did not observe that the performance was consistently sensitive to batch size. Encoding the full dataset in a single batch did perform better in most scenarios tested.

B.2. Training in two phases

In ODPFL-HN two main components are trained using a federation of labeled clients: A hypernetwork and a client encoder. We train them together end to end. When training the client encoder, batches from the same client yield different representations, and we found that this variability might hurt training end to end. We design 2 approaches to alleviate this issue: (1) Calculating the embedding using all client data, without breaking it into mini-batches. (2) Training in two steps by first learning an embedding of each training client, namely, a mapping from a client identity i to a dense descriptor e_i. This embedding layer was trained in a standard way jointly with the hypernetwork. Then, we trained the client encoder. We tested both methods, and choose between them using cross validation. Practically, for the iNaturalist experiments, the second method was better. For all other datasets, the first method was better. We now explain the second approach in detail.

Training the client encoder and the HN in two phases was done in the following way: The hypernetwork optimizes the L_{HN} loss defined in 12 by updating both its own weights θ and client representations $\{e_k\}_{k=1}^N$.

$$L_{HN}(\theta, e_1, ..., e_N) = \sum_{i=1}^{n}\sum_{j=1}^{m_i} l(f_\theta(e_i)(x_j^i), y_j^i) \tag{12}$$

The client encoder trains to predict the representations learned by the hypernetwork from raw client data by minimizing $L_{encoder}$ defined in 13. At inference time, a novel client feeds its data to the client encoder and gets an embedding vector. Then, feeding the embedding vector to the hypernetwork produces a custom model for the client.

$$L_{encoder} = \sum_{i=1}^{n} L_2(g_\gamma(\{x_j^i\}_{j=1}^{m_i}), e_i) \tag{13}$$

In detail, in each communication step: (1) The server selects a random client and feeds its embedding e_i to the HN to create the personal model $h_i = h(\cdot, w_i)$. (2) The servers sends h_i, e_i, and the current encoder g_γ to the client. (3) The client then locally trains this network on its data and communicates the delta between the weights before and after training Δw_i back to the server. Using the chain rule, the server can train the hypernetwork and the embedding layer itself. (4) The client trains the encoder locally using the current given embedding e_i by optimizing $L_{encoder}$, then, the updates of the encoder are sent back to the server for aggregation. Inference is done in the same way as in end-to-end training.

Up to this point, the client encoder trains in parallel to the hypernetwork and has no influence on the hypernetwork weights or the embeddings of the labeled clients. We found that freezing the encoder and fine-tuning the hypernetwork using the trained encoder predictions improve the results of our method. This is done by optimizing the hypernetwork parameters θ using $L_{Fine-tune}$.

$$L_{Fine-tune}(\theta) = \sum_{i=1}^{n}\sum_{j=1}^{m_i} l(f_\theta(g_\gamma(\{x_j^i\}_{j=1}^{m_i}))(x_j^i), y_j^i) \tag{14}$$

However, this fine-tuning step reduces the performance of labeled clients. Note that in a real-world application, the server may save a version of the hypernetwork before fine-tuning it and use it when generating models for the original federation.

C. Experimental Details

For all experiments presented in the main text, we use a fully connected hypernetwork with 3 hidden layers of 100 hidden units each. The size of the embedding dimension is $\frac{N_{clients}}{4}$. Experiments are limited to 500 communication steps. In each step, communication is done with $0.1 \cdot N_{train}$.

Hyperparmeter Tuning We divide the training samples of each training client into 85% / 15% train / validation sets. The validation sets are used for hyperparameter tuning and early stopping of all baselines and datasets. The hyperparameters searched and the corresponding values by method: **FedAVG**: The local momentum μ_{local} is set to 0.5. We search over local learning-rate $\eta_{local} \in \{1e-1, 5e-2, 1e-$

$2, 5e-3, 1e-3\}$, number of local epochs $K \in \{1, , 2, 5, 10\}$ and batch size $\{16, 32, 64\}$. **FedProx** and **FedMA**: We used the hyperparameters used by [33] in the official code that provided by the authors. **pFEdHN**: We set $\mu_{local} = 0.9$. We search over learning-rates of the hypernetwork, embedding layer and local training: $\eta_{hn}, \eta_{embedding}, \eta_{local} \in \{1e-1, 5e-2, 1e-2, 5e-3, 1e-3\}$, weight decays $wd_{hn}, wd_{embedding}, wd_{local} \in \{1e-3, 1e-4, 1e-5\}$, number of local epochs $K \in \{1, , 2, 5, 10\}$ and batch size $\{32, 64\}$. **ODPFL-HN**: We perform the optimization using the same parameters and values as in pFEdHN. In addition, we search for the learning rate of the client encoder $\eta_{encoder} \in \{1e-1, 5e-2, 1e-2, 5e-3, 1e-3\}$.

CIFAR(Section 6.3) We use a LeNet-based target network with two convolution layers with 16 and 32 filters of size 5 respectively. Following these layers are two fully connected layers of sizes 120 and 84 that output logits vector. The client encoder follows the same architecture with an additional fully connected layer of size 200 followed by Mean-global-pooling for the first 100 units and Max-global-pooling for the other 100 units. Global pooling is done over the samples of a batch.

iNaturalist, Landmarks and Yahoo Answers Data(Sections 6.4-6.6) We use a simple fully-connected network with two Dense layers of size 500 each, followed by a Dropout layer with a dropout probability of 0.2. The client encoder is a fully-connected network with three Dense layers of size 500. The first layer is followed by Mean Global Pooling for the first 250 units and Max Global Pooling for the other 250 units.

D. Differential Privacy

Proof.

$$\begin{aligned}\Delta g \quad &:= \max_{D,D'} ||g(D) - g(D')|| \\ &= \max_{D,D'} ||\psi(\tfrac{1}{|D|}\textstyle\sum_{x\in D}\phi(x)) \\ &\quad -\psi(\tfrac{1}{|D'|}\textstyle\sum_{x\in D'}\phi(x))||.\end{aligned} \tag{15}$$

Denote $d \in D$ and $d' \in D'$ as the only nonidentical instance between D and D', so $D/d = D'/d'$. Then

$$\begin{aligned}\Delta g = \quad &\max_{D,D'} ||\psi\left(\tfrac{1}{|D|}[\textstyle\sum_{x\in D/d}\phi(x) + \phi(d)]\right) \\ &-\psi\left(\tfrac{1}{|D'|}[\textstyle\sum_{x\in D'/d'}\phi(x) + \phi(d')]\right)|| \\ = \quad &max_{d,d'} \tfrac{1}{|D|}||\psi\left(\phi(d) - \phi(d')\right)||\end{aligned} \tag{16}$$

Assume that ϕ is bounded by B_ϕ, so $|\phi(d) - \phi(d')| < 2B_\phi$. Then from the linearity of ψ:

$$\Delta g \le \frac{1}{|D|}L_\psi|\phi(d) - \phi(d')| \le \frac{2}{|D|}L_\psi B_\phi \tag{17}$$

$\square$

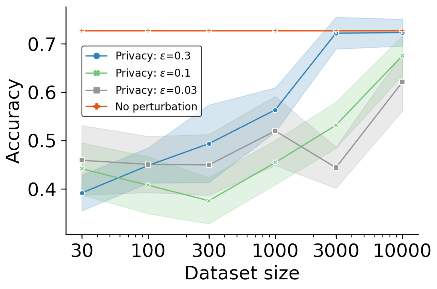

Figure 5. Test accuracy (±SEM) for CIFAR-10 novel client while applying DP. As ϵ decreases, we need more data to preserve the same accuracy of the model.

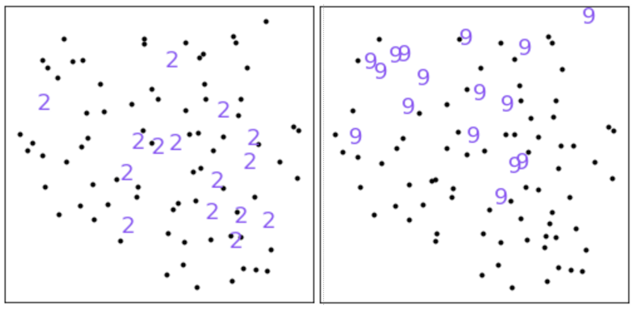

Figure 6. A tSNE plot of the embedding space for the pathological split of CIFAR10. In this setting, each client has a task of categorizing samples from two classes. Each dot and digit correspond to one client. The digits 2 marks clients that had samples from class 2; specifically (2,1), (2,3) (2,4) etc. Same for the digit 9. Similar plots can be made for the remaining classes. Clients involving the same class tend to be closer to each other (2 on the right, 9 at the top).

We also compare different levels of privacy, using lower values of ϵ. Figure 5 shows that more privacy (lower ϵ), requires a larger dataset to maintain the same accuracy of the model.

E. Embedding Space Visualization

To get intuition for the way the client encoder captures similarities between clients, we wish to visualize the space of client embeddings $\mathcal{E}$. Since each client involves samples from different classes, it is somewhat hard to visualize which client should be close. To capture some of those similarities, we mark clients that contain samples from some class with that class number. Figure 6 shows all clients that have samples from class #2 (left) or #9 (right). Clients that share a class tend to be located closer. Note that those clients have another class that is not shown, so they are not expected to fully cluster together.