Multi-Task Federated Learning for Personalised Deep Neural Networks in Edge Computing

Jed Mills, Jia Hu, and Geyong Min

Abstract—Federated Learning (FL) is an emerging approach for collaboratively training Deep Neural Networks (DNNs) on mobile devices, without private user data leaving the devices. Previous works have shown that non-Independent and Identically Distributed (non-IID) user data harms the convergence speed of the FL algorithms. Furthermore, most existing work on FL measures global-model accuracy, but in many cases, such as user content-recommendation, improving individual User model Accuracy (UA) is the real objective. To address these issues, we propose a Multi-Task FL (MTFL) algorithm that introduces non-federated Batch-Normalization (BN) layers into the federated DNN. MTFL benefits UA and convergence speed by allowing users to train models personalised to their own data. MTFL is compatible with popular iterative FL optimisation algorithms such as Federated Averaging (FedAvg), and we show empirically that a distributed form of Adam optimisation (FedAvg-Adam) benefits convergence speed even further when used as the optimisation strategy within MTFL. Experiments using MNIST and CIFAR10 demonstrate that MTFL is able to significantly reduce the number of rounds required to reach a target UA, by up to $5\times$ when using existing FL optimisation strategies, and with a further $3\times$ improvement when using FedAvg-Adam. We compare MTFL to competing personalised FL algorithms, showing that it is able to achieve the best UA for MNIST and CIFAR10 in all considered scenarios. Finally, we evaluate MTFL with FedAvg-Adam on an edge-computing testbed, showing that its convergence and UA benefits outweigh its overhead.

Index Terms—Federated learning, multi-task learning, deep learning, edge computing, adaptive optimization

1 Introduction

Multi-access Edge Computing (MEC) [1] moves Cloud services to the network edge, enabling low-latency and real-time processing of applications via content caching and computation offloading [2], [3]. Coupled with the rapidly increasing quantity of data collected by smartphones, Internet-of-Things (IoT) devices, and social networks (SNs), MEC presents an opportunity to store and process huge quantities of data at the edge, close to their source.

Deep Neural Networks (DNNs) for Machine Learning (ML) are becoming increasingly popular for their huge range of potential applications, ease of deployment, and state-of-the-art performance. Training DNNs in supervised learning, however, can be computationally expensive and require an enormous amount of training data, especially with the trend of increasing DNN size. The use of DNNs in MEC has typically involved collecting data from mobile phones/IoT devices/SNs, performing training in the cloud, and then deploying the model at the edge. Concerns about data privacy, however, mean that users are increasingly unwilling to upload their potentially sensitive data, raising the question about how these models will be trained.

Federated Learning (FL) [4] opens new horizons for ML at the edge. In FL, participating clients collaboratively train an ML model (typically DNNs), without revealing their private data. McMahan *et al.* [5] published an initial investigation into FL with the *Federated Averaging* (FedAvg) algorithm. FedAvg works by initialising a model at a coordinating server before distributing this model to clients. These clients perform a round of training on their local datasets and push their new models to the server. The server averages these models together before sending the new aggregated model to the clients for the next round of training. We refer to the people/institutions/etc. that own data for FL as ‘users’, and to the devices that actually participate in FL as ‘clients’.

FL is a very promising approach for distributed ML in situations where data cannot be uploaded for protecting clients’ privacy. Therefore, FL is well suited for real-world scenarios such as analysing sensitive healthcare data [6], [7], next-word prediction on mobile keyboards [8], and content-recommendation [9]. However, FL presents multiple unique challenges:

- Clients usually do not have Independent and Identically Distributed (IID) training data. Each client has data generated by itself, and can have noisy data or only a subset of all features/labels. These factors can all substantially hinder training of the FL model.
- FL research typically uses the performance metric of global-model accuracy on a centralised test-set. However, in many cases, individual model accuracy on clients is the real objective - motivating ‘personalised FL’ that creates unique models for FL clients to improve local performance. However, the best way of incorporating personalisation into FL remains an under-researched topic.
- Due to the non-IID nature of client datasets, the performance of the global FL model may be higher on

- *The authors are with the Department of Computer Science, University of Exeter, EX4 4QF Exeter, U.K. E-mail: {jm729, j.hu, g.min}@exeter.ac.uk.*

Manuscript received 16 Apr. 2021; revised 6 July 2021; accepted 7 July 2021. Date of publication 21 July 2021; date of current version 11 Aug. 2021.
(Corresponding authors: Jia Hu and Geyong Min.)
Recommended for acceptance by B. Di Martino.
Digital Object Identifier no. 10.1109/TPDS.2021.3098467

some clients than others. This could even lead some clients to receive a worse model than the one they could have trained independently.

This paper addresses the above challenges by proposing a Multi-Task FL algorithm (MTFL), that allows clients to train personalised DNNs that both improve local model accuracy, and help to further enhance client privacy. MTFL has lower storage cost of personalisation, and lower computing cost compared with other personalised FL algorithms (not requiring extra steps of SGD on clients during the training loop or at personalisation time) [10], [11], [12], [13].

As client datasets in FL are usually non-IID, clients can be viewed as attempting to optimise their models during local training for disparate tasks. Our MTFL approach takes the Batch-Normalisation (BN) layers that are commonly incorporated into DNN architectures, and keeps them private to each client. Mudrarkarta *et al.* [14] previously showed that private BN layers improved Multi-Task Learning (MTL) performance for joint training on ImageNet/Places-365 in the centralised setting.

Using private BN layers has the dual benefit of personalising each model to the clients' local data as well as helping to preserve data privacy: as some parameters of client models are not uploaded to the server, less information about a client's data distribution can be gleaned from the uploaded model. Our MTFL approach using BN layers also has a storage-cost benefit compared to other personalised FL algorithms: BN layers typically contain a tiny fraction of the total parameters of a DNN, and only these BN parameters need to be stored between FL rounds, compared to entire personalised DNN models of competing algorithms [10], [12], [13].

MTFL adds personalisation on top of the typical iterative FL framework. FedAvg and other popular algorithms are instances of this iterative optimisation framework [5], [15], [16]. Most of these FL algorithms use vanilla Stochastic Gradient Descent (SGD) on clients, however, momentum-based optimisation strategies such as Adam [17] have the potential to improve convergence speed of FL training. We show that a distributed optimisation technique using Adam (FedAvg-Adam) shows substantial speedup in terms of communication rounds compared to FedAvg, and works very well within the MTFL algorithm.

Our work makes the following contributions:

- We propose an MTFL algorithm that adds Multi-Task learning on top of general iterative-FL algorithms, allowing users to learn DNN models that are personalised for their own data. MTFL uses private Batch Normalisation (BN) layers to achieve this personalisation, which provides an added privacy benefit.
- We propose a new metric for measuring the performance of FL algorithms: User model Accuracy (UA). UA better reflects a common objective of FL (increasing test accuracy on clients), as opposed to the standard global-model accuracy.
- We analyse the impact that private BN layers have on the activations of MTFL models during inference, providing insights into the source of their impact. We also analyse the training and testing performance of MTFL when keeping either the trained parameters or statistics of BN layers private, demonstrating that MTFL provides a better balance between convergence and regularisation compared to FL or independent training.
- We conduct extensive simulations on the MNIST and CIFAR10 datasets. The results show that MTFL with FedAvg is able to reach a target UA in up to $5\times$ less rounds than when using only FL, with FedAvg-Adam providing a further $3\times$ improvement. Other experiments show that MTFL is able to significantly improve average UA compared to other state-of-the-art personalised FL algorithms.
- We perform experiments using an MEC-like testbed consisting of Raspberry Pi clients and a FL server. The results show that MTFL with FedAvg-Adam's overheads are outweighed by its substantial UA and convergence speed benefits.

The rest of this paper is organised as follows: Section 2 describes related work; Section 3 details the proposed MTFL algorithm, the effect that keeping private BN layers within MTFL has on training and inference, and the FedAvg-Adam optimisation strategy; Section 4 presents and discusses experiments using both simulations and an MEC-like testbed; and Section 5 concludes the paper.

2 Related Work

As this work addresses several challenges to existing FL algorithms, we overview the related work in three subtopics of FL: works considering personalisation, works dealing with practical and deployment challenges, and works aiming to improve convergence speed and global-model performance.

2.1 Personalised Federated Learning

Several authors have considered the approach of 'personalising' FL models in order to tailor model performance to non-IID user datasets.

Meta-Learning aims to train a model that is easy to finetune with few samples. Fallah *et al.* [12] proposed the Per-FedAvg algorithm based on Model Agnostic Meta-Learning (MAML), that adds a first-order adaptation term to the client loss functions, so they can be tuned to client datasets with one step. Jiang *et al.* [11] highlighted the connection between FedAvg and first-order MAML updates, and proposed a three-stage training algorithm to improve personalisation.

Other authors propose training a combination of local and global models in FL to improve personalisation. Hanzely and Richtárick [18] added a learnable parameter to allow clients to control the extent of local and global model mixing. Dinh *et al.* [10] kept a global model and a personal model for each user, performing SGD on their personal model and then updating their copy of the global model in an outer loop. Huang *et al.* [19] kept a local model on each client, and added a proximal term to client loss functions to keep these models close to a 'personalised' cloud model, for the cross-silo FL setting.

Smith *et al.* [13] proposed MOCHA, which performs Federated MTL formulates FL as a function of the model weight matrix and a relationship matrix. Their algorithm takes into account the heterogeneous hardware of clients, meaning MOCHA is not directly comparable to our MTFL scheme. Recently, Dinh *et al.* [20] generalised MOCHA and other algorithms into the FedU framework, including proposing a decentralised version.

Our work proposes a Multi-Task learning approach to achieve personalisation in FL (MTFL). We later show that our approach has substantial converge speed, personalisation performance, privacy and storage coast benefits compared to existing personalised FL algorithms.

2.2 Federated Learning in Edge Computing

FL performs distributed computing at the network edge. Some authors have considered the system design and communication costs of FL in this environment. Jiang *et al.* [21] proposed an FL system that reduces the total data clients upload by selecting model weights with the largest gradient magnitudes. They also considered implementation details such as asynchronous or round-robin client updates. Bonawitz *et al.* [22] produced a FedAvg system design, specifying clients/server roles, fault handling, and security. They also provide analytics for their deployment of this system with over 10 million clients. To address the non-IID nature of client datasets in FL, Duan *et al.* [23] proposed the Astrea framework: client datasets are augmented to help reduce local class imbalances, and mediators are introduced to the global aggregation method.

Several authors have also investigated the impacts of wirelessly connected FL clients. Yang *et al.* [24] studied different scheduling policies in a wireless FL scenario. Their analysis showed that with a low Signal-to-Interference-plus-Noise Ratio (SINR), simple FL schemes perform well, but that as SINR increases, more intelligent methods of selecting clients are needed. Ahn *et al.* [25] proposed a Hybrid Federated Distillation scheme for FL with wireless edge devices, including using over-the-air computing and compression methods. Their results showed that their scheme gave better performance in high-noise wireless scenarios.

Other authors have proposed schemes for considering the computing, networking and communication resources of FL clients in edge computing. Wang *et al.* [26] performed experiments with smartphones to argue that the computation-time (as opposed to communication-time) of FedAvg is the most significant bottleneck for real-world FL, and propose algorithms to accommodate this computational heterogeneity. Nishio and Yonetani [27] designed a system that collects information about the computing and wireless resources of clients before initiating a round of FL, reducing the real-time taken to reach a target accuracy for FedAvg.

These previous works have proposed implementations of FL systems. However, they do not consider MTL within FL, which is a main contribution of our work with the MTFL algorithm.

2.3 Federated Learning Performance

The seminal FedAvg algorithm [5] collaboratively trains a model by sending an initial model to participating clients, who each perform SGD on the model using their local data. These new models are sent to the server for averaging and a new round is begun. Some progress has been made towards improving the convergence rate of FedAvg. Leroy *et al.* [15] used Adam adaptive optimisation when updating the global model on the server. Reddi *et al.* [16] also generalised other adaptive optimisation techniques in the same style and provided convergence guarantees. Our FedAvg-Adam algorithm differs from these as clients in FedAvg-Adam perform Adam SGD (as opposed to vanilla SGD), and the Adam parameters are averaged alongside model weights at the server. Liu *et al.* [28] used momentum-SGD on clients, and aggregated the momentum values of clients on the server alongside the model weights as an alternative method of accelerating convergence.

Some works have been produced investigating FL with non-IID or poor-quality client data. Zhao *et al.* [29] proposed sharing a small amount of data between clients to decrease the differences in their data distributions and improve global model accuracy. Konstantinov and Lampert [30] evaluated which clients have poor-quality data by finding the difference between a client model's local predictions and predictions using a trusted dataset. Wang *et al.* [31] ignored *irrelevant* client updates during training by checking if each client's update aligns with the global model.

The FedAvg-Adam optimisation method presented here uses adaptive optimisation on clients, rather than SGD, which we later show converges much faster than when using FedAvg or Adam optimisation purely on the server.

3 Multi-Task Federated Learning (MTFL)

Fig. 1 shows a high-level overview of how the MTFL algorithm would operate in the edge-computing environment. More detailed descriptions of the use of BN patches in MTFL, and optimisation on clients is given in the later subsections.

The MTFL algorithm is based on the client-server framework, however, rounds are initiated by the server, as shown in Fig. 1. First, the server selects all, or a subset of all, known clients from its database and asks them to participate in the FL round (*Step 1*), and sends a *Work Request* message to them. Clients will accept a *Work Request* depending on user preferences (for example, users can set their device to only participate in FL if charging and connected to WiFi). All accepting clients then send an *Accept* message to the server (*Step 2*). The server sends the global model (and any associated optimization parameters) to all accepting clients, who augment their copy of the global model with private patches (*Step 3*). Clients then perform local training using their own data, creating a different model. Clients save the patch layers from their new model locally, and upload their non-private model parameters to the server (*Step 4*).

The server waits for clients to finish training and upload their models (*Step 5*). It can either wait for a maximum time limit, or for a given fraction of clients to upload before continuing, depending on the server preferences. After this, the server will aggregate all received models to produce a single global model (*Step 6*) which is saved on the server, before starting a new round.

MTFL therefore offloads the vast majority of computation to client devices, who perform the actual model training. It preserves users' data-privacy more strongly than

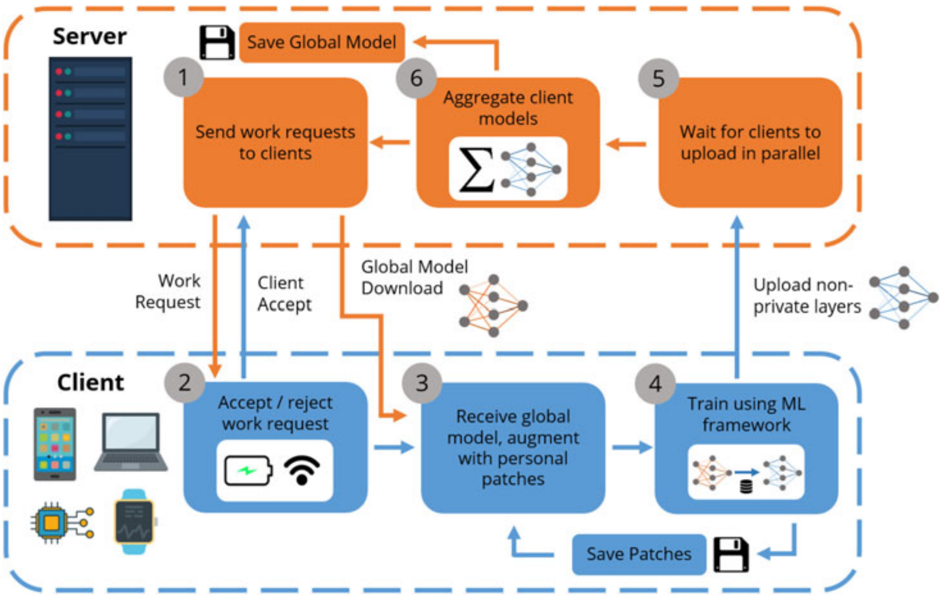

Server
Save Global Model
1
Send work requests to clients
6
Aggregate client models
5
Wait for clients to upload in parallel
Work Request
Client Accept
Global Model Download
Upload non-private layers
Client
2
Accept / reject work request
3
Receive global model, augment with personal patches
4
Train using ML framework
Save Patches

Fig. 1. Operation of the MTFL algorithm in Edge Computing. Training is performed in rounds until a termination condition is met. *Step 1*: the server selects a subset of clients from its database to participate in the round, and sends a work request to them. *Step 2*: clients reply with an accept message depending on physical state and local preferences. *Step 3*: clients download the global model (and any optimisation parameters) from the server, and update their copy of the global model with private patches (in this work, we use BN layers as patches). *Step 4*: clients perform local training, before saving their personal patches for the next round. *Step 5*: the server waits for C fraction of clients to upload their non-private model and optimiser values, or until a time limit. *Step 6*: the server averages all models, saves the aggregate, and starts a new round.

FedAvg and other personalised-FL algorithms: not only is user data not uploaded, but key parts of their local models are not uploaded. The framework also accounts for client stragglers with its round time/uploading client fraction limit. Moreover, MTFL utilises patch layers to improve *local* model performance on individual users' non-IID datasets, making MTFL more personalised.

3.1 User Model Accuracy and MTFL

In many FL works, such as the original FedAvg paper [5], the authors use a central IID test-set to measure FL performance. Depending on the FL scenario, this metric may or may not be desirable. If the intention is to create a single model that has good performance on IID data, then this method would be suitable. However, in many FL scenarios, the desire is to create a model that has good performance on individual user devices. For example, Google have used FedAvg for their GBoard next-word-prediction software [8]. The objective was to improve the prediction score for individual users. As users do not typically have non-IID data, a single global model may display good performance for some users, and worse performance for others.

We propose using the average User model Accuracy (UA) as an alternative metric of FL performance. UA is the accuracy on a client using a local test-set. This test-set for each client should be drawn from a similar distribution as its training data. In this paper, we perform experiments on classification problems, but UA could be altered for different metrics (e.g., error, recall).

In FL, user data is often non-IID, so users could be considered as having different but related learning tasks. It is possible for an FL scheme to achieve good global-model accuracy, but poor UA, as the aggregate model may perform poorly on some clients' datasets (especially if they have a small number of local samples, so are weighted less in the FedAvg averaging step). We propose the MTFL algorithm that allows clients to build different models, while still benefiting from FL, in order to improve the average UA. Mudrakarta *et al.* [14] have previously shown that adding small per-task 'patch' layers to DNNs improved their performance in MTL scenarios. Patches are therefore a good candidate for training personalised models for clients.

In FL, the aim is to minimise the following objective function:

$$F_{\mathrm{FL}} = \sum_{k=1}^{K} \frac{n_k}{n} \ell_k(\Omega), \tag{1}$$

where K is the total number of clients, n_k is the number of samples on client k, n is the total number of samples across all clients, ℓ_k is the loss function on client k, and Ω is the set of global model parameters. Adding unique client patches to the FL model changes the objective function of MTFL to

$$F_{\mathrm{MTFL}} = \sum_{k=1}^{K} \frac{n_k}{n}\ \ell_k(\mathcal{M}_k) \tag{2}$$

$$\mathcal{M}_k = (\Omega_1 \cdots \Omega_{i_1}, P_{k_1}, \Omega_{i_1+1} \cdots \Omega_{i_m}, P_{k_m}, \Omega_{i_m+1} \cdots \Omega_j), \tag{3}$$

where $\mathcal{M}_k$ is the patched model on client k, composed of Federated model parameters $\Omega_1 \cdots \Omega_j$ (j being the total number of Federated layers) and patch parameters $P_{k_1} \cdots P_{k_m}$ (m being the total number of local patches, $\{i\}$ being the set of indexes of the patch parameters) unique to client k. Fig. 2 shows an example composition of a DNN model used in MTFL.

MTFL is a general algorithm for incorporating MTL into FL. Different optimisation strategies (including FedAvg-

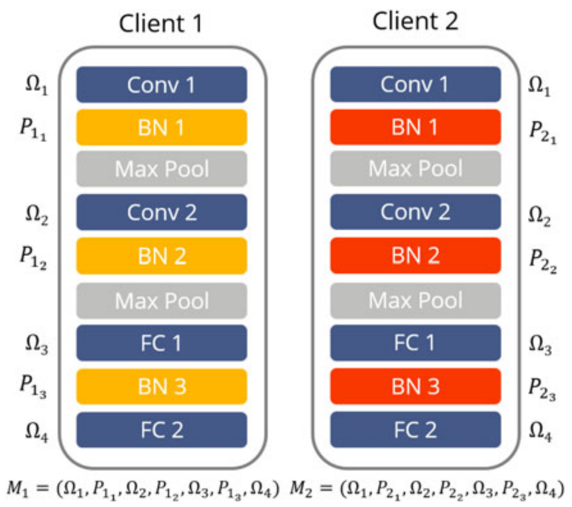


Fig. 2. Example composition of a DNN model used in MTFL. Each client's model consists of shared global parameters ($\Omega_1 - \Omega_4$) for Convolutional (Conv) and Fully-Connected (FC) layers, and private Batch-Normalization (BN) patch layers ($P_{k_1}, P_{k_2}, P_{k_3}$).

Adam described in Section 3.3) can be used within MTFL, and we later show that MTFL can substantially reduce the number of rounds to reach target UA, regardless of the optimisation strategy used.

Algorithm 1. MTFL

```
1:  Initialise global model $\Omega$ and global optimiser values $V$
2:  **while** termination criteria not met **do**
3:    Select round clients, $S_r \subset S, |S_r| = C \cdot |S|$
4:    **for each** client $s_k \in S_r$ in parallel **do**
5:      Download global parameters $\mathcal{M}_k \leftarrow \Omega$
6:      Download optimiser values $V_k \leftarrow V$
7:      **for** $i \in$ patchIdxs **do**          ▷ Apply local patches
8:        $\mathcal{M}_{k,i} \leftarrow P_{k,i}, V_{k,i} \leftarrow W_{k,i}$
9:      **end for**
10:     **for** batch $b$ drawn from local data $D_k$ **do**
11:       $\mathcal{M}_k, V_k \leftarrow$ LocalUpdate($\mathcal{M}_k, V_k, b$)
12:     **end for**
13:     **for** $i \in$ patchIdxs **do**          ▷ Save local patches
14:       $P_{k,i} \leftarrow \mathcal{M}_{k,i}, W_{k,i} \leftarrow V_{k,i}$
15:     **end for**
16:     **for each** $i \notin$ patchIdxs **do**
17:       Upload $\mathcal{M}_{k,i}, V_{k,i}$ to server
18:     **end for**
19:   **end for**
20:   **for** $i \notin$ nonPatchIndexes **do**
21:     $\Omega_i \leftarrow$ GlobalModelUpdate($\Omega_i, \{\mathcal{M}_{k,i}\}_{k \in S_r}$)
22:     $V_i \leftarrow$ GlobalOptimUpdate($V_i, \{V_{k,i}\}_{k \in S_r}$)
23:   **end for**
24: **end while**
```

As shown in Algorithm 1, MTFL runs rounds of communication until a given termination criteria (such as target UA) is met (Line 2). At each round, a subset S_r of clients are selected to participate from the set of all clients S (Line 3). These clients download the global model Ω, which is a tuple of model parameters, and the global optimiser V, if used (Lines 5-6). The clients then update their copy of the global model and optimiser with their private patch layers (Lines 7-9), where the 'patchIdxs' variable contains the indexes of patch layer placement in the DNN. Clients perform training using their now-personalised copy of the global model and optimiser on their local data (Line 10). Depending on the choice of FL optimisation strategy to be used within MTFL, the LocalUpdate function represents local training of the model. For FedAvg, LocalUpdate is simply minibatch-SGD. We discuss this, and the proposed FedAvg-Adam optimisation strategy, further in Section 3.3. After local training, the updated local patches are saved (Lines 11-13), and the non-patch layers and optimiser values are uploaded to the server (Lines 14-16).

At the end of the round, the server makes a new global model and optimiser according to the GlobalModelUpdate and GlobalOptimUpdate functions (Lines 18-20). These functions are again dependent on the FL optimisation strategy used, and are discussed further in Section 3.3. FedAvg, for example, uses a weighted average of client models for GlobalModelUpdate. The updated global model marks the end of the round and a new round is begun.

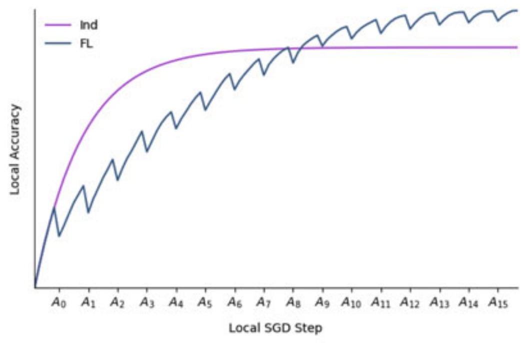
The total per-round computation complexity of MTFL scales with $|S_r|$, where $|S_r|$ is the number of clients participating per round. The computation performed by each client is independent of the total number of clients. As clients perform local computation in parallel, MTFL (like FedAvg) is eminently scalable. Scalability is important in FL as real-world deployments are expected to have huge numbers of low-powered clients [4], [8]. The global model and optimiser updates (Lines 20-23 in Algorithm 1) depend on the optimisation strategy used. For FedAvg and FedAvg-Adam, GlobalModelUpdate is essentially map-reduce (averaging after local training) - also $\mathcal{O}(|S_r|)$. For FedAdam, the Adam step following the map-reduce in GlobalOptimUpdate is not dependent on the number of clients (only on the DNN architecture).

There are numerous works investigating FL in the Peer-To-Peer (p2p) setting, which we do not consider in this paper. Simple p2p FL algorithms involve sending all client models to all participating peers for decentralised aggregation. Extension of MTFL to these schemes is trivial: peers would simply just send/aggregate the non-private layers. More sophisticated p2p FL algorithms may require more complex ways of incorporating private layers – an interesting direction we leave for future works.

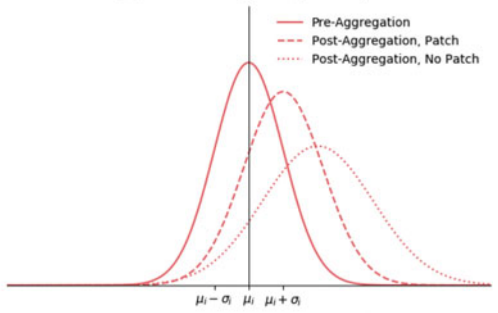
Mudrakarta *et al.* [14] showed that Batch Normalisation (BN) layers can act as model patches for MTL in the centralised setting. We show later that BN layers work well as patches in MTFL, considering that they are very lightweight in terms of number of parameters. BN layers are given by

$$\begin{aligned} \hat{x}_i &= \frac{z_i - \mathbb{E}(z_i)}{\sqrt{\mathrm{Var}(z_i) + \epsilon}} \\ \mathrm{BN}(\hat{x}_i) &= \gamma_i \hat{x}_i + \beta_i, \end{aligned} \tag{4}$$

where $\mathbb{E}(z_i)$ and $\mathrm{Var}(z_i)$ are the mean and variance of a neuron's activations (z_i, post-nonlinearity) across a minibatch, and γ_i and β_i are parameters learned during training. BN layers track a weighted moving average of $\mathbb{E}(z_i)$ and $\mathrm{Var}(z_i)$ during training: μ_i and σ_i^2, for use at inference time. In Section 4 we investigate the benefit of keeping statistics μ, σ and/or trainable parameters γ, β as part of private patch layers.



(a) Local accuracy during training



(b) Effect of patch layers on activations

Fig. 3. (a) Federated Learning (FL) results in an accuracy curve where the UA decreases after aggregations and increases during local training, compared with the smoother accuracy curve when training independently (Ind). (b) Patch BN-layers help bring the distribution in outputs for neuron i closer to the pre-aggregation distributions.

We have chosen to use BN layers for personalisation within MTFL. The reason for this choice is twofold: 1) they show excellent personalisation performance and 2) the storage cost of BN parameters is very small ($< 1\%$ of total model size for the tested model architectures). Mudrakarta *et al.* [14] also investigated the use of depthwise-convolutional patches for centralised Multi-Task learning. Any model layers could in principle be kept private during MTFL, however, there is an inherent trade-off between the number of parameters kept private and the ability of the global model to converge.

3.2 Effect of BN Patches on Inference

To understand the impact that BN-patch layers have on UA, we consider the change in internal DNN activations over a client's local test-set immediately *before* and immediately *after* the FL aggregation step.

As illustrated in Fig. 3a, UA typically drops after the aggregation step in iterative FL. This is because the model has been tuned on the local training set for several epochs, and suddenly has its model weights replaced by the Federated weights, which are unlikely to have better test performance than the pre-aggregation model. This idea is further examined in [32] and showed later in our experimental section. Consider a simple DNN consisting of dense layers followed by BN and then nonlinearities. The vector of first-layer neuron activations over the client's test-set (X) from applying weights and biases (W_0, b_0), can be modelled as a normal distribution, which BN relies on to work

$$\begin{aligned} z_i &\triangleq [W_0 X + b_0]_i \\ z_i &\sim N(\mathbb{E}[z_i], Var[z_i]). \end{aligned} \tag{5}$$

During local training, the client's model has been adapted to the local dataset, and the BN-layer statistics used for inference (μ, σ^2) have been updated from the layer activations. Assuming, after local training (and before aggregation), $\mu_i \approx \mathbb{E}[z_i]$, and $\sigma_i^2 \approx Var[z_i]$, then the BN-layer (ignoring ϵ) computes

$$\begin{aligned} \hat{x}_i &\triangleq \frac{z_i - \mu_i}{\sigma_i} \\ \hat{x}_i &\sim N(0,1) \\ \mathrm{BN}(\hat{x}_i) &\sim N(\beta_i, \gamma_i^2), \end{aligned} \tag{6}$$

where β_i, γ_i^2 are the learned BN parameters. If the client is participating in FL or MTFL, then the model parameters W_0, b_0 are updated after downloading the global model with federated values: $\overline{W}_0$, $\overline{b}_0$. The activations of the first layer are then

$$\begin{aligned} \overline{z}_i &\triangleq [\overline{W}_0 X + \overline{b}_0]_i \\ \overline{z}_i &\sim N(\mathbb{E}[\overline{z}_i], \mathrm{Var}[\overline{z}_i]). \end{aligned} \tag{7}$$

Defining the difference in mean and variance between pre- and post-aggregation activations, $\Delta\mu_i = \mathbb{E}[\overline{z}_i] - \mathbb{E}[z_i]$ and $\Delta\sigma_i^2 = \mathrm{Var}[\overline{z}_i] - \mathrm{Var}[z_i]$, the output from a BN-patch layer as part of MTFL (which maintains μ, σ, β, γ after aggregation) is

$$\begin{aligned} \hat{\overline{x}}_i &\sim N\left(\frac{\Delta\mu_i}{\sigma_i}, 1 + \frac{\Delta\sigma_i^2}{\sigma_i^2}\right) \\ \mathrm{BN}(\hat{\overline{x}}_i) &\sim N\left(\gamma\frac{\Delta\mu_i}{\sigma_i} + \beta_i, \gamma_i^2\left(1 + \frac{\Delta\sigma_i^2}{\sigma_i^2}\right)\right). \end{aligned} \tag{8}$$

If the BN layer is *not* a patch layer (i.e., the client is participating in FL, with federated BN values $\bar{\mu}$, $\bar{\sigma}$, $\overline{\beta}$, $\bar{\gamma}$), the output of the BN layer is

$$\begin{aligned} \hat{\overline{x}}_i &\sim N\left(\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i}, \frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right) \\ \overline{\mathrm{BN}}(\hat{\overline{x}}_i) &\sim N\left(\bar{\gamma}\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i} + \overline{\beta}_i, \bar{\gamma}_i^2\frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right). \end{aligned} \tag{9}$$

We posit that using BN-patch layers in MTFL constrains neuron activations to be closer to what they were before the aggregation step, compared to non-patch BN layers as part of FL (as illustrated in Fig. 3b. I.e., the difference in means and variances pre- and post-aggregation using MTFL is smaller than when using FL

$$\begin{aligned} \left|\gamma\frac{\Delta\mu_i}{\sigma_i}\right| &< \left|\beta_i - \bar{\gamma}\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i} - \overline{\beta}_i\right| \\ \left|\gamma_i^2\frac{\Delta\sigma_i^2}{\sigma_i^2}\right| &< \left|\gamma_i^2 - \bar{\gamma}_i^2\frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right|. \end{aligned} \tag{10}$$

Assuming the above inequality holds, it is easy to see how the values propagated through the network after the first layer are closer to the pre-aggregation values when using BN-patches as opposed to federated BN layers. If BN-patches are added throughout the network, the intermediate DNN values will be regularly 'constrained' to be closer to the pre-aggregation values, resulting ultimately in network outputs closer to the pre-aggregation outputs.

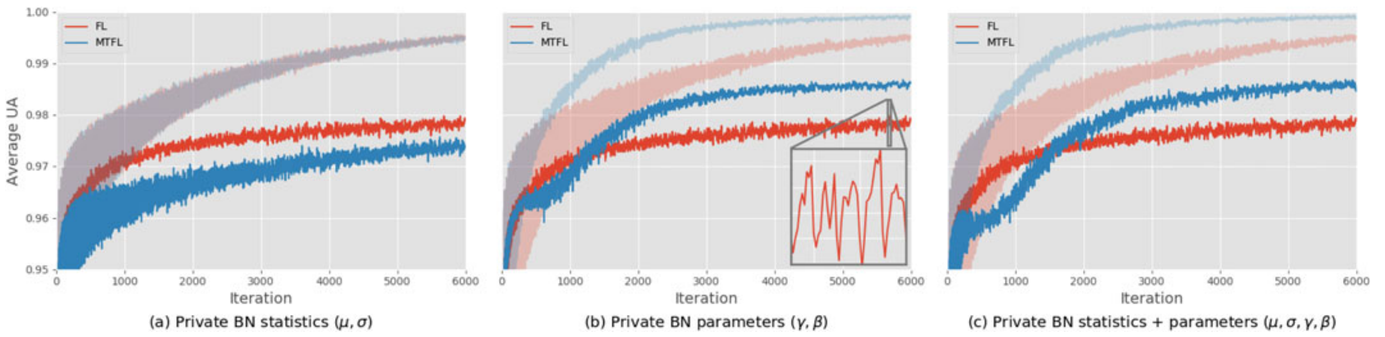


Fig. 4. Average training (faint) and testing (solid) User Accuracy (UA) curves for every step of local SGD on the MNIST, $W = 200$, $C = 1.0$ scenario, using FL(FedAvg) (red), and MTFL(FedAvg) (blue). Each plot compares keeping different values within the BN layers of MTFL private: either statistics (μ, σ) and/or trainable parameters (γ, β), to FL. All curves have been smoothed with an averaging kernel for presentation, except the inset of plot (b), which shows the cyclic drops in accuracy due to model averaging characteristic of FL.

Looking at Eq. (8), if $\Delta\mu_i$ and $\Delta\sigma_i^2$ for neuron i are large, then the output distribution of the neuron after the BN-patch layer $(\mathrm{BN}(\hat{\tilde{x}})_i)$ over the test-set will be quite different than $\mathrm{BN}(\hat{x})_i$. The BN-patch layer will therefore provide little benefit over a federated BN layer, as the left hand sides of the inequalities in Eq. (10) are unlikely to be much smaller than the right hand sides. Large differences in pre- and post-aggregated model parameters are seen during the early stages of training, when gradients are large and client models diverge more during local training. This therefore implies that MTFL has less benefit during the early stages of training, and its benefit increases during training as gradient magnitudes decrease (as shown in Fig. 4).

3.3 Federated Optimisation Within MTFL

As shown in Algorithm 1, MTFL applies private patch layers for each client, and trains them alongside the federated (non-private) layers during LocalUpdate. At the end of each round, the server aggregates the uploaded federated layers from clients (and any distributed optimiser values used), producing a new global model using the GlobalModelUpdate function. If distributed adaptive-optimisation is used, then the GlobalOptimUpdate function will also be called. Table 1 details different FL training algorithms as characterised by their implementations of these functions.

In FedAvg, LocalUpdate is simply minibatch-SGD, and GlobalModelUpdate produces the new global model as a weighted (by number of local samples) average of uploaded client models. FedAvg uses SGD with no adaptive optimisation, so the variable V in Algorithm 1 is a tuple of empty values, and GlobalOptimUpdate performs no function. For FedAdam [15], [16], clients also perform SGD during LocalUpdate. However, during GlobalModelUpdate, the server takes the difference (Δ_r) between the previous global model and the average uploaded client model. The server treats Δ_r as a 'psuedogradient', and uses a set of 1st and 2nd moment values stored on the server to update the global model using an Adam-like update step. Clients do not use distributed adaptive optimisation in FedAdam, so V is also a tuple of empty values and GlobalOptimUpdate performs no function.

We propose using adaptive optimisation (namely, Adam) as the distributed optimisation strategy. We call this strategy FedAvg-Adam. In FedAvg-Adam, clients share a global set of Adam 1st and 2nd moments, stored in the V variable in Algorithm 1. Clients store private optimiser values for their patch layers (W_k), as we find performance is better when keeping private optimiser values for patches. During LocalUpdate, clients perform Adam SGD, and the federated model layers and Adam values are uploaded by clients at the end of the round. To produce a new global model, the server averages the client models in GlobalModelUpdate and averages the Adam moments in GlobalOptimUpdate. FedAvg-Adam therefore has a $3\times$ communication cost per round compared to FedAvg or FedAdam. However, in many FL scenarios, the major concern is reducing the number of communication rounds required for the model to converge. We later show that FedAvg-Adam considerably improves the convergence speed of FL and MTFL.

For the rest of the paper, we refer to iterative FL schemes that do not keep any private model patches as FL, with the optimisation strategy in brackets, e.g., FL(FedAvg). If clients keep private model patches, we refer to the scheme as MTFL, again with the optimisation strategy in brackets, e.g., MTFL(FedAvg).

TABLE 1
LocalUpdate, GlobalModelUpdate and GlobalOptimUpdate Used by the FedAvg [5], FedAdam [15], [16] and FedAvg-Adam FL Training Strategies

Optimisation Strategy	LocalUpdate	GlobalModel Update	GlobalOptim Update
FedAvg	SGD	Average	-
FedAdam	SGD	Adam	-
FedAvg-Adam	Adam	Average	Average

All of these strategies can be used within MTFL.

4 Experiments

In this section, we first give details of the datasets, models and data partitioning scheme used for all the experiments. We then present extensive experiments analysing the impact that MTFL has on the number of rounds taken to reach a target UA. These experiments also examine which BN values, when kept private, give the best performance, and compare FL and MTFL with different optimisation strategies. After that, we investigate why different private BN values have different impacts on training, and compare

TABLE 2
Communication Rounds Required to Reach Target Average User Accuracies for Different Tasks Using FL and MTFL (With Private Statistics μ, σ and/or Trained Parameters γ, β), for Different Numbers of Total Clients W, Client Participation Rates C, and Optimisation Strategies

MNIST - 2NN																
	FL				**MTFL**											
	Private values = None				$\mu, \sigma, \gamma, \beta$				μ, σ				γ, β			
Optimisation	W = 200		400		200		400		200		400		200		400	
Strategy	C = 0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0
FedAvg	99	102	107	110	85	58	101	68	X	X	X	X	29	21	34	26
FedAdam	85	69	88	65	56	37	75	77	109	90	194	262	31	25	31	27
FedAvg-Adam	44	49	40	50	17	41	19	32	131	151	170	198	**9**	**9**	**10**	**9**
CIFAR10 - CNN																
FedAvg	139	138	171	164	49	33	55	36	231	280	258	266	37	24	45	30
FedAdam	105	90	83	80	21	14	22	16	67	45	48	38	24	14	25	16
FedAvg-Adam	57	43	36	31	11	9	14	**8**	82	79	62	63	**10**	**7**	**11**	**8**

Cases unable to reach the target UA within 500 rounds are denoted by X. Best results for each scenario (combination of W and C) given in bold.

TABLE 3
Communication Rounds Required to Reach Target Average User Accuracies (of Non-Noisy Clients) for Different Tasks Using FL and MTFL (With Private Statistics μ, σ and/or Trained Parameters γ, β), When 20 percent of Clients Have Noisy Training Data, for Different Numbers of Total Clients W, Client Participation Rates C, and Optimisation Strategies

MNIST - 2NN																
	FL				**MTFL**											
	Private values = None				$\mu, \sigma, \gamma, \beta$				μ, σ				γ, β			
Optimisation	W = 200		400		200		400		200		400		200		400	
Strategy	C = 0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0
FedAvg	276	X	290	X	115	76	144	144	85	58	102	68	50	36	65	48
FedAdam	X	X	X	X	76	47	110	89	56	37	75	77	43	33	46	53
FedAvg-Adam	133	260	191	X	20	16	24	27	17	41	19	32	**12**	**8**	**15**	**40**
CIFAR10 - CNN																
FedAvg	148	208	202	250	47	32	52	35	239	186	260	88	36	24	43	28
FedAdam	159	91	92	93	21	14	21	15	74	49	51	42	34	16	21	14
FedAvg-Adam	193	X	X	X	14	10	16	**9**	103	111	67	74	**12**	**8**	**13**	**9**

Cases unable to reach the target UA within 500 rounds are denoted by X. Best results for each scenario (combination of W and C) given in bold.

MTFL to other state-of-the-art personalised FL algorithms. Finally, we evaluate the cost in terms of computation time of MTFL(FedAvg-Adam) on an MEC-like testbed.

4.1 Datasets and Models

We conduct experiments using two image-classification datasets: MNIST [33] and CIFAR10 [34], and two DNN architectures.

MNIST: (28×28) pixel greyscale images of handwritten digits from 10 classes. The '2NN' network used on this dataset had one Fully Connected (FC) layer of 200 neurons, a BN layer, a second 200-neuron FC layer, and a softmax output layer.

CIFAR10: (32×32) pixel RGB images of objects from 10 classes. The 'CNN' network used on this dataset had one (3×3) convolutional (conv) layer with 32 filters followed by BN, ReLU and (2×2) max pooling; a second (3×3) conv ReLU layer with 64 filters, BN, ReLU and (2×2) max pooling; a 512 neuron ReLU FC layer; and a softmax output layer.

Experiments were run with different numbers of clients W, client participation rates C and optimisation strategies, on non-IID clients. To produce non-IID client data, we take the popular approach from [5]: order the training and testing data by label, split each into $2W$ shards, and assign each client two shards at random. Using the same assignment indexes for the testing data means that the classes in each client's training set are the same as those in their test set. This splitting produces a strongly non-IID distribution across clients. All results are an average over 5 trials with different random seeds.

4.2 Patch Layers in FL

Setup - First, we compare how many rounds are needed to reach a target average UA (97 percent for MNIST, 65 percent for CIFAR10) for MTFL and FL. In FL, no model parameters are kept private (i.e., there are no patches), whereas in MTFL, some model parameters are kept private. For the MTFL columns in Tables 2 and 3, we present the effects of keeping BN-layer statistics (μ, σ) and/or trainable parameters (γ, β) private, as part of the patch layers.

For these results, we fixed number of local epochs $E = 1$ and tuned the learning-rate hyperparameters for every scenario such that the target was reached in the fewest rounds. For FedAvg and FedAvg-Adam, we had to tune only one hyperparameter for each scenario, but FedAdam required training both client and server learning rate. Entries with 'X' in Tables 2 and 3 denote cases that could not reach the target within 500 communication rounds.

In Table 3, we also investigate the robustness of MTFL to clients with noisy training data. Here, 20 percent of the clients at random had 0-mean Gaussian noise added to their training data. The average UA taken for Table 3 was for the non-noisy clients only, to test how MTFL helps to mitigate the effect of noisy clients on non-noisy clients. We used Gaussian noise with standard deviation 3 for MNIST and 0.2 for CIFAR10 (MNIST is a much simpler image classification task than CIFAR10, so required more noise to significantly hinder training).

Results - Table 2 shows that for MTFL, when all BN-layer values $(\mu, \sigma, \gamma, \beta)$ are kept private, the number of rounds to reach a target average UA is substantially lower in almost all scenarios when compared to FL. For example for the CIFAR10 $W = 400$, $C = 1.0$ scenario, FL(FedAvg) took 164 rounds to reach the target average UA, whereas MTFL (FedAvg) with private $(\mu, \sigma, \gamma, \beta)$ took only 36 rounds. However, Tables 2 and 3 show that when keeping only the tracked statistics of BN patches private, UA is actually harmed. Conversely, MTFL with only private trainable parameters took even fewer rounds than MTFL will all-private $(\mu, \sigma, \gamma, \beta)$. For the same scenario, MTFL(FedAvg) with private (μ, σ) took 266 rounds, whereas MTFL(FedAvg) with private (γ, β) took just 30 rounds. We investigate the reason behind these differences further in Section 4.3.

MTFL naturally increases the variance of UAs during training, as non-identical user models in MTFL would contribute to the variance of UAs. However, in the experiments we performed, the difference between the variance of UA for FL and MTFL is very small: at less than 1 percent.

Table 3 shows that MTFL also helped to mitigate the impact of noisy clients on non-noisy clients. With FL, noisy clients prevented the average non-noisy UA from reaching the target in many scenarios. However, in most cases, MTFL allowed the non-noisy clients to reach the target average UA in a similar number of rounds than the corresponding non-noisy scenarios in Table 2. For example, for the CIFAR10 $W = 400, C = 1.0$ scenario, FL(FedAvg) took 250 rounds to reach the target, however MTFL(FedAvg) with private (γ, β) parameters, took just 28 rounds.

As Table 3 displays the rounds required for the non-noisy clients to reach the target average UA, the improvements shown when using MTFL may be due to the non-noisy clients being more 'decoupled' from the noisy ones. As they do not share all model parameters, the harmful effect of receiving a global model that has been harmed by the participation of noisy clients has been reduced, allowing them to reach higher accuracies, faster.

Tables 2 and 3 also show that in most scenarios, the FedAvg-Adam optimisation strategy reached the target average UA in the fewest rounds, regardless of whether FL or MTFL is used. Taking the same CIFAR10 scenario in Table 2, FL(FedAvg) took 164 rounds, FL(FedAdam) 80 rounds, and FL(FedAvg-Adam) only 31 rounds to reach the target. Similarly, MTFL(FedAvg) took 36 rounds, MTFL (FedAdam) 16 rounds and MTFL(FedAvg-Adam) just 8 rounds with private $(\mu, \sigma, \gamma, \beta)$.

4.3 Training and Testing Results Using MTFL

Setup - To investigate why MTFL with BN patches using private (μ, σ) and/or private (γ, β) give such different results (as shown in Tables 2 and 3), we plotted training and testing UA during one scenario from Table 2: namely MNIST with $W = 200$, $C = 1.0$ for FL(FedAvg) and MTFL(FedAvg). We ran the algorithms for 600 communication rounds, where clients performed 10 steps of local training each round, and calculated the average training and testing UA for every local step. These graphs therefore present $600 \times 10 = 6000$ total steps. Measuring in this way allowed us to present the train/test accuracy trade-off, the impact that averaging has during training, and the effect on training and testing with different private BN values.

Results - Fig. 4 shows the (smoothed) training and testing UAs of the different combinations of BN layer statistics/parameters for the MNIST problem. Note that because the lines are smoothed for presentation, the steps where the curves reach the target accuracies may not correspond to the values in Table 2. In Fig. 4a, the training curves for FL (FedAvg) and MTFL(FedAvg) with private (μ, σ) are the same: this is because the BN statistics are only used at test-time and do not influence training. The test accuracy for private (μ, σ) is lower than FedAvg, mirroring the results in Tables 2 and 3. The lower test accuracy may be due to mismatch in BN values: γ and β have been averaged, so output a different distribution than these private statistics have been tracking, thus harming the ability of the model. This seems to be supported by Fig. 4c. When keeping both private (μ, σ) and (γ, β) there is no substantial performance drop when compared to Fig. 4b, when only (γ, β) are kept private.

Figs. 4b and 4c show that keeping private (μ, σ) significantly increases the rate at which the training accuracy can improve (see faint lines in Figs. 4b and 4c). Previous authors [5] have commented that FedAvg can work as a kind of regularisation for client models. When clients have small local datasets, their training error would quickly reach near-0 as it is easy for independently-trained models to overfit. However, they would have poor generalisation performance (which is the motivation behind FL). Keeping some model parameters private (here μ and σ from the BN layers) seems to strike a balance between fast convergence (which would be achieved by a fully-private model) and regularisation due to FL (which is achieved by averaging client model parameters).

4.4 Personalised FL Comparison

Setup - We compare the personalisation performance of MTFL(FedAvg) with two other state-of-the-art Personalised FL algorithms: Per-FedAvg [12] and pFedMe [10], and FL (FedAvg) (where no model layers are private) [5]. We tuned the hyperparameters of each algorithm to achieve the maximum average UA within 200 communication rounds. We present MTFL(FedAvg) with private (γ, β), not MTFL (FedAvg-Adam), as we wish only to compare the personalisation algorithms, not the benefit of the adaptive

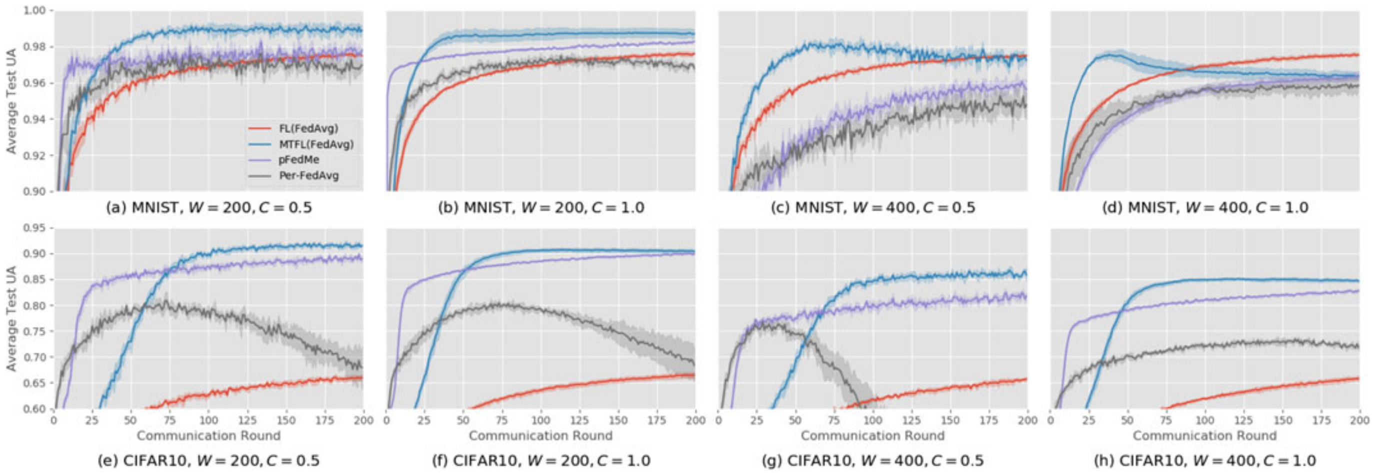


Fig. 5. Per-round testing User Accuracy (UA) of four FL algorithms: FL(FedAvg) [5], MTFL(FedAvg) (using private γ, β), pFedMe [10] and Per-FedAvg [12]. Experiments are conducted on MNIST and CIFAR10, with data divided in a non-IID fashion between $W = 200$ or 400 clients, and $C = 0.5$ or 1.0 fraction of users participating per round. Shaded regions show 95 percent confidence intervals per round over 5 trials with different random seeds.

optimisation strategy. We also fixed the amount of local computation to be roughly constant for the algorithms: we perform $E = 1$ epoch of local training for MTFL(FedAvg) and FL(FedAvg). For MNIST, using a batch size of 20, this is equivalent to 15 and 8 steps of local SGD for $W = 200$ and $W = 400$, respectively. For CIFAR10, this is equivalent to 13 and 7 steps of local SGD for $W = 200$ and $W = 400$, respectively. Per-FedAvg uses the value K for local iterations, so we fix that the the same number of steps for FL and MTFL. pFedMe uses has two inner loops, and we set the number of outer-loops R to the same value as K from Per-FedAvg, and fix the inner-loop number for pFedMe to 1 for all scenarios. This setup results in the same number of local steps performed for each algorithm, however, the cost per local step of Per-FedAvg and pFedMe is considerably higher than FL (FedAvg) and MTFL(FedAvg). Note also that MTFL (FedAvg) and FL(FedAvg) have only one hyperparameter, η to tune, whereas Per-FedAvg and pFedMe both have two. This makes the hyperparameter search for Per-FedAvg and pFedMe considerably more costly.

Results - The plots in Fig. 5 show that MTFL(FedAvg) was able to achieve a higher UA compared to the other schemes in all tested scenarios. Per-FedAvg and pFedMe were able to reach a higher UA than FL(FedAvg) in the $W = 200$ cases for MNIST, but were actually slower in the $W = 400$ cases. All the personalised-FL schemes were able to achieve good UA faster than FL(FedAvg) for the CIFAR10 experiments, however. This is likely due to the CIFAR10 task being a much harder one than MNIST. It is interesting to note that Per-FedAvg appeared to overfit quickly on this task. Also worthy of note is the fact that MTFL(FedAvg) was able to beat Per-FedAvg and pFedMe whilst also having one less hyperparameter to tune, and being computationally cheaper. MTFL also provides the extra benefit to privacy of keeping some model parameters private (pFedMe and Per-FedAvg both upload entire models).

4.5 Testbed Results

Setup - To test MTFL in a more realistic MEC environment, we set up a testbed consisting of 10 clients: 5 Raspberry Pi (RPi) 2B's and 5 RPi 3B's, connected over WiFi to a server, in order to emulate a low-powered, heterogeneous set of clients. The RPi's used Tensorflow to perform local training. The server did not perform any model testing, only receiving, averaging and distributing models. The average time over 10 rounds was taken, along with the percentage of time spent per round in downloading models from the server, local training, uploading models and work performed on the server.

Results - Table 4 shows the average time taken per round for FL(FedAvg), MTFL(FedAvg-Adam), and Independent learning, when one local epoch of training is performed. Each round is also split by time spent for each task within the round. As would be expected, Independent learning took the least time per round as clients did not have to download / upload any models. FL(FedAvg) took longer per round due to uploading / downloading, and MTFL(FedAvg-Adam) took the longest per round due to the increased number of weights that FedAvg-Adam communicates over FedAvg, indicated by the higher percentage of round time spent downloading and uploading models. However, the increase in communication time is likely to be outweighed in most cases by the far fewer

TABLE 4
Average Time Per Round of Different Learning Schemes on the MNIST and CIFAR10 Datasets, and Percentage of Time Spent Downloading the Model (*Down*), Training the Model (*Client*), Uploading the Model (*Up*), and Model Aggregation/Distribution on the Server (*Server*) Took

Learning Scheme	Round Time (s)	Percentage of Round Time (%) *Down*	*Client*	*Up*	*Server*
	MNIST - 2NN				
FL(FedAvg)	30	5	88	6	1
MTFL(FedAvg-Adam)	38	11	76	12	1
Independent	29	0	100	0	0
	CIFAR10 - CNN				
FL(FedAvg)	108	5	86	5	4
MTFL(FedAvg-Adam)	136	11	74	12	3
Independent	100	0	100	0	0

rounds required to reach a target average UA (see Tables 2 and 3).

The majority of the round times were spent in local training rather than in communication for FL or MTFL. This is due to the low computing power of the RPi's and the high computational cost of training DNN models. In real-world FL scenarios, the round times are influenced by the compute abilities of client devices, the computational cost of the models used, and the communication conditions.

5 Conclusion

We proposed a Multi-Task Federated Learning (MTFL) algorithm that builds on iterative FL algorithms by introducing private patch layers into the global model. Private layers allow users to have personalised models and significantly improves average User model Accuracy (UA). We analysed the use of BN layers as patches in MTFL, providing insight into the source of their benefit. MTFL is a general algorithm that requires a specific FL optimisation strategy, and we also proposed the FedAvg-Adam optimisation scheme that uses Adam on clients. Experiments using MNIST and CIFAR10 show that MTFL with FedAvg significantly reduces the number of rounds to reach a target average UA compared to FL, by up to $5\times$. Further experiments show that MTFL with FedAvg-Adam reduces this number even further, by up to $3\times$. These experiments also indicate that using private BN trainable parameters (γ, β) instead of statistics (μ, σ) in model patches gives better convergence speed. Comparison to other state-of-the-art personalised FL algorithms show that MTFL is able to achieve the highest average UA given limited communication rounds. Lastly, we showed in experiments using a MEC-like testbed that the communication overhead of MTFL with FedAvg-Adam is outweighed by its significant benefits over FL with FedAvg in terms of UA and convergence speed.

Acknowledgments

This work was supported by EPSRC DTP Studentship and the EU Horizon 2020 INITIATE Project under Grant 101008297.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[2] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, "Computation offloading in multi-access edge computing: A multi-task learning approach," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.

[3] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang, "Learning-aided computation offloading for trusted collaborative mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 12, pp. 2833–2849, Dec. 2020.

[4] T. Li, A. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.

[5] B. McMahan, E. Moore, D. Ramage, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Stat.*, 2017, pp. 1273–1282.

[6] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," 2019, *arXiv:1905.06731*.

[7] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records," *J. Biomed. Inform.*, vol. 99, 2019, Art. no. 103291.

[8] A. Hard *et al.*, "Federated learning for mobile keyboard prediction," 2018, *arXiv:1811.03604*.

[9] M. Ammad-ud-din, E. Ivannikova, S. A. Khan, W. Oyomno, Q. Fu, K. E. Tan, and A. Flanagan, "Federated collaborative filtering for privacy-preserving personalized recommendation system," 2019, *arXiv:1901.09888*.

[10] C. T. Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," in *Proc. Conf. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 21394–21405.

[11] Y. Jiang, J. Konečný, K. Rush, and S. Kannan, "Improving federated learing personalization via model agnostic meta learning," 2019, *arXiv:1909.12488*.

[12] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach," in *Proc. Conf. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 3557–3568.

[13] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," in *Proc. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4427–4437.

[14] P. K. Mudrakarta, M. Sandler, A. Zhmoginov, and A. Howard, "K for the price of 1: Parameter efficient multi-task and transfer learning," in *Proc. Int. Conf. Learn. Representations*, 2019.

[15] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2019, pp. 6341–6345.

[16] S. Reddi *et al.*, "Adaptive federated optimization," in *Proc. Int. Conf. Learn. Representations*, 2021.

[17] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2014.

[18] F. Hanzely and P. Richtárik, "Federated learning of a mixture of global and local models," 2020, *arXiv:2002.05516*.

[19] Y. Huang *et al.*, "Personalized cross-silo federated learning on non-IID data," in *Proc. Assoc. Adv. Artif. Intell.*, 2021, pp. 7865–7873.

[20] C. Dinh, T. Vu, N. Tran, M. Dao, and H. Zhang, "Fedu: A unified framework for federated multi-task learning with laplacian regularization," 2021, *arXiv:2102.07148*.

[21] H. Jiang, J. Starkman, Y.-J. Lee, H. Chen, X. Qian, and M.-C. Huang, "Distributed deep learning optimized system over the cloud and smart phone devices," *IEEE Trans. Mobile Comput.*, vol. 20, no. 1, pp. 147–161, Jan. 2021.

[22] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *Proc. Syst. Mach. Learn. Conf.*, 2019.

[23] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, "Self-balancing federated learning with global imbalanced data in mobile systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 59–71, Jan. 2021.

[24] H. Yang, Z. Liu, T. Quek, and H. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Trans. Commun.*, vol. 68, no. 1, pp. 317–333, Jan. 2020.

[25] J. Ahn, O. Simeone, and J. Kang, "Wireless federated distillation for distributed edge learning with heterogeneous data," in *Proc. IEEE 30th Ann. Int. Symp. Personal, Indoor Mobile Radio Commun.*, 2019, pp. 1–6.

[26] C. Wang, Y. Yang, and P. Zhou, "Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 394–410, Feb. 2021.

[27] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–7.

[28] W. Liu, L. Chen, Y. Chen, and W. Zhang, "Accelerating federated learning via momentum gradient descent," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 8, pp. 1754–1766, Aug. 2020.

[29] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," 2018, *arXiv:1806.00582*.

[30] N. Konstantinov and C. Lampert, "Robust learning from untrusted sources," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3488–3498.
[31] L. Wang, W. Wang, and B. Li, "CMFL: Mitigating communication overhead for federated learning," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 954–964.
[32] T. Yu, E. Bagdasaryan, and V. Shmatikov, "Salvaging federated learning by local adaptation," 2020, *arXiv:2002.04758*.
[33] Y. LeCun and C. Cortes, "Mnist handwritten digit database." Accessed: Jul. 23, 2021. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[34] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2019.

Jed Mills received the BSc degree in natural science from the University of Exeter in 2018. He is currently working toward the PhD degree in computer science with the Department of Computer Science, the University of Exeter, U.K. His research interests include machine learning, federated learning, and mobile edge computing.

Jia Hu received the BEng and MEng degrees in electronic engineering from the Huazhong University of Science and Technology, China, in 2004 and 2006, respectively, and the PhD degree in computer science from the University of Bradford, U.K., in 2010. He is currently a senior lecturer in computer science with the University of Exeter. His research interests include edge cloud computing, resource optimization, applied machine learning, and network security.

Geyong Min received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995 and the PhD degree in computing science from the University of Glasgow, U.K., in 2003. He is currently a professor of high-performance computing and networking with the Department of Computer Science, the University of Exeter, U.K. His research interests include computer networks, wireless communications, parallel and distributed computing, ubiquitous computing, multimedia systems, and modeling and performance engineering.