Personalized Federated Learning with Exact Distributed Stochastic Gradient Descent Updates

Sotirios Nikoloutsopoulos[1] Iordanis Koutsopoulos[1] Michalis K. Titsias[2]
[1]Athens University of Economics and Business
[2]DeepMind

ABSTRACT

We propose a novel Stochastic Gradient Descent (SGD)-type algorithm for personalized Federated Learning. The model to be trained includes a set of common weights for all clients, and a set of personalized weights that are specific to each client. At each optimization round, randomly selected clients perform full gradient-descent updates over their client-specific weights towards optimizing the loss function on their own datasets, without updating the common weights. At the final update, each client computes the joint gradient over both the client-specific and the common weights and returns the gradient of common weights to the server. With this procedure, which we define as exact distributed SGD, an exact and unbiased SGD step is performed over the full set of weights in a distributed manner, i.e. the updates of the personalized weights are performed by the clients and those of the common ones by the server. For this optimization scheme we rigorously prove convergence in non-convex settings such as for training neural networks. Our obtained theoretical guarantees translate to superior performance in practice against baselines such as FedAvg and FedPer. We demonstrate this in several multi-class classification datasets, such as in Omniglot, CIFAR-10, MNIST, Fashion-MNIST, and EMNIST. Further, we show that our optimization scheme has low computational cost per iteration round.

KEYWORDS

Federated Learning, Distributed Learning, Personalization

1 INTRODUCTION

Federated learning (FL) aims at training a global Machine Learning (ML) model out of distributed datasets that reside in different clients. One of its basic assets is that datasets do not need to be transferred to a central location, and thus FL overcomes data privacy concerns during model training [33]. Chronologically, the first FL algorithm is FedAvg [25]. In FedAvg, at each round, the server selects a subset of clients and sends to them the current model parameters (weights). The clients update model parameters locally by performing a number E of local gradient update steps towards optimizing their loss function, and they return the locally optimized parameters to the server. The server then averages model parameters and sends them back to the clients. For $E = 1$, FedAvg is called Federated Stochastic Gradient Descent [14]. FedAvg notoriously suffers from data heterogeneity, when each client has to solve its own *personalized* task, i.e. when the clients' datasets are drawn from different distributions. In such cases, a shared global model and an naive aggregation of its parameters could lead to convergence to a stationary point which is far from optimal [22, 35].

Several extensions of FedAvg have been proposed, see e.g. [20, 21, 24, 28, 35, 39], which try to improve the behaviour of the algorithm in heterogeneous data settings. While such methods can improve convergence, they still try to learn a fully shared/global ML model across heterogeneous clients. From a modelling perspective, this may not be the best choice since it ignores client personalization.

Personalized Federated Learning arises often in real-life applications, such as next word prediction [12], emoji prediction [18, 27], health monitoring [37], and personalized healthcare via wearable devices [6]. If local client datasets are drawn from different data distributions [15], then the prevalent way in which FL is done is not suitable for producing a global model that will perform well at the specific ML task of each client. As a result, the training process can be slow or even diverge.

One important class of personalized FL methods dictates clients to share some common weights anyway, since clients cannot produce a good model on their own due to insufficient amount of data. However, there also exist some client-specific (personalized) weights for each client as well that are trained on their own datasets [2, 15]. In this work, we consider this approach to address data heterogeneity in personalized FL. Along with the global (shared) model parameters, obtained by a set of backbone neural network layers, for each client there exists an additional set of client-specific output layers with client-specific parameters. Each such parameter set is exclusive to the client, and its purpose is to allow for more flexible modelling of the client-specific data distribution or task. Such a multi-task setting was also used by [2, 34], from which our work differs significantly in the way we train the FL system. A taxonomy of related work is presented in Section 2.

We propose a novel approach for personalized FL, which we call Personalized Federated Learning with Exact Gradient-based Optimization (PFLEGO) [1]. Its main asset is that the updates are engineered so that the overall algorithm achieves exact stochastic gradient descent (SGD) [3, 30] minimization of the training loss function, which is the *exact equivalent* of training with all data concentrated in one place. Our starting point is the FedPer neural network (NN) architecture for personalization [2] (see also [34]), whereby the NN has two types of layers: the first ones are common layers, while the few last ones are the client-specific ones used for personalization.

At each optimization round, our algorithm performs the following steps: (a) The server sends to a randomly selected subset of clients the updated weights corresponding to common layers; (b) Given the common weights, each client performs a number of local gradient descent updates of its client-specific weights on its own local dataset, towards optimizing its loss function, without updating the common weights; (c) Contrary to other methods that

Proc. of the Adaptive and Learning Agents Workshop (ALA 2023), Cruz, Hayes, Wang, Yates (eds.), May 29-30, 2023, London, UK, https://alaworkshop2023.github.io/. 2023.

[1]We make our source code available at https://github.com/sotirisnik/PFLEGO.

we compare against i.e. FedPer, FedAvg, in our approach, at its *final* optimization step, each client computes the joint gradient over both the client-specific and the common weights and sends to the server the *gradient* of the common weights, rather than the raw weights themselves; (d) Finally the server aggregates the gradients of common weights. The process continues until convergence. The sequence of the last two steps (c),(d) turns out to be the precise equivalent of an SGD update over the full set of parameters that is effectively performed by the clients and the server in a distributed manner.

We validate the superiority of our method over FedAvg, FedPer and other state-of-the-art methods on benchmark datasets in multi-class classification such as MNIST, CIFAR-10, Fashion-MNIST, EMNIST and Omniglot. The experimental study shows that our algorithm leads to lower training loss, and thus much more effective learning especially in cases where personalization is needed most, i.e. when the data distribution differ most. Further, our algorithm has much lower, $O(1)$ computational complexity per round compared to the $O(\tau)$ one of baselines, leading to less energy consumption. Finally, the convergence of the proposed method is ensured through a rigorous proof in non-convex settings e.g. neural networks.

2 RELATED WORK

Personalized FL by fine tuning a global model. One approach to deal with personalization in FL is to allow clients to fine tune the shared global model using local adaptation [36, 41], or techniques inspired by meta-learning [5, 8, 9, 14]. These methods differ significantly from ours since they require to communicate the full set of parameters of a shared global model (i.e. there is no client-specific model part) between the server and the clients. Note also that fine tuning can be expensive since it requires the individual clients to adapt the full parameter vector of a deep NN.

In other works [19–21], the local objective of the clients incorporates a regularization term to keep the local model close to the optimal global model. These methods differs from ours since our approach does not incorporate any regularization term. FedDane [20] has an additional computational step that requires to communicate with two different subset of clients. In Ditto [19] each client has two separate model parameters of the same dimension of the global model. In the first model, the client copies the global parameters from the server and executes a number of optimization steps to the received parameters. Then the client performs a number of optimization step to the second model parameters and uses a regularization term to keep the second model close to the first model.

Personalized FL by feature transfer. Our approach mostly relates to methods that achieve personalization in FL by using a multi-task of feature transfer model, similarly to traditional non-distributed multi-task architectures [4, 31]. This aims to learn a NN whose parameters consist of a common part (also called global or shared parameters) for all clients, and a client-specific part (also called personalized parameters). This approach was followed by [2]. However, the optimization algorithm in [2] differs from our method, as it is based on the standard FedAvg scheme. Specifically, the clients update both the global and the personalized parameters by executing joint gradient descent steps. Then, each client sends back to the server the locally updated global parameters and the server updates the global parameters through averaging. Their method reduces to standard FedAvg when there are no client-specific parameters. In contrast, in our method the clients return gradients over the global parameters, in a way that our FL algorithm performs exact SGD steps.

In another related work FedRecon [34], similarly to FedPer [2], a set of global and client-specific parameters are learned, but at each round their optimization algorithm does not update simultaneously the global and personalized parameters, and therefore their method can be considered as performing block coordinate optimization. In contrast, our distributed optimization algorithm incorporates *exact* SGD steps where the full set of model parameters are simultaneously updated. This latter property means that our method enjoys theoretical convergence guarantees similar to SGD schemes [3, 30].

Another work FedBaBu [26] and FedRep [7], similarly to FedPer, decouples the model parameters to a set of global and client-specific parameters. However at FedBabu the client-specific parameters have an orthogonal weight initialization at the server side, and the client-specific weights are never trained. FedBabu differs from our method, since the client-specific parameters are initialized at the server-side, and they remain fixed across all rounds. The orthogonal initialization is essential to achieve desirable personalized performance. At FedRep each client performs a number of updates steps to their client-specific parameters, and then the clients proceeds to perform a number of updates to their global parameters. While our method uses a weighted unbiased gradient scheme on the global parameters of the clients, FedRep employs a weighted averaging scheme on the global parameters. Another difference between our method and FedRep is that the final gradient update for client-specific parameters in our method is unbiased. Our distributed unbiased scheme produces estimates that are on average equal to the true value of the parameter being estimated, without over or underestimating the true value, therefore, we expect our method to outperform FedRep.

FL with gradient return. Finally, there exist non-personalized FL algorithms optimizing a set of global parameters, where gradients are returned to the server [29, 38]. In [38], each client performs many training steps over local copies of the global parameters and returns to the server the final gradient (after the final iteration) of these parameters, which then aggregates them by performing a gradient step. However, convergence is guaranteed only in the special case that the client performs a single iteration so that their algorithm reduces to SGD optimization. The work by [29] tries to accelerate training by optimizing batch size. Specifically, clients sample a batch over their private datasets and perform a single gradient step to update their local copies of global parameters, and then they return the gradient to the server. This work differs from our method, since it does not deal with personalized settings. In our scheme, the clients may perform multiple gradient steps over the client-specific parameters before returning the gradient of global parameters to the server.

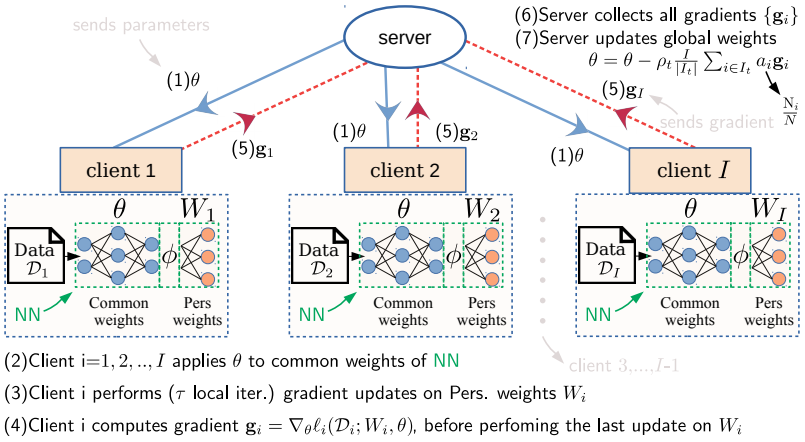

Figure 1: Training process of the PFLEGO algorithm with I clients and a single server for one communication round. The numbers in the arcs display the sequence of the execution steps.

3 PROPOSED FRAMEWORK

3.1 Personalized FL Setting

We consider a supervised FL setting in which there is a server and I clients. Each client has a locally stored dataset $\mathcal{D}_i = (X_i, Y_i)$, where $X_i = \{x_{i,j}\}_{j=1}^{N_i}$ are the input data samples (e.g. images) and $Y_i = \{y_{i,j}\}_{j=1}^{N_i}$ are the corresponding target outputs (e.g. class labels). The objective of FL is to optimize a shared or global model, together with personalized or client-specific parameters, by utilising all client datasets. As a shared backbone model, we assume a deep neural network that consists of a number of common layers with overall parameter vector θ. The number of outputs of the common layers, i.e. the size of the *feature vector*, is M. More precisely, the shared model receives an input x and constructs in its final output a representation or feature vector $\phi(x;\theta) \in \mathbb{R}^M$. Each client has a copy of the same network architecture corresponding to this shared representation. Each client i also has an additional set of personalized layers attached as a head to $\phi(x;\theta)$. For simplicity, we assume that personalized layers consist of a single linear output layer with weights W_i; see Figure 1 for a pictorial description.

Training Objective for Classification. The learning objective is to train the neural network model by adapting the global parameters θ and the personalized parameters $\{W_i\}_{i=1}^I$. The full set of parameters is denoted by $\psi = \{\theta, W_1, \ldots, W_I\}$. Learning requires the minimization of the following training loss function that aggregates all client datasets:

$$\mathcal{L}(\psi) = \sum_{i=1}^{I} \alpha_i \ell_i(W_i, \theta), \tag{1}$$

where scalar $\alpha_i = \frac{N_i}{\sum_{j=1} N_j}$ quantifies the data proportionality of different clients, and $\ell_i(W_i, \theta)$ is the client-specific loss,

$$\ell_i(W_i, \theta) = \frac{1}{N_i} \sum_{j=1}^{N_i} \ell(y_{i,j}, x_{i,j}; W_i, \theta). \tag{2}$$

The form of each data-individual loss $\ell(y_{i,j}, x_{i,j}; W_i, \theta)$ depends on the application, e.g. on whether the task is a regression or a classification one. The case of multi-class classification, that we consider in our experiments, is detailed below while other cases can be dealt with similarly.

Multi-class classification. A standard task that each client may need to solve is multi-class classification, where the input x is assigned a class label $y \in \{1, \ldots, K_i\}$. In our personalized setting, each client i can tackle a separate classification problem with K_i classes, where classes across clients can be mutually exclusive or partially overlap. The set of personalized weights W_i becomes a $K_i \times M$ matrix that allows to compute the logits in the standard cross entropy loss, $\ell(y_{i,j}, x_{i,j}; W_i, \theta) = -\log\left\{Pr(y_{i,j}|x_{i,j}; W_i, \theta)\right\}$. The class probability is modeled by the Softmax function, i.e. $\Pr(y_{i,j}|x_{i,j}; W_i, \theta) = \frac{e^{\alpha_{y_{i,j}}}}{\sum_{k=1}^{K_i} e^{\alpha_k}}$ where the K_i-dimensional vector of logits is $\alpha = W_i \times \phi(x_{i,j}; \theta)$ and $\phi(x_{i,j}; \theta)$ is a $M \times 1$ vector.

3.2 The Proposed Algorithm

The objective of personalized FL is to minimize the global training loss in (1) over the full set of parameters ψ. To this end, we propose a distributed optimization algorithm that incorporates exact Stochastic Gradient Descent (SGD) steps over ψ. The stochasticity arises due to a random *client participation* or selection process defined in the sequel, in Section 3.2.1. Without stochasticity, i.e. if all clients participate at every round, these previous steps become exact gradient descent (GD) steps. This means that the proposed algorithm (detailed in Section 3.2.2) converges similarly to standard GD or SGD methods respectively; see subsection 3.3, and subsection 3.4.

3.2.1 Client Participation Process . We assume that the optimization of the global loss in (1) is performed in different *rounds*, where each round involves a communication of the server with some of the clients. Specifically, at the beginning of each optimization round t, a subset of clients $\mathcal{I}_t \subset \{1, \ldots, I\}$ is selected uniformly at random to participate. For instance, two sensible options are: (a) the number of clients $r_t := |\mathcal{I}_t|$ follows a Binomial distribution $\mathrm{B}(I, \rho)$, i.e. each client participates independently with probability ρ, or (b) a fixed

number $0 < r \leq I$ of clients are always selected, i.e. $|\mathcal{I}_t| = r$ for any t. For both cases an arbitrary client $i \in \{1, \ldots, I\}$ participates in each round with probability $\Pr(i \in \mathcal{I}_t) = \frac{r}{I}$, where for case (a) $r = I\rho$ can be a float number, while for (b), r is strictly an integer.

3.2.2 Client and Server Updates .

Client side. Having selected the subset of clients $\mathcal{I}_t$ to participate in round t, the server sends the global model parameters θ to these clients. Then, each client $i \in \mathcal{I}_t$ trains locally the task specific parameters W_i by performing a total number of τ gradient descent steps. For the first $\tau - 1$ steps, the global parameter θ is "ignored" (i.e. it remains fixed to the value sent by the server), and only the gradient $\nabla_{W_i} \ell_i(W_i, \theta)$ over the client-specific parameters W_i is computed. The gradient steps of these $\tau - 1$ steps have the form

$$W_i \leftarrow W_i - \beta \nabla_{W_i} \ell_i(W_i, \theta),$$

where β is the learning rate. In fact, these $\tau - 1$ updates could be replaced by any other optimization procedure, as long as the final loss value $\ell_i(W_i, \theta)$, i.e. after these $\tau - 1$ steps, is smaller or equal to the corresponding initial value. In contrast, for the final (τ-th) iteration, the client simultaneously computes the joint gradient $(\nabla_{W_i} \ell_i, \nabla_\theta \ell_i)$ of both W_i and the shared parameters θ, and it performs the final (τ-th) gradient step for W_i using the rule

$$W_i \leftarrow W_i - \rho_t \frac{I}{r} \nabla_{W_i} \ell_i(W_i, \theta), \tag{3}$$

where ρ_t is the learning rate at round t, and the multiplicative scalar $\frac{1}{\Pr(i \in \mathcal{I}_t)} = \frac{I}{r}$ ensures unbiasedness of the full gradient update over all parameters ψ; For more details about that important property of our proposed algorithm, please see subsection 3.3.

Server side. The server gathers all gradients from the participating clients, and then it performs a gradient update to the parameters θ by taking into account also the data proportionality weight of each client. Specifically, the update it performs takes the form

$$\theta \leftarrow \theta - \rho_t \frac{I}{r} \sum_{i \in \mathcal{I}_t} \alpha_i \nabla_\theta \ell_i(W_i, \theta), \tag{4}$$

where again the term $\frac{I}{r}$ is included to ensure unbiasedness as detailed next. The whole optimization procedure across rounds is described by Algorithm 1.

Algorithm 1 PFLEGO

Input: T rounds, τ local gradient updates, I clients, $r \ll I$ (average) sampled clients per round, N_i data samples at the i-th client
Server:
  Initialize global parameters θ_1
  for round $t = 1, 2, ..., T$ **do**
    $\mathcal{I}_t \leftarrow$ (Select a random subset of clients)
    Receive $\mathbf{g}_i$ from each `client i` $\in I_t$
    Aggregate: $\theta_{t+1} \leftarrow \theta_t - \rho_t \frac{I}{r} \sum_{i \in I_t} a_i \mathbf{g}_i$
  end for

ClientI_Update(θ_t):#runs on client i
  Initialize W_i, the first time client i is visited
  for local gradient update = $1, 2, ..., \tau - 1$ **do**
    $W_i \leftarrow W_i - \beta \nabla_{W_i} \ell_i(W_i, \theta_t)$
  end for
  Compute joint grad $(\nabla_{W_i} \ell_i(W_i, \theta_t), \nabla_{\theta_t} \ell_i(W_i, \theta_t))$
  $W_i \leftarrow W_i - \rho_t \frac{I}{r} \nabla_{W_i} \ell_i(W_i, \theta_t)$
  Return $\mathbf{g}_i := \nabla_{\theta_t} \ell_i(W_i, \theta_t)$ to server

3.3 Exact Stochastic Gradient Descent Optimization Convergence

An important property of our proposed algorithm is that the final iteration τ over W_i's at the selected set of clients $\mathcal{I}_t$, combined with the update over the global parameter θ at the server results in an unbiased SGD step over all parameters $\psi = \{\theta, W_1, \ldots, W_I\}$. To prove this rigorously, we introduce the stochastic gradient vector $\nabla^s_\psi \mathcal{L} = \{\nabla^s_\theta \mathcal{L}, \nabla^s_{W_1} \mathcal{L}, \ldots, \nabla^s_{W_I} \mathcal{L}\}$ where we use the symbol s in ∇^s to indicate that these gradient vectors are stochastic. For any client $i = 1, \ldots, I$ the vector $\nabla^s_{W_i} \mathcal{L}$ is defined as

$$\begin{aligned} \nabla^s_{W_i} \mathcal{L} &= \mathbf{1}(i \in \mathcal{I}_t) \frac{I}{r} \nabla_{W_i} \mathcal{L}(\psi) \\ &= \mathbf{1}(i \in \mathcal{I}_t) \frac{I}{r} \alpha_i \nabla_{W_i} \ell_i(W_i, \theta). \end{aligned} \tag{5}$$

Here, $\mathbf{1}(i \in \mathcal{I}_t)$ is an indicator function that equals one if $i \in \mathcal{I}_t$, i.e. if client i was selected in round t, and zero otherwise. We also used the fact that $\nabla_{W_i} \mathcal{L}(\psi) = \alpha_i \nabla_{W_i} \ell_i(W_i, \theta)$.Note that for selected clients in set $\mathcal{I}_t$, the corresponding vector, $\nabla^s_{W_i} \mathcal{L} = \frac{I}{r} \alpha_i \nabla_{W_i} \ell_i(W_i, \theta)$ is precisely the gradient used in the client update in (3), while for the remaining clients $i \notin \mathcal{I}_t$, $\nabla^s_{W_i} \mathcal{L} = 0$. The stochastic gradient $\nabla^s_\theta \mathcal{L}$ is defined as

$$\nabla^s_\theta \mathcal{L} = \frac{I}{r} \sum_{i=1}^{I} \mathbf{1}(i \in \mathcal{I}_t) \alpha_i \nabla_\theta \ell_i(W_i, \theta). \tag{6}$$

We can see that in Algorithm 1, the final client update together with the server update can be compactly written as the following gradient update over all parameters ψ:

$$\psi \leftarrow \psi - \rho_t \nabla^s_\psi \mathcal{L}.$$

This is now a proper SGD step as long as the stochastic gradient $\nabla^s_\psi \mathcal{L}$ is unbiased, as we state next.

Proposition 1. *The stochastic gradient $\nabla^s_\psi \mathcal{L}$ is unbiased, i.e. it holds $\mathbb{E}[\nabla^s_\psi \mathcal{L}] = \nabla_\psi \mathcal{L}(\psi)$ where $\nabla_\psi \mathcal{L}(\psi)$ denotes the exact gradient and the expectation is taken under the client participation process (either case i or ii) defined in Section 3.2.1.*

Proof. By taking the expectation $\mathbb{E}[\nabla^s_{W_i}\mathcal{L}]$ for any i, and the expectation $\mathbb{E}[\nabla^s_{\theta}\mathcal{L}]$, the indicator function $\mathbf{1}(i \in \mathcal{I}_t)$ is replaced by its expected value $\Pr(i \in \mathcal{I}_t) = \frac{r}{I}$ (this value is the same for cases i and ii), which gives the exact gradient. □

As a consequence of Proposition 1, when the number of client updates is $\tau = 1$, Algorithm 1 is precisely an SGD algorithm with the standard convergence guarantees, as long as $\sum_{t=1}^{\infty} \rho_t = \infty$ and $\sum_{t=1}^{\infty} \rho_t^2 < \infty$ [3, 30]. When the selected clients at each round perform more than one gradient steps, i.e. $\tau > 1$, convergence can speed up since the first $\tau - 1$ GD steps in these clients can result in a systematic/deterministic improvement of the global loss in (1), i.e. $\mathcal{L}(\psi^{\text{after } \tau-1 \text{ steps}}) \leq \mathcal{L}(\psi^{\text{init}})$. Indeed, in practice we observe that convergence gets faster as τ increases, please see Figure 4

Since during the first $\tau - 1$ client updates the global parameters θ are fixed, we can pass the data from the NN only twice for any value of τ. At the beginning of each round, we pass once the data from the NN, store all feature vectors and then carry out $\tau - 1$ GD steps to update only the client-specific parameters. For the final τ-th iteration, we need to pass the data for a second time from the NN to compute the joint gradient. This is a great computational advantage of PFLEGO against FedAvg and FedPer, since PFLEGO roughly runs $\frac{\tau}{2}$ times faster. Concretely, given that the complexity per round is dominated by the NN evaluations, then PFLEGO is $O(1)$ while others are $O(\tau)$, consequently the clients consume much less energy per round. The minimization of energy consumption is important when the size of the model parameter vector gets large, i.e. the NN has a large number of layers.

3.4 Convergence Rate

In Proposition 2 we state the results from our detailed proof in non-convex settings [40]. We provide the convergence rate, and how to set the values for ρ, and β at the server-side and at the client-side respectively to ensure convergence guarantees. We follow the same convention as in [1, 11, 23, 40] where the average expected squared gradient norm is used to characterize the convergence rate. We show that our proposed method is able to achieve linear-speedup [13, 40], which is desired in distributed training across multiple clients [13].

Proposition 2. *The convergence rate of PLEGO is able to achieve linear-speedup [13, 40] with respect to the number of rounds T and of the clients $\mathcal{I}$. If we set $\rho = \frac{\sqrt{r}}{L\sqrt{T}}$ and $0 < \beta < \frac{2}{L}$ then the algorithm is guaranteed to converge and the convergence rate is*

$$O(\sqrt{\frac{\mathcal{I}}{T}}).$$

Proof. We provide only a summary of the proof. We assume that the clients perform a number of steps until they find the optimal client-specific parameters. Furthermore we use the following assumptions for overall loss $\mathcal{L}$.

- Lipschitz continuity of $\nabla\mathcal{L}$ w.r.t. to the ℓ_2 norm, where L denotes the Lipschitz constant.
  $||\nabla\mathcal{L}_{\theta_t} - \nabla\mathcal{L}_{\theta_{t-1}}|| \leq L||\theta_t - \theta_{t-1}||$
- The ℓ_2 norm of gradients are bounded by a constant G.
  $||\nabla_{\theta_{t-1}}\ell_i(., \theta_{t-1})|| \leq G$
- The computed gradients are unbiased gradients.
  $\mathbb{E}[\nabla_{\theta_{t-1}}\ell_i(., \theta_{t-1}) - \nabla\mathcal{L}_{\theta_{t-1}}] = 0$

Then for each optimization round $t \in [1, T]$ by the Lipschitz smoothness we have

$$\mathbb{E}[\mathcal{L}_{\theta_t}] \leq \mathbb{E}[\mathcal{L}_{\theta_{t-1}}] + \mathbb{E}[\langle \nabla\mathcal{L}_{\theta_{t-1}}, \underbrace{\theta_t - \theta_{t-1}}_{\text{stochastic}} \rangle] + \frac{L}{2} E[|| \underbrace{\theta_t - \theta_{t-1}}_{\text{stochastic}} ||^2]$$

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[||\nabla\mathcal{L}_{\theta_{t-1}}||^2] \leq \frac{1}{\rho T}(\mathcal{L}_{\theta_0} - \mathcal{L}_{\theta^*}) + \frac{L\rho G^2}{2} \frac{\mathcal{I}\sum_{i=1}^{\mathcal{I}} a_i^2 + r\sum_{i\neq j}^{\mathcal{I}} a_i a_j}{r}$$

If we choose $\rho = \frac{\sqrt{r}}{L\sqrt{T}}$, and $r = \mathcal{I}$ then we have

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[||\nabla\mathcal{L}_{\theta_{t-1}}||^2] \leq \frac{L}{\sqrt{IT}}(\mathcal{L}_{\theta_0} - \mathcal{L}_{\theta^*}) + \frac{\mathcal{I}G^2}{\sqrt{IT}}$$

$$= O(\frac{1}{\sqrt{\mathcal{I}T}} + \frac{\mathcal{I}}{\sqrt{\mathcal{I}T}}) = O(\sqrt{\frac{\mathcal{I}}{T}}).$$

□

4 EXPERIMENTS

In order to validate the performance benefits of the proposed PFLEGO algorithm, we compare it against state-of-the-art algorithms such as FedPer [2], FedAvg [25], FedRecon [34], FedRep[7], FedBabu[26], FedProx[21], and Ditto[19]. We experiment with the Omniglot, CIFAR-10, MNIST, Fashion-MNIST and EMNIST datasets.

We report *training loss* (given by (1)) and *test accuracy* for the compared methods. Training loss allows to visualize how fast each FL algorithm optimizes the model, i.e. how many rounds are required, while test accuracy quantifies predictive classification performance. Results are averages over all clients.

4.1 Dataset description

Omniglot. This dataset was introduced by [17] and consists of 1623 105x105 handwritten characters from 50 different alphabets, and 20 samples per handwritten character. Omniglot can be a natural choice for personalized learning due to its small number of samples per character and large number of different handwritten characters per alphabet. Each alphabet can be considered as a classification problem with a certain number of classes, e.g. the English alphabet has 24 classes. We use 4 convolutional layers, each layer is followed by one max pooling layer, the architecture we use is the same as the one from [10].

CIFAR-10. It consists of 32×32 RGB images of 10 different classes of visual categories [16]. We use the same architecture as in [38]. It includes two convolutional layers of 64 filters each, and a kernel of size 5. Each layer is followed by a max pooling layer of size 3×3 with stride 2. The output of convolutional layers passes through two additional fully connected layers of size 384 and 192. The activation function for the convolutional and fully connected layers is ReLU.

MNIST. It consists of 28×28 handwritten grayscale images of single digits from classes 0 to 9. For MNIST, we use an MLP architecture that consists of one fully connected layer of 200 units with a ReLU activation function.

Table 1: Test accuracy for MNIST, CIFAR-10, EMNIST and Fashion-MNIST datasets on different degrees of personalization.

	MNIST			CIFAR-10		
Method / Deg of Pers.	High-Pers	Medium-Pers	No-Pers	High-Pers	Medium-Pers	No-Pers
FedPer	97.88 ± 0.25	92.83 ± 0.46	87.23 ± 0.33	85.15 ± 1.08	61.01 ± 0.84	37.88 ± 0.59
FedAvg	97.54 ± 0.25	92.83 ± 0.52	**93.12 ± 0.24**	85.18 ± 0.96	64.53 ± 0.75	**61.33 ± 0.52**
PFLEGO	**98.43 ± 0.21**	**94.22 ± 0.43**	90.68 ± 0.17	**87.81 ± 0.94**	**74.83 ± 0.66**	58.98 ± 0.81
	EMNIST			Fashion-MNIST		
Method / Deg of Pers.	High-Pers	Medium-Pers	No-Pers	High-Pers	Medium-Pers	No-Pers
FedPer	97.78 ± 0.51	74.19 ± 0.36	48.12 ± 0.23	96.14 ± 0.35	88.22 ± 0.64	77.44 ± 0.59
FedAvg	97.29 ± 0.54	68.82 ± 0.29	**69.4 ± 0.10**	96.35 ± 0.47	87.51 ± 0.73	**83.59 ± 0.35**
PFLEGO	**98.49 ± 0.43**	**74.43 ± 0.40**	55.42 ± 0.19	**96.34 ± 0.43**	**89.84 ± 0.52**	81.49 ± 0.51

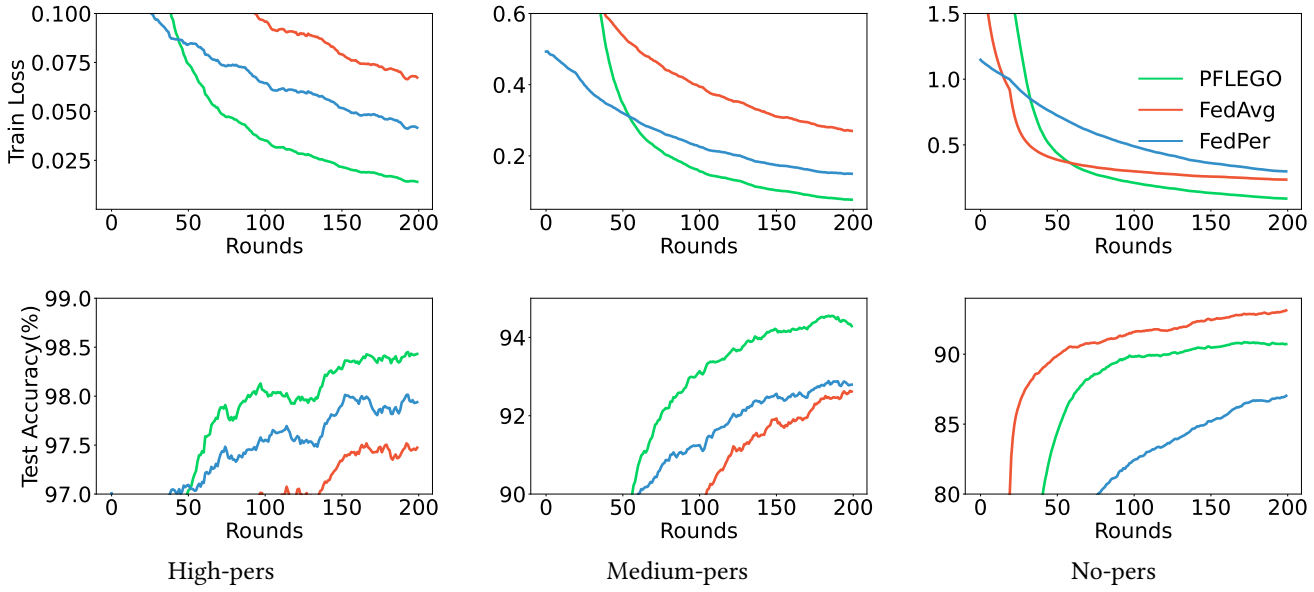

Figure 2: Training loss (top row) and test accuracy (bottom row) for MNIST dataset for PFLEGO, FedAvg and FedPer over $T = 200$ rounds, $I = 100$ clients, $r = 20\%$ client participation per round. Each column corresponds to a degree of personalization (high, medium, no personalization).

Table 2: Test accuracy for Omniglot dataset on high degree of personalization. The average is measured by averaging the last 10 global rounds of the first 1000 global rounds.

FedRep	FedBabu	FedProx	Ditto	FedPer	FedAvg	FedRecon	PFLEGO
69.65 ± 2.47	68.45 ± 1.53	48.41 ± 1.83	50.88 ± 1.72	68.02 ± 1.74	49.65 ± 1.74	74.06 ± 1.19	**75.85 ± 1.21**

Fashion-MNIST. It is similar to MNIST but more challenging and consists of 28×28 grayscale images of 10 different classes of clothing. We use the same MLP architecture as the one in MNIST.

EMNIST. It extends the MNIST dataset with grayscale handwritten digits. In total there are 62 different classes of handwritten letters and digits. We use the same MLP architecture as the one in MNIST.

4.2 Experimental Setup

For Omniglot, we assume that a single alphabet is stored in each client. Due to the fact that each handwritten system of each alphabet is unique, there is no class label set overlap among the clients, which makes Omniglot the hardest and highly personalized FL problem in our experiments. At each alphabet, data of each class are split into 75% for training and 25% for testing. Also standard data augmentation is used by including rotated image samples of multiples of 90° [10]. The setup we follow is based on [32] that uses $I = 50$ clients, $\tau = 50$ inner steps per client, and $T = 5,000$ communication rounds, and each client has an unique alphabet. For the FedAvg algorithm, the final layer of the common weights is set

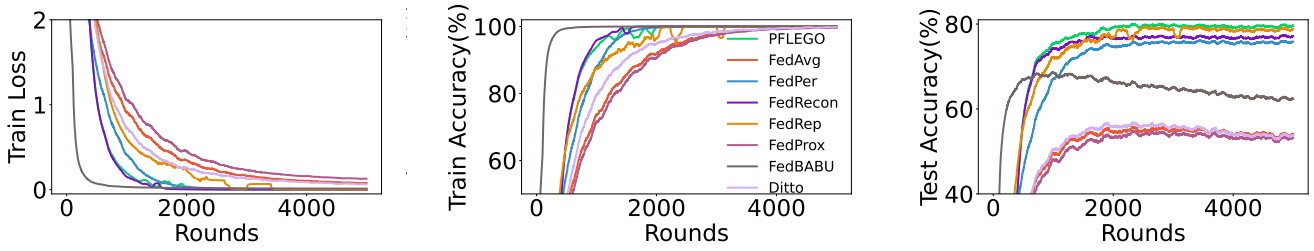

Figure 3: Training loss (left), training accuracy (middle) and test accuracy (right) for the Omniglot dataset. All FL methods were run for $5,000$ rounds, with 50 inner client steps and $r = 20\%$ client participation per round.

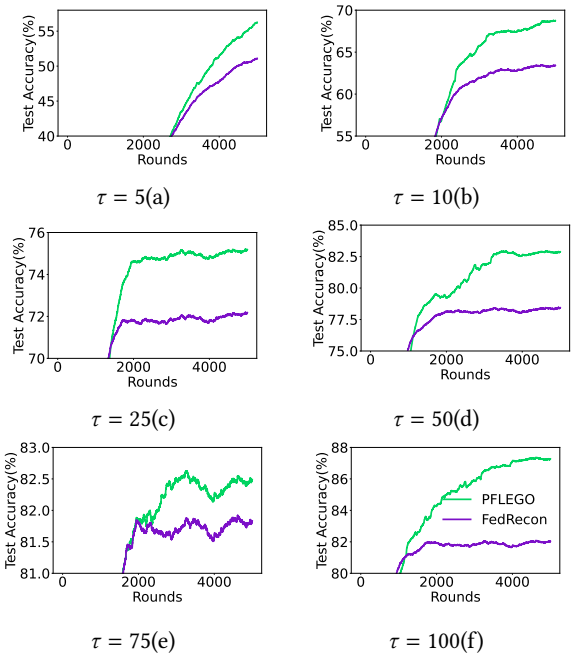

Figure 4: PFLEGO vs the block coordinate descent FedRecon algorithm. Settings: 50 clients, 5000 rounds, r ∼ [4,28], $\rho = 0.001, \beta = 0.007$ and Inner Steps τ: 5 (a), 10 (b) 25 (c), 50 (d), 75 (e), and 100 (f) for the Omniglot dataset.

to 55 outputs, which is equal to the maximum number of classes among all 50 alphabets.

For MNIST, Fashion-MNIST, EMNIST and CIFAR-10, we use $I = 100$ clients, $\tau = 50$ inner steps per client, and $T = 200$ communication rounds. MNIST, Fashion-MNIST and CIFAR-10 are well-balanced datasets and contain 10 classes. EMNIST is more challenging since it has 62 classes and varying number of examples per class. We simulate several FL scenarios by varying the amount of task-personalization among clients. This involves varying the degree of class label set overlap among clients, so that different clients can have different classes in their private datasets.

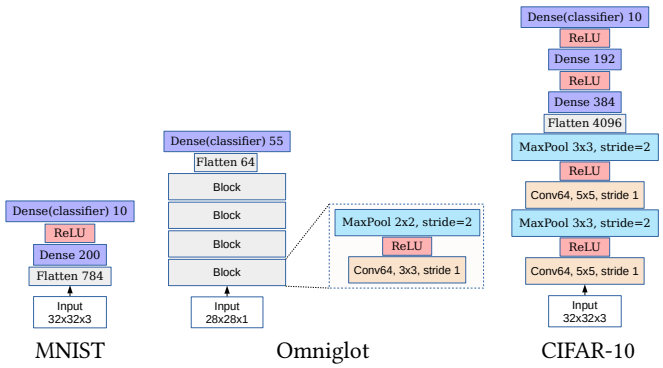

Figure 5: Model architectures. Left: MLP architecture for MNIST, Fashion-MNIST and EMNIST; Middle: 4-convolutional layer architecture for Omniglot; Right: 2-convolutional layer architecture for CIFAR-10.

4.3 Degree of personalization

For all datasets except Omniglot which is personalized by design, i.e. for MNIST, CIFAR-10, Fashion-Mnist and EMNIST, we artificially simulate personalized FL problems by varying the *degree of personalization*. This is quantified by the size K of classes randomly assigned to each client from the total set of classes, so that the smaller the K is, the higher the degree of personalization is, since the probability of class overlap among clients reduces with smaller K values.

We consider three degrees of personalization. (*i*) "high-pers" where each client has $K = 2$ randomly chosen classes from the total set of C classes ($C = 10$ for MNIST, CIFAR-10, Fashion-MNIST and $C = 62$ for EMNIST), (*ii*) "medium-pers" where $C/2$ classes are randomly assigned to each client, and (*iii*) "no-pers" where all clients have data points from all C classes, i.e. all clients solve the same task.

4.4 Results and discussion

Table 1 reports test accuracy scores for FedPer, FedAvg and our approach PFLEGO for all datasets, except Omniglot, and degrees of personalization (high/medium/no personalization). The degree of personalization is quantified by the size of classes that are randomly assigned to each client from the total set of classes, so that the

smaller the size is, the higher the degree of personalization is, since the probability of class overlap among clients reduces when the clients have fewer classes. Table 2 reports the test accuracy for Omniglot. Each reported accuracy value and confidence interval is the mean of the corresponding values at the final 10 global rounds. We observe that our algorithm has significantly higher accuracy from the other baselines for high degree of personalization; see High-Pers columns in Table 1 and Omniglot results in Table 2. In the case of Medium-Pers, our algorithm has better performance than the baselines in all datasets as well. Note the significant difference in performance in CIFAR-10, where our method has achieved 10% higher accuracy than the baselines. Finally, as expected, in the case of low personalization, FedAvg outperforms all methods.

To visualize optimization and learning speed, Figure 2 shows training loss and test accuracy with respect to the number of rounds for MNIST and for different degrees of personalization.Figure 3 plots the same quantities for the Omniglot dataset. These plots clearly indicate that PFLEGO achieves high classification accuracy faster in the high-personalized regime and it requires fewer communication rounds to minimize the overall training loss.

Further, for the most challenging Omniglot benchmark dataset, we also compare our approach to other recently proposed personalized algorithms: (i) FedRep and FedBabu[26] which have the FedPer[2] NN architecture, (ii) personalized variants of FedAvg: one that uses a regularization term such as FedProx[21], and Ditto[19] which trains 2 NN per client, and (iii) FedRecon [34], which follows a stochastic block coordinate descent scheme, as opposed to our SGD approach.

In Table 2, we include the test accuracy of block coordinate descent algorithm FedRecon, from which we observe that our approach outperforms this baseline as well. We perform an additional ablation study between our method PFLEGO and the block coordinate descent algorithm FedRecon [34] at the Omniglot dataset. At the client side we test various values on the number of the inner steps τ. At the server side, at the beginning of each round the server randomly selects at least 4 participants up to 28 participants. We observe that in each case PFLEGO has higher test accuracy than FedRecon regardless of the number of inner steps τ. In Figure 3, we observe that FedBabu[26] minimizes the training loss faster than PFLEGO, but PFLEGO achieves higher accuracy. That is because the client-specific parameters in FedBabu have an orthogonal weight initialization at the server side, and the client-specific weights are never trained.

5 ADDITIONAL EXPERIMENTS

5.1 Effect of participation rate r

Effect of participation rate r. We examine the effect of *clients participation rate* r in each optimization round, i.e. the average percentage r of clients participating in each round where a single server update is performed over θ. We consider $r \in \{40, 100\}$. We use the CIFAR-10 dataset, with $T = 200$ rounds and $\tau = 50$ inner client GD steps per round. The client learning rate for all algorithms is $\beta = 0.001$ while for PFLEGO the learning rate of the server is $\rho = 0.001$. From Figure 6 we observe that for FedAvg and FedPer, the convergence speed does not vary with r, while for PFLEGO the optimization algorithm converges faster as r increases. The more we increase the number of participation, the more gradients will be sent back to the server for aggregation. Then the server can perform an SGD step to the common weights that minimizes the total loss. (1).

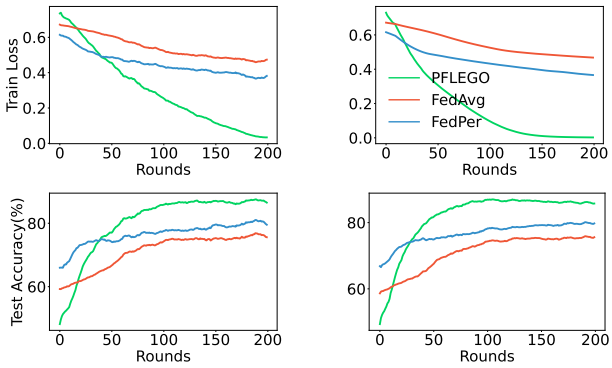

Figure 6: Ablation study of client participation using the CIFAR-10 for the High-pers case. PFLEGO is compared against FedAvg and FedPer in terms of the ability to minimize the train loss(top row) across rounds. The panel in each column corresponds to a different participation percentage r. Test accuracy is is given in the bottom row.

6 CONCLUSION

We propose PFLEGO, a new algorithm for personalized FL with the unique, and novel to the best of our knowledge, property that it achieves exact SGD minimization of the total training loss. We use a NN architecture that includes shared and client-specific layers. We show rigorously that PFLEGO performs exact SGD-based optimization, where the stochasticity arises due to randomness in client participation, and therefore it achieves faster convergence and lower training loss than state-of-the-art alternatives such as FedAvg, FedPer and others. Importantly, PFLEGO's advantage arises in regimes where a high degree of personalization is needed.

In this work, the degree of personalization i.e. the overlap between the subsets of classes available to each client was assumed to be known a priori. If this were not known, a learning algorithm would need to be devised, which gradually learns the degree of personalization needed e.g through estimating similarity of tasks of different clients, so as to decide whether a personalization algorithm (e.g. PFLEGO) would be needed or not. In this work, we aimed at optimizing total training loss over clients. Another issue would be to elaborate on fairness aspects by ensuring similar loss across clients, through a carefully selected objective function and subsequent decentralized FL process.

ACKNOWLEDGMENTS

This work was supported by the CHIST-ERA grant CHIST-ERA-18-SDCDN-004 (project LeadingEdge, grant number T11EPA4- 00056) through the General Secretariat for Research and Innovation (GSRI).

REFERENCES

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/6c340f25839e6acdc73414517203f5f0-Paper.pdf

[2] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. 2019. Federated Learning with Personalization Layers. arXiv:1912.00818 [cs.LG]

[3] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 177–186.

[4] Rich Caruana. 1997. Multitask Learning. *Mach. Learn.* 28, 1 (July 1997), 41–75.

[5] Fei Chen, Mi Luo, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. 2019. Federated Meta-Learning with Fast Convergence and Efficient Communication. arXiv:1802.07876 [cs.LG]

[6] Yiqiang Chen, Jindong Wang, Chaohui Yu, Wen Gao, and Xin Qin. 2021. FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare. arXiv:1907.09173 [cs.LG]

[7] Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. 2021. Exploiting Shared Representations for Personalized Federated Learning. arXiv:2102.07078 [cs.LG]

[8] Canh T. Dinh, Nguyen H. Tran, and Tuan Dung Nguyen. 2022. Personalized Federated Learning with Moreau Envelopes. arXiv:2006.08848 [cs.LG]

[9] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. 2020. Personalized Federated Learning: A Meta-Learning Approach. arXiv:2002.07948 [cs.LG]

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. arXiv:1703.03400 [cs.LG]

[11] Saeed Ghadimi and Guanghui Lan. 2013. Stochastic First- and Zeroth-order Methods for Nonconvex Stochastic Programming. https://doi.org/10.48550/ARXIV.1309.5549

[12] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2019. Federated Learning for Mobile Keyboard Prediction. arXiv:1811.03604 [cs.CL]

[13] Peng Jiang and Gagan Agrawal. 2018. A Linear Speedup Analysis of Distributed Deep Learning with Sparse and Quantized Communication. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/17326d10d511828f6b34fa6d751739e2-Paper.pdf

[14] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. 2019. Improving Federated Learning Personalization via Model Agnostic Meta Learning. arXiv:1909.12488 [cs.LG]

[15] Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. arXiv:1610.02527 [cs.LG]

[16] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images, Technical report.

[17] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350 (2015), 1332 – 1338.

[18] Sangsu Lee, Xi Zheng, Jie Hua, Haris Vikalo, and Christine Julien. 2021. Opportunistic Federated Learning: An Exploration of Egocentric Collaboration for Pervasive Computing Applications. arXiv:2103.13266 [cs.LG]

[19] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. 2020. Ditto: Fair and Robust Federated Learning Through Personalization.

[20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. FedDANE: A Federated Newton-Type Method. arXiv:2001.01920 [cs.LG]

[21] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. arXiv:1812.06127 [cs.LG]

[22] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2020. On the Convergence of FedAvg on Non-IID Data. arXiv:1907.02189 [stat.ML]

[23] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/f75526659f31040afeb61cb7133e4e6d-Paper.pdf

[24] Wei Liu, Li Chen, Yunfei Chen, and Wenyi Zhang. 2019. Accelerating Federated Learning via Momentum Gradient Descent. arXiv:1910.03197 [cs.LG]

[25] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629 [cs.LG]

[26] Jaehoon Oh, Sangmook Kim, and Se-Young Yun. 2021. FedBABU: Towards Enhanced Representation for Federated Image Classification.

[27] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated Learning for Emoji Prediction in a Mobile Keyboard. arXiv:1906.04329 [cs.CL]

[28] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. 2021. Adaptive Federated Optimization. arXiv:2003.00295 [cs.LG]

[29] Jinke Ren, Guanding Yu, and Guangyao Ding. 2020. Accelerating DNN Training in Wireless Federated Edge Learning Systems. arXiv:1905.09712 [cs.LG]

[30] Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* 22, 3 (1951), 400–407.

[31] Sebastian Ruder. 2017. An Overview of Multi-Task Learning in Deep Neural Networks. *ArXiv* abs/1706.05098 (2017).

[32] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. 2021. Personalized Federated Learning using Hypernetworks. *arXiv preprint arXiv:2103.04628* (2021).

[33] Sheng Shen, Tianqing Zhu, Di Wu, Wei Wang, and Wanlei Zhou. 2020. From distributed machine learning to federated learning: In the view of data privacy and security. *Concurrency and Computation: Practice and Experience* (2020).

[34] Karan Singhal, Hakim Sidahmed, Zachary Garrett, Shanshan Wu, Keith Rush, and Sushant Prakash. 2021. Federated Reconstruction: Partially Local Federated Learning. arXiv:2102.03448 [cs.LG]

[35] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. arXiv:2007.07481 [cs.LG]

[36] Kangkang Wang, Rajiv Mathews, Chloé Kiddon, Hubert Eichner, Françoise Beaufays, and Daniel Ramage. 2019. Federated Evaluation of On-device Personalization. arXiv:1910.10252 [cs.LG]

[37] Qiong Wu, Xu Chen, Zhi Zhou, and Junshan Zhang. 2020. FedHome: Cloud-Edge based Personalized Federated Learning for In-Home Health Monitoring. arXiv:2012.07450 [cs.NI]

[38] Xin Yao, Tianchi Huang, Rui-Xiao Zhang, Ruiyu Li, and Lifeng Sun. 2020. Federated Learning with Unbiased Gradient Aggregation and Controllable Meta Updating. arXiv:1910.08234 [cs.LG]

[39] Hao Yu, Rong Jin, and Sen Yang. 2019. On the Linear Speedup Analysis of Communication Efficient Momentum SGD for Distributed Non-Convex Optimization. arXiv:1905.03817 [math.OC]

[40] Hao Yu, Sen Yang, and Shenghuo Zhu. 2019. Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence* (Honolulu, Hawaii, USA) *(AAAI'19/IAAI'19/EAAI'19)*. AAAI Press, Article 698, 8 pages. https://doi.org/10.1609/aaai.v33i01.33015693

[41] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. 2021. Salvaging Federated Learning by Local Adaptation. arXiv:2002.04758 [cs.LG]