IEEE ComSoc | IEEE COMPUTER SOCIETY | IEEE Signal Processing Society | VTS | IEEE Transactions on Machine Learning in Communications and Networking

Received 21 April 2023; revised 6 September 2023, 25 November 2023, and 22 December 2023; accepted 24 December 2023.
date of current version 11 January 2024.

The associate editor coordinating the review of this article and approving it for publication was N. Pappas.

Digital Object Identifier 10.1109/TMLCN.2023.3347680

Congruent Learning for Self-Regulated Federated Learning in 6G

JALIL TAGHIA[1], FARNAZ MORADI[1], HANNES LARSSON[1], XIAOYU LAN[1], ADAM ORUCU[1,2], MASOUMEH EBRAHIMI[2], AND ANDREAS JOHNSSON[1,3] (Senior Member, IEEE)

[1]Ericsson AB, Ericsson Research, 164 40 Stockholm, Sweden
[2]Division of Electronics and Embedded Systems, KTH Royal Institute of Technology, 114 28 Stockholm, Sweden
[3]Department of Information Technology, Uppsala University, 753 10 Uppsala, Sweden

CORRESPONDING AUTHOR: J. TAGHIA (jalil.taghia@ericsson.com)

This work was supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) under Project Celtic-Next ANIARA (C2019/3-2) and Project Celtic-Next IMMINENCE (C2020/2-2).

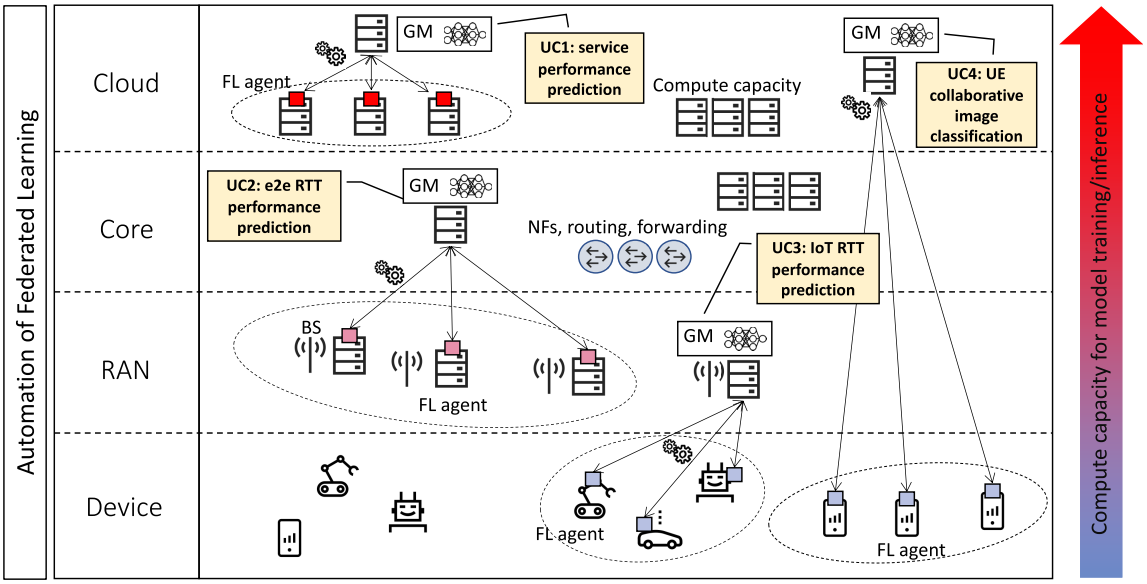
ABSTRACT Future 6G networks are expected to be AI-native with distributed machine learning functionalities responsible for improving and automating a variety of network- and service-management tasks. To enable a privacy-preserving approach to distributed learning, federated learning (FL) has become prevalent in the communication-and-networking domain. However, for efficient management of the networks, FL needs to be automated requiring minimal hyperparameter tuning. An outstanding challenge towards automation of FL is regarding difficulties in handling overfitting. Existing techniques tackle overfitting via regularization heuristics that rely on hyperparameter tuning and as such presume availability of representative validation data. However, in the dynamic and heterogeneous network environments, this assumption is limiting. Even if existence of validation data can be assumed, hyperparameter tuning comes with added communication and compute overhead cost which grows prohibitively as the federation scales in size. Here, we propose the *congruent federated learning* (CFL) as a self-regulated method of learning that is robust to overfitting and achieves the robustness without reliance on hyperparameter tuning. CFL employs a self-taught regularization mechanism that refrains local models from overfitting to the local data. This is enabled via introduction of the *congruent activation functions* as a class of similarity-promoting activation functions that discourage learning local models which differ excessively from the global (federated) model. Across four networking use cases on several tasks, reflecting different profiles of data heterogeneity and limited availability of data, it is shown that CFL greatly reduces overfitting and in nearly all cases improves the performance—a relative gain of about 21% averaged across all use cases.

INDEX TERMS Collaborative intelligence, distributed and federated machine learning for efficient network performance, scalability and complexity of machine learning in networks.

I. INTRODUCTION

MACHINE learning (ML) and artificial intelligence (AI) techniques play an integral role in achieving the goal that is automation in telecommunication networks. They have already proven advantageous in a wide range of use cases including spectrum management and beamforming, resource and slice orchestration, service assurance, energy efficiency optimization, and root-cause analysis [1], [2]. The future 6G network is envisioned to be intelligent as the network becomes AI-native with ML functionalities used pervasively to accommodate increasingly intelligent services and devices [3], [4], [5], [6]. As illustrated in Fig. 1, AI and ML-based functionalities are expected to be integrated into different levels of the future 6G network, ranging from devices to base stations, the core network, and the central cloud.

The flexibility aspect of the network is essential for realizing a vast number of emerging 6G use cases with challenging requirements on latency and throughput such as tele-presence, collaborative driving, and immersive communication [7], [8]. However, the increased flexibility comes naturally with a new set of challenges related to complexity,



BS – Base station, GM – Global Model, FL – Federated Learning, NF – Network Function, UC – Use Case, UE – User Equipment

FIGURE 1. Machine learning will be ubiquitous in 6G networks, for improving and automating a variety of network- and service-management tasks, and for providing AI-compute capabilities as a service. Automation is essential for scalable management and configuration of model hyperparameters as they may depend on data availability, compute capabilities, model latency requirements, and network dynamicity.

performance, and automation. These challenges are due to various sources of data and system heterogeneity, resource limitations, dynamicity of the environment, and data ownership and privacy guidelines [9], [10]. To enable automation of a variety of network and service management tasks, the 6G network will need to base its operation on ML functionalities that can leverage the distributed nature of the environment (in terms of resources) while being able to cope with its challenges.

Given the vital role of ML in achieving the goal of automation of telecommunication networks, it is important to consider requirements and challenges for effective training of ML models with respect to data and compute resources. Enabling pervasive use of ML functionalities in the network necessitates collection of massive volumes of data across the network. To avoid the high cost from overhead of data collection and raw data transmission while safeguarding private data, privacy-preserving distributed ML techniques such as *federated learning* are becoming more prevalent in the communication and networking domain [11], [12], [13], [14], [15].

Federated learning facilitates collaborative learning in a distributed environment while providing certain guarantees on the data privacy [16], [17], [18]. Specifically, one can see it as an approach to distributed learning where agents (residing in nodes with varying capabilities) participate in a federation to collaboratively learn a *global model* without having to share their data. Under orchestration of a server, the standard formulation of federated learning involves cycling through two phases of the *local learning* at the agents and the *global aggregation* at the server. At the local learning phase, agents update their local models given the global model and using their local data. At the global aggregation phase, an aggregated model is constructed by aggregating (e.g., averaging) the local models into a single global model.

However, to enable the automation of federated learning, an important challenge is to avoid the overfitting problem of the learning agents. In this direction, we introduce a method of federated learning, named *congruent federated learning*, which tries to promote similarity between parameters of the agents' models and the global model. This is done via the introduction of a novel class of similarity-promoting parameter activation functions named *congruent activation functions*. Integration and learning through such activation functions in the federated learning process help local models refrain from overfitting to the local data leading to improved convergence characteristics.

We discuss our approach in the context of neural networks as the underlying predictive model and describe the concept of congruent learning through introduction of the congruent activation functions. Unlike the standard activation functions that are applied to the layer representations of the neural networks, congruent activation functions are applied to the parameters of the neural networks (i.e., weight matrices and bias vectors). Conceptually, they discourage learning local models that contrast excessively from the global model, ultimately resulting in local models that are less prone to overfitting.

Our solution is general in the sense that the commonly used federated learning frameworks (such as FedAvg [17], FedSGD [18], and many other recent variants of them [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]) can be turned into a congruent federated learning method with minimal effort. We motivate the rationale behind the idea and

verify the method on four use cases, namely, (1) service performance prediction on data traces collected from a data center (DC), (2) end-to-end (e2e) round-trip-time (RTT) prediction on data traces from a 5G-mmWave testbed, (3) RTT prediction on data traces from an IoT testbed, and (4) user equipment (UE) image classification; these use cases are shown in Fig. 1. The results show that enabling congruent learning in the standard federated learning framework considerably improves its robustness to overfitting and improves optimality of the solutions while requiring no hyperparameter tuning; this translates into reducing the need for labor-intensive configuration and tuning, and as such relaxes the requirement for availability of representative validation data.

The remainder of this paper is organized as follows. Section II describes the problem formulation and gives a sketch of our approach. Section III introduces the proposed family of congruent activation functions and describes the construction of neural networks using them. Section IV introduces congruent federated learning. Section V evaluates the approach empirically on our use cases. Section VI provides a high-level discussion of the main results. Further, Section VII discusses related work, and finally, Section VIII concludes the paper.

II. PROBLEM STATEMENT AND APPROACH

A key step towards automation of federated learning in 6G networks is effectively regulating the model fitness at the local phase of learning, which reduces the requirements on manual configuration and hyperparameter tuning for each ML task. In this section, we provide a problem description, a formal statement of the problem, a background on the existing class of solutions and their limitations, and a high-level description of our approach.

A. PROBLEM DESCRIPTION

Figure 1 shows examples of different federated learning clusters that could be formed across the 6G network. Within each cluster, models with varying resource constraints and requirements (such as latency, accuracy, and privacy) will be trained in a federated learning setting. In use cases 1-3, the models are used for different performance prediction and verification tasks of interest for an operator, whereas use case 4 exemplifies how the network can enable compute offloading and training of a collaborate image classification model. Note that the requirements on the data privacy may vary across use cases. However, common for all is the need to reduce the overhead in terms of data transmission.

These use cases highlight the fact that for the broader applicability of federated learning, there are important challenges to be addressed, among others, with respect to system heterogeneity, data heterogeneity, and communication overhead cost associated with transferring the models from agents to the server and vice versa [29]. Addressing these challenges has been subject of much research since introduction of federated learning [16], [17] and remains an active area of research; we refer readers to a recent survey in [30] for further discussion.

However, a challenge that has been fairly under-studied is regarding the *automation* of federated learning. The dynamic and heterogeneous environment of the future networks poses unique challenges in this regard. One outstanding challenge is related to difficulties in determining the *degree of model fitness* at the local phase of learning. In the context of neural networks as the underlying predictive model, the degree of model fitness is determined largely by the number of epochs per round of federation. At the local phase of learning, if agents learn underfitted models to their local data (too few epochs per round), the global model would require many rounds to converge to a desirable solution; conversely, if agents learn overfitted models (too many epochs per round), the global model may diverge or converge to a poor solution. The difficulty of the problem is that it is unclear under which circumstances a trained model shall be regarded as an underfitted or as an overfitted model. The problem becomes particularly pronounced when agents' data distributions are heterogeneous to a degree that they can be seen as non-independently and identically distributed (non-IID), or when agents have limited data samples representative of the underlying oracle distribution of their data [24], [31], [32]. The degree of model fitness affects not only the needed number of rounds before a reasonable solution is reached but also the optimal properties of the final solution [33], [34].

Indeed, determining the exact degree of model fitness at the local learning phase of the federated learning remains unsolved under presence of severe data heterogeneity (non-IID settings) or insufficient amounts of data. However, to enable automation of federated learning, it might be sufficient to instead study design of learning methods that are robust to overfitting.

Existing solutions aim at reducing overfitting through regularization via heuristics that require hyperparameter tuning. Thus, to enable the hyperparameter tuning through cross-validation, they presume availability of representative validation data. However, in a highly dynamic and heterogeneous environment of the 6G networks, this assumption is not realistic. Additionally, hyperparameter tuning can become costly as it requires performing multiple training and evaluation steps. Considering the resource limitations (in terms of compute and data) in the environment, there is a need for development of federated learning methods that achieve robustness to overfitting through self-taught mechanisms that do not rely on the hyperparameter tuning.

B. PROBLEM FORMULATION

For a neural network with a given architecture, the number of epochs per round is one of the influential factors determining the overall model fitness[1] in terms of convergence

[1]There are a number of other factors that can affect the degree of model fitness, such as size and quality of training data, architecture of the neural network, optimizer, and loss function. However, here, we study the effect of number of epochs per round while fixing the effect from other factors.

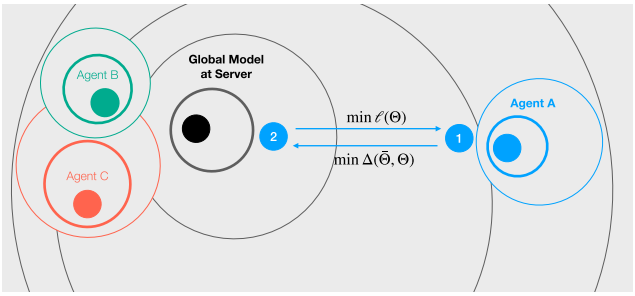


FIGURE 2. A conceptual figure visualizing the optimization problem in (1), described from Agent A prospective. Merely minimizing the loss at the agent node moves the space of solutions in the direction of the evidence from the local data (position 1), resulting in an overfitted model. However, merely minimizing the deviation of the agent's parameter set Θ from the global parameter set $\bar{\Theta}$ moves the space of solutions in direction of the evidence from the global model (position 2), resulting in an underfitted model. As the federation progresses, the challenge is to arrange an interplay between the two competing objectives that leads to improved convergence characteristics.

characteristics, including both the optimal properties of the final solution and the convergence rate.

Let J_r denote the training number of epochs at the round r of a federated learning. A typical assumption is that

$$J_r = J \quad \forall r = 1, 2, \ldots R,$$

where R is the number of rounds. However, this assumption ignores that at different stages of learning, the same number of epochs J can result in models with different degrees of fitness. As an example, for a given agent, training locally for J' epochs at the round r' may result in an overfitted model while training for $J'' \gg J'$ at the round r'' may result in an underfitted model.

The problem can be stated as follows: *at each round of federated learning and at the local learning phase, for how many epochs should agents train their models for the federation's learning objective to converge to an optimal solution?* A theoretically grounded solution to the problem may not be readily available. However, in practice, it might suffice to instead learn models that are *robust to overfitting*. That effectively means training models for many epochs locally and then relying on some clever regularization heuristics for regulating the extra complexity.

Perhaps, the first heuristic that comes to mind is using early stopping at the local learning phase based on data samples from a validation set (i.e., a portion of the train set) or model selection based on choosing the model that performs best on the validation set. However, a validation set that is representative of the data is not always available; for example, in scenarios where there are only a few data samples available for training at the agents, or where data of different agents are non-IID such that agents' local data are poorly representative of the underlying distribution of data. Arguably, federated learning is of utmost relevance in precisely these scenarios.

An alternative heuristic is to instead construct models that are robust to overfitting. One way to achieve this is through solving a constrained optimization problem where, at the training phase, the aim is to find a setting of the local parameter set Θ that minimizes a given agent's learning loss ℓ while deviating as little as possible from the global parameter set $\bar{\Theta}$, that is:

$$\min_{\Theta \in \mathfrak{P}} \ell(\Theta) \quad \text{subject to :} \quad \min \Delta(\bar{\Theta}, \Theta), \tag{1}$$

where $\Delta(\bar{\Theta}, \Theta)$ quantifies the deviation of Θ from $\bar{\Theta}$, and $\mathfrak{P}$ denotes the set of all possible parameters. Figure 2 conceptualizes the optimization problem in (1).

C. LIMITATIONS OF EXISTING SOLUTIONS

Existing solutions (e.g., in [19], [20], [21], [22], [23], [24], [25], [26], [27], and [28]) solve the constrained optimization problem in (1) via a class of heuristic techniques based on hyperparameter tuning known as *penalty methods*. This is done by adding a *penalty function* to the objective function ℓ that consists of a *penalty parameter* multiplied by a measure of violation of the constraint. Effectively, they solve a surrogate objective function,

$$\min_{\Theta \in \mathfrak{P}} \ell(\Theta) + \mu\, \ell_{(\text{reg})}(\Theta, \bar{\Theta}), \tag{2}$$

where $\ell_{(\text{reg})}$ is the penalty term and μ is the penalty parameter that dictates the strength of the penalization. Although optimizing the surrogate loss in (2) can potentially help reduce overfitting, choosing the right penalty parameter μ is consequential. In practice, optimal settings of the penalty parameters are data dependent, and they are often selected through cross-validation techniques. Hence, in the scope of this study that is concerned with the cases where cross-validation may not be feasible, the existing techniques are not suitable due to the need for data-dependant hyperparameter tuning of the penalty parameters; refer to the related work in Section VII for additional discussion.

D. OUR APPROACH: LEARNING THROUGH CONGRUENCE

In this work, we introduce an alternative class of heuristic methods for relaxation of the constrained optimization problem in (1) via introduction of the concept of congruent learning in the context of federated learning. Compared to the existing class of heuristics based on penalty methods that require hyperparameter tuning, the proposed heuristics are notably free from hyperparameters.

Learning through congruence is enabled via introduction of a similarity-promoting class of parameter activation functions referred to as the congruent activation functions. Such activation functions directly operate on the model parameters and are designed to promote similarity by discouraging contrasts between the local parameters Θ and the global parameters $\bar{\Theta}$. Contrary to the existing class of solutions for relaxation of the constrained optimization problem in (1) into an unconstrained one through optimizing a surrogate objective function as in (2), our proposed class of solutions optimizes the objective function in (1) directly and regulates

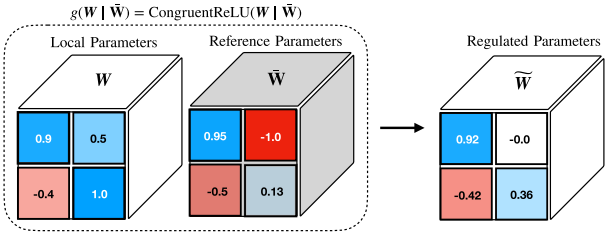


FIGURE 3. A conceptual example visualizing effect of application of congruent activation function, CongruentReLU, in regulating the model parameters $\boldsymbol{W}$ given the reference parameters $\bar{\mathbf{W}}$. The resulting regulated parameters are shown here with $\widetilde{\boldsymbol{W}} \leftarrow \text{CongruentReLU}(\boldsymbol{W} \mid \bar{\mathbf{W}})$.

the parameter set Θ through the congruent activation functions. The fundamental distinction is that, in our case, it is the model parameters that are regulated directly and not the objective function.

III. CONGRUENT LEARNING IN NEURAL NETWORKS

In this section, we begin with providing the necessary background and defining our notations. Next, we introduce congruent activation functions and their integration in the construction of neural networks. This is a first step towards the design of congruent federated learning as described in Section IV.

A. BACKGROUND AND NOTATIONS

Let $\{\underline{\mathbf{x}} = \mathbf{x}_n \mid n = 1, \ldots, N\}$ denote the input data where $\mathbf{x}_n$ indicates the n-th feature vector, and let $\underline{\mathbf{y}} = \{\mathbf{y}_n \mid n = 1, \ldots, N\}$ denote the output task response where $\mathbf{y}_n$ indicates the n-th response vector. Furthermore, let $f_{\underline{\Theta}}$ define a neural network consisting of L layers with the parameter set $\underline{\Theta} = \{\Theta^l \mid l = 1, \ldots, L\}$, specified as:

$$f_{\underline{\Theta}}(\mathbf{x}) = \left(f^1_{\Theta^1} \circ f^2_{\Theta^2} \circ \ldots \circ f^L_{\Theta^L}\right)(\mathbf{x}), \tag{3}$$

where $\circ$ denotes the function composition and $f^l_{\Theta^l}$ indicates the layer l of the neural network with the layer parameter set Θ^l. Depending on the class of neural networks, a layer parameter set may include different types of learnable parameters. As an example, for a multi-layer perceptron (MLP) neural net, a layer parameter set Θ includes a weight matrix and a bias vector.

In standard training of a neural network, learning through error back-propagation involves finding a setting of $\underline{\Theta}$ that minimizes the error loss $\ell(\mathbf{y}, \hat{\mathbf{y}})$ between $\mathbf{y}$ and its prediction, $\hat{\mathbf{y}} := f_{\underline{\Theta}}(\mathbf{x})$.

B. CONGRUENT ACTIVATION FUNCTIONS

Here, we formally introduce the class of congruent activation functions and provide an example of such activation functions that will be used throughout the paper.

We begin with a conceptual example. As visualized in Fig. 3, a congruent activation function, g, takes as its inputs: (i) a parameter $\boldsymbol{W}$ that requires optimization and (ii) a reference parameter $\bar{\mathbf{W}}$ that does not require optimization, where these two parameters belong to the same space of parameters. The reference parameter $\bar{\mathbf{W}}$ can be seen as a mask that is applied element wise to the optimizable parameter $\boldsymbol{W}$. The activation function is designed to preserve agreements and penalize disagreements between the optimizable parameter and the reference parameter.

To keep the notation uncluttered, let $\boldsymbol{W}[i, j] := w$, and $\bar{\mathbf{W}}[i, j] := \bar{\mathrm{w}}$ be the (i, j)-th element of $\boldsymbol{W}$, and $\bar{\mathbf{W}}$. Further, let sgn denote the signum function. A congruent activation function, denoted by g, satisfies the following conditions:

1) $\text{sgn}(w) \neq \text{sgn}(\bar{\mathrm{w}}) \quad \Rightarrow \quad g(w \mid \bar{\mathrm{w}}) \to 0;$
2) $w \to \bar{\mathrm{w}} \quad \Rightarrow \quad g(w \mid \bar{\mathrm{w}}) \to \bar{\mathrm{w}};$
3) $\text{sgn}(w) = \text{sgn}(\bar{\mathrm{w}}),\ w \gg \bar{\mathrm{w}} \ \Rightarrow \ \bar{\mathrm{w}} < g(w \mid \bar{\mathrm{w}}) < w;$
4) $\text{sgn}(w) = \text{sgn}(\bar{\mathrm{w}}),\ w \ll \bar{\mathrm{w}} \ \Rightarrow \ w < g(w \mid \bar{\mathrm{w}}) < \bar{\mathrm{w}};$
5) g must be approximately differentiable almost everywhere.

In the above, the first condition enforces maximal penalty for disagreements in signs; the second condition ensures preservation of the agreements; the third and fourth conditions introduce smoothing effect; the fifth condition is needed to enable back-propagation in training neural networks via automatic differentiation tools.

An example of a congruent activation function g satisfying these conditions is given by

$$\begin{aligned} g\left(\boldsymbol{W} \mid \bar{\mathbf{W}}\right) &= \text{sgn}\left(\bar{\mathbf{W}}\right)\sqrt{\epsilon + \text{ReLU}\left(\bar{\mathbf{W}} \odot \boldsymbol{W}\right)}, \\ &:= \text{CongruentReLU}(\boldsymbol{W} \mid \bar{\mathbf{W}}) \end{aligned} \tag{4}$$

where $\odot$ indicates the element-wise multiplication, ReLU denotes the rectified linear activation function, ϵ is a small positive number added for numerical stability ($\epsilon \to +0$). We refer to the congruent activation function in (4) as CongruentReLU. Conceptually, if $\boldsymbol{W}$ and $\bar{\mathbf{W}}$ have opposite signs, the contrast between the two is maximal; the disagreement is settled by setting it to a value that approaches zero. If $\boldsymbol{W}$ is in full agreement with the reference $\bar{\mathbf{W}}$, both in the sign and the strength, the agreement is preserved. If $\boldsymbol{W}$ and $\bar{\mathbf{W}}$ have the same signs but contrast in their strength values such that the strength of one is much larger than the other, the output is skewed towards the one with the smaller strength.

Figure 4 visualizes CongruentReLU for a case example showing the behaviour of the function at and about the reference parameters. As shown in Fig. 4.A,B, depending on the sign and the strength of the reference parameter, the function behaviour varies. Specifically, let $g(w \mid \bar{\mathrm{w}}) = \text{CongruentReLU}(w \mid \bar{\mathrm{w}})$, then for all $w \in \boldsymbol{W}$ and $\bar{\mathrm{w}} \in \bar{\mathbf{W}}$, it follows that:

$$\begin{cases} g(w \mid \bar{\mathrm{w}}) \to 0, & \bar{\mathrm{w}} > 0,\ \forall\, w \leq 0, \\ w < g(w \mid \bar{\mathrm{w}}) < \bar{\mathrm{w}}, & \bar{\mathrm{w}} > 0,\ \forall\, 0 < w < \bar{\mathrm{w}}, \\ \bar{\mathrm{w}} < g(w \mid \bar{\mathrm{w}}) < w, & \bar{\mathrm{w}} > 0,\ \forall\, 0 < \bar{\mathrm{w}} < w, \end{cases}$$

$$\begin{cases} g(w \mid \bar{\mathrm{w}}) \to 0, & \bar{\mathrm{w}} < 0,\ \forall\, w \geq 0, \\ w < g(w \mid \bar{\mathrm{w}}) < \bar{\mathrm{w}}, & \bar{\mathrm{w}} < 0,\ \forall\, w < \bar{\mathrm{w}} < 0, \\ \bar{\mathrm{w}} < g(w \mid \bar{\mathrm{w}}) < w, & \bar{\mathrm{w}} < 0,\ \forall\, \bar{\mathrm{w}} < w < 0, \end{cases}$$

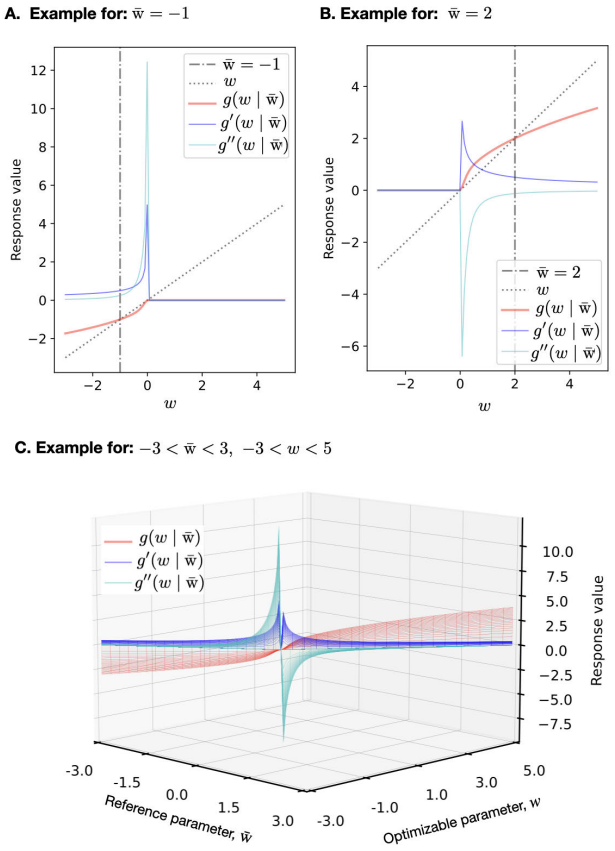


FIGURE 4. Example visualization for the congruent activation function, $g(w \mid \bar{w})$ = CongruentReLU($w \mid \bar{w}$), where $w \in W$ is a learnable parameter requiring optimization and $\bar{w} \in \bar{W}$ is a reference parameter which does not require optimization. Here, g' and g'' denote the first and second derivatives with respect to w. (A, B) Function and its first and second derivatives for fixed negative and positive values of $\bar{w}$, respectively. (C) Function and its first and second derivatives for a range of w and $\bar{w}$.

satisfying the requirements for the congruent activation functions. In addition to the function values, the figure also visualizes its first and second derivatives with respect to w; Fig. 4.C. visualizes the function values and the derivatives for a range of w and $\bar{\mathbf{w}}$.

At the training phase of a neural network model during the forward propagation, CongruentReLU is applied at each epoch. Upon application, the optimizable parameters $\boldsymbol{W}$ are updated by a regulated version of them according to: $\boldsymbol{W} \leftarrow \text{CongruentReLU}(\boldsymbol{W} \mid \bar{\mathbf{W}})$.

C. CONSTRUCTION OF NEURAL NETWORKS WITH CONGRUENT ACTIVATION FUNCTIONS

Here, we formally describe construction of neural networks with congruent activation functions, and next describe their application in the context of federated learning. For the ease of discussion, we begin with a simple example and next proceed with the general formulation.

Figure 5.B visualizes transformation of a neural network as specified in Fig. 5.A into the one with congruent activation functions. In practice, enabling congruent learning in a neural network involves integration of the congruent activation functions to the layers of the neural network. This integration requires minimal effort; it involves only changing the forward pass. The congruent activation functions are designed to be differentiable so that the error back-propagation can be followed naturally using standard automatic differentiation techniques (e.g, as implemented in PyTorch or TensorFlow).

Consider an MLP neural network denoted as $f_{\underline{\Theta}}(\mathbf{x}) = \left(f^1_{\Theta^1} \circ f^2_{\Theta^2} \circ f^3_{\Theta^3}\right)(\mathbf{x})$ and specified according to:

$$f_{\underline{\Theta}}(\mathbf{x}) = \begin{cases} \hat{\mathbf{y}} &= a^1\left(\boldsymbol{W}^1\mathbf{h}^2 + \boldsymbol{b}^1\right), \\ \mathbf{h}^2 &= a^2\left(\boldsymbol{W}^2\mathbf{h}^3 + \boldsymbol{b}^2\right), \\ \mathbf{h}^3 &= a^3\left(\boldsymbol{W}^3\mathbf{x} + \boldsymbol{b}^3\right), \end{cases}$$

where $\underline{\Theta} = \{\Theta^l \mid l = 1, 2, 3\}$, l indicates the neural network layer, a^l is a layer activation function (e.g., ReLU and tanh), $\mathbf{h}^l$ is the vector of latent layer representations, and the pair of $\boldsymbol{W}^l \in \Theta^l$ and $\boldsymbol{b}^l \in \Theta^l$ denotes the weight matrix and the bias vector at the l-th layer which require optimization.

The corresponding neural network with congruent activation functions is denoted as

$$f_{g(\underline{\Theta}|\bar{\underline{\Theta}})}(\mathbf{x}) = \left(f^1_{g(\Theta^1|\bar{\Theta}^1)} \circ f^2_{g(\Theta^2|\bar{\Theta}^2)} \circ f^3_{g(\Theta^3|\bar{\Theta}^3)}\right)(\mathbf{x}),$$

and constructed according to:

$$f_{g(\underline{\Theta}|\bar{\underline{\Theta}})}(\mathbf{x}) = \begin{cases} \hat{\mathbf{y}} &= a^1\left(g(\boldsymbol{W}^1 \mid \bar{\mathbf{W}}^1)\mathbf{h}^2 + g(\boldsymbol{b}^1 \mid \bar{\mathbf{b}}^1)\right), \\ \mathbf{h}^2 &= a^2\left(g(\boldsymbol{W}^2 \mid \bar{\mathbf{W}}^2)\mathbf{h}^3 + g(\boldsymbol{b}^2 \mid \bar{\mathbf{b}}^2)\right), \\ \mathbf{h}^3 &= a^3\left(g(\boldsymbol{W}^3 \mid \bar{\mathbf{W}}^3)\mathbf{x} + g(\boldsymbol{b}^3 \mid \bar{\mathbf{b}}^3)\right), \end{cases}$$

where $\bar{\underline{\Theta}} = \{\bar{\Theta}^l \mid l = 1, 2, 3\}$ denotes the set of reference parameters, g is a congruent activation function, such as CongruentReLU given by (4), and $\bar{\mathbf{W}}^l \in \bar{\Theta}^l$ and $\bar{\mathbf{b}}^l \in \bar{\Theta}^l$ are the reference weights and biases which do not require optimization.

As in MLPs, other classes of neural networks, including recurrent neural nets and convolutional neural nets, can be transformed into the ones with congruent activation functions. In a general form, for a neural network f as defined in (3), we express the corresponding neural network with congruent activation functions as:

$$f_{g(\underline{\Theta}|\bar{\underline{\Theta}})}(\mathbf{x}) = \left(f^1_{g(\Theta^1|\bar{\Theta}^1)} \circ f^2_{g(\Theta^2|\bar{\Theta}^2)} \circ \ldots \circ f^L_{g(\Theta^L|\bar{\Theta}^L)}\right)(\mathbf{x}), \quad (5)$$

where the notation of $g(\Theta^l \mid \bar{\Theta}^l)$ is used to emphasize on the dependence to the reference parameter set at the layer l through the congruent activation function g. Given $\bar{\underline{\Theta}}$, the training involves finding a setting of $\underline{\Theta}$ that minimizes the loss $\ell(\mathbf{y}, \hat{\mathbf{y}})$ defined between $\mathbf{y}$ and $\hat{\mathbf{y}} := f_{g(\underline{\Theta}|\bar{\underline{\Theta}})}(\mathbf{x})$.

IV. CONGRUENT FEDERATED LEARNING

In this section, we describe the concept of congruent federated learning which corresponds to federated learning using neural networks with congruent activation functions. Without

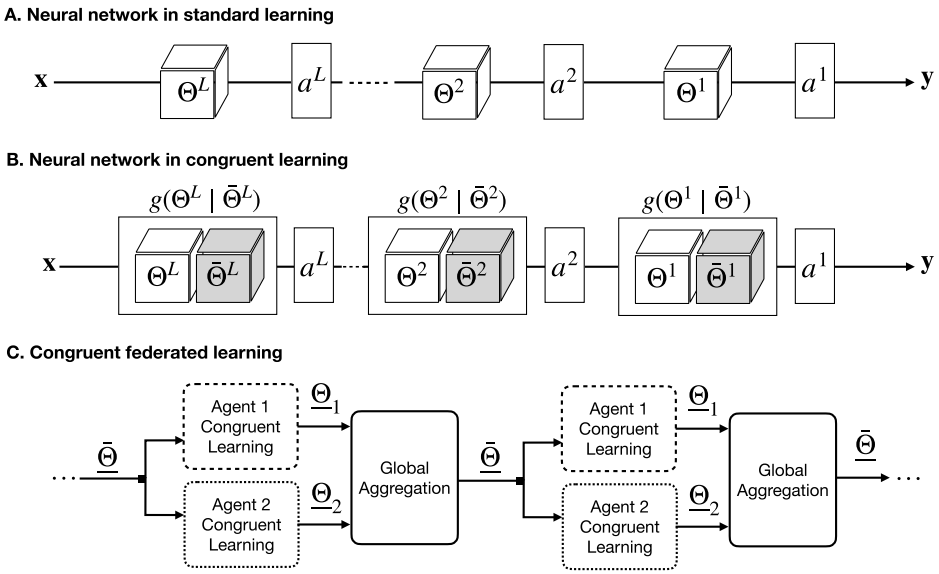


FIGURE 5. **Transformation of a neural network into the one with congruent activation functions and its implementation in federated learning: (A) a neural network as defined in (3); (B) integration of the congruent activation functions g into the layers of the neural network for the construction of the neural network in (5); (C) two consecutive rounds of federated learning through congruent learning in a federation of two agents. Here, a^l denotes the layer activation function, Θ^l denotes the layer parameter set, and $\bar{\Theta}^l$ denotes the corresponding reference parameter set. The horizontal lines in (A) and (B) indicate the layer depth while in (C) the timeline by training rounds.**

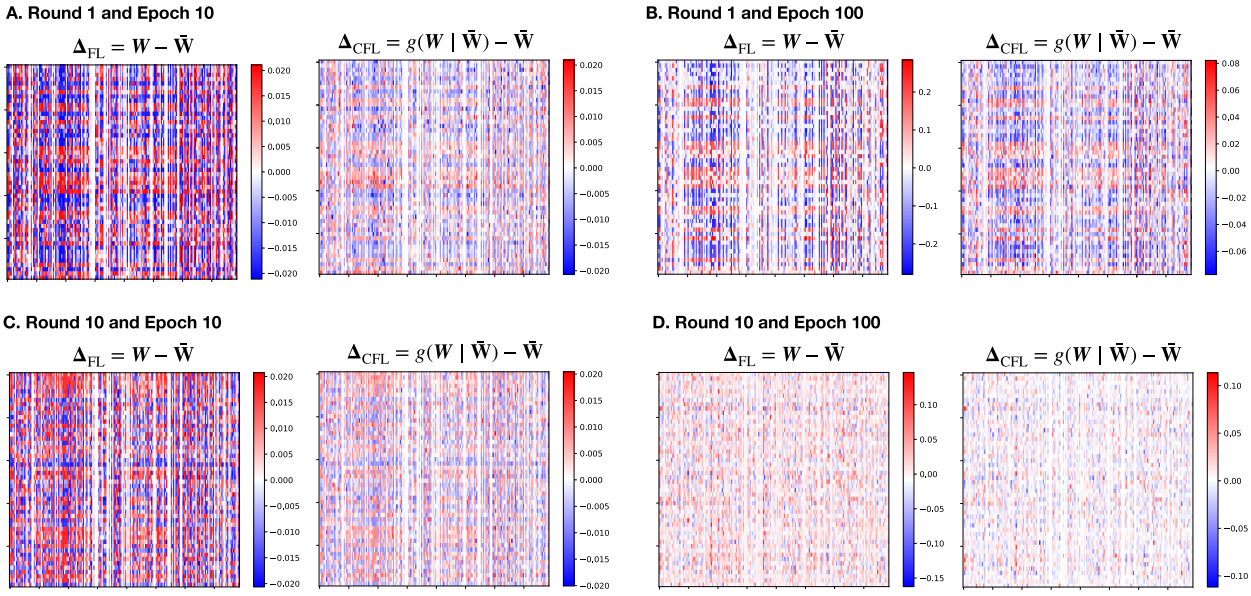


FIGURE 6. **Visualization of an example showing the changes in the weight matrix parameter of the input layer of a randomly chosen agent during federated learning with and without congruent activation functions, referred to as CFL and FL, respectively. Note the difference in the scales of color maps.**

loss of generality, we consider federated learning framework of FedAvg [17] and construct a variant of it that uses our construction in (5).

A. LEARNING THROUGH CONGRUENT ACTIVATION FUNCTIONS

Figure 5.C visualizes two consecutive rounds of federated learning between two agents where each agent accommodates a neural network model with congruent activation functions (as shown in Fig. 5.B), and shares their regulated parameters with the server for the global aggregation. This procedure is described formally as follows.

Consider a federation of M agents, and let us assume that the agents participate in the federation for solving the same task (such as a classification or a regression task) and share the same input feature attributes. However, we make

Algorithm 1 A Variant of FedAvg [17] Described in Congruent Federated Learning Framework; the Main Differences Are Highlighted by Box Frames. All Notations Are Defined in Sections III and IV

Outputs:

- $\underline{\Theta}_{(\text{global})}$: final global model parameter set
- $\underline{\Theta}_m$, $\forall m$: final local model parameter sets

Server executes:

initialize the global model parameter set $\underline{\Theta}_{(\text{global})}$ randomly;
for $r = 1, \ldots, R$ **do**
  for $m = 1, \ldots, M$ **do**
    send the global model $\underline{\Theta}_{(\text{global})}$ to the m-th agent;
    $\underline{\Theta}_m \leftarrow \text{AgentLocalLearning}(m, \underline{\Theta}_{(\text{global})})$;
  end
  $\underline{\Theta}_{(\text{global})} \leftarrow \text{GlobalAggregation}\left(\underline{\Theta}\right)$;
end
GlobalAggregation $\left(\underline{\Theta}\right)$**:**
  return: $\underline{\Theta}_{(\text{global})} = \text{Aggregation}(\underline{\Theta}_m \,|\, m = 1, \ldots, M)$
for $l = 1, \ldots, L$ **do**
  for $\theta^l_{(\text{global})} \in \Theta^l_{(\text{global})}$ **do**
    $\theta^l_{(\text{global})} \leftarrow \sum_{m=1}^{M} \frac{N_m}{\sum_{i=1}^{M} N_i} \theta^l_m$ (as in (7));
  end
  $\underline{\Theta}_{(\text{global})} = \{\Theta^l_{(\text{global})} \mid l = 1, \ldots, L\}$;
end
AgentLocalLearning $\left(m, \underline{\bar{\Theta}}\right)$**:**
  return: $\underline{\Theta}_m = \{\Theta^l_m \mid l = 1, \ldots, L\}$
initialize local parameter set as: $\underline{\Theta}_m \leftarrow \underline{\Theta}_{(\text{global})}$ as in (6a);
update the reference parameter set: $\underline{\bar{\Theta}} \leftarrow \underline{\Theta}_{(\text{global})}$ as in (6b);
for $j = 1, \ldots, J$ **do**
  for each batch of data $\mathcal{B}_m = \{(\mathbf{x}_m, \mathbf{y}_m)\} \subset \mathcal{D}_m$ **do**
    $\hat{\mathbf{y}}_m := f_{g(\underline{\Theta}_m|\underline{\bar{\Theta}})}(\mathbf{x})$ (as in (5)) ;
    $\ell_m = \text{Loss}(\mathbf{y}_m, \hat{\mathbf{y}}_m)$;
    $\Theta^l_m \leftarrow \text{Optimizer}(\Theta^l_m, \nabla_{\Theta^l_m} \ell_m)$;
  end
end

no assumptions about the agents' data distributions, meaning that they can be non-IID. Let $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n) \mid n = 1, \ldots, N\}$ denote the data consisting of the pairs of inputs and outputs, and let $\mathcal{D}_m$ denote the local data from the m-th agent. Furthermore, following the general form of (5), let $f_{g(\underline{\Theta}_m|\underline{\bar{\Theta}})}$ denote the local model of the m-th agent with the optimizable parameter set $\underline{\Theta}_m$ given the reference parameter set $\underline{\bar{\Theta}}$, that is,

$$f_{g(\underline{\Theta}_m|\underline{\bar{\Theta}})}(\mathbf{x}) = \left(f^1_{g(\Theta^1_m|\bar{\Theta}^1)} \circ \ldots \circ f^L_{g(\Theta^L_m|\bar{\Theta}^L)}\right)(\mathbf{x}), \ \forall m.$$

Under orchestration of a server, the global parameter set is initialized randomly and sent to the agents. Learning then follows by cycling through the two phases of the local learning at the agents and the global aggregation at the server, described in the following.

Let $\underline{\Theta}_{(\text{global})} = \{\Theta^l_{(\text{global})} \mid l = 1, \ldots, L\}$ denote the set of global parameters received from the server by the agents. The m-th agent initiates the local learning with updating both its reference parameter set $\underline{\bar{\Theta}}$ and initializing its optimizable parameter set $\underline{\Theta}_m$ with the global parameter set $\underline{\Theta}_{(\text{global})}$, that is:

$$\underline{\bar{\Theta}} = \left\{\bar{\Theta}^l \leftarrow \Theta^l_{(\text{global})} \mid l = 1, \ldots, L\right\}, \tag{6a}$$

$$\underline{\Theta}_m = \left\{\Theta^l_m \leftarrow \Theta^l_{(\text{global})} \mid l = 1, \ldots, L\right\}. \tag{6b}$$

Next, it trains its model locally given the local data $\mathcal{D}_m$ for J epochs which means optimizing the local parameter set $\underline{\Theta}_m$ through minimizing the local loss $\ell_m = \text{Loss}(\mathbf{y}_m, \hat{\mathbf{y}}_m)$ where $\hat{\mathbf{y}}_m = f_{g(\underline{\Theta}_m|\underline{\bar{\Theta}})}(\mathbf{x}_m)$ as defined in (5). It is important to note that, during training at a given round, the reference parameter set $\underline{\bar{\Theta}}$ remains *unaltered* while the optimizable parameter set $\underline{\Theta}_m$ *adapts* in the direction of minimizing the loss ℓ_m. The reference parameter set changes once a new global parameter set is provided to the agent at the next round of federated learning. At the end of the local training phase, the agent sends its optimized local parameter set $\underline{\Theta}_m$ to the server. The same procedure will be carried out by all M agents.

Server constructs the global model which is an aggregated model computed from the agent models. Among others, the aggregated model could simply be obtained by taking the arithmetic average of the local model parameter sets or, following the framework of FedAvg, by taking the weighted average based on the number of samples per agent. In the case of the latter, for a parameter $\theta^l \in \Theta^l$ (such as a weight matrix or a bias vector), it is expressed as:

$$\theta^l_{(\text{global})} = \sum_{m=1}^{M} \tau_m \theta^l_m, \quad \tau_m = \frac{N_m}{\sum_{i=1}^{M} N_i}, \tag{7}$$

where $N_m = |\mathcal{D}_m|$ is the number of training data points of agent m. Similarly, aggregation is done for all the parameters at all layers and accordingly the global parameter set $\underline{\Theta}_{(\text{global})} = \{\Theta^l_{(\text{global})} \mid l = 1, \ldots, L\}$ is updated. In a general form, the global aggregation step is shown as

$$\underline{\Theta}_{(\text{global})} = \text{Aggregation}(\underline{\Theta}_m \mid m = 1, 2, \ldots, M).$$

The resulting global parameter set is sent to the agents which will be used by the agents at the local learning phase. The procedure continues until a convergence or a stopping criterion is met.

An algorithmic[2] description of the congruent federated learning is shown in Algorithm 1 built on FedAvg. The main differences are at the local learning phase and, more precisely, learning through neural networks with congruent activation functions.

Regarding the number of epochs per round, J, it shall be set to a "sufficiently large" value. In practice, it means setting it to a larger value than expected (or a value much larger

[2]An implementation of the congruent federated learning in PyTorch is available through the following public repository under BSD 3-Clause License: https://github.com/EricssonResearch/congruent-federated-learning.git.

that expected if added computational complexity is not the main concern) and then relying on the congruent learning to regulate additional complexity and reduce risks of overfitting.

B. EXAMPLE VISUALIZATION

Figure 6 visualizes an example where we monitor the changes in the weight matrix parameter of the input layer of an agent during federated learning with and without congruent activation functions, referred to as CFL and FL, respectively. More specifically, the figure shows $\Delta_{\text{FL}} := \boldsymbol{W} - \bar{\mathbf{W}}$ and $\Delta_{\text{CFL}} := g(\boldsymbol{W} \mid \bar{\mathbf{W}}) - \bar{\mathbf{W}}$ at rounds 1 and 10 of the learning at selected epochs 10 and 100, where $\boldsymbol{W}$ is the optimizable weight matrix at the input layer, $\bar{\mathbf{W}}$ is the corresponding reference weight matrix from the global model, and g is the CongruentReLU activation function.

Figure 7.A,B show the results at the early phase of learning, at round 1. Going from epoch 10 to epoch 100, in the case of FL, the majority of the weight parameters quickly deviate from the reference. However, in the case of CFL, the congruent activation function regulates the weight parameters such that they do not deviate from the reference as quickly. As the learning continues, at round 10 and at epoch 100 (shown in Fig. 7.D), the profile of the learned weight matrix obtained from CFL differs notably from the one using FL which is reflected in Δ_{CFL} and Δ_{FL}. Importantly, we can see the effect of self-regulation in CFL in encouraging the parameters to stay in the neighborhood of the reference.

V. EXPERIMENTS

Experiments are designed in connection to the problem statement in Section II and our use cases illustrated in Fig. 1. Our main focus is on federated learning scenarios where agents have access to limited amounts of data for training and where agents' data distributions are non-IID. A direct consequence of which is that in all experiments, we assume there is no validation data available to be used for the purpose of improving model generalization capabilities through heuristics such as hyperparameter tuning via cross-validation, early stopping of the learning, or model selection.

A. USE CASES

In the following, we describe the four studied use cases in relation to Fig. 1.

1) USE CASE 1: DC SERVICE-PERFORMANCE PREDICTION

In this use case, the goal is to predict the service-level metrics (SLMs) on clients, which is required for automated service and network management in 5G and beyond networks. The SLM prediction is done using data collected from a DC infrastructure, as well as labels collected from the clients. Here, data traces are collected from a DC testbed at KTH University [35], and are publicly available[3] [36]. The testbed consists of a server cluster and six client machines. There are two services running on these machines, namely, Video-on-Demand (VoD) and a Key-Value (KV) store (database). The traces are generated by executing experiments with different configurations of services and load patterns. The features are collected from the Linux kernels on the server cluster machines and the SLMs are collected on the client machines. Examples of such features are CPU utilization per core, memory utilization, network utilization and disk I/O. The task underlying the use case is prediction of SLMs given the features. For KV traces, there are two SLMs, average read and write latency, while for VoD traces, we consider six SLMs such as number of played audio samples, and average read delay. Table 1 lists all the SLMs considered in this study for KV and VoD services.

Specifically, data used in this study are an emulation of a multi-operator environment of 24 operators (agents). Each agent node has a unique configuration based on the execution type, load pattern, and the client server machine. Agents have roughly equal number of samples. Table 1 summarizes the data specifications for KV and VoD services. Data from these traces are by construction heterogeneous making them suitable for our study [34]. Additionally, obtaining labeled data from agents requires instrumentation of the agents which can be costly and therefore, in an operational network, the number of labeled data samples at each agent can be limited.

2) USE CASE 2: 5G E2E RTT PERFORMANCE PREDICTION

The goal of this use case is to predict e2e RTT values experienced by a UE connected to the network based on data collected from the base station which is valuable for applications to proactively solve performance issues in 5G and beyond networks. The traces are collected from a 5G-mmWave testbed in which the equipment corresponds to a 5G non-standalone system [37]. In this testbed, there are two UEs, one for performance measurements and the other for traffic-load generation. The traces obtained from the testbed contain RTT values as experienced by a UE in the network, and approximately 200 metrics and events related to the analogue beamforming function, te UEs connected to the base station, and the uplink and downlink events [38], [39], [40].

Traces considered here include four experiments specified according to the traffic-load generation, and the UE movement. In two experiments, a constant bit rate traffic-load (uplink) is generated and in others no traffic load is added. In two experiments, the UE is moving for a duration of 10 minutes, and in the others the UE is stationary. The e2e RTT is measured every 10ms using ICMP ping [41] with a measurement packet size of 1400 bytes. Further, as a preprocessing step, the RTT and the base-station metrics are averaged with different time intervals (e.g., every 100ms, 1000ms). In total, 495 features are generated from around 200 metrics based on different averaging intervals.

For construction of the federation, we treat each experiment as an agent. The objective is to predict e2e RTT as

[3]Direct link to data: https://www.kaggle.com/datasets/jaliltaghia/data-traces-from-a-data-center-testbed.

TABLE 1. Data specifications for DC traces (KV and VoD services), and split of data into federations of 24 agents (use case 1).

Agent	Configuration			Tasks	Dataset	
	Load Pattern	*Execution Type*	*Machines*	(SLMs)	D_x	D_y
KPS1-6	Periodic	SingleApp	1 - 6	KV (*)	197	1
KPB1-6	Periodic	BothApps	1 - 6	KV (*)	197	1
KFS1-6	Flashcrowd	SingleApp	1 - 6	KV (*)	197	1
KFB1-6	Flashcrowd	BothApps	1 - 6	KV (*)	197	1
VPS1-6	Periodic	SingleApp	1 - 6	VoD (**)	182	1
VPB1-6	Periodic	BothApps	1 - 6	VoD (**)	182	1
VFS1-6	Flashcrowd	SingleApp	1 - 6	VoD (**)	182	1
VFB1-6	Flashcrowd	BothApps	1 - 6	VoD (**)	182	1
* KV SLMs:	(1) ReadsAvg, (2) WritesAvg.					
** VoD SLMs:	(1) AvgInterAudioPlayedDelay, (2) AvgInterDispDelay, (3) NetReadAvgDelay, (4) NetReadBytes, (5) noAudioPlayed, (6) DispFrames.					

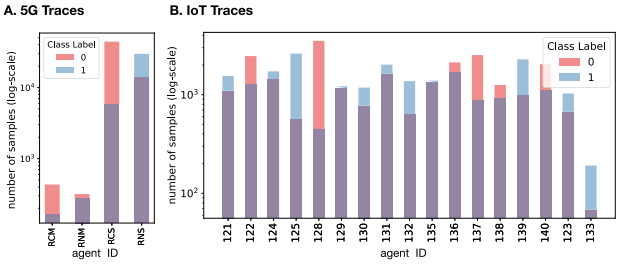


FIGURE 7. Number of samples per label for (A) 5G traces (use case 2), and (B) IoT traces (use case 3). Note that, for ease of visualization, y-axis is in logarithmic scale. Data are divided into train and test set as specified in Table 3.

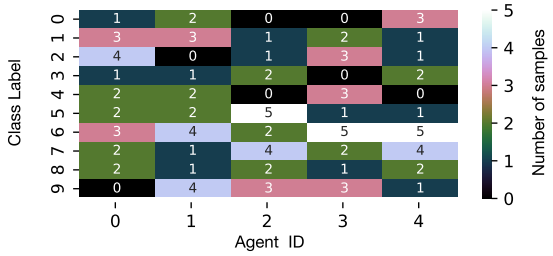


FIGURE 8. Number of samples per class label for FMNIST (use case 4) at the training phase. The figure exemplifies the case for federation of 5 agents.

TABLE 2. Data specifications for 5G traces and split of data into 4 agents (use case 2).

Agent Name	Configuration		Task (*)	Dataset	
	Traffic Load	*UE Status*		D_x	D_y
RNS	None	Stationary	RTT	495	2
RCM	Constant bit rate	Moving	RTT	495	2
RCS	Constant bit rate	Stationary	RTT	495	2
RNM	None	Moving	RTT	495	2
Note: *	RTT prediction cast as a binary classification problem.				

TABLE 3. Data split into train and test sets for use cases 1-4 (uc.1-4).

Data Trace UC.X	DC-KV UC.1	DC-VOD UC.1	5G UC.2	IoT UC.3	FMNIST UC.4
Train (%)	1	10	50	20	4
Test (%)	99	90	50	80	96
N_{samples}	∼24000	∼24000	(**)	(**)	500 (*)
Notes	(*) Number of samples per label varies (refer to Fig. 8). (**) Number of samples per label varies (refer to Fig. 7).				

experienced by the UEs given traces from the 5G-mmWave base station. Data traces are summarized in Table 2. The use case is formulated as a binary classification problem based on different RTTs (low versus high) according to a threshold.[4] Note that, the total number of samples per agent varies noticeably, and agents do not have the same number of samples per label, as shown in Fig. 7.A.

3) USE CASE 3: IOT RTT PERFORMANCE PREDICTION

IoT devices are expected to be ubiquitous and 6G is expected to enhance and accelerate the performance of IoT devices compared to 5G [42]. In this use case, the goal is to predict the RTT between an IoT device and an IoT gateway. The traces are generated from the EWSN'17 testbed located at Uppsala University [43], and are publicly available. The testbed consists of 18 Tmote Sky Motes (IoT devices). A mote is a compact device with a complete execution environment including: CPU, memory, radio, antenna, battery, and various sensors including temperature, humidity, and light. The motes reside in a university building constituting a challenging environment for wireless communication.

A mote either acts as a Two-Way Active Measurement Protocol (TWAMP) controller or a reflector [44]. The TWAMP controller is placed on one of the motes. For our purpose, we build a federation of 17 agents. This is done by treating the 17 reflector motes as agents and the controller as the server. The task is to predict RTT between the mote node itself and

[4]The threshold corresponds to emerging 6G use cases explored and therefore is confidential. However, the difference between low and high is large enough to lead to different distributions across outputs, corresponding to service violations.

the gateway (TWAMP controller), given 14 input features containing network statistics; example of input features are: RPL rank, RSSI, NBR, LQI, CPU and transmitted packets. The use case is formulated as a binary classification problem based on different RTTs (high versus low) corresponding to the threshold of 1 second. Note that, the total number of samples per agent varies noticeably, and agents do not have the same number of samples per label, as shown in Fig. 7.B. The IoT devices in general have resource constraints and therefore available data for training and validation will be very limited in real life as well.

4) USE CASE 4: UE COLLABORATIVE IMAGE CLASSIFICATION

For this use case, we consider collaborative image classification on mobile devices connected to the cellular network. We use data from FMNIST [46], a popular image data set that is publicly available. The data set comprises of 28×28 gray-scale images of fashion products from 10 categories, such as: dress, coat, sandals.

We construct federations with different sizes consisting of 2, 5, 50, and 100 agents. All agents have equal number of training data samples and the number of samples per agent remains the same for different sizes of federations. This is done to evaluate how federated learning methods scale with the number of agents and the extent they are able to leverage the available data distributed across agents.

Aiming at construction of heterogeneous federations, data are divided across agents in a non-IID fashion such that none of the agents in their training set have data representing all 10 class labels; as an example, Fig. 8 shows the number of training samples per class label for a federation of 5 agents. However, importantly, at the evaluation phase, data samples from the test set include all 10 class labels.

B. EXPERIMENTAL SETUP AND EVALUATION FRAMEWORK

Evaluation framework is designed to reflect scenarios where the federation is heterogeneous with respect to the underlying distribution of data and where there are limited data samples available at the training phase—a common theme for datasets and tasks in operational networks. Data samples from all use cases are randomly split into train and test sets. Table 3 summarizes the data split considered in the experiments.

We compare performance of the standard federated learning (FedAvg) denoted as FL, against the proposed variant of it based on congruent federated learning denoted as CFL.

All methods use a similar neural network architecture as their underlying predictive model. In all experiments, the model architecture is an MLP specified in Table 4. The distinguishing factor between the MLP architecture in CFL and other methods is that CFL uses congruent activation functions, as defined in (4), at all layers. In any other ways, the MLP models are identical and use the same initialization (with the same random seed).

We consider federations of 20 rounds, $R = 20$. To evaluate the effect of the degree of model fitness on the performance, we consider different training strategies through varying the number of epochs J at the local phase of learning including 10, 100, 1000 epochs. As a convention, for instance, we use the notation CFL-10 to denote the congruent federated learning with 10 epochs per round; similarly, FL-10 to denote the standard federated learning with 10 epochs per round. Note that at the training phase, there is no early stopping (or model selection) in place, and all models are trained for the specified number of epochs. In fact, there is no possibility of model selection, as across all experiments, agents' training data are highly heterogeneous and in some cases non-IID. This agrees with the goals of the study towards automation of federated learning and provides an opportunity to evaluate model sensitivity to overfitting in real-world use cases.

Models are learned on the train set and evaluated on the unseen data samples from the test set. For use case 1, the regression performance is evaluated in terms of the normalized mean absolute error (nMeanAE) between measured and predicted SLMs, given by:

$$\text{nMeanAE} := \frac{1}{\bar{y}} \left(\frac{1}{N_{\text{test}}} \sum_{n=1}^{N_{\text{test}}} |y_n - \hat{y}_n| \right),$$

where $\hat{y}_n$ is the model prediction for the n-th measured performance metric y_n, and $\bar{y}$ is the average quantity across all samples in the test set. The nMeanAE scores closer to zero are preferred. For use cases 2-4, classification performance is evaluated in terms of f1-score which combines the precision and recall scores of the classification into a single metric by taking their harmonic mean. The use of f1-score is preferred often in evaluation of imbalanced datasets which is the case for use cases 2-4. The f1-scores closer to one are preferred.

For the same training set, experiments are repeated 10 times with random initialization of the models. At a given experiment, all models use the same initialization. The average and standard deviation values of the performance metrics across all independent runs are reported.

C. EXPERIMENTAL RESULTS

In all experiments, we show the performance metrics, the nMeanAE for regression tasks in use case 1 and the f1-score for classification tasks in use cases 2-4, averaged across all agents. The results are shown for selected rounds, 5, 10, 15, and 20. The *main observations* are with respect to (i) robustness to overfitting and (ii) the optimality of the solution in terms of the final performance metrics.

The first main observation is that the number of epochs per round for FL is an influential hyperparameter that greatly affects the optimality of the solution, whereas it is far less of a concern for CFL. The second main observation is that, in several cases, CFL improves the optimal properties of the solutions resulting in improved performance metrics where in some cases the improvement is substantial.

TABLE 4. Neural network architecture of the MLP used in experiments.

MLP	Number of Units	Batch Normalization	Dropout	Layer Activation Function	Parameter Activation Function	
					CFL	FL
Input Layer	*	True	0.2	Tanh	CongruentReLU	False
Hidden Layer 1	50	True	0.2	ReLU	CongruentReLU	False
Hidden Layer 2	50	True	0.2	ReLU	CongruentReLU	False
Output Layer	*	False	False	Linear	CongruentReLU	False
Loss Function	Cross-Entropy loss for FMNIST, 5G and IoT traces (use cases 2-4); Smooth-L1 loss for DC traces (use case 1).					
Optimizer	Adam optimizer [45] using default settings (learning rate=0.001).					
Note: *	The number of units for input and output layers are set according to dimensionality of the data trace.					

TABLE 5. Summary of the results across all use cases and their respective tasks at the final round of learning (round 20). The table compares the methods in the statistical sense using paired t-Test (With the statistical significance of 0.05). In cases where multiple methods perform equally well with no statistical significance, all the methods are stated. Relative gain is computed at round 20 according to $\text{gain}_{\text{relative}} = \frac{\text{abs}(Q_{\text{CFL}} - Q_{\text{FL}})}{Q_{\text{FL}}} \times 100$ where Q_{CFL} indicates the performance metric obtained using CFL-1000, and similarly Q_{FL} indicates that of obtained using FL-1000. For regression tasks (use case 1), the performance metric is nMeanAE and for classification tasks (use case 2-4), it is f1-score. Refer to Fig. 9 and 10 for a detailed presentation of the results.

Use case		Task	Best performer(s) across CFL methods	Best performer(s) across FL methods	Best performer(s) overall	Relative gain (%)
Use case 1 -VoD		*AvgInterAudioPlayedDelay*	CFL-1000	FL-10	CFL-1000	35
		AvgInterDispDelay	CFL-10, 100, 1000	FL-10	CFL-10, 100, 1000	46
		NetReadAvgDelay	CFL-10, 100, 1000	FL-100	CFL-10, 100, 1000	44
		NetReadBytes	CFL-10, 100, 1000	FL-100	CFL-10, 100, 1000, FL-100	5
		noAudioPlayed	CFL-10, 100, 1000	FL-10	CFL-10, 100, 1000	34
		DispFrames	CFL-10, 100, 1000	FL-10	FL-10	29
Use case 1 - KV		*ReadsAvg*	CFL-1000	FL-100	CFL-1000, FL-100	25
		WritesAvg	CFL-100	FL-100	FL-100	18
Use case 2 - 5G		*e2e RTT classification*	CFL-1000	FL-10	CFL-1000	12
Use case 3 - IoT		*RTT classification*	CFL-1000	FL-100	CFL-1000	6
Use case 4	**2 agents**	*Image classification*	CFL-10, 100, 1000	FL-10	FL-10	—
	5 agents	*Image classification*	CFL-10, 100	FL-100	CFL-10, 100	4
	50 agents	*Image classification*	CFL-10, 100, 1000	FL-100	CFL-10, 100, 1000	14
	100 agents	*Image classification*	CFL-10, 100, 1000	FL-10	CFL-10, 100, 1000	16

Table 5 provides a summary of the results at the final round of learning, round 20. It is shown that CFL-1000 is among the best performing CFL methods in all cases with the exception of "ReadsAvg" task from KV traces in use case 1. In comparison, for FL methods, in some cases FL-10 and in other cases FL-100 are the best performers—in none of the cases FL-1000 is among the best performers. This is important from an automation point of view: for the case of CFL, it suffices to set the number of epochs to a large value and then rely on the self-regularization capability enabled through congruent learning for handling the overfitting. In the context of our experiments, it means CFL-1000 can be used safely across all use cases and nearly all tasks with little signs of overfitting. When the number of epochs per round increases to an arbitrary large value—from 100 in CFL-100 to 1000 in CFL-1000—in most cases, there is no sign of overfitting such that CFL-100 and CFL-1000 perform equally well, while for the case of FL-100 and FL-1000, there is a noticeable decrease in performance due to overfitting as the number of epochs grows. More specifically, when the performance of CFL-1000 is compared directly to FL-1000, across all use cases and all tasks, CFL-1000 surpasses FL-1000 to a markedly greater degree in the statistical sense; Table 5 summarizes the relative gain achieved by CFL-1000 over FL-1000.

In the following, we further describe the results for our four use cases in Section V-A.

1) USE CASE 1: DC SERVICE-PERFORMANCE PREDICTION

Figure 9 shows the regression results in terms of nMeanAE as the performance metric. Regarding the convergence rate,

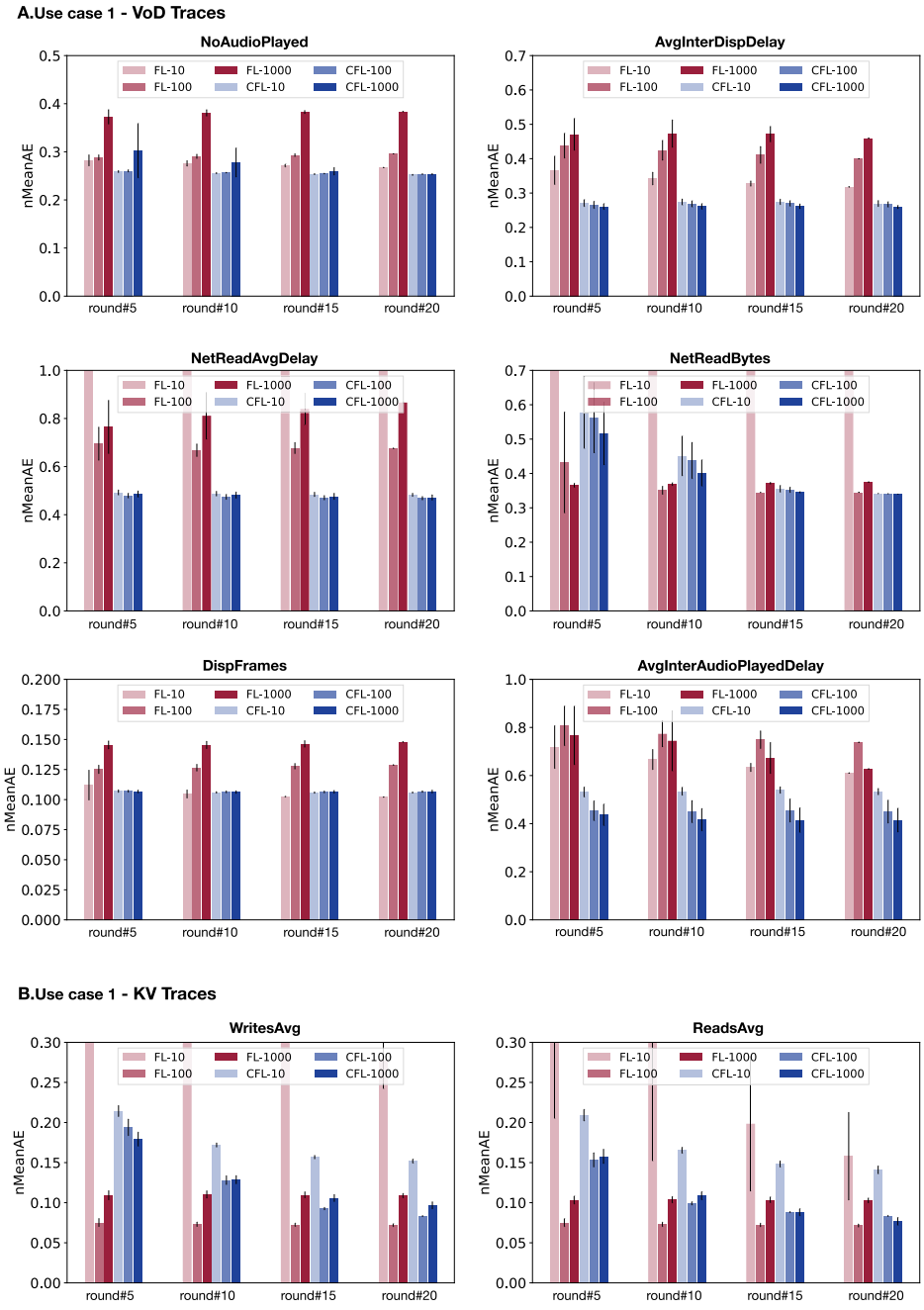


FIGURE 9. Regression performance evaluation on use case 1 in terms of nMeanAE (smaller values are preferred). The figure compares the performance of the congruent federated learning (CFL) against the standard federated learning (FL) at selected rounds. As an example, FL-10 and CFL-10 indicate FL and CFL trained using 10 epochs per round, respectively.

across a majority of VoD tasks (with the exception of "NetReadBytes"), CFL converges faster while for KV tasks FL has a faster convergence.

Considering the performance of the methods over the course of learning, there are two main observations. Firstly, FL shows signs of overfitting for the majority of tasks: "NoAudioPlayed", "NetReadAvgDelay", "DispFrames", and "AvgInterDispDelay", "NetReadBytes" from VoD tasks, and both "WritesAvg" and "ReadsAvg" from KV tasks. The overfitting issue is most visible with the increase in the number of epochs from 100 to 1000 which corresponds to FL-100 and FL-1000, respectively. In comparison, CFL can effectively avoid overfitting across most tasks with the exception of a few tasks such as "WritesAvg" where slight overfitting is observed going from CFL-100 to CFL-1000 at round 20. Secondly, for VoD and KV traces, across a

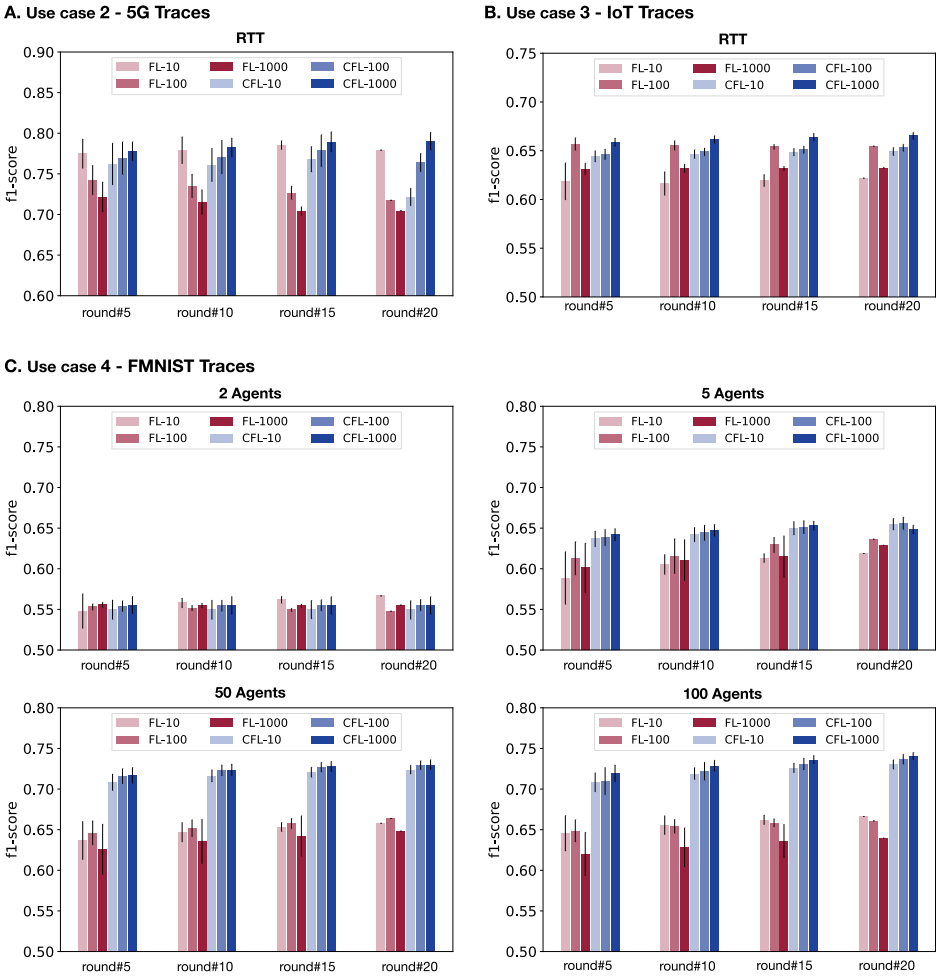


FIGURE 10. Classification performance evaluation on use cases 2-4 in terms of f1-score (values closer to one are preferred). The figure compares the performance of the congruent federated learning (CFL) against the standard federated learning (FL) at selected rounds. As an example, FL-10 and CFL-10 indicate FL and CFL trained using 10 epochs per round, respectively.

majority of their respective tasks, CFL achieves the highest performance where the improvement is most noticeable for "AvgInterDispDelay", "NetReadAvgDealy", and "AvgInterAudioPlayedDelay".

Comparing the performance of CFL-1000 and FL-1000 directly, CFL-1000 is among the best performing methods across nearly all tasks, with the exception of "WritesAvg". Among CFL methods, in most cases, CFL-1000 and CFL-100 perform equally well and they are the best performers, which highlights effective handling of overfitting in CFL. However, among FL methods, in some cases FL-10 is clearly the best performer while in some other cases FL-100, and in none of the cases FL-1000 is among the best performers. The observations here point at the overfitting problem in FL.

2) USE CASE 2: 5G E2E RTT PERFORMANCE PREDICTION

Figure 10.A shows the classification results in terms of f1-score as the performance metric. Regarding the convergence rate, both classes of methods, FL and CFL, show similar characteristics.

Considering the performance of the methods over the course of learning, it is shown that FL methods are largely affected by overfitting which is reflected in the classification accuracy; overfitting is most pronounced in FL-1000. In comparison, for CFL there is no sign of overfitting and the performance improves with the increase in the number of epochs; CFL-1000 is shown to achieve the highest f1-score.

Comparing the performance of CFL-1000 and FL-1000 directly, CFL-1000 performs considerably better than FL-1000; indeed FL-1000 is the worst performer among FL methods whereas CFL-1000 is the best performer among CFL methods.

3) USE CASE 3: IOT RTT PERFORMANCE PREDICTION

Figure 10.B shows the classification results in terms of f1-score as the performance metric. Regarding the convergence rate, FL and CFL show similar characteristics.

Considering the results over the course of learning, similar observations can be made as in use case 2. Firstly, it is shown that the FL methods are largely affected by overfitting while CFL shows no sign of overfitting. Secondly, CFL-1000 is shown to achieve the highest f1-score.

Comparing the performance of CFL-1000 and FL-1000 directly, CFL-1000 performs substantially better than FL-1000.

4) USE CASE 4: UE COLLABORATIVE IMAGE CLASSIFICATION

Figure 10.C shows the results for image classification on FMNIST. In this experiment, the number of agents is varied to reflect federations with different sizes. For the federation of 2 agents, FL and CFL perform equally well. However, importantly, as the federation grows in size from 2 agents to 5, 50 and 100 agents, this is the CFL that benefits the most from the added number of agents. For example, by increasing the size of federation from 2 to 50 agents, the performance of the best performing FL method goes from about 0.57 to about 0.67 while the performance of the best performing CFL method goes from about 0.55 to about 0.74.

Considering the results over the course of learning, as in the previous use cases, there are two main observations. Firstly, FL is largely affected by overfitting while CFL is only marginally affected. For federations of smaller sizes, 2 and 5 agents, both FL and CFL show slight signs of overfitting. As the federation grows in size to 50 and 100 agents, FL shows clear signs of overfitting while CFL shows no sign of overfitting. Secondly, CFL is shown to achieve the highest f1-score as the federation grows in size.

Comparing the performance of CFL-1000 and FL-1000 directly, CFL-1000 outperforms FL-1000 where the improvement becomes more notable as the federation grows in size. Finally, across CFL methods, CFL-100 and CFL-1000 perform equally well and both are among the best performing methods. The results once again show effective handling of the overfitting, that is arbitrarily increasing the number of epochs from 100 to 1000 (corresponding to CFL-100 and CFL-1000) will not affect the optimality of the final solution in a statistical sense. On the other hand, across FL methods, performance of FL-100 differs from FL-1000 where in all cases FL-100 is preferred to FL-1000 which is a sign of an overfitting problem.

D. COMPARISON WITH PENALTY BASED METHODS

As discussed in Section II, while both family of techniques, the penalty-based approach and the proposed congruent-learning-based approach, aim at handling overfitting through regularization, they achieve this in two different ways. The penalty-based methods achieve the regularization effect through optimizing the surrogate objective function (2) whereas the proposed approach directly optimizes the original objective function (1) and instead regularizes the learnable parameters through the application of the congruent activation function. Importantly, the penalty based methods require hyperparameter tuning of a penalty parameter which dictates the strength of the regularization while the proposed approach is free from such hyperparameter tuning.

In the following, we compare the performance and computational complexity of CFL against a penalty-based candidate approach in selected experiments. Specifically, we consider FedProx [19] as a well-known example of the penalty-based methods and compare its performance against CFL on use cases 2-4.

1) PERFORMANCE COMPARISON WITH FedProx

The method FedProx relies on the hyperparameter tuning of a penalty term, referred to as the *proximal term* and denoted by μ in [19]. One goal for this comparative study is to compare the performance of CFL against FedProx in handling overfitting, and further to demonstrate the need for the hyperparameter tuning of the proximal term in FedProx and its overall effect on the performance. The second goal is to show that the congruent learning can be integrated into other federated learning frameworks, such as FedProx. This is to highlight that the proposed approach to handling overfitting can be seen as complementary to the penalty-based family of methods including FedProx.

With the above goals in mind, we explicitly compare the performance of the following learning methods, namely: (i) FL, the standard federated learning based on FedAvg framework with no mechanism for handling overfitting; (ii) FedProx as described in [19]; (iii) CFL based on FedAvg framework; (iv) FedProxCFL, a variant of CFL based on FedProx framework.

For FedProx and FedProxCFL, the proximal term is varied and chosen from the non-exhaustive set of $\mu \in \{0.001, 0.01, 0.1, 1.\}$ as recommended in [19]. We limit the experiments to use cases 2-4 and compare the performance of the above methods.

Figure 11 summarizes the results of the experiments for use cases 2 and 3. Similarly, Fig. 12 summarizes the results for use case 4 for the federations of 5 agents and 50 agents. The first main observation is that FedProx can potentially outperform FL for certain settings of the proximal term. As an example, in use case 2, for the case of 10 epochs per round, the proximal term $\mu = 0.01$ is the best setting which results in a noticeable gain over FL at the final round of federation (round 20) while for the case of 1000 epochs per round, $\mu = 0.1$ is the best setting. For use cases 3 and 4, similar observations can be made. Table 6 summarizes the best settings of μ at the final round of federation for FedProx and FedProxCFL. As shown in this table, the optimal setting of the proximal term depends on a number of factors, including the data type, the number of epochs per round, and the size of federation.

The second main observation is that CFL either performs equally as good or outperforms the best performing FedProx in a majority of cases. An exception is in use case 2 for the case of 10 epochs per round, Fig. 11.A.1. This is explained by the fact that in this case, CFL is underfitted. As discussed earlier, CFL requires that the number of epochs per round be

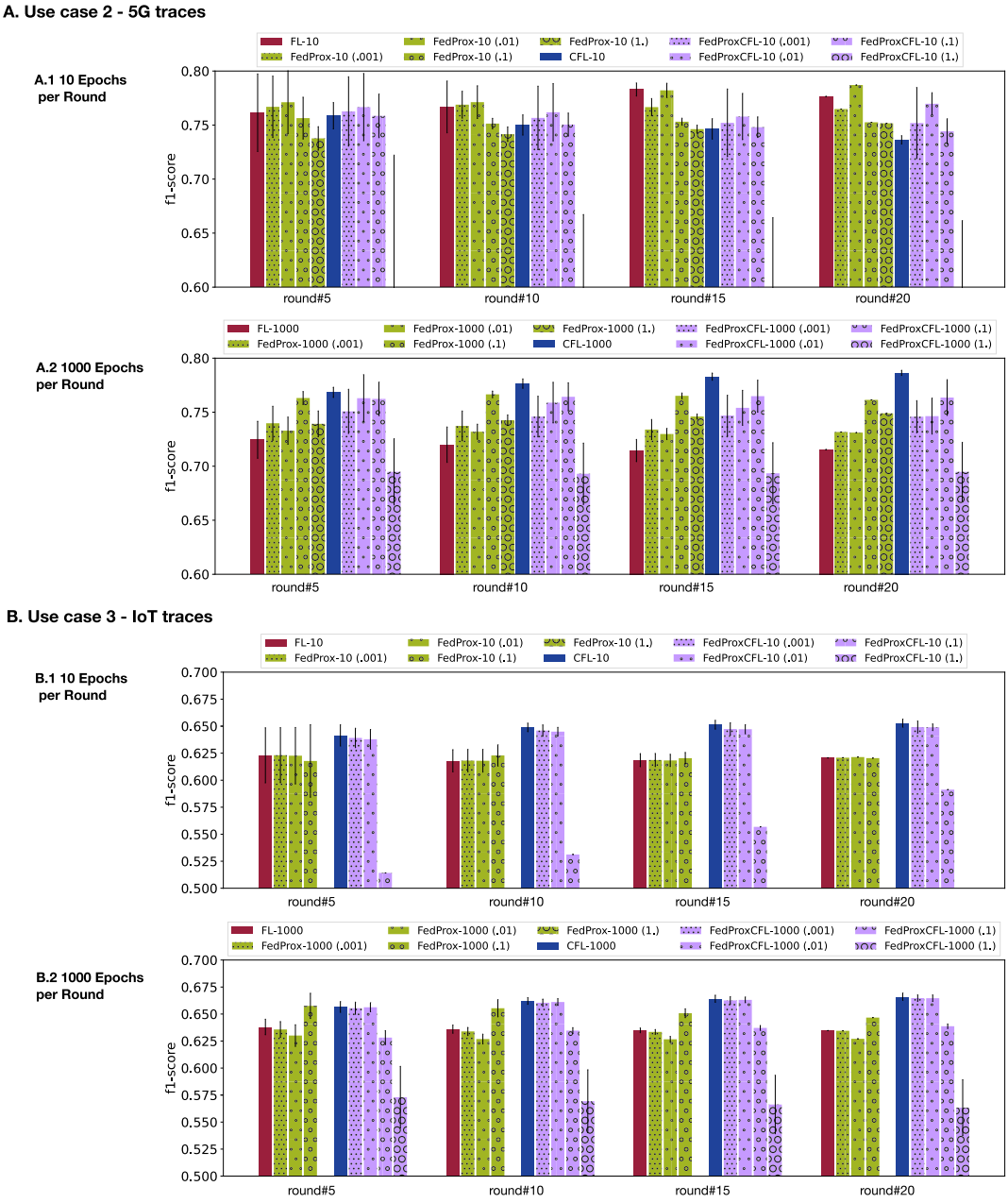


FIGURE 11. Classification performance evaluation on use cases 2 and 3 in terms of f1-score (values closer to one are preferred). The figure compares the performance of FL, FedProx, CFL and FedProxCFL at selected rounds. For instance, FedProxCFL-10 (0.1) indicates FedProxCFL trained using 10 epochs per round with the proximal term $\mu = 0.1$.

TABLE 6. Optimal settings of the proximal term μ for FedProx and FedProxCFL across use cases 2-4.

Use case		Use case 2		Use case 3		Use case 4 (5 agents)		Use case 4 (50 agents)	
Method		FedProx	FedProxCFL	FedProx	FedProxCFL	FedProx	FedProxCFL	FedProx	FedProxCFL
Condition	*10 epochs per round*	0.01	0.01	0.001, 0.01, 0.1	0.001, 0.01	0.1	0.1	0.1	0.01
	1000 epochs per round	0.1	0.1	0.1	0.001, 0.01	0.001	0.01, 0.1	0.01	0.001, 0.01, 0.1

sufficiently large. As shown in Fig. 11.A.2, once the number of epochs per round increases to 1000 epochs per round, CFL is shown to outperform FedProx noticeably by effectively regulating the additional complexity.

The third main observation is that there are cases where the best performing FedProxCFL has a marginal advantage over CFL. As an example, this is true in use case 4 where FedProxCFL with the proximal term $\mu = 0.1$ outperforms

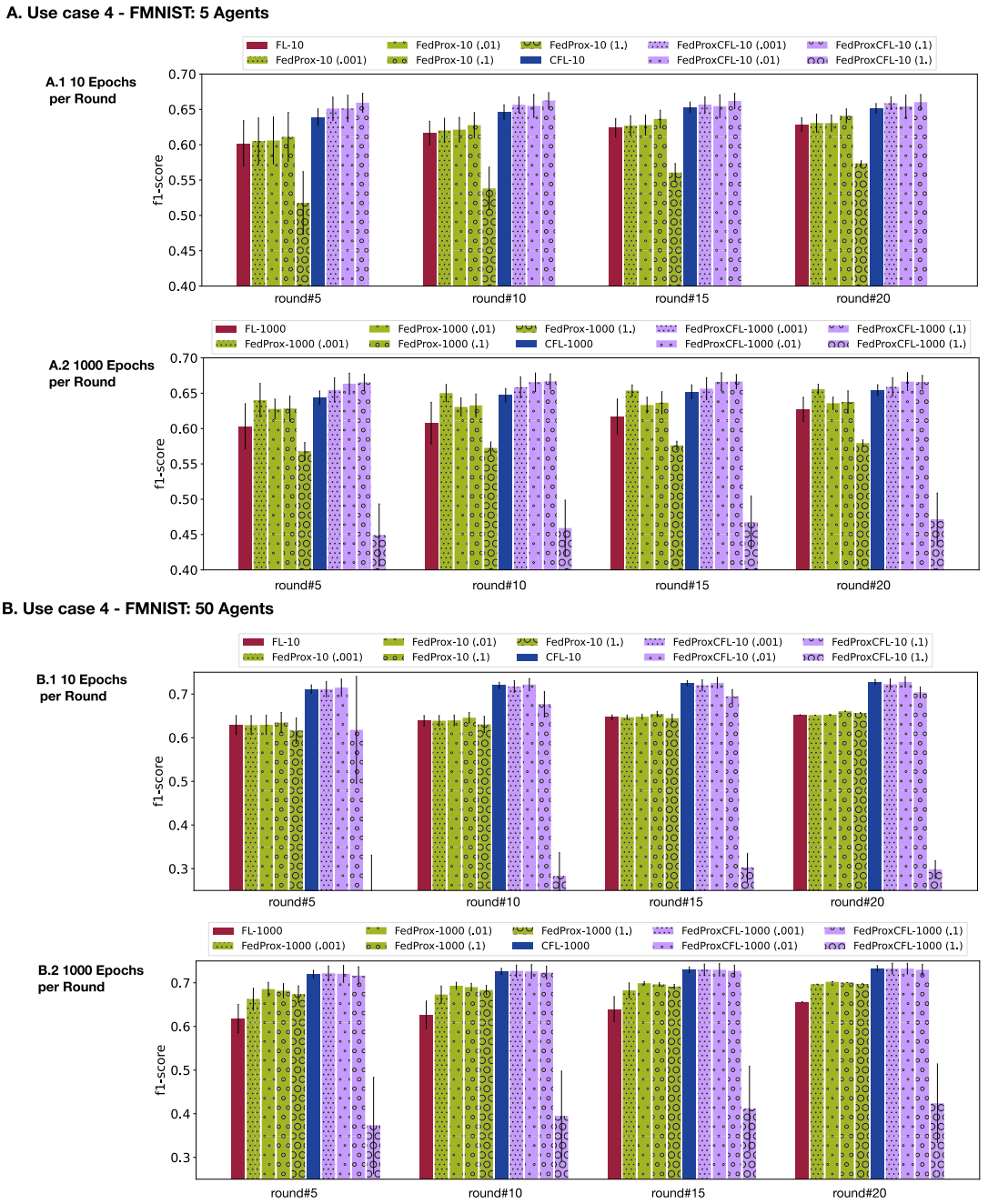


FIGURE 12. Classification performance evaluation on use case 4 in terms of f1-score (values closer to one are preferred). The figure compares the performance of FL, FedProx, CFL and FedProxCFL at selected rounds. For instance, FedProxCFL-10 (0.1) indicates FedProxCFL trained using 10 epochs per round with the proximal term $\mu = 0.1$.

CFL marginally as shown in Fig. 12.A. The results here suggest that while CFL can potentially gain from the integration with the penalty based method of FedProx, the gain is marginal and it comes with the cost of hyperparameter tuning of the proximal term. Table 6 shows that the performance of FedProxCFL as in FedProx depends upon the choice of the proximal term μ.

As reflected in the results, across all use cases, $\mu = 1$ seems to be a poor setting for both FedProx and FedProxCFL in the context of our experiments. This basically highlights that the framework of FedProx is sensitive to the tuning of the hyperparameter μ that controls the degree of model fitness. Additionally, it can be seen that FedProxCFL for $\mu = 1$ performs worse than FedProx for $\mu = 1$. This is due to the fact that for FedProxCFL, there are effectively two mechanisms in parallel controlling over-fitting at the local phase of learning. The first mechanism is enabled by regulating the model parameters through congruent activation

TABLE 7. Complexity analysis for different learning methods.

Computational complexity	FL	FedProx	CFL
MACs	12,660K	12,660K	22,788K

functions, and the second mechanism is enabled by regulating the optimization loss function by setting the proximal term μ which dictates the degree of regularization. Both mechanisms discourage overfitting by slowing down learning. For larger values of μ (such as $\mu = 1$), for the case of FedProxCFL, the local models are under-fitted to a more severe extent than FedProx that only employs the second mechanism of regularization. The key takeaway is that the optimal settings of μ are different for FedProx and FedProxCFL which can be seen in Figs. 11,12 and Table 6.

2) COMPLEXITY ANALYSIS

In terms of computational complexity, when compared to FL, both CFL and FedProx (as an example of a penalty-based method) come with added complexity albeit in two distinct ways. For CFL, the added complexity is at the forward pass (forward propagation) as well as the backward pass (backward propagation) due to the application of the CongruentReLU on the model parameters, whereas for FedProx the added complexity is in the backward pass due to the added cost in performing error backpropagation through the penalty term in the surrogate objective function (2).

Figure 13 compares the computational time for CFL against FL and FedProx in terms of the average time needed for the completion of the forward and backward computations at the training phase. At the backward pass, the error backpropagation, CFL has a higher complexity than FL and roughly similar complexity to FedProx. Even though all methods contain models with the same number of parameters, CFL and FedProx require additional backpropagation steps which add to the computational time. At the forward pass, CFL has a larger computational time than FedProx. Considering computational time needed for the completion of the forward and backward computations, CFL has a larger computational complexity than FedProx and FL at the training phase. However, given that CFL does not require hyperparameter tuning, it might be preferred to FedProx. Additionally, CFL can be applied to the cases where there is no validation set while FedProx would require having access to one.

At the inference which involves a single forward pass, CFL has a larger complexity than FedProx. Table 7 lists the number of Multiply-Accumulate Operations (MACs) at the inference phase for the MLP model used in our experiments computed using the tool developed in [47].

Finally, at each round of federation, the amount of information transferred from the server to the agent and vice versa is the same for all methods.; across all methods, every agent receives only a single copy of the global model.

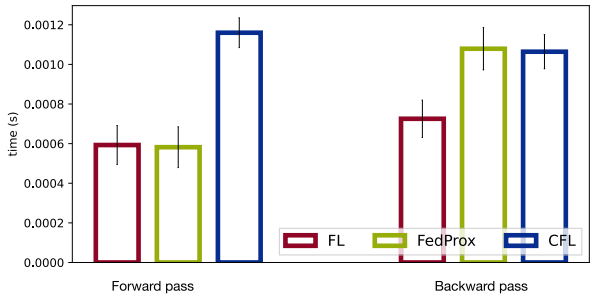


FIGURE 13. Computational time at the forward and backward propagation for different learning methods at the training phase.

VI. DISCUSSION

Federated learning is already becoming prevalent in the networks and has been suggested by 3GPP Release 17 as an alternative to exchanging raw training data amongst network data analytics function (NWDAF) instances [48]. Additionally, its presence in future networks is expected to be indispensable [49]. Given the expected increase in the number of ML tasks and federations in the network, as well as the dynamicity and flexibility in the network, manual search for the best hyperparameters for each model will become infeasible or prohibitively costly. Therefore, automating federated learning is becoming essential and as such there is a need for federated learning methods that are robust to overfitting and can achieve the robustness without reliance on time-consuming and costly hyperparameter tuning that requires availability of representative validation data. In this direction, we introduced an approach for enhancing federated learning in the network and communication domain by eliminating the need to tune the degree of model fitness during local training.

More specifically, we introduced the concept of congruent learning through introduction of congruent activation functions as a class of similarity-promoting parameter activation functions. We hypothesized that congruent learning in the context of federated learning—congruent federated learning—enables a self-taught regularization mechanism that refrains local models from overfitting to the local data, resulting in improved convergence characteristics both in terms of the convergence rate of the federation and the optimality of the final solution.

To validate the essence of our solution, we considered the standard federated averaging, FedAvg, as a baseline, referred to as FL. We then constructed a variant that follows the same approach during the global aggregation phase but differs in using congruent learning at the local learning phase, referred to as CFL. We then empirically evaluated the performance of our approach in handling model complexity in our four use cases across several tasks (a total of 11 regression and classification tasks including 5G and IoT RTT prediction and DC service performance prediction). These use cases allowed us to evaluate our approach in scenarios where data heterogeneity and limited access to labeled data samples for training and validation are the main hurdles for federated learning methods that rely on the hyperparameter tuning.

Additionally, we compared the performance of the proposed solution, CFL, against a penalty-based method, FedProx. The results of this comparative study showed that, in the majority of cases, CFL performs either equally as good or outperforms the best performing FedProx. We also showed that the integration of CFL in FedProx framework, named FedProxCFL, can lead to only a slight gain over CFL provided that the proximal term is well-calibrated.

In our experiments, it was shown that the degree of model fitness (model complexity) at the local learning phase of FL has a consequential effect on the optimality of the global model and convergence characteristics of the learning. Contrary to FL, it was shown that the self-regularization mechanism embedded in CFL greatly reduces overfitting and ,in nearly all cases, improves the performance characteristics, where in some cases, the improvement was considerable (as summarized by the relative gain in Table 5).

The improved results for CFL are achieved with added cost of computational complexity. This is due to application of the congruent activation functions to the optimizable model parameters at all layers. However, the added complexity at the training phase may be justified given that the robustness to overfitting is achieved without reliance on the hyperparameter tuning. Firstly, hyperparameter tuning necessities having access to a representative validation dataset that is reflective of the underlying (oracle) distribution of data; indeed, obtaining such a validation dataset in some 6G use cases is not even possible making hyperparameter tuning impracticable. Secondly, hyperparameter tuning comes with added computational and communication overhead itself.

Regarding communication overhead, and potential negative impact on 6G network performance, congruent federated learning does not increase the amount of information that needs to be transferred at each round of federation. However, it may help reducing communication overhead by relaxing the need for hyperparameter tuning.

We laid out conditions for a parameter activation function to act as a congruent activation function and provided an example of which that satisfies those conditions, named CongruentReLU, given by (4). Although CongruentReLU can be used for different classes of neural networks, other congruent activation functions might be more suitable for specific classes of neural network architectures, such as recurrent neural nets. We leave design of such network-specific congruent activation functions and evaluation of their performance in the framework of congruent federated learning for the future work.

In the analysis of the convergence characteristics of a federated learning solution, there are two main aspects to consider. First, the optimality of the solution after convergence which is indicative of the performance, and second, the convergence rate which indicates how quickly convergence has been achieved. We leave the theoretical study of the convergence characteristics of CFL to the future work. From a purely observational point of view, in our experiments in Section V, it can be seen that CFL achieves faster convergence than FL with improved optimality, reflected in the performance metrics, in the majority of cases but not in all cases. However, theoretical analysis is needed to establish the necessary conditions where CFL is guaranteed to outperform FL.

VII. RELATED WORK

Hyperparameter optimization in federated learning is receiving growing interest, [50], [51], [52], [53], [54]. Authors in [53] discuss challenges in hyperparameter optimization in federated learning with respect to the availability of a federated validation set, resource limitations, and the cost associated with hyperparameter tuning. In another study in [54], authors propose an adaptive method, referred to as AMBLE that aims at adjusting different influential factors on the model complexity at the local phase of training (such as, number of epochs, batch size and learning rate) with the aim of improving the convergence rate and the performance. However, the method assumes existence of a validation set for the purpose of hyperparameter tuning through cross validation. This group of methods aims at construction of frameworks for structured and efficient hyperparameter tuning. While they are not directly related to our work, they highlight the influential role that the hyperparameters play in the overall performance of the federated learning methods. The works in this area can be seen as a motivation towards development of federated learning techniques that do not require extensive hyperparameter tuning.

Overfitting problem is a known challenge in machine learning affecting the generalization capability of the models [55]; in this regard, federated learning is of no exception. There has been recent effort in reducing the overfitting problem in federated learning. Existing solutions in [19], [20], [21], [22], [23], [24], [25], [26], [27], and [28] are based on a class of heuristic techniques known as penalty methods that optimize a surrogate objective function as defined in Section II and in (2). The surrogate objective function relaxes the constrained optimization problem in (1) into an unconstrained one by penalizing the objective function with a penalty parameter that dictates the strength of the penalization. In the following, we provide examples of these solutions. In FedProx [19], the penalty function takes on a simple form and it is the Euclidean norm between the local model parameters and the global (aggregated) model parameters. In Scaffold [20], the penalty function is relatively more involved and it is a measure of the drift between the parameters. Inspired by Scaffold, in FedDC [21], the objective loss relies on a hyperparameter as a control variable that dynamically bridges the gap between the local model and the global model with the learned local drift variable. In MOON [22], the penalty function is a model-contrastive loss added to the local optimization loss measuring similarity between model representations. In ConTre [23] (as in MOON), a contrastive regularization loss is added to the local optimization loss that uses a temperature coefficient to adjust the regularization strength. In FedAlign [24], authors propose a distillation-based regularization that aims at regularizing the Lipschitz

constants of the final block in a network that is the portion of the network most prone to overfitting. This is achieved by adding a penalty term to the local objective function that is the mean squared error between the approximated Lipschitz constant vectors. However, as in other penalty-based techniques, it requires tuning of a penalty parameter that balances out the two competing objectives. In FedBN [25], authors propose use of the local batch normalization to alleviate the feature shift before averaging models where the scaling parameter of the batch normalization requires hyperparameter tuning. Other examples of penalty-based regularization techniques are GradAug [26], StochasticDepth [27], and Mixup [28] which all rely on some penalty parameters that dictate the strength of regularization. Although the class of penalty-based methods could be useful in reducing overfitting, in the scope of this paper that is concerned with automation in federated learning, the need for data-dependant hyperparameter tuning of the penalty parameters is a bottleneck which makes their application limited to the cases where existence of a representative validation dataset can be assumed.

The degree of model fitness at the local phase of federated learning has been previously shown to be an influential factor dictating the optimal properties of the solution. In this regard, in [33], authors carried out experiments using several state-of-the-art federated learning approaches, namely, FedAvg [17], FedProx [19], Scaffold [20], and FedNova [56], on non-IID data sets with different number of local epochs per round. The results show that the number of local epochs can have a large effect on the accuracy of the existing algorithms. The optimal value of the number of local epochs is shown to be particularly sensitive for federation of agents with non-IID data distributions. A similar observation was made in [34] where authors showed that training federated learning models that perform reasonably well in non-IID and heterogeneous settings is challenging. Importantly, it was shown that the performance of the federated model depends largely on the degree of model fitness at the local training phase, determined by the number of training epochs.

VIII. CONCLUSION

Future 6G networks are envisioned to be AI-native with ML functionalities used pervasively as enablers for improving and automating a large variety of network- and service-management tasks and for providing AI capabilities as a service. Further, federated learning is already being considered by 3GPP as a critical component for effective use of distributed resources without the need for sharing data. However, automation is essential for configuration of the vast number of models and their hyperparameters as they may depend on data availability, compute capabilities, model-latency requirements, and network dynamicity.

In this paper, we proposed congruent federated learning as a method of distributed learning that is robust to overfitting and achieves the robustness without reliance on hyperparameter tuning. Learning through congruence was enabled via the introduction of the congruent activation functions as a class of similarity-promoting activation functions that are applied as masks to the parameters of a neural network. Importantly, such parameter activation functions enable a self-taught mechanism that refrains local models from overfitting to their local data via penalizing contrasts in between the parameters of the local models and the global model.

Implementation of the congruent activation functions and their integration into the existing federated learning frameworks require minimal effort. We used the standard federated averaging as an example and constructed a variant of it that uses congruent learning at the local learning phase. We then evaluated both variants on multiple use cases relevant for 6G networks with different degrees and profiles of heterogeneity, namely, service-performance prediction on data traces collected from a data center, end-to-end RTT prediction on data traces from a 5G-mmWave testbed, RTT prediction on data traces from an IoT testbed, and user equipment image classification. When compared to the standard federated learning, in the majority of evaluated scenarios, it was shown that the proposed method of learning can effectively, and automatically, reduce overfitting and improve optimal properties of the solutions resulting in improved performance metrics—a relative gain of about 21% averaged across all use cases.

REFERENCES

[1] R. Boutaba et al., "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *J. Internet Services Appl.*, vol. 9, no. 1, pp. 1–99, Dec. 2018.

[2] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine learning for 6G wireless networks: Carrying forward enhanced bandwidth, massive access, and ultrareliable/low-latency service," *IEEE Veh. Technol. Mag.*, vol. 15, no. 4, pp. 122–134, Dec. 2020.

[3] I. F. Akyildiz, A. Kak, and S. Nie, "6G and beyond: The future of wireless communications systems," *IEEE Access*, vol. 8, pp. 133995–134030, 2020.

[4] L. U. Khan, I. Yaqoob, M. Imran, Z. Han, and C. S. Hong, "6G wireless systems: A vision, architectural elements, and future directions," *IEEE Access*, vol. 8, pp. 147029–147044, 2020.

[5] H. Yang, A. Alphones, Z. Xiong, D. Niyato, J. Zhao, and K. Wu, "Artificial-intelligence-enabled intelligent 6G networks," *IEEE Netw.*, vol. 34, no. 6, pp. 272–280, Nov. 2020.

[6] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Netw.*, vol. 32, no. 2, pp. 92–99, Mar. 2018.

[7] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G networks: Use cases and technologies," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 55–61, Mar. 2020.

[8] M. A. Uusitalo et al., "6G vision, value, use cases and technologies from European 6G flagship project Hexa-X," *IEEE Access*, vol. 9, pp. 160004–160020, 2021.

[9] Z. Zhang et al., "6G wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Veh. Technol. Mag.*, vol. 14, no. 3, pp. 28–41, Sep. 2019.

[10] W. Saad, M. Bennis, and M. Chen, "A vision of 6G wireless systems: Applications, trends, technologies, and open research problems," *IEEE Netw.*, vol. 34, no. 3, pp. 134–142, May 2020.

[11] O. Nassef, W. Sun, H. Purmehdi, M. Tatipamula, and T. Mahmoodi, "A survey: Distributed machine learning for 5G and beyond," *Comput. Netw.*, vol. 207, Apr. 2022, Art. no. 108820.

[12] Y. Liu, X. Yuan, Z. Xiong, J. Kang, X. Wang, and D. Niyato, "Federated learning for 6G communications: Challenges, methods, and future directions," *China Commun.*, vol. 17, no. 9, pp. 105–118, Sep. 2020.

[13] S. A. Khowaja, K. Dev, P. Khowaja, and P. Bellavista, "Toward energy-efficient distributed federated learning for 6G networks," *IEEE Wireless Commun.*, vol. 28, no. 6, pp. 34–40, Dec. 2021.

[14] M. Al-Quraan et al., "Edge-native intelligence for 6G communications driven by federated learning: A survey of trends and challenges," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 7, no. 3, pp. 957–979, Jun. 2023, doi: 10.1109/TETCI.2023.3251404.

[15] O. A. Wahab, A. Mourad, H. Otrok, and T. Taleb, "Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1342–1397, 2nd Quart., 2021.

[16] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016. [Online]. Available: http://arxiv.org/abs/1610.05492

[17] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2017, pp. 1–10.

[18] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput.*, Sep. 2015, pp. 909–910.

[19] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, vol. 2, pp. 429–450, Mar. 2020.

[20] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5132–5143.

[21] L. Gao, H. Fu, L. Li, Y. Chen, M. Xu, and C.-Z. Xu, "FedDC: Federated learning with non-IID data via local drift decoupling and correction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10102–10111.

[22] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 10708–10717.

[23] Z. Chen et al., "Contractible regularization for federated learning on non-IID data," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2022, pp. 61–70.

[24] M. Mendieta, T. Yang, P. Wang, M. Lee, Z. Ding, and C. Chen, "Local learning matters: Rethinking data heterogeneity in federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 8387–8396.

[25] X. Li, M. Jiang, X. Zhang, M. Kamp, and Q. Dou, "FedBN: Federated learning on non-IID features via local batch normalization," 2021, *arXiv:2102.07623*.

[26] T. Yang, S. Zhu, and C. Chen, "GradAug: A new regularization method for deep neural networks," 2020, *arXiv:2006.07989*.

[27] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 646–661.

[28] H. Zhang, M. Cissé, Y. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2017, *arXiv:1710.09412*.

[29] P. Kairouz et al., "Advances and open problems in federated learning," 2021, *arXiv:1912.04977*.

[30] T. Li, A. K. 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.

[31] S. Zawad et al., "Curse or redemption? How data heterogeneity affects the robustness of federated learning," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 10807–10814.

[32] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-IID data with reinforcement learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 1698–1707.

[33] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-IID data silos: An experimental study," 2021, *arXiv:2102.02079*.

[34] X. Lan et al., "Federated learning for performance prediction in multi-operator environments," *ITU J. Future Evolving Technol.*, vol. 4, no. 1, pp. 166–177, 2023.

[35] R. Yanggratoke et al., "A service-agnostic method for predicting service metrics in real-time," *Int. J. Netw. Manage.*, vol. 28, no. 2, p. e1991, 2018, doi: 10.1002/nem.1991.

[36] F. S. Samani. (2021). *Data Traces for Efficient Learning on High-Dimensional Operational Data*. [Online]. Available: https://github.com/foroughsh/KTH-traces

[37] A. Rao et al., "Prediction and exposure of delays from a base station perspective in 5G and beyond networks," in *Proc. ACM SIGCOMM Workshop 5G Beyond Netw. Meas., Modeling, Use Cases*. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 8–14.

[38] *NR; Physical Layer Procedures for Data*, 3GPP document Version 16.5.0, 3rd Generation Partnership Project (3GPP), Technical Specification (TS), 2021.

[39] *NR; Physical Layer Measurements*, 3GPP document Version 16.5.0, 3rd Generation Partnership Project (3GPP), Technical Specification (TS), 2021.

[40] *NR; Medium Access Control (MAC) Protocol Specification*, 3GPP document Version 16.5.0, 3rd Generation Partnership Project (3GPP), Technical Specification (TS), 2021.

[41] J. Postel. (1981). *Internet Control Message Protocol*. IETF RFC 792. [Online]. Available: https://www.rfc-editor.org/rfc/rfc792.txt

[42] D. C. Nguyen et al., "6G Internet of Things: A comprehensive survey," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 359–383, Jan. 2022.

[43] M. Schuss, C. A. Boano, M. Weber, and K. Römer, "A competition to push the dependability of low-power wireless protocols to the edge," in *Proc. EWSN*, 2017, pp. 54–65.

[44] K. Hedayat, R. M. Krzanowski, A. Morton, K. Yum, and J. Babiarz, *A Two-Way Active Measurement Protocol (TWAMP)*, document RFC5357, 2008, pp. 1–26.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2015, *arXiv:1412.6980*.

[46] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," Aug. 2017, *arXiv:1708.07747*, doi: 10.48550/arXiv.1708.07747.

[47] V. Sovrasov. (2023). *ptflops: A FLOPs Counting Tool for Neural Networks in PyTorch Framework*. [Online]. Available: https://github.com/sovrasov/flops-counter.pytorch

[48] *Study on Enablers for Network Automation for the 5G System (5GS); Phase 2*, Standard 3GPP Version 17.0.0, 3rd Generation Partnership Project (3GPP), Technical Specification Group Services and System Aspects, 2020.

[49] E. Muscinelli, S. S. Shinde, and D. Tarchi, "Overview of distributed machine learning techniques for 6G networks," *Algorithms*, vol. 15, no. 6, p. 210, Jun. 2022.

[50] H. Zhang, M. Zhang, X. Liu, P. Mohapatra, and M. DeLucia, "FedTune: Automatic tuning of federated learning hyper-parameters from system perspective," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Nov. 2022, pp. 478–483.

[51] Y. Zhou, P. Ram, T. Salonidis, N. Baracaldo, H. Samulowitz, and H. Ludwig, "FLoRA: Single-shot hyper-parameter optimization for federated learning," 2021, *arXiv:2112.08524*.

[52] Z. Wang, W. Kuang, C. Zhang, B. Ding, and Y. Li, "FedHPO-B: A benchmark suite for federated hyperparameter optimization," 2022, *arXiv:2206.03966*.

[53] M. Khodak et al., "Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 19184–19197.

[54] J. Park, D. Yoon, S. Yeo, and S. Oh, "AMBLE: Adjusting mini-batch and local epoch for federated learning with heterogeneous devices," *J. Parallel Distrib. Comput.*, vol. 170, pp. 13–23, Dec. 2022.

[55] R. Roelofs et al., "A meta-analysis of overfitting in machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2019.

[56] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 33, Dec. 2020, pp. 7611–7623.