MLO DISCUSSION PAPER
February 19, 2021

Measuring the Impact of Model and Input Heterogeneity in Personalized Federated Learning

Semester Project by Freya Behrens

Supervised by Anastasia Koloskova and Tao Lin
Machine Learning and Optimization Lab EPFL

Abstract

Federated Learning is explicitly designed for learning a global model from distributed, possibly sensitive non-i.i.d. data at different clients. In reality, this scenario is not always legible since in some cases the heterogeneous needs of clients cannot be packaged into a single global model, e.g. in next-word prediction on mobile phones where different clients might express themselves differently. Thus, in personalized federated learning the goal is to learn a personal model for each client individually by training a global model to be easy to adapt locally. Such a formulation is also known in meta-learning, where the goal is to learn a global model that can be easily adapted to different tasks rather than clients. In this project, we examine the differences and similarities between algorithms from the two fields. We furthermore develop a benchmarking dataset that captures different aspects of heterogeneity between the clients. On this we evaluate both when personalization in federated learning is actually beneficial over individual training and compare which algorithms from meta-learning and personalized federated learning fare best.

1 Introduction

In the past, the focus of machine learning algorithms has been on algorithms for data sources that are available centrally. Nowadays, an increasing amount of data is generated on edge devices such as mobile phones, raising privacy concerns about saving them in central collections. In combination with an increased computation power available at the edge, this generated more interest in distributed training algorithms as opposed to centralized versions.

Federated Learning [KMA+19] is such a distributed training framework where a central server coordinates local training on several clients. In the process, the server repeatedly sends the global model to the clients, they perform local optimization and return the result of the optimization to the server, where all clients results are combined into an updated global model. In any case, clients never directly share their local dataset with the server or others but communicate only via model updates. Formally, the objective is to converge to a consensus solution of model parameters via the local empirical loss functions f_i at clients $i = 1, ...n$. This is expressed as the objective

$$w_{\text{fl}}^* = \arg\min_{w \in W} \frac{1}{n} \sum_{i=1}^{n} f_i(w).$$

However, in the real-world, one naturally encounters local losses that are heterogeneous and not identical at each client. This is caused by e.g. differences in the training data distributions of local datasets, changing availability of clients and the local power of computation. Furthermore, the training process might be exposed to adversarial clients that modify their own functions to deliberately attack the model. Consequently, the goal of reaching a global consensus model using only optimization on the individual local empirical losses is somehow inconsistent.

In this project, we specifically consider two types of heterogeneity between clients under a classification task (prediction of the label y for an input $\mathbf{x}$). This heterogeneity can be expressed as difference in the local training distributions represented by some probability distribution $p_i(x, y)$ for client i. In a factorized version, $p_i(x, y) = p_i(x)p_i(y|x)$, we consider $p_i(x)$ the input distribution which may vary between clients as *input heterogeneity*. Some clients may have examples from the input space that others do not have. Looking at $p_i(y|x)$ we describe this as the (ground truth) model of the client. This describes how, according to client i, an input is assigned a class label. As such, a high *model heterogeneity* describes clients that label the same input inconsistently.

On the positive side, these types of heterogeneity may also be desired, as the global model might not corresponds to the actual needs of a client [YBS20]. It is reasonable to assume that inconsistent data between clients is not undesirable model heterogeneity that increases the difficulty of learning, but rather that this represents the data to which the client actually wants to fit a model to. In this case we expect that they can profit from the information of others via training without sharing these private datasets, but still being able to learn a personal version of the global model which differs from the other clients. As an example, in next-word prediction for keyboards on mobile devices [HRM$^+$18] different clients may be writing about different topics or using a different slang, e.g. typing in a document using formal terms as opposed to colloquial terms in private chat applications. Learning a single global model might not be able to simultaneously cover all predictions accurately. Instead, each client needs a personal model. As the local data (and computation power) is likely insufficient to train a full-fletched language model, they should still profit from a distributed process where they can acquire information about the general structure of the language from other clients data without accessing it.

Such a scenario corresponds to *personalized federated learning* (PFL) [KKP20], which wants to find individual personalized models for each client that fit their data well locally while still profiting from useful information of other clients data via the federated training. Several methods to achieve this goal have been proposed that involve different techniques like learning some layers of model locally or directly learning a mixture of local and global models [DKM20, HR20, MMRS20, MMRS20]. In this project we consider a group of PFL methods where the goal is to derive a global model in the training phase via several communication rounds that is a good initial point for local adaption. Formally we consider

$$w^*_{\text{pfl}} = \arg\min_{w \in W} \frac{1}{n} \sum_{i=1}^{n} f_i(\mathcal{A}_i(w))$$

where $\mathcal{A}_i$ is a function which adapts the global parameters w to better fit the local objective locally at client i based on the clients data.

Interestingly, this problem has a history in meta-learning, where the famous optimization based MAML [FAL17] explicitly optimizes the ability of a global model to adapt to different tasks. This is accomplished by using a similar local update structure as in federated learning, whereby the tasks correspond to clients. In fact, different optimization algorithms used and analysed in the two disciplines show a high overlap. One of the first algorithms used in federated learning was FedAvg [KMA$^+$19] which simply uses a global averaging step after several local SGD updates. Using ideas from MAML, personalized versions like pFedAvg [FMO20] have been developed that increased the complexity over FedAvg by local second order updates in favour of better personalized performance. Contrarily, in meta-learning, the high computational complexity of MAML inspired researchers to develop algorithms that approximate its properties but require less computation. This includes FOMAML [FAL17], which has a similar structure structure as pFedAvg. Later, iMAML [RFKL19] and Reptile [NAS18] were introduced, the last of which is algorithmically the same as FedAvg. Even more, FedProx [LSZ$^+$20],

an algorithm for federated learning that has been shown to deal well with stragglers and varying local computation power has been independently suggested for solving meta-learning problems [GRF+20a].

Even though these algorithms appear in both fields for the same formal objective, the goals and circumstances under which this objective should be minimized vary. Federate learning is very much constrained by the number of global rounds since updating locally is a lot cheaper than communicating the parameters of the model, whereas this is not an issue in meta-learning. In addition, while the heterogeneity in federated learning arises from the circumstances and cannot be directly changed, e.g. the clients availability at each round, this can be controlled in meta-learning, e.g. by determining exactly how tasks are sampled. Moreover, optimization based methods in meta-learning are often specifically employed for few-shot learning, where the local adaptions has only a few steps. In federated settings local clients may have access to non-uniform sizes of datasets, possibly creating a 'varying'-shot learning setting in which different amounts of local adaption should lead to good models. Overall, this similarity calls for leveraging the broad research in meta-learning to tackle model and input heterogeneity in personalized federated learning (PFL). In this project, we

- draw further connections between meta-learning methods and algorithms from (personalized) federated learning
- define several types of heterogeneity that can occur at the client level
- devise a synthetic dataset that allows to vary the model and input heterogeneity independently and experimentally compare the introduced algorithms on this dataset.

In the following two Sections we formally define the PFL framework and relevant algorithms. In Section 4 we review the literature. In Section 5 we specify the different types of heterogeneity encountered on which we base the definition of a synthetic dataset in Section 6. Finally, we present results from experiments on this in Section 7 and summarize our findings in Section 8.

2 Notation and General Framework

General Framework of personalized federated learning Overall, the personalized federated training algorithms for n clients we consider here are structured as in Algorithm 1. We assume that all clients and the server use the same parameterized model architecture $m : \mathcal{X} \times \mathcal{W} \to \mathcal{Y}$ that for some input $x \in \mathcal{X}$ predicts the label y based on the parameters $w \in \mathcal{W}$ via $m(x; w) = y$. The local empirical loss functions $f_i : \mathcal{W} \to \mathbb{R}$ then differ only in the clients local target data with $f_i(w) = \frac{1}{n} \sum_{(x,y) \in \mathcal{D}_l ocal} l(w, x, y)$ with the local target dataset $\mathbf{D}_i^{\text{target}} \subset \mathcal{D} = \mathcal{X} \times \mathcal{Y}$. For training $\mathbf{D}_i^{\text{source}} \subset \mathcal{D}$ is available. The server has no access to no data whatsoever.

The federated learning training itself is initialized locally at the server and clients. The server initializes the server model $w \in \mathcal{W}$ and possibly its state c which we leave undefined here, as different algorithms use different variables in the state. The clients also initialize their states c_i independently.

Then, the training phase with R global communication rounds starts. In each round r a subset of $\mathcal{S}_r$ clients is selected. After receiving the global model w_r and state c^r (of which c^r is empty for some algorithms), the clients initiate a training process, calling `train` on this and their own state c_i^r. This returns an updates local state c_i^{r+1} and local model θ_i^{r+1}. Note that only states that *participate in round r* update their local states. This implies that clients can freely drop in and out of training, rather than being required to participate continuously. Finally, the updated models θ_i are communicated back to the server that combines all results into an update w^{r+1}.

Importantly, the training function does not need to correspond to the personalization function. This is why after finishing all training rounds R another rounds of communication with all clients is required. They receive the global model w^R and personalize it with their local data.

Also, the amount of training data that can be used in one call of the training function `train` is defined in terms of number of batches T, where we sample batches of size B from the local client i's training data $\mathbf{D}_i^{\text{source}}$ without replacement in `sample_data_batches`. The list $[\mathbf{D}_{i,t}^{\text{source}}]_{t=0}^{T}$ possibly includes several epochs in an ordered form, when $T \cdot B > |\mathbf{D}_i^{\text{source}}|$.

Algorithm 1: Personalized FL Structure

Result: w^R, the final global model parameters
initialize server (global) model w^0 and state c^0;
for each client i initialize local state c_i^0;
// **training phase**
for $r = 0$ *to* R **do**
  for $i \in \mathcal{S}_r$ **do**
    receive global model and state w^r, c^r;
    $[\mathbf{D}_{i,t}^{\text{source}}]_{t=0}^{T}$ = `sample_data_batches`(i, T, B);
    $w_i^{r+1}, c_i^r + 1$ = `train`$(T, w_r, [\mathbf{D}_{i,t}^{\text{source}}], c_i^r, c^r)$;
    send ^{r} to server ;
  end
  receive local results $\{\theta_i^r\}_{i \in \mathcal{S}_r}$;
  w^{r+1} = `update_global`$(w^r, \{\theta_i^r\}_{i \in \mathcal{S}_r})$;
end
// **personalization phase**
for $i \in [1...N]$ **do**
  $[\mathbf{D}_i^{source}]_{t=0}^{T}$ = *sample_data_batches*(i, T, B);
  θ_i = *personalize*$(T, w_r, [\mathbf{D}_i^{source}], c_i^r)$;
end

Objective Deriving a global model in the training phase via several communication rounds that is a good initial point for the local adaption is the goal of personalized federated learning. Formally, we want to find the minimizer of the average of the local empirical losses f_i applied to the personalized models θ_i as a result of the local personalization function which we abbreviate as $\mathcal{A}_i : \mathcal{W} \times \mathcal{D} \to \mathcal{W}$:

$$w_{\text{pfl}}^* = \arg\min_{w \in W} \frac{1}{n} \sum_{i=1}^{n} f_i(\theta_i; \mathbf{D}_i) \tag{2.1}$$

$$\text{with } \theta_i = \mathcal{A}_i(w; \mathbf{D}_i') \tag{2.2}$$

The empirical loss function naturally requires some data $\mathbf{D}_i \subseteq \mathbf{D}_i^{\text{source}}$ from the client i to be calculated. Also, as to an algorithm the only difference between clients is in their training data, the personalization function also requires a dataset to adapt to client i, in this case $\mathbf{D}_i' \subseteq \mathbf{D}_i^{\text{source}}$. Note that in the context of meta-learning algorithms, $\mathbf{D}_i$ and $\mathbf{D}_i'$ are usually not distributed equally. In the setting of few shot tasks, the adaption uses source "training" data and the calculation of the empirical loss uses source "testing" data. This allows a differentiation between examples that will be encountered to adapt the model compared to those that the adapted model should fit. As this difference is rarely made in PFL, we consider only $\mathbf{D}_i$ and $\mathbf{D}_i'$ that stem from the same distribution.

We now turn to different algorithms that are commonly used to solve problems of this type.

3 Algorithms

Given on the previously introduced framework, the following introduces several algorithms from meta-learning and personalized federated learning.

MAML Model-agnostic meta-learning directly optimizes the PFL objective (2.1) using gradient methods [FAL17]. This requires calculating the gradient of the composition of the empirical loss with the personalization step with respect to the model parameters w in the update step of the so-called

meta-loss:

$$w^{R+1} = \frac{1}{n}\sum_{i=1}^{n} \nabla_w (f_i \circ \mathcal{A}_i(w^R; \mathbf{D}_i, \mathbf{D}_i')).$$

In this case the local adaption via optimization $\mathcal{A}_i(w)$ itself is the result a couple of steps of an iterative optimization method that uses first-order information and approximates them via the data, e.g.

$$\mathcal{A}_i(w; \mathbf{D}_i) = w - \eta \nabla_w f_i(w; \mathbf{D}_i)$$

Thus, taking the derivative of $\mathcal{A}_i$ with respect to w can involve higher-order derivatives, e.g. here we require the Hessian $\nabla_w^2 f_i(w; \mathbf{D}_i')$ for $\nabla_w \mathcal{A}_i(w; \mathbf{D}_i')$.

As we require both data for the personalization routine and the meta-gradient step, the available data needs to be distributed to these two scenarios. The algorithm does not require stateful clients.

In practice deriving $\mathcal{A}_i(w)$ is generally possible as methods for back-propagating through the computational graph of gradient computations are available in many practical frameworks. For large deep-learning models the model size becomes problematic as the Hessian is difficult to store and training behaviour overall exhibits similar difficult behaviours as training very deep networks [AES19].

This is why many methods have been developed that approximate the behaviour of MAML itself with cheaper updates and a more controllable training behaviour, some of which are presented in the following paragraphs.

Algorithm 2: MAML: personalization, training

Result: update for global w, personalized model θ_i
$\theta_i^0 = w_r$;
for $t \in [1, ..., T-1]$ **do**
  $\theta_i = \theta_i - \eta \nabla_{\theta_i} f_i(\theta_i; \mathbf{D}_{i,t})$;
end
return $w^r - \eta \nabla_{w_r} f_i(\theta_i; \mathbf{D}_{i,T})$;

Algorithm 3: pFedAvg: training

Result: update for global model
$w^0 = w_r$;
for $t \in [1, ..., T]$ **do**
  if *t % 1 == 0* **then**
    $\theta_i = w^{t-1} - \eta \nabla_{w^{t-1}} f_i(w^{t-1}; \mathbf{D}_{i,t})$;
  else
    $w^t = w^{t-2} - \eta \nabla_{\theta_i} f_i(\theta_i; \mathbf{D}_{i,t})$;
  end
end
return w^T;

Algorithm 4: FedAvg: training

Result: update for global model
$theta_i = w_r$;
for $t \in [1, ..., T]$ **do**
  $\theta_i = \theta_i - \eta \nabla f_i(\theta_i; \mathbf{D}_{i,t})$;
end
return θ_i;

pFedAvg/FOMAML FOMAML (First-Order MAML) approximates the higher order derivatives present in MAML with first order terms. In this case, the meta-gradient step is substituted by a first-order approximation. Instead of deriving through $\mathcal{A}_i$, we simply use its *value* directly and do not back-propagate through its computation. As the communication rounds play a crucial role in federated learning, pFedAvg which uses the same idea as FOMAML allows several of local meta-update steps per communication round. In Algorithm 3 we depict it with one step of gradient descent. This entails that the true MAML objective is not optimized anymore [check paper of pfedavg].

FedAvg/Reptile Reptile completely gets rid of the meta-gradient and directly uses the last iterative of the local adaption/personalization procedure, resulting in the local steps as in Algorithm 4. This is equivalent to FedAvg, the classical algorithm for federated learning with several local update steps. This is also equivalent to the lookahead optimizer.

FedProx The algorithm FedProx is equivalent to fedAvg but uses l2 regularization during the local optimization phase to prevent the clients local models to shift away too far from the global model.

This amounts to adapting the optimization to the local steps

$$\theta_i = \theta_i - \eta\nabla(f_i(\theta_i; \mathbf{D}_{i,t}) + \frac{1}{\lambda}||\theta_i - w_r||_2^2$$

This introduces an additional regularization parameter λ which needs to be tuned.

Scaffold Scaffold has been developed to address the problem of heterogeneous inputs at different clients which can cause a shift of the global model towards the local optima of the clients rather than the global optimum. Using a technique based on variance reduction the In contrast to other models, this requires clients to keep a state $c_i \in \mathcal{W}$ of the control variate and the global model to keep a control variate $c \in \mathcal{W}$ which is send to the local clients along with the global model at each start of a local loop. Initially both values are set to zero. The local steps then amount to

$$\theta_i^{t+1} = \theta_i^t - \epsilon_l \cdot \nabla l_i(\theta_i^t) + (c - c_i).$$

Each round r in which the client participates the c_i is updated according to

$$c_i' = c_i + \frac{1}{t \cdot \epsilon_l} \cdot (\theta_i^T - w) - c.$$

The c_i' is thus an approximation of the gradient direction of all local steps. The difference between the $\Delta_i = c_i' - c_i$ is then sent to the server and averaged according to $c' = c + \frac{\epsilon_g}{n} \sum_{i \in S_r} \Delta_i$.

Since this training procedure explicitly prevents the client shift which is desired for the personalization, we remove the control variate term in the personalization procedure of the PFL framework, essentially fine-tuning the model.

Separate As a baseline for PFL we assume that each client only uses their data for local training without communication to anyone. This means they keep a state with their own local model θ_i during training time and update it for T steps whenever they would have otherwise participated in the global federated training.

4 Related Work

Federated Learning In standard federated learning [KMA+19] the classic algorithm has been FedAvg and has been applied successfully in real world applications [HRM+18]. For non-iid distributed data this algorithm can perform very poorly in highly skewed settings [HQB19a]. The miss-alignment of the local objectives with the global objectives leads to a client drift that prevents FedAvg from converging to the true optimum of the FL objective [KKM+19]. With methods for variance reduction Scaffold [KKM+19] is able to reliably converge to the global optimum despite the input heterogeneity using stateful clients. FedSplit [PW20] is able to do the same based on operator splitting without states. Next to the problem of heterogeneous inputs, a problem in federated learning can be that training a global model can actually be worse for some clients than using their own data as they might require different models than other clients [KMA+19]. This leads to consider the setting of Personalized Federated Learning (PFL) where the goal is to learn personalized models instead of a single global one.

Personalized Federated Learning and Meta-Learning In meta-learning, the goal is "learning how to learn". The connection between learning to quickly find good models for different tasks and learning to adapt a global model to local clients has been drawn in [JKRK19]. The authors noted how the meta-learning algorithm Reptile [NAS18] essentially corresponds to FedAvg [**?**]. One meta-gradient update over a set of tasks corresponds to updating the server with all client results in one communication round. This algorithm is also known as the lookahead optimizer [ZLHB19] and SlowMo [WTBR19].

As Reptile is only one of many in the zoo of meta-learning, the PFL community has drawn on this existing body of work. The popular model-agnostic meta-learning algorithm MAML [FAL17] and its first order approximation FOMAML have inspired the personalized federated learning algorithm PFedAvg [FMO20]. Different to the analogy of Reptile with FedAvg, pFedAvg allows several meta-gradient updates per communication rounds and thus fails to replicate the FOMAML exactly, but enjoys similar convergence guarantees. In [CK20] it has been shown that the aforementioned meta-learning methods can be generalized in a common form parameterized by the local learning rates, which the authors termed local update methods.

Other work from the meta-learning community proposes algorithms that regularize the local objective with the pairwise distances between the personalized models [GRF+20a] to speed up convergence. The FedProx [LSZ+20] algorithm similarly regularizes the l2 distance to the global model during the local updated steps. In PFL, this idea has been used to jointly train a local and global model, requiring the local model to stay close in terms of l2 distance to the global model [DTN20]. For this algorithm client participation in all rounds of training is mandatory as they need to train keep their local model training in sync with the global model, which is why we do not consider it in here, but restrict ourselves to exploring FedProx.

Further methods for PFL There are more methods available in personalized federated learning. Most generally, in [DKM20] and [HR20, HHHR20] personalized federated learning is viewed as learning a mixture of a local and global model. Specifically, [DKM20] allow tuning the degree of personalization required at each client adaptively. While this assumption is not required for the MAML type algorithms we investigate, we do use this idea to generate synthetic datasets that require personalization. Just as mixing the global and local model, in [MMRS20] the client data itself is interpolated, but requires special features to do so. Another approach is using contextualization, in this way user information can be fed into the model along with the input data in order to predict the label while taking the clients properties into account [MMRS20]. Some degree of personalization can be achieved by individually learning representations of the data which will differ from client to client [MMRS20]. In transfer learning for federated learning [WMK+19] a globally trained model is fine-tuned to the local training data, possibly freezing a few of the global models layers. This is close to our approach, although we consider only methods were we take very few fine-tuning steps rather than training until convergence. Personalization can also be accomplished by creating personalized models on the server side. FedAMP [HCZ+20a] creates personalized models from the local updates by averaging them according to their similarities leading to collaboration between similar clients. There is another branch of work that is not framed as personalized federated learning but similarly seeks to increase the local accuracy by providing different models to different clients at the server. Clustering federated learning methods learn several models and assign each client to such a model. In [SMS20] the global model is split up during training when it converges, clients are clustered according to their gradients at this point and each cluster continues to train their model independently of one another.

Empirical Evaluation of PFL The main difference in the application context between meta-learning and PFL is that the first focuses on the learning of new and very different tasks while PFL considers personalized clients that are all rather close to the global model. The question is then, when is personalization better than training a global mode? Or training separately? These question have already been raised in a few works [KMA+19, DKM20, YBS20]. One attempt at answering them has been done by developing a method to adapt the degrees of personalization on a client basis during training [DKM20]. Overall, empirically either very large datasets without the possibility to change data or easy datasets have been chosen that do not require personalization at all. Real world dataset may encounter both input and model similarity and the question is how to determine which algorithms are actually helpful for which clients - Personalization based or global model based ones. It is difficult to judge when personalization becomes worth pursuing and does not worsen the experience due to overfitting on the local data. Investigating the performance of both objectives for different levels of input heterogeneity *and* model heterogeneity has not been thoroughly investigated before. So far, the

PFL literature handselects the problem with ranging client numbers (e.g. between 20 [SMS20] and 50 [DTN20] for the MNIST digit dataset), different complexities of datasets (e.g. mostly MNIST or CIFAR-10, but notably some complex datasets as Reddit [YBS20]) or different model architectures (from a logistic regression [DKM20] to ResNet-18 [HCZ+20a] for MNIST), all of which make it very difficult to compare the individual outcomes and make general assessments. There is no benchmark dataset for personalized federated learning, although there has been an effort in creating benchmarking datasets and environments for standard federated learning, like recently the OARF Benchmark Suite [HLL+20].

5 Input and Model Heterogeneity

In this Section, we further look at the different types of heterogeneity that one encounters in federated learning. We especially look at sources of heterogeneity that lend themselves to approaches that make use of personalization. From an optimization perspective heterogeneity translates into a higher variability between the parameters/gradients received globally from the different clients [KKM+19]. But the sources of this variation can be of very different. Assuming that all clients use the same architecture to retrieve a model, what still differs is the local target and source distributions as well as the rate of participation, amount of local work and local data that then produce heterogeneous settings. The target data distributions could be seen as representing the heterogeneous interests client have in the federated learning setting, whereas the source distribution and its availability, the rate of participation and the ability to conduct local work all represent the heterogeneous abilities of the clients.

To make a more fine-grained distinction, we factorize both local source and target distribution for each client i as $p^i_{\mathrm{s(ource)}}(x,y) = p^i_{\mathrm{s}}(x)p^i_{\mathrm{s}}(x|y)$ and analogously for the target $p^i_{\mathrm{t(arget)}}(x,y)$ (dropping the dependence on i for readability whenever clear from context). The first factor is the *input heterogeneity*, the second the *model heterogeneity*. Each of the following quantities may vary between the different clients.

- $p^i_s(x)$: this is the evidence of the local source data, which is used in federated training.
- $p^i_s(y|x)$: this is the conditional distribution that is approximated via the local/global model during the federated training.
- $p^i_t(x)$: this is the evidence of the local target data, i.e. what the client actually expects to encounter after the federated training has finished.
- $p^i_t(y|x)$: this is the distribution of the labels that the client will encounter after the federated training has finished.

The range in which these distributions can vary between the clients is extensive. In the following, we focus on a couple of scenarios where they vary in different degrees.

A - Non-iid source distributions but equal targets. The target distribution is the same for all clients: $p^i_t(x,y) = p^j_t(x,y) \forall i,j$ with $p^(_t y|x)$ as its common conditional. The conditional source distribution is exactly this for all clients, s.t. $p^(_t y|x) = p^i_s(y|x) \forall i$. This means that clients are interested in learning a single shared global model. What differs, is the source distribution in terms of $p_s(x)$ at each client. To be able to generalize well, it is mostly assumed that sampling a data point from the globally identical $p_t(x)$ is like first sampling a client i uniformly at random (or weighted according to the amount of data they have) and then sampling a data point from $p^i_s(x)$ [HQB19b]. This also corresponds to the examples presented in [KKM+19, DTN20, FMO20, HCZ+20a], since even though every client has a different source distribution, the performance is assessed in terms of the global target distribution, identical for all clients. In this case, federated learning is helpful since the *few examples per client do not suffice to train a good model locally*, instead the clients profit from each

others additional information to achieve a better generalization on the target distribution quickly on a shared global model. One reason to learn local models in addition to a global one despite the fact that in theory a global model could be satisfactory to all clients, is that e.g. learning representations locally can *reduce need for communication* since it requires a smaller global model [LLL+20]. Even though this scenario uses local models, it does not correspond to personalized federated learning, since the goal is not fitting different 'personal' target distributions $p_t(x, y)$ but only to lighten the load on the communication channels. Only once the local target distributions vary between clients, the need for true personalization arises, which we summarize as scenario B with different subcases:

B - Non-iid target distributions, where source and target distributions are identical per client. When a clients source and target distribution are the identical, i.e. $p_s^i(x, y) = p_t^i(x, y) \forall i$, the client could in principle learn a local model without collaborating with others via federated learning. But factors like the *amount of local source data available and the local training capacity could hamper successfully fitting the source distribution* and make collaboration with other clients that have useful data available profitable or even necessary. Personalized federate learning postulates that this can also be the case when the target distributions themselves differ between clients, i.e. clients want to learn personalized models instead of a common global one. This is similar to ideas from multi-task learning where similar tasks are learnt jointly [FAL17, FMO20]. The performance of the overall federated learning system is then not only measured in terms of the global models performance but in terms of the performance of the personalized models at the clients on their own target distribution. Another motivating factor for personalized federated learning with different target distributions is that it might be *inefficient to learn a local model separately on each and every individual device.* Instead, clients that join later on should be able to arrive at a personalized local model based on the global one more quickly than training locally from scratch.

Obviously, the more different the $p_t^i(x, y)$ for different clients i, the more difficult it becomes to utilize the common information. In the following, B.1 and B.2 are two subcategories of such variations.

B.1 - $p_t^i(y|x) = p_t^j(y|x) \forall i, j$**.** Given that $p_s^i(y|x) = p_t^i(y|x)$ for all clients and even identical between all clients, we can still learn exactly one global model that correctly assigns a label y to an input x for all clients. We also have $p_s(x) = p_t(x)$ to ensure that $p_s(x, y) = p_t(x, y)$ locally, but different $p_s(x)$ between clients create the different target distributions. One example of this could be non-iid distributed labels at each client as in scenario A. The difference is only that the performance of a global model would be measured differently at each client since their input target distributions are different (unless the global fits $p_t(y|x)$ perfectly, which is unlikely). Next to non-iid $p_s(x)$ another scenario of varying $p_s(x)$ between clients would be having only a subset of the classes available at each client, an extreme case of a non-iid distribution. Some of these ideas were explored in [YBS20].

B.2 - $p_t^i(y|x) \neq p_t^j(y|x)$ **for** $i \neq j$**.** Another subcase is an inconsistent label distribution $p_t(y|x)$ at different clients. This implies that a global model can no longer fit all these distributions simultaneously (unless they the inputs x with non-zero probability in the target distributions are disjoint). Still, clients might profit from relevant information in each others data or be able to learn features of the data that are relevant to all clients jointly. Clustering or message passing approaches identify groups of clients that are particularly similar and profit from one another's consistent distributions [SMS20, HCZ+20b]. A more general approach is using ideas from multi-task learning, to learn an initial model that works well for adapting it locally e.g. [FAL17, FMO20]. In all cases, the global model (or several models available globally) need to lie in a cluster of the true best local models to give an especially good performance [GRF+20b].

C - Non-iid target and non-identical source distributions. Of course, we could also encounter problems where not only the target distributions vary between clients, but also the source distribution is not identical to the target distribution locally. For example, the source distribution could be dependent on time, so that it varies in different number of communication rounds at each client. If then some

form of running average is then actually identical to the target distribution, there is still the possibility to recover a relevant model that generalizes well to fit this target distribution. But unless there is some type of such a proxy for the target distribution, as it was the case with the sum of the local source distributions in A or the local source distribution in B, there is no hope in finding an algorithm that generalizes well to the target distribution.

6 Datasets

To empirically evaluate different PFL and meta-learning algorithms in a common scenario in Section 7, we define two datasets. Specifically, both are parameterized by two values that allow to adapt model and input heterogeneity. The model similarity α quantifies the similarity of the models $p_i(y|x)$ over all clients. The input variance γ reflects the variance in the input distributions $p_i(x)$. Note that the values correspond to similarities that are based on the generation process and to not relate to distributional similarity measures like the KL-Divergence or the Wasserstein distance.

6.1 Synthetic Data: Federated Hidden Manifold Model

For the synthetic dataset, each clients dataset $\mathcal{D}_i$ is generated via a sampling strategy for the input samples $\mathbf{x} \sim p_i(x)$ and a model g_i that follows a probability distribution $p_i(x|y)$.

The input samples are drawn i.i.d. from a Gaussian $\mathcal{N}(\mu_i, 1)$ where $\mu_\mathbf{i} \sim \mathcal{N}(0, \gamma)$. Thus, adjusting the variance γ of the distribution from which the means are sampled produces clients whose input distributions are closer or further away from one another. This strategy which gives *input heterogeneity* is inspired by [DTN20].

The models themselves are parameterized functions $g : \mathcal{X} \times \mathcal{K} \to \mathcal{Y}$ used to assign labels to the sampled inputs. For example, such a function g could be a neural network with one hidden layer, where the parameters are the weights and biases and lie in $\mathcal{K}$. Note that the parameterization of g_i is independent of the model architecture used in federated learning, which is why the parameter space $\mathcal{K}$ may differ from $\mathcal{W}$ introduced earlier. Taking a parameter vector $\kappa \in \mathcal{K}$ together with the input sampling gives a training dataset: Sample x from $p_i(x)$ and label it according to $g_i(\mathbf{x}) = y$ to generate a sample for client i.

The parameter vectors $\kappa_i \in \mathcal{K}$ for each client are generated in the following way: First, the *common model* κ^{common}'s entries are sampled i.i.d. from a Gaussian distribution $\mathcal{N}(0, 1)$. Then, for each client a vector $\tilde{\kappa_i}$ is sampled from the same distribution. Depending on the model similarity $\alpha \in [0, 1]$ each resulting client model interpolates between the client parameter vector and the common model

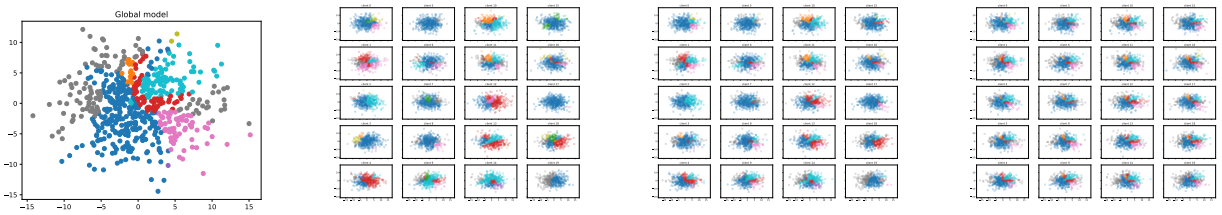
$$\kappa_i = (1 - \alpha) \cdot \tilde{\kappa_i} + \alpha \cdot \kappa^{\text{common}}$$

. This yields more similar models when α is bigger, and completely independent models when α is small, so we take it to model some *model heterogeneity*. Examples of datasets for different sets of parameters that generated via this process are shown in Figures 1 and 2.

To make the dataset slightly more realistic and taking into account that for real-world data, e.g. natural images, relevant inputs lie only on manifolds of the input space, we allow the option to adapt the model to allow for a hidden manifold [GMKZ20].

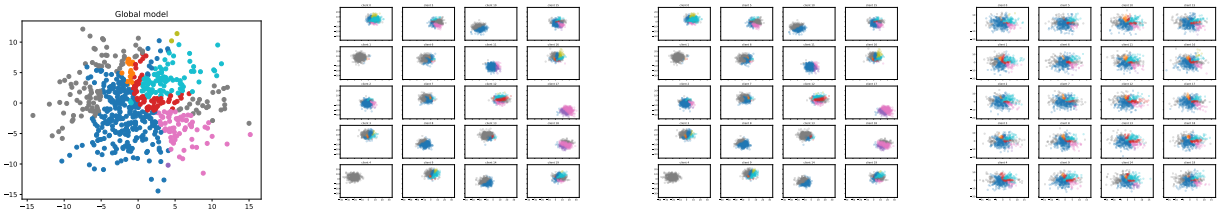
6.2 (E)MNIST

We also propose a method to change model similarity and input variance simultanously to different degrees for classification datasets like (E)MNIST [CATvS17]. To introduce variance in the inputs, each clients data is composed of two differently composed subset. One part of relative size $(1 - \gamma)$ is i.i.d. data, i.e. randomly sampled from the 47 available classes at each client. The remaining subset of size γ is filled for one client after another by the remaining sorted, so that they are generally made up of no more than two classes for n clients and $n < |\mathcal{Y}|$.



(a) common model (b) $\alpha = 0.75$ (c) $\alpha = 0.85$ (d) $\alpha = 0.95$

Figure 1: *Data with different model similarity α and no input variance γ.* Given a common model θ on the left this shows a sample of the derived client distributions in b), c) and d) for different values of the model similarity α. The input data is two dimensional and ten labels can be assigned in the model output, although only 8 are actually used in the common model.



(a) common model (b) $\gamma = 10$ (c) $\gamma = 5$ (d) $\gamma = 0$

Figure 2: *Dataset with varying input variance γ and high model similarity.* For the common model (a) different clients sample the input space according to a Gaussian with variance 15 and a mean μ_i with every entry sampled i.i.d from $\mathcal{N}(0, \gamma)$. For small γ as in, the input spaces overlap, for $\gamma = 0$ they overlap completely. In addition, the model similarity is $\alpha = 0.95$ so we have only small differences between the models.

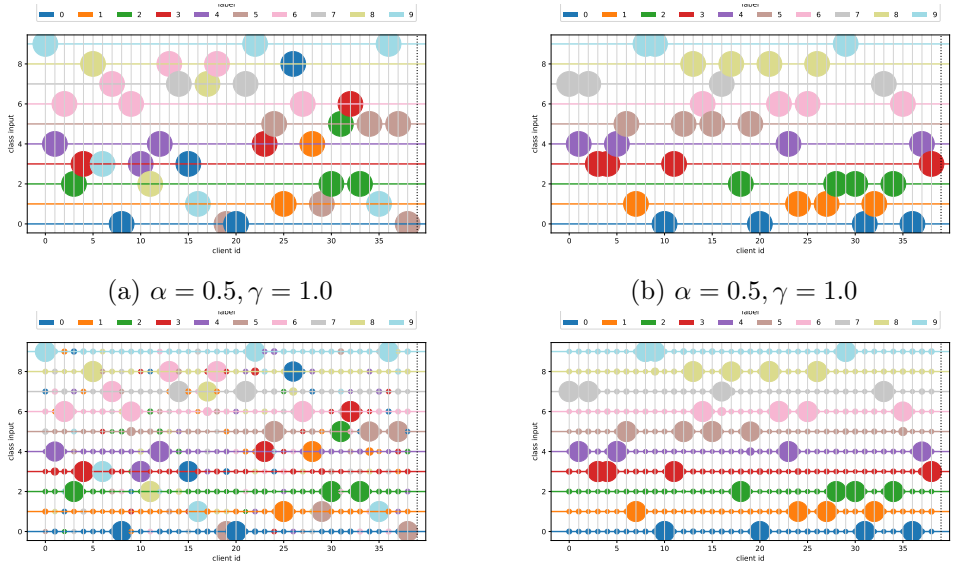
To get different models at each client and thus create the need for personalization, we shuffle subsets of the labels locally independent of other clients. This is inspired by the fact that fully shuffling all labels over the task is commonly used to train few shot learning. Here, we parameterize this model similarity α by the number of labels that are kept as in the original dataset. This way, $\alpha = 0.5$ means that half of the labels are shuffled and $\alpha = 0$ means that the model labels are shuffled completely. An example for a generated dataset for MNIST is in Figure 3.

7 Experiments

As we want to investigate how the different meta-learning and PFL algorithms from Section 3 compare under different model and input heterogeneity, we run federated training on the datasets from the previous section. We refer to $w \in \mathcal{W}$ learned by the training phase of federated learning as the "global model" and the adaptions by the clients $\theta_i = \mathcal{A}_i(w)$ as the "local models". We also distinguish between training and testing clients, the first of which participate in training while the latter do not participate. This enables us to measure the usefulness of the federated learning process for newly joint clients.

In our experiments we now specifically ask under different model heterogeneity α and input heterogeneity γ

- Which algorithms are advantageous for different model or input heterogeneity?
- When is personalized federated learning advantageous over standard federated learning? Local model does not improve of the local model?



(a) $\alpha = 0.5, \gamma = 1.0$ (b) $\alpha = 0.5, \gamma = 1.0$

(c) $\alpha = 0, \gamma = 0.5$ (d) $\alpha = 0, \gamma = 0.5$

Figure 3: *MNIST dataset.* Examples for differing degrees of model similarity (consistent labels per client) and input variance (amount of single-class data per client). Furthermore, $\alpha = 1$ and $\gamma = 0$ gives the standard federated learning setting with identical model and i.i.d. data over all clients. Changing only γ from there on gives B.1. Finally, we get B.2 by changing both parameters.

- Do different types of input heterogeneity (sampling specific regions of the input space vs. selecting different labels) influence these outcomes differently?

To answer these questions we keep track of the average generalization error of the global and local models on the clients via the local test data.

In addition, for comparability, every optimizer used in the algorithms is only allowed the same number of maximal local epochs, the same number of maximal local gradient evaluations and the batchsize is kept constant. All optimizers have to optimize the same architecture. Clients participating in training are selected in the same order. Also, methods like batchnorm or dropout are not used.

7.1 Experiments on the Snthetic Dataset

Setup. *(Dataset)* As a dataset we use the synthetic dataset in Section 6.1. We select the same random seed for generating the common and client models θ^{glob} and θ_i^{loc} for all subsequent experiments unless otherwise mentioned. For creating different scenarios we either change the model similarity via the interpolation factor α or by changing the sampling for the inputs. As an architecture we use a neural network acting on 4 input dimensions to give an output of 10 dimensions of which the argmax is interpreted as the label. It has 3 hidden layers (relu activations, 80,50,40 units each). The common model κ^{common} was hand selected to contain a large number of classes (8/10) for a Gaussian input distribution of $\mathcal{N}([0,0],[[15,0],[0,15]])$.

Changing α from 0 (no similarity) to 1 (identical clients) changes the model heterogeneity. Since for a value of $\alpha > 0.75$ the models are already so heterogeneous, that in all cases federated training does not improve over the Separate baseline we conclude that for this specific dataset the models are already too dissimilar to be of value for one another, so we confine our experiments to the range $[0.75, 1.0]$. For the input variance γ we vary it from 0 (identical means) to 10 (variance of 10 between the means in each dimension independently).

Given this parameterization setting $\alpha = 1$ and varying $\gamma \neq 0$ and measuring only the average error on each clients test data is scenario A. Accounting for differences and introducing new clients gives scenario B.1. To create a B.2 scenario, we do as in B.1 but also allow a non-zero α to decrease model similarity.

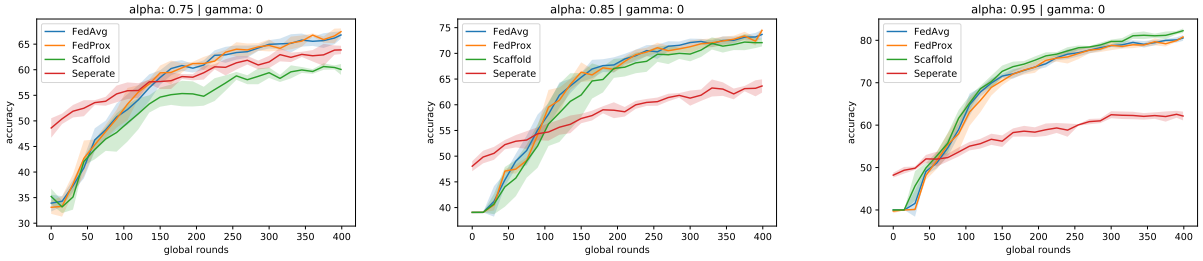
(Algorithms) We compare the algorithms FedAvg, FedProx and Scaffold in addition to the baseline, Separate, in a setting with 30 training clients and 10 test clients. Unless otherwise mentioned we run the training for 400 rounds and the clients each train for 2 epochs locally with a batch size of 16. Each client has 64 distinct samples from the source distribution available. This is equal for the target distribution, for testing we use four times as many samples.

When training classic SGD centrally on all training data at all clients taken together for $\alpha = 1$ and $\gamma = 0$ we can achieve an accuracy of more than 93% with a neural network network with hidden layer sizes $[240, 120, 80, 50, 40]$ for training until convergence.

We tune the local learning rates of the algorithms in the range of $\{0.2, 0.1, 0.05, 0.01\}$ and the regularization parameter of FedProx for $\{1.0, 0.1, 0.01\}$.

Varying the model similarity α under constant input heterogeneity γ. We first set $\gamma = 0$ and thus consider only datasets where input samples are i.i.d. at each client, i.e. $p_s^i(x) = p_s^j(x)$ for all clients i, j. Varying the model similarity α increasingly changes the similarity of the datasets as visualized in Fig. 1 from the Section 6. The average accuracy of the clients personalized models in the federated setting is much higher than training separately for $\alpha = 0.95$ as shown in Figure 4 (c). This is closest to the standard federated learning setting where Scaffold is essentially equivalent to FedAvg. Interestingly, in this setting it takes about 40 global rounds until the federated methods outperform Separate training, which converges much slower overall but starts with a higher accuracy. For decreasing the similarity α, Separate outperforms the best federated method for approximately the first 70 and 110 global rounds. *As the model similarity decreases, Separate training becomes a viable option if the available number of global communication rounds is small.* Also, the less similar the clients are, the more the performance of Scaffold is reduced. When $\alpha = 0.75$, Scaffold even fails to reach the baseline when training ends while FedAvg and FedProx surpass it.

Overall, Scaffold finds personalized models with higher norms, see Table 1. This suggests that *FedAvg and FedProx learn models that generalize better to different clients.*



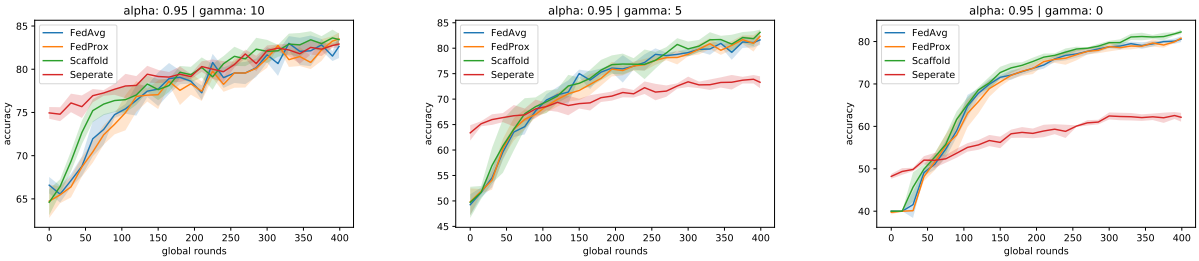
(a) $\alpha = 0.75$ (b) $\alpha = 0.85$ (c) $\alpha = 0.95$

Figure 4: Average training accuracy of local models for varying the model similarity α. The input variance γ is set to zero, so all clients sample from the same Gaussian distribution. The Separate baseline performs consistently over all values of α, as it does not profit from shared gradient updates. For (a) Scaffold fails to outperform the baseline and profit from the federated training process at all. For (c) Scaffold outperforms FedAvg and FedProx. *Scaffolds performance decreases to FedAvg and FedProx as the model similarity decreases, but outperforms them when the models are very similar.*

$\alpha =$	0.75	0.85	0.95
FedProx	205.59	221.87	252.35
FedAvg	214.95	226.88	259.36
Scaffold	**237.59**	**251.94**	**269.77**

Table 1: Average norms of the personalized models after federated training. FedProx generally finds smaller models while Scaffold consistently reaches the biggest model norm. Average for three different client selection orders $\{S_r\}_{r=0}^{T}$.

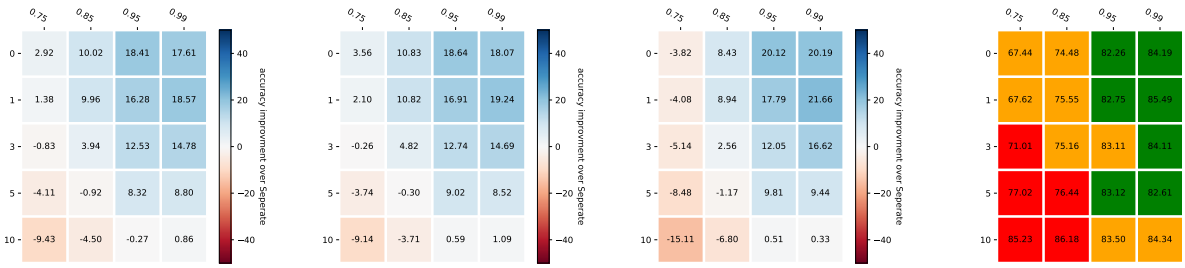
Varying the input distributions γ under constant α. This setting corresponds to what is often understood as heterogeneous federated learning. The clients have different data available but sample from different subsets of the global distribution, i.e. $p_s^i(x) \neq p_s^j(x)$ for $i \neq j$, but the goal is to learn a common model. Varying the input variance γ between clients we can investigate this aspect with datasets generated as in Figure 2. The average accuracy of the training clients is depicted in Figure 5. First of all, for higher variance γ learning individual models via Separate becomes easier. This is because as the input spaces of the client's training input distributions move away from $\mathbf{0}$ they move to less complicated regions of the space, where they only have to learn to distinguish a few classes rather than the many that are encountered around the center. It seems that in this case, *for varying γ under very similar models, the algorithms choice is not too relevant, even though federated techniques reach or improe over the baseline.*



(a) $\gamma = 10$ (b) $\gamma = 5$ (c) $\gamma = 0$

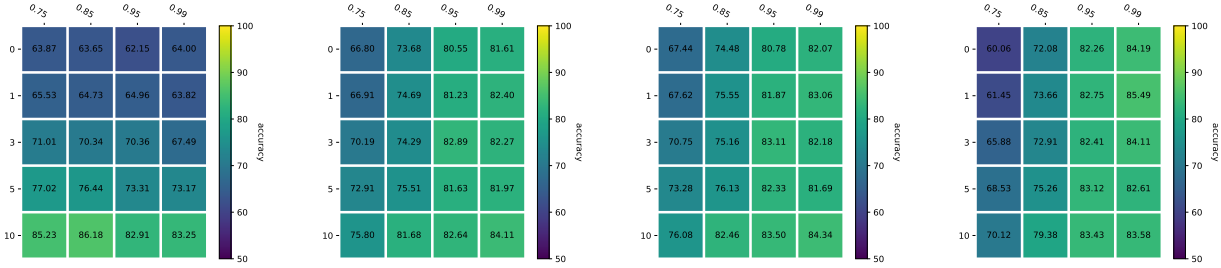
Figure 5: Average training accuracy of local models for varying the input variance γ. For $\gamma = 0$ Scaffold fares better than FedAvg and FedProx, but for the other parameters there is hardly a difference between the performance taking the variance between the 5 runs conducted into account. *Under high model similarity, moving the input distributions far away from another leads to disjoint distributions for which the federated training does help in the given number of rounds.*

Varying model similarity α and input variance γ concurrently. In addition to evaluating α and γ for fixed values of one another, we investigate how different algorithms perform under concurrently changing model similarity and input variance. Overall, we find that *for improving the personalized accuracy of training clients under high model similarity Scaffold works best as long as the input variance is not to high*, see Figure **??**. In the case of very high input variance $\gamma = 10$ the input spaces are completely disjoint and it might actually be easier to find different models via personalization than fit one that works for all of them. This is consistent for decreasing the model similarity, as the input distributions show almost no overlap and FedAvg performs best, suggesting that the personalized models learned do not benefit from the information of other clients anymore as their data is too far away in the input space from their own. While Scaffold fails to improve over the Separate baseline for a low model similarity $\alpha = 0.75$, FedAvg and FedProx work better in some cases (Figure 6 (a)-(c)). Eventually, when both model similarity is low and input variance is high, federated training does not improve over the separate baseline anymore. In this case, federated training is only useful when we consider the test clients that did not participate in training and join afterwards, see Figure 8. For them, the federated training is still better, as is provides a good initialization for their training and find better parameters than starting from the initial model. FedAvg and FedProx provide a better initialization than Scaffold (Figure 8).



(a) FedAvg (b) FedProx (c) Scaffold (d) Best

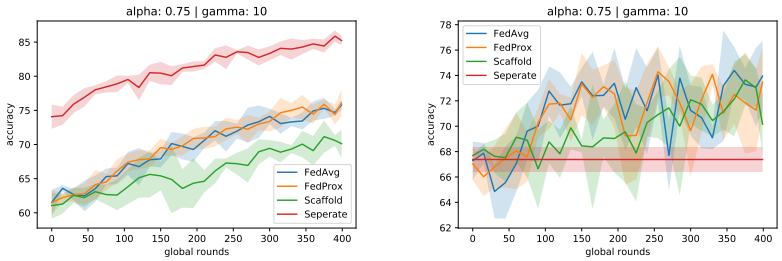
Figure 6: Improvement over the Separate baseline of federated training algorithms for 400 rounds of communication. (α in column, γ in rows). In (d) we depict the best of the four algorithms FedAvg, FedProx, Scaffold and Separate (average over 5 runs). *FedAvg and FedAvg and especially FedProx improve over Separate on a wider range of parameters than Scaffold, but for high model and input similarity Scaffold outperforms them in terms of accuracy. If both models and input are very dissimilar, a federated training process is counterproductive.*



(a) Separate (b) FedAvg (c) FedProx (d) Scaffold

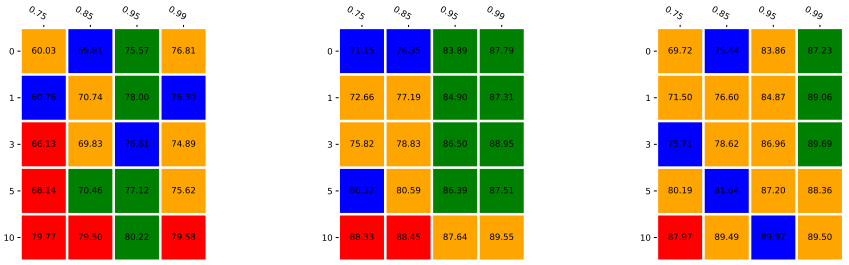
Figure 7: Training client accuracy for different algorithms. (α in columns, γ in rows) Average of 5 runs with different client selection orders. *Overall, the individual models are easier to fit when the input variation is higher, as their input space captures less complex regions of the input space.* Only given enough similarities in either model or input similarity the algorithms achieve higher accuracy.

Changing the amount of local computation available We also change the number of epochs a client conducts locally. Specifically, we consider $\{1, 5, 10\}$ epochs. This corresponds also to the amount of fine tuning that is allowed for the personalization after training. We report the average training accuracy after 400 rounds of communication. For one epoch and 4 local steps (4 batches with 16 samples each) the different federated learning algorithms do not differ much. As the amount of local work is increased to 20 and 40 steps, the picture becomes a bit clearer. While Scaffold has an advantage for a moderate number of steps for similar models, for a high number of steps the difference to FedAvg in terms of training accuracy becomes smaller, see Figure 9.



(a) training clients (b) test clients

Figure 8: Average training accuracy on training and test clients. Even though the Separate baseline outperforms federated training for training clients, we note that with Separate training, for clients that did not participate in training no local model has been trained. Using only personalization of the global model they achieve a higher accuracy more quickly than with the normally initialized model, although in this case just a little bit. *Even though federated training might not give an advantage for clients that participate in training, for those that do not participate, being provided with a good global model as an initialization can be better than starting from scratch when only limited computation is available locally.*



(a) 1 local epoch (b) 5 local epochs (c) 10 local epochs

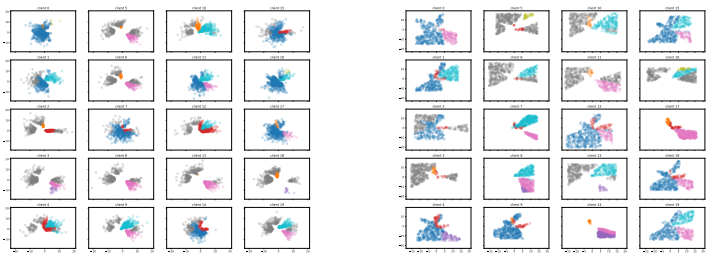
Figure 9: Varying the amount of local epochs during training and personalizing. To compare these plots note that the amount of communication rounds is equal, but the local computation and personalization are increased from left to right. The algorithms are FedAvg, FedProx, Scaffold and Separate). *As the number of local computation steps is increased, datasets with disjoint more dissimilar inputs and models profit more.*

A different type of input heterogeneity: Subsampling the labels. Instead of just shifting a Gaussian input distribution by changing its mean, we also investigate how subsampling the labels at each clients affects the training accuracy. This also increases the variance between the input distributions although in a different way to what we investigated before. This has been done before to show how Scaffold outperforms other algorithms on the global objective, but not for a personalized version of the model [KKM+19].

For subsampling labels we use the same overall input distribution for all models, setting $\gamma = 0$. Then, we determine the number of labels available at the client as the number of different class labels that occur by sampling 500 input points $\mathcal{X}_0 \subset \mathcal{D}$. From these we select k labels at each client uniformly at random (independent of their evidence). For generating samples for a complete dataset of those labels we either (a) sample again Gaussian from the input and keep only the selected labels or (b) sample uniformly in the bounding box of $\mathcal{X}_0$ until we reach the desired number of training and test samples. For this experiment we resort to a two dimensional input space, but keep all other parameters as before, so we are able to visualize the result of this process in Figure 10.

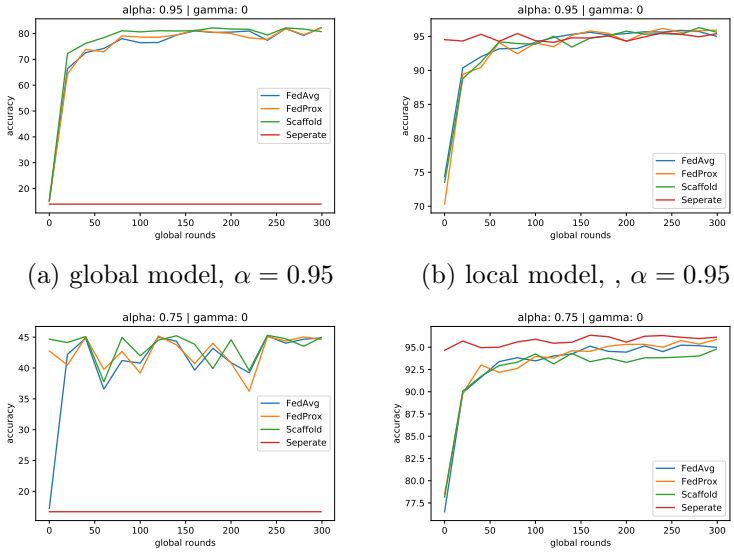
A visible difference between FedAvg and Scaffold when we subsample only three classes can be only observed with high enough model similarity between the clients at $\alpha = 0.95$ as in Figure 11. In this

case Scaffold converges to a higher accuracy of the global model. Eventually, this has little influence on the accuracy of the personalized models, since with so many local steps its actually possible to converge to a reasonable personal solution quickly. For a lower model similarity $\alpha = 0.75$ the global model varies very much and does not gain much. In contrast, for the local model FedAvg and FedProx work better than Scaffold.



(a) (b)

Figure 10: *Subsampling of 3 labels at each client.* In (a) the selected labels are sampled from a Gaussian and in (b) they are sampled uniformly from a bounding box determined by a Gaussian. the model heterogeneity is 0.95.



(a) global model, $\alpha = 0.95$ (b) local model, , $\alpha = 0.95$

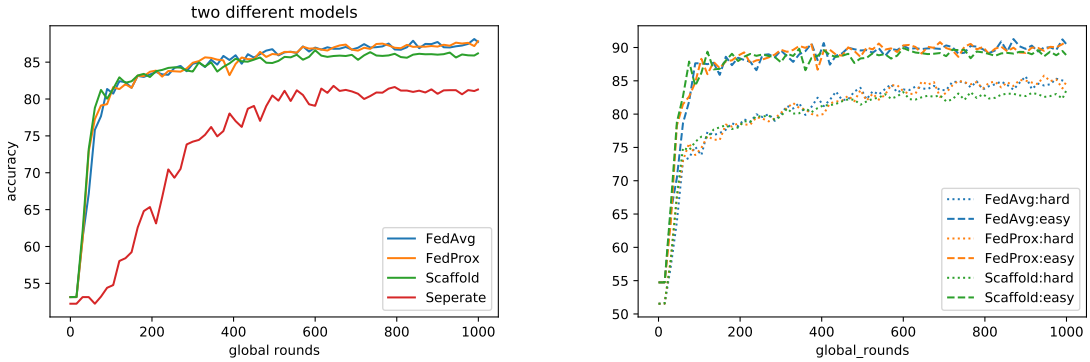
(c) global model, $\alpha = 0.75$ (d) local model, $\alpha = 0.95$

Figure 11: Generalization error on global and local models for subsampling 3 labels. While Scaffold outperforms FedAvg and FedProx on the global model for high similarity between the model $\alpha = 0.95$, this improvement does not translate to the performance of the local model. *For less similar models Scaffolds global model is on par with the other algorithms in terms of the accuracy of the global model but performs worse on the local model.*

Examining the personalization phase. To further understand what exactly happens in the personalization phase in the parameter space, we devise another dataset. It contains two equally sized groups of clients that have one of two types of models for generating the data. The first group is close to a "hard" common model $\kappa \in \mathcal{K}$ which is the previously used neural net and likewise sampled from $\mathcal{N}(0, 1)$. By fitting a linear regression to a large amount of data generated from the hard model, we obtain a model that is "easy". One could view it as a linear approximation of the hard dataset. This

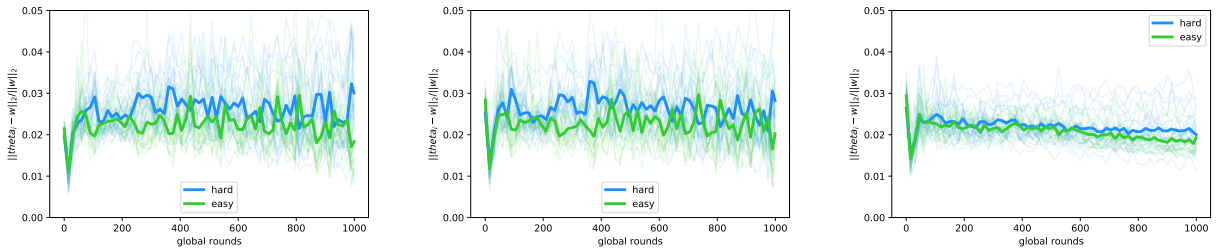
model is close to the second group of clients models (in the parameter space of the linear regression). The inputs for all clients are generated from a common input distribution.

Training our algorithms on this dataset, we observe that the performance of the clients with the data that is separable via the logistic regression is indeed higher and reaches a higher accuracy more quickly, see Figure 12. The accuracy curve of the difficult clients takes longer to converge. For FedAvg it looks like this convergence is even not completed after 1000 rounds, whereas Scaffold stagnates. This also translates to the global model. We also measure the distances between the global and personalized models during training for the individual clients in Figure 13. For Scaffold this difference decreases during training whereas it stays rather constant for FedAvg and FedProx. Note that for all models, the global model is closer to the easy model.



(a) Average over all clients. (b) Average over clients by model type.

Figure 12: Personalized accuracy for training clients in a dataset with clients from two different groups, some that want to learn a "hard" model and some that want to learn an "easy" model. *For this dataset with clients of different difficulty FedAvg and FedProx learn a better personalized model, which takes its advantage mostly from outperforming Scaffold on the hard model.*



(a) FedAvg (b) FedProx (c) Scaffold

Figure 13: Average l_2-distance of personalized models $\theta_i \in \mathcal{W}$ to the global model $w \in \mathcal{W}$ during training. Normalized by the size $||w||_2$ of the global model. Thin lines are individual clients. The distance between the global and local model of FedAvg and FedProx varies more strongly for those algorithms than for Scaffold. All datasets stay closer to the easy model. *In the case of this dataset the better performance of FedAvg is not due to a closer placement to the hard models in terms of l_2 distance, in contrast to a previous work's conclusion [CMS20].*

7.2 Experiments on EMNIST

Now, we want to analyse a dataset that uses real data, namely the "balanced" EMNIST dataset [CATvS17]. For this we use the parameterization as in Section 6.2. The input dissimilarity γ for 0 means that no part of the data at each client stems from an i.i.d. assignemnt. $\gamma = 1$ means that the data is completely i.i.d. distributed over all clients. This is analogous to the experiment on

input variance of Scaffold [KKM+19]. In our case we also add the possibility to shuffle labels on the client level. The model similarity α is here defined in terms of the number of class labels of a client that is consistent with the original assignment of images to the labels. An $\alpha = 0$ means completely inconsistent, $\alpha = 1.0$ means completely consistent.

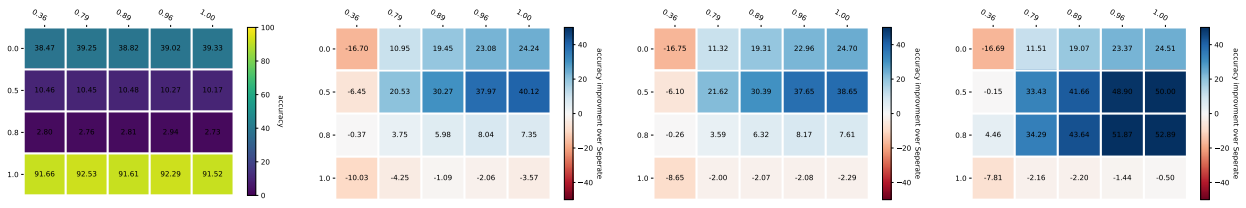
We use a two layer neural net with 600 hidden neurons during training of $n = 100$ clients on the "balanced" EMNIST split which contains 47 balanced classes of handwritten letters and numbers. This means at each client there are 810 samples available per client. We employ a batchsize of 162 which leads to 5 steps per local epoch. The following experiments we conducted local training for 1 local epoch. The learning rates are again optimized in the range of $\{0.1, 0.01, 0.001\}$. The experiments are run for 1000 rounds.

First note that due to the local shuffling of the labels per client, the Separate baseline (Figure 14 (a)) which only trains locally performs equally well for all model similarities. This is because the model similarity in this case is changed by permuting a subset of the labels to which the difficulty of local training is obviously invariant.

Also, for an input similarity of zero, the Separate baseline easily achieves a very high accuracy due to the fact that this means that all clients have only one class available which is trivial to fit and thus leads to a good local test error.

We now compare the performance increase of the different federated learning algorithms over Separate in Figure 14 (b), (c) and (d). Firstly, using Separate training is better when the training instance is trivial to fit. Separate is also reaches a higher accuracy quicker when the models are so different, that almost all labels are inconsistent (while still maintaining the clusters). Note that in this case 1000 global communication rounds were not sufficient for either of the methods to converge, highlighting that such a difficult dataset also takes more time to be learned. If the time to train and number of global communication rounds is scarce though, here Separate training should be preferred. Unfortunately, there was no time to conduct more extensive experiments with more communication rounds on this issue.

In between the federated training methods, Scaffold consistently outperforms or performs on par with FedAvg and FedProx. For i.i.d input distribution and not input variation they perform equally well. This also holds true for i.i.d input data in the case of varying degrees of model similarity, already suggesting that for this dataset with more diverse models FedAvg does have an advantage over Scaffold. For non-i.i.d. inputs Scaffold gains an advantage as the input similarity decreases. Especially for the regime of high input variance of 0.8 Scaffold outperforms FedAvg by a margin of almost 40%, across all model similarities. In this case, dealing with the input variance seems to be the biggest challenge for the optimization algorithm.



(a) Separate (b) FedAvg (c) FedProx (d) Scaffold

Figure 14: Average training client accuracy for EMNIST under changing model (columns) and input (rows) similarities. The Separate baseline's accuracy in (a) is invariant to the model similarity, i.e. shuffling the labels locally. Scaffold consistently outperforms FedAvg and FedProx, even for low model similarity. *For the current values of α and γ the input variance that helps to keep up Scaffolds advantage also for 0.8 seems the most important factor.*

8 Conclusion

Overall, we saw that the similarity between methods from meta-learning and personalized federated learning is indeed striking. For high model similarity, the specialized federated methods for non-i.i.d. distributed input Scaffold fares best on in our synthetically generated datasets. As model similarity decreases and the input spaces in the training data become disjoint, Scaffold is outperformed by FedAvg i.e. the meta-learning method Reptile. In addition, we saw that FedProx is performing similar to FedAvg as it equates FedAvg for $\mu = 0$, and has the advantage of being optimized over more hyperparameters. Thus using FedProx would be preferable, if the capacity for tuning its parameters is available. For practical algorithm design in PFL we found that it is specifically important to be aware of the alternatives for federated training, i.e. separate individual training as this can already lead to good results. Finally, we provide the code for the framework with the our experiments that can also be swiftly adapted to using other datasets, model architectures and PFL algorithms, while keeping track of relevant metrics up onto the individual client level.

References

[AES19] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml, 2019.

[CATvS17] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017.

[CK20] Zachary Charles and Jakub Konečný. On the outsized importance of learning rates in local update methods, 2020.

[CMS20] Liam Collins, Aryan Mokhtari, and Sanjay Shakkottai. Why does maml outperform erm? an optimization perspective, 2020.

[DKM20] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. Adaptive personalized federated learning, 2020.

[DTN20] Canh T. Dinh, Nguyen H. Tran, and Tuan Dung Nguyen. Personalized federated learning with moreau envelopes, 2020.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[FMO20] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach, 2020.

[GMKZ20] Sebastian Goldt, Marc Mézard, Florent Krzakala, and Lenka Zdeborová. Modeling the influence of data structure on learning in neural networks: The hidden manifold model. *Physical Review X*, 10(4), Dec 2020.

[GRF+20a] Micah Goldblum, Steven Reich, Liam Fowl, Renkun Ni, Valeriia Cherepanova, and Tom Goldstein. Unraveling meta-learning: Understanding feature representations for few-shot tasks, 2020.

[GRF+20b] Micah Goldblum, Steven Reich, Liam Fowl, Renkun Ni, Valeriia Cherepanova, and Tom Goldstein. Unraveling meta-learning: Understanding feature representations for few-shot tasks, 2020.

[HCZ+20a] Yutao Huang, Lingyang Chu, Zirui Zhou, Lanjun Wang, Jiangchuan Liu, Jian Pei, and Yong Zhang. Personalized federated learning: An attentive collaboration approach, 2020.

[HCZ+20b] Yutao Huang, Lingyang Chu, Zirui Zhou, Lanjun Wang, Jiangchuan Liu, Jian Pei, and Yong Zhang. Personalized federated learning: An attentive collaboration approach, 2020.

[HHHR20] Filip Hanzely, Slavomír Hanzely, Samuel Horváth, and Peter Richtárik. Lower bounds and optimal algorithms for personalized federated learning, 2020.

[HLL+20] Sixu Hu, Yuan Li, Xu Liu, Qinbin Li, Zhaomin Wu, and Bingsheng He. The oarf benchmark suite: Characterization and implications for federated learning systems, 2020.

[HQB19a] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification, 2019.

[HQB19b] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification, 2019.

[HR20] Filip Hanzely and Peter Richtárik. Federated learning of a mixture of global and local models, 2020.

[HRM+18] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *CoRR*, abs/1811.03604, 2018.

[JKRK19] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning, 2019.

[KKM+19] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for on-device federated learning. *CoRR*, abs/1910.06378, 2019.

[KKP20] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. Survey of personalization techniques for federated learning, 2020.

[KMA+19] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2019.

[LLL+20] Paul Pu Liang, Terrance Liu, Ziyin Liu, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *CoRR*, abs/2001.01523, 2020.

[LSZ+20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2020.

[MMRS20] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning, 2020.

[NAS18] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018.

[PW20] Reese Pathak and Martin J. Wainwright. Fedsplit: An algorithmic framework for fast federated optimization, 2020.

[RFKL19] Aravind Rajeswaran, Chelsea Finn, Sham M. Kakade, and Sergey Levine. Meta-learning with implicit gradients. *CoRR*, abs/1909.04630, 2019.

[SMS20] F. Sattler, K. Müller, and W. Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 2020.

[WMK+19] Kangkang Wang, Rajiv Mathews, Chloé Kiddon, Hubert Eichner, Françoise Beaufays, and Daniel Ramage. Federated evaluation of on-device personalization, 2019.

[WTBR19] Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael Rabbat. Slowmo: Improving communication-efficient distributed sgd with slow momentum, 2019.

[YBS20] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation, 2020.

[ZLHB19] Michael R. Zhang, James Lucas, Geoffrey E. Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. *CoRR*, abs/1907.08610, 2019.

A Summary of Parameters

R	# of global communication rounds
T	# of local steps per one global round
N	# of clients
$\mathcal{S}_r$	# of clients that were selected for sending update to server in communication round r
B	batch-size for local training
$\mathcal{X} = \mathbb{R}^d$	input space with dimension d
$\mathcal{Y} = \mathbb{R}$	class label
$\mathbf{D} \in (\mathcal{X} \times \mathcal{Y})^n$	set of n datapoints
$\mathbf{x} \in \mathcal{X}$	instance of an input
$y \in \mathcal{Y}$	instance of a label
$\mathbf{d} = (\mathbf{x}, \mathbf{y})$	datapoint
$\mathcal{D}_i$	local data distribution
$\mathcal{D}_i^{\text{source}}$	local source data distribution for client i referred to as $p_s(x, y)$ for a given input x and a
$\mathcal{D}_i^{\text{target}}$	local target data distribution for client i, sometimes referred to as $p_t(x, y)$ for a given in
$\mathcal{W}$	parameter space
$w \in \mathcal{W}$	parameters of the global model
$\theta_i \in \mathcal{W}$	parameters of the local model for client i
$f : \mathcal{W} \times \mathcal{X} \to \mathcal{Y}$	predictive model
$l : \mathcal{W} \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$	loss function
$l_i : \mathcal{W} \to \mathbb{R}$	empirical loss for the local test data of client i i.e. $l_i(\theta) = \mathbb{E}_{\mathbf{D} \sim \mathcal{D}_i^{\text{target}}} [l(\theta; \mathbf{D})]$
σ	sigmoid function