

FedSL: Federated Split Learning on Distributed Sequential Data in Recurrent Neural Networks

ALI ABEDI and SHEHROZ S. KHAN, KITE, University Health Network, Canada

Federated Learning (FL) and Split Learning (SL) are privacy-preserving Machine-Learning (ML) techniques that enable training ML models over data distributed among clients without requiring direct access to their raw data. Existing FL and SL approaches work on horizontally or vertically partitioned data and cannot handle sequentially partitioned data where segments of multiple-segment sequential data are distributed across clients. In this paper, we propose a novel federated split learning framework, FedSL, to train models on distributed sequential data. The most common ML models to train on sequential data are Recurrent Neural Networks (RNNs). Since the proposed framework is privacy preserving, segments of multiple-segment sequential data cannot be shared between clients or between clients and server. To circumvent this limitation, we propose a novel SL approach tailored for RNNs. A RNN is split into sub-networks, and each sub-network is trained on one client containing single segments of multiple-segment training sequences. During local training, the sub-networks on different clients communicate with each other to capture latent dependencies between consecutive segments of multiple-segment sequential data on different clients, but without sharing raw data or complete model parameters. After training local sub-networks with local sequential data segments, all clients send their sub-networks to a federated server where sub-networks are aggregated to generate a global model. The experimental results on simulated and real-world datasets demonstrate that the proposed method successfully train models on distributed sequential data, while preserving privacy, and outperforms previous FL and centralized learning approaches in terms of achieving higher accuracy in fewer communication rounds.

CCS Concepts: • **Computing methodologies** → **Distributed algorithms**; **Distributed artificial intelligence**.

Additional Key Words and Phrases: federated learning, split learning, recurrent neural network, distributed sequential data

ACM Reference Format:

Ali Abedi and Shehroz S. Khan. 2020. FedSL: Federated Split Learning on Distributed Sequential Data in Recurrent Neural Networks. In ., ACM, New York, NY, USA, 23 pages.

1 INTRODUCTION

In conventional Machine-Learning (ML) algorithms, models are usually developed in a centralized setting, i.e. the data generated by local clients are firstly aggregated into a server or PC, and the training is performed on the server or PC. However, the client's personal data is privacy-sensitive and can be large in quantity, which disqualifies centralized ML methods from sending local data to the server. Federated Learning (FL) [21] and Split Learning (SL) [10] are distributed collaborative ML techniques that decouple model training from the need for direct access to the raw data of clients.

FL [21] enables to collaboratively train a shared model by aggregating locally trained models in clients using their local privacy-sensitive data. On the other hand, SL [10] trains a model, e.g. a neural network, where the network is split into two sub-networks; one sub-network is trained on

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

client and the other is trained on server. Existing FL and SL approaches are applicable to horizontally [12, 15, 21, 30] or vertically [5, 7, 8, 19] partitioned data, and cannot handle sequentially partitioned data occurring frequently in practice. In sequentially partitioned data, where training sequences are sequentially distributed across clients, consecutive segments of multiple-segment sequential data are generated and available on different clients.

As a concrete example of sequentially partitioned data, suppose one patient is consecutively admitted to two different hospitals each recording sequential vital signs of the patient [23], e.g. heart rate. In this case, the heart rate of the patient is sequentially distributed across two hospitals/clients. This multiple-segment sequential data can be used to train ML models to predict, for instance, mortality of patient. The most common ML models to train on sequential data are Recurrent Neural Networks (RNNs). Due to privacy concerns, the clients are not allowed to share the segments of sequential data. One solution to decouple training RNN from segment sharing between clients is to split RNN into sub-networks each being trained on single clients containing single segments of sequential data. Existing SL approaches work on feed-forward neural networks, and cannot be used for RNNs [1]. RNNs, in order to train on sequentially partitioned data, should be split in a way that preserve latent dependencies between segments of multiple-segment sequential data available on different clients.

In this paper, we present FedSL, a novel collaborative ML framework utilizing FL, along with a novel SL approach tailored for RNNs, to take advantage of both approaches and to train models on sequentially partitioned data, while preserving privacy of clients' local data. Due to privacy-preserving property of FL and SL approaches, data sharing between clients and also between clients and server is not allowed. Therefore, in FedSL, model training is performed locally on each client in collaboration with other clients and with the server. SL is performed between clients, and FL is performed between clients and server. A RNN is split into two sub-networks to be trained with sequential training samples with multiple segments available on different clients. The first sub-network is trained with the first segment in one client, and the second sub-network is trained using the second segment in another client. During SL, these sub-networks of the split RNN, on different clients, collaborate and communicate to learn from their local segments without sharing data or model parameters to other clients. These rounds of SL are repeated whenever new training sequential segments are generated in consecutive clients. After each round of SL, all clients send their locally trained sub-networks to the server where all received sub-networks are aggregated. These aggregated sub-networks are used as the initial model for the next rounds of FedSL. This process continues until the server acknowledges the convergence. Our primary contributions in this paper are summarized as follows:

- We are first to define and address the problem of training models in a federated setting with multiple-segment sequential data distributed across clients. This data distribution is different from vertical and horizontal partitioned data [30], and none of the previously presented FL or SL approaches have studied this data-partitioning structure [12, 15, 16, 30].
- To address the above problem, we propose a novel architecture that integrates FL and SL. In this architecture, none of the *data*, or *label*, or *complete model parameters* are shared between clients or between clients and the server.
- We propose a novel SL approach for RNNs. Consecutive splits of RNNs are trained on consecutive clients containing consecutive segments of multiple-segment sequential data distributed across clients.
- We conduct extensive experiments on simulated and real-world sequential multiple-segment datasets, showing that our proposed framework successfully extracts patterns and train

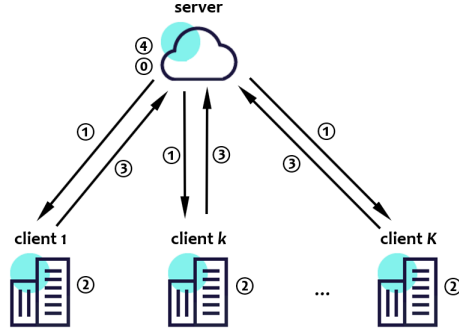


Fig. 1. The vanilla FL framework, ① server initializes a global model, and ① sends the global model to all participating clients. ② each client updates the received model using its local data, and ③ sends the updated model back to the server. ④ the server aggregates all received models to generate the updated version of the global model. ①-④ continue until convergence. see Section 2.1.

sequential models, while preserving privacy, and outperforms previous FL and centralized learning approaches in terms of achieving higher accuracy in fewer communication rounds.

The paper is structured as follows. Section 2 introduces background and related works on FL and SL. In Section 3, the proposed pathway for analyzing distributed sequential data through splitting RNNs is presented. Section 4 describes experimental settings and results on the proposed methodology. In the end, Section 5 presents our conclusions and directions for future works.

2 BACKGROUND AND LITERATURE REVIEW

In this Section, after briefly explaining FL and SL, we discuss their related work, differences, and various ways these two frameworks are integrated in previous works.

2.1 Federated Learning

FL is a collaborative distributed learning paradigm firstly developed by Google [21] aiming to train ML models on distributed devices having locally generated training samples. There are three major components in a FL setting, K participants (clients), a curator (server), and the communication-computation framework in which clients and the server collaborate to train the ML models, Figure 1.

Figure 1 depicts different steps of the vanilla FL algorithm [21]. At the starting round of the FL ($t = 0$), ① the curator server initializes a global model W_0 randomly or using a small number of training samples which are similar to the data generated by clients, and ① sends this initial model to selected clients participating in the current round of FL. After receiving the initial model, ② each client k starts training, updating the initial model, using its local training samples. Then, ③ each client sends the updated model back to the server. ④ the server aggregates all received models to generate the updated version of the global model and get W_{t+1} . Each time these local model updates by clients, model aggregation by server, and communication between them are performed is called one round of FL. These rounds of computation-communication continue until the server acknowledge the global model to be converged.

In the vanilla FL algorithm [21], the model aggregation on the server is called Federated Averaging (FedAvg) where the global model is generated by averaging local model updates [21]

$$W_{t+1} = \sum_{k=1}^K \frac{n_k}{n} W_{t+1}^k \quad (1)$$

n_k is the number of training samples in client k , and n is the total number of samples in all the K clients. Over the entire rounds of FL, only the model parameters are communicated between clients and the server. Therefore, the clients do not require to share their raw training data or labels to the server or to other clients. The local data remains local and confidential which makes FL a privacy-preserving ML algorithm.

From data-distribution point of view, existing FL approaches are categorized into vertical and horizontal, [12, 15]. The horizontal FL approaches focus on analyzing horizontal data structure in which training samples in different clients are represented by the same features, and training sample IDs in clients are unique, i.e., there is not different versions of one sample ID in different clients. Contrarily, in vertical FL, different clients contain different versions of same training sample IDs, represented by different features.

Most existing FL approaches are in the category of horizontal FL [12, 15, 21, 30]. The vanilla FL (FedAvg) approach [21], described in this subSection, is a horizontal FL approach. Next-word prediction [11] and Emoji prediction [24] in virtual keyboards in smartphones are two well-known applications of horizontal FedAvg where Long Short-Term Memory (LSTM) RNNs are trained on local mobile devices using text data generated by clients in a federated setting to train personalized models for next-word and emoji prediction on Google virtual keyboard app (Gboard). While the training samples on clients share the same features, i.e., all clients generate text data by their Gboards, they do not share sample IDs (If we assume each user with a unique ID uses one single smartphone in which (s)he types text using Gboard).

Chen et al. [5] proposed a vertical asynchronous FL (VAFL) approach which allows clients to participate in FL asynchronously without coordination with other clients. As it is a privacy-preserving approach, instead of sharing raw data of clients to the server, a local embedding, a mapping reducing the dimensionality of the raw data, is sent to the server, and client's local model updating is performed with collaboration with the server, independent of other clients. Different classification models were used in VAFL setting, including Logistic Regression (LR), Convolutional Neural Networks (CNNs), and LSTMs on various image and Electronic Health Record (EHR) datasets.

Cheng et al. [7] proposed a vertical FL for training gradient boosting decision trees using vertically partitioned training datasets. A privacy-preserving entity alignment method is used to find common users in different clients. They used financial datasets and trained their models in the vertical federated setting to solve binary classification problems. For more information on different horizontal and vertical FL approaches, see [12, 15, 16, 30].

Other important points about data distribution in FL are the balance of data among clients, and if the data is independent and identically distributed (IID) among clients. The data is unbalanced, if different users generate and participate in FL with various amounts of local training data. The data is non-IID, if clients contain samples in one or a small number of classes and do not contain samples of other classes. Imbalanced and non-IID data is considered to be a challenging problem in FL in terms of test accuracy and communication efficiency [17, 21]. Many works have been done to address the non-IID data issue in FL [12]. For instance, Xiang et al. [17] proposed an improved version of FedAvg with data sharing in which distributing a small amount of holdout IID data between clients results in a significant improvement in the performance of FL on non-IID data. Briggs et al. [4] proposed to separate clients in clusters based in the similarity of their locally trained models to the global model, and then train clusters of clients independently. In this way, higher accuracy on non-IID data is achieved compared to the conventional FL.

As the main goal of FL is to train a ML model, all participating clients train their own models which will be aggregated on the server. Based on the application and datasets, different models are utilized in FL setting, including, LR [5], decision trees [7], 1-D CNNs [1, 9], 2-D CNN [21, 28], RNNs [11, 21, 24] (when dealing with privacy-sensitive distributed sequential data), and more sophisticated deep convolutional neural networks [6, 12, 28]. In addition, in many FL settings, transfer learning is performed on client side, i.e. clients start from a pre-trained model and update that model by their local data [6].

While, most FL frameworks work based on Stochastic Gradient Descent (SGD) optimization, some other methods proposed modified versions of SGD to improve learning performance. In FedAvg [21], clients run local SGD for a predetermined number of epochs. A modified version of FedAvg, LoAdaBoost, is presented in [13] where for each client, after a certain number of epochs, if the local loss is higher than a threshold, the local epochs of training continue to decrease the local loss, otherwise local training finishes. Another modification of FedAvg is presented by Baheti et al. in [2] in which the weights in FedAvg are modified based on the local loss of clients. The clients with lower local loss will have greater weights in FedAvg (see Equation 1).

One of the main reasons for avoiding centralized learning and turn to FL is to preserve clients' privacy. Although in FL the raw data of clients remains on local devices, sending complete model parameters to the network is not safe and may lead to sensitive data leakage. There are mechanisms which can derive the raw data having the complete model parameters. In differentially private FL approaches [12, 15, 30], adding noises, random sub-sampling, and distorting local data or the model parameters, increases differential privacy and protects FL framework against inference attacks.

2.2 Split Learning

SL is a collaborative privacy-preserving technique in which the model (e.g. neural network) is split into at least two sub-networks, client-side and server-side sub-networks, [10]. In FL, each client trains a complete model using its local data. While, in SL, each of the client and server trains a partial model (sub-network). The client-side sub-network is trained on client device where the local training data exists, and the server-side sub-network is trained on the server. Due to privacy concerns, the server has no direct access to raw data or model details generated at client-side. Since each client only trains a sub-network containing a few layers of the complete network, the computational burden on clients is reduced compared to FL.

The vanilla SL algorithm is firstly presented in [10]. From one specific layer, called split layer or cut layer, the neural network is split into two sub-networks. The client, containing training data, performs forward propagation and computes the output of its sub-network, up to the cut layer. The cut layer's output is sent to the server. The server computes the final output, until the last layer of the network. At server's sub-network, the gradients are back propagated from the last layer to the cut layer, and the gradient of the cut layer is sent back to the client. The client performs the rest of back propagation process from the cut layer to the first layer of the network. This process continues until the client has new training data.

In the above described vanilla SL architecture [1, 9, 10, 28], the server has no direct access to the raw data of clients, and also complete model parameters are not sent to the server. The only information being communicated between clients and the server is the output of the cut layer, from client to server, and the gradient of cut layer, from server to clients. However, one drawback of the vanilla SL [10] is that the labels of training samples need to be transmitted from clients to server. In [29], a U-shaped configuration for SL is presented to alleviate the problem of label sharing in SL, where four sub-networks are used to make to model training possible without label sharing.

In Section 2.1, FL algorithms were studied from data-partitioning structure point of view. SL algorithms can also be described based on the structure of data they analyze. The vanilla SL [10]

and U-shaped SL approaches [29] described above are suitable for horizontal data where training samples in different clients share the same feature space but do not share the same sample ID space. In contrast, vertical SL approaches deal with data structures in which different features of the same training samples are available at different clients. Vepakomma et al. [29] proposed a vertical SL configuration in which two clients containing different modalities of same training samples train their specific sub-networks up to the cut layer. Then, the outputs of their sub-networks are concatenated and sent to the server. The server performs forward and backward propagation and sends gradient to each of clients to complete the backward propagation and train the overall network.

Gupta and Raskar in [10] evaluated the performance of horizontal SL with one or more clients collaborating with one server using various image classification datasets. They achieved classification accuracy comparable to the centralized learning using different feed-forward 2D CNNs. According to their experiments, participating more clients in SL causes accuracy to improve significantly. Abuadbbba et al. in [1] examined application of SL in 1D CNNs for detecting heart abnormalities using medical ECG data and to classify speech signals. They concluded that 1D CNN classifiers under SL are able to achieve the same accuracy as centralized learning.

2.3 Comparing Federated Learning with Split Learning

FL and SL approaches have their own benefits and shortcomings; while SL better address privacy concerns because of splitting model architecture between clients and the server and also due to not sharing complete model parameters between participants, model training in FL is relatively faster and need less communication compared to SL due to the parallel and simultaneous model updates in clients.

Based on the experiments in [9], evaluating classification performance of the vanilla FL [21] and SL [10] on ECG and speech signals, SL achieves better classification accuracy than that of FL when data is IID, but works worse than FL under extreme non-IID data distribution. They also concluded that by increasing the number of participating clients, FL converges slower than SL and needs more local epochs per client.

In [29], experiments on CIFAR-10 and CIFAR-100 datasets using deep CNNs show significant reductions in client-side computations and communication bandwidth when using SL compared to the vanilla FL [21]. The reason for the reduction in client-side computation is that while vanilla FL requires computation of forward and backward propagations for the entire neural network on clients-side, SL needs these computations for only the first few layers of neural network up to the cut layer. By the same token, the reduction in transmitted data is due to the fact that in the vanilla FL the parameters of the entire neural network are transmitted, while in SL only the activations and gradients of the cut layers are communicated between clients and the server

Singh et al. in [27] compared communication efficiency of SL and FL and reported that while SL is more communication efficient with increasing the number of clients, FL becomes more efficient with increasing the number of data samples especially when the number of clients or model size is small.

It is noteworthy that the works in this subSection compared different SL architectures with the vanilla FL [21]. However, as described in Section 2.1, many updates on the original FL have been provided. These updates significantly improved the communication and computation efficiency, convergence speed, and privacy of the vanilla FL [12].

2.4 Integrating Federated Learning with Split Learning

In this subSection, we study a method integrating FL and SL to take advantage of the both approaches. Thapa et al. [28] proposed SplitFed framework combining FL and SL to train models on horizontally

partitioned data. This framework splits a neural network between each participating client and a *main server*. The clients carry out SL with the main server to train their local models. Then, the clients send their local models to a *fed server* which conducts federated averaging of the received models to create a global model. In this configuration, the label sharing is needed, i.e. all clients send the label of their local data to the main server to be able to perform SL. The SplitFed has been evaluated using different deep CNNs for image classification. It achieved classification accuracy very close to that of FL and SL, while being faster than SL. The SplitFed has the same communication efficiency as SL, and an improved communication efficiency compared to FL.

Based on the descriptions of the related works, Table. 1 briefly compares different FL and SL approaches from different points of view. The previous FL [5, 8, 21] and SL [1, 10] methods work on horizontally or vertically partitioned data. Our method, FedSL (described in Section 3), is the first framework that handles sequentially partitioned data in a privacy-preserving manner. Previous SL methods split feed-forward neural networks [1, 10, 28], while FedSL splits recurrent neural networks. In the SplitFed method [28], SL and FL are combined to work on horizontally partitioned data. In SplitFed, after SL between clients and a main server, complete model parameters are sent from clients to a fed server. It is against the main goal of SL which is to avoid complete model sharing among clients and server. In addition, in the SL step of SplitFed, label sharing is required among clients and the split server. However, in FedSL, the complete model is not shared and label sharing is not required between clients, or between clients and server.

In this paper, we split recurrent neural networks into sub-networks on individual clients to analyze distributed sequential data without data sharing. Then, these sub-networks collaborate and aggregate in a federated setting to generate global trained neural networks. In this way, we will take advantage of both SL and FL approaches and compensate their shortcomings.

Table 1. Comparing FL [21], VAFL [5], MMVFL [8], SL [10], 1D-CNN SL [1], SplitFed [28], and our proposed method, FedSL (described in Section 3), from different perspectives.

Method	Data partitioning	Label sharing	Model aggregation	Model split	Split neural network
FL [21]	Horizontal	No	Yes	No	-
VAFL [5], MMVFL [8]	Vertical	Yes	Yes	No	-
SL [10]	Horizontal	Yes	No	Yes	Feed-forward (2D-CNN)
1D-CNN SL [10]	Horizontal	Yes	No	Yes	Feed-forward (1D-CNN)
SplitFed [28]	Horizontal	Yes	Yes	Yes	Feed-forward (2D-CNN)
FedSL (proposed)	Sequential	No	Yes	Yes	Sequential (RNN)

3 FEDSL - A NOVEL FEDERATED SPLIT LEARNING FRAMEWORK

In this Section, after defining the problem of training models on multiple-segment sequential data distributed among clients in Section 3.1, SL in RNNs is introduced in Section 3.2, and finally the overall FedSL framework for analyzing sequentially partitioned data is presented in Section 3.3.

3.1 Problem Definition

As an example of a system generating privacy-sensitive sequentially partitioned data, suppose hospitals as clients participating in collaborative ML. Multiple-segment sequences of vital signs of each single patient is considered as one sequential training sample with a unique patient/sample ID. Consecutive segments of a sequential training sample are distributed among hospitals/clients, if the

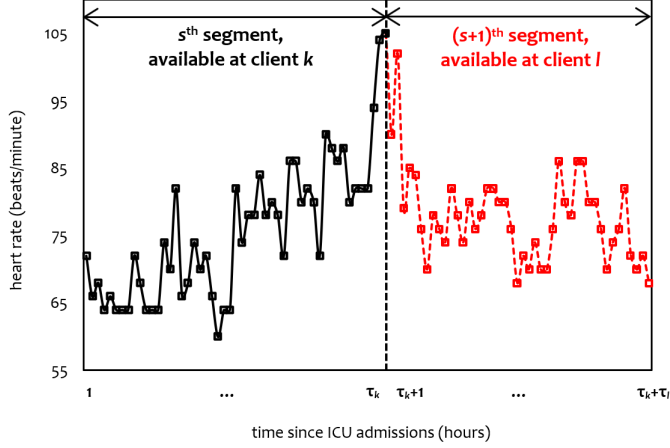


Fig. 2. An example of a sequentially partitioned data distributed among clients, (artificially generated) heart rate of one patient (one training sample X_j) admitted to two different hospitals (two clients, k and l). The s^{th} segment (time steps 1 to τ_k) of X_j is generated and is available in client k , and the $(s+1)^{\text{th}}$ segment (time steps $\tau_k + 1$ to $\tau_k + \tau_l$) of X_j is in client l . see Section 3.1

corresponding patient has multiple admissions to the hospitals. Figure 2 exemplifies a sequentially partitioned data, heart rate of one patient (one training sample, X_j) admitted to two different hospitals (two clients). The s^{th} segment (time steps 1 to τ_k) of the sequential training sample X_j is generated and is available in client k , and the second segment (time steps $\tau_k + 1$ to $\tau_k + \tau_l$) of X_j is in client l .

In general, there are K clients generating consecutive segments of multiple-segment sequential data. The j^{th} distributed multiple-segment sequential data which its two consecutive segments k and l are available at clients k and l , is denoted by $X_j = \{\dots, X_{j_s}^k, X_{j_{s+1}}^l, \dots\}$. The two consecutive segments in X_j are sequences of length τ_k and τ_l , $X_{j_s}^k = \{x_j^1, \dots, x_j^{\tau_k}\}$, and $X_{j_{s+1}}^l = \{x_j^{\tau_k+1}, \dots, x_j^{\tau_k+\tau_l}\}$. X_j can be rewritten as

$$X_j = \{\dots, x_j^1, \dots, x_j^{\tau_k}, x_j^{\tau_k+1}, \dots, x_j^{\tau_k+\tau_l}, \dots\}. \quad (2)$$

The index j is the sample ID of the training sample X_j . The index k is the segment ID of the segment $X_{j_s}^k$ which means the s^{th} segment is generated and is available at client k , Figure 2.

Each multiple-segment sequence X_j has a single label Y_j , being available at the client l , containing the last segment of the sequence. For instance, in the problem of in-hospital mortality prediction (see Section 4.2), the last hospital knows if the patient is alive or dead.

An ID bank is available on the curator server, containing a set of sample IDs $S_{\text{ID}} = \{0, 1, \dots, j, \dots\}$ and sets of segment IDs $S_{\text{segment}_j} = \{0, 1, \dots, k, \dots\}$. These sets of IDs are updated at each round of communication between clients and the server when the new generated training samples on clients are used in FL process, see Section 3.3. For instance, in the current round of FL, the segment $X_{j_{s+1}}^l$ is generated in the client l . The client l checks the sample ID j with the set of sample IDs S_{ID} in the ID bank:

If j is not in S_{ID} , j is added to S_{ID} , and $X_{j_{s+1}}^l$ is considered as the first segment of the multiple-segment sequence X_j . In addition, the set S_{segment_j} is created with one element l , $S_{\text{segment}_j} = \{l\}$, which means the training sample with ID j has had one segment so far, on client l .

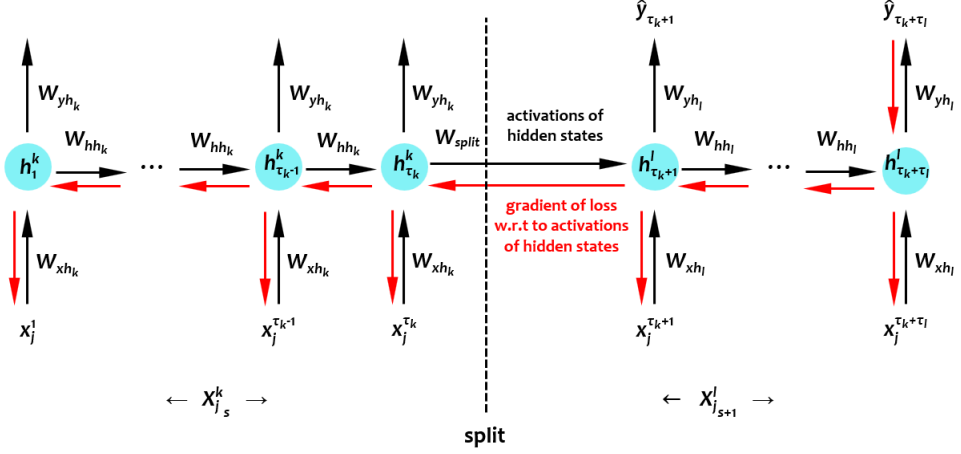


Fig. 3. SL in RNNs. Two consecutive segments of a multiple-segment sequence X_j ($X_{j_s}^k$ and $X_{j_{s+1}}^l$) are available at clients k and l . From the dashed line, the RNN is split into two sub-networks. The left-side sub-network is trained on client k , while the right-side sub-network is trained on client l . The two sub-networks communicate to perform forward (black arrows) and backward (red arrows) propagations and update their weights. The left-side sub-network sends the activations of its hidden states to the right-side sub-network, while the right-side sub-network sends the gradient of the loss w.r.t. the activations of the hidden states of the left-side sub-network to the left-side sub-network. x 's, y 's, h 's, and w 's are inputs, outputs, hidden states, and weights of RNN at different time steps, respectively (see Section 3.2).

If j is in S_{ID} , l is added to $S_{segment_j}$, and $X_{j_{s+1}}^l$ is considered as the latest segment (up to now) of the multiple-segment sequence X_j . It should be noted that in the problem defined in this subsection, only sample IDs (j) and segment IDs (k) are shared between clients and the ID bank on the server, and no label sharing (Y_j) or raw-data sharing (X_j) is required.

3.2 Split Learning in Recurrent Neural Networks

Our aim is to train ML models, in a privacy-preserving setting, on multiple-segment sequential data with segments distributed among clients (see Section 3.1). The common ML models for dealing with sequential data are RNNs, such as the vanilla RNNs, GRUs, LSTMs, etc. [26]. We introduce SL for RNNs. A RNN is split into sub-networks each being trained on one client and with one segment of multiple-segment training sequences.

In the previous SL algorithms (see Sections 2.2 to 2.4), the split layer is a feed-forward layer. From a feed-forward layer, a feed-forward neural network is split between clients (containing training data) and server (containing training labels). However, RNNs have sequential (recurrent) structure to analyze sequential data. In the following, we explain how RNNs are split from recurrent layers.

In RNNs, recurrent connections between hidden states create a directed graph along a temporal sequence which enables RNNs to model temporal sequential behavior (see Figure 3). Likewise, in our problem, segments of a multiple-segment sequential data ($X_{j_s}^k$ and $X_{j_{s+1}}^l$) are temporally related and continuation of each other, all together constituting one sequence X_j (see Section 3.1). In order to model this temporal relation between segments, all segments should be given to one RNN. However, segments are available on different clients. In addition, in privacy-preserving SL algorithms, sharing segments between clients is not allowed. Therefore, A RNN (from the connections between hidden states) is split into sub-networks, each being trained on one client

with one segment. The sub-networks communicate with each other through their hidden states in order to preserve the temporal relation between segments available on different clients.

Figure 3 illustrates how a multiple-segment sequential data, available on different clients (See Figure 2 and Equation 2), is given to an (unsplit) RNN. X_j contains multiple segments, and its two consecutive segments $X_{j_s}^k$ of length τ_k , and $X_{j_{s+1}}^l$ of length τ_l are available at clients k , and l , respectively. These two segments can be considered as a result of a split in X_j between time steps τ_k and $\tau_k + 1$. Likewise, we split RNN from the hidden layer between hidden states $h_{\tau_k}^k$ and $h_{\tau_k+1}^l$. The split layer is indicated by a dashed line and its weight is named W_{split} in Figure 3. This split results in two sub-networks, the sub-network at the left side of the split line is trained on client k with $X_{j_s}^k$, while the sub-network at the right side of the split line is trained on client l with $X_{j_{s+1}}^l$.

The values of the hidden state, output, loss, and the gradient of loss w.r.t. W_{split} at time step $\tau_k + 1$, in the right-side sub-network (on client l) are computed as follows

$$h_{\tau_k+1}^l = \varphi_h(W_{\text{split}} h_{\tau_k}^k + W_{xh_l} x_j^{\tau_k+1}) \quad (3)$$

$$\hat{y}_{\tau_k+1} = \varphi_y(W_{hy_l} h_{\tau_k+1}^l) \quad (4)$$

$$L_{\tau_k+1} = \text{loss}(\hat{y}_{\tau_k+1}, y_{\tau_k+1}) \quad (5)$$

$$\frac{\partial L_{\tau_k+1}}{\partial W_{\text{split}}} = \frac{\partial L_{\tau_k+1}}{\partial \hat{y}_{\tau_k+1}} \frac{\partial \hat{y}_{\tau_k+1}}{\partial h_{\tau_k+1}^l} \frac{\partial h_{\tau_k+1}^l}{\partial W_{\text{split}}} \quad (6)$$

The first two derivatives in the right-hand side of Equation 6 are computable by the inputs, activations, weights, and the labels in the right-side sub-network (on client l). However, the third derivative $\partial h_{\tau_k+1}^l / \partial W_{\text{split}}$ equals to $h_{\tau_k}^k \varphi'_h(W_{\text{split}} h_{\tau_k}^k + W_{xh_l} x_j^{\tau_k+1})$ and is dependent to $h_{\tau_k}^k$ in the left-side sub-network. Therefore, the right-side sub-network needs the activation of hidden state of the left-side sub-network, $h_{\tau_k}^k$. By the same token, the left-side sub-network needs the gradient of the loss in the right-side sub-network w.r.t. the activation of hidden state of the left-side sub-network, $\partial L_{\tau_k+1} / \partial h_{\tau_k}^k$. Therefore, the two sub-networks need to communicate with each other to be able to perform forward and backward propagations, calculate the gradients, and update their parameters.

Note that the above gradient computations and the Equations 3 to 6 in split RNN are different from the computations in a conventional RNN resulting in the Back Propagation Through Time (BPTT) equations [18]. Nevertheless, the BPTT equations are true for each sub-network separately.

Based on the above computations, Algorithm 1 describes SL in RNNs. The input to the algorithm is multiple-segment sequential data $X_j = \{\dots, X_{j_s}^k, X_{j_{s+1}}^l, \dots\}$ with consecutive segments distributed among clients, and the labels Y_j being available at clients containing the last segments. Client k contains s^{th} segments $X_{j_s}^k$ and trains sub-network W_s^k , while client l contains $(s+1)^{\text{th}}$ segments $X_{j_{s+1}}^l$ and trains sub-network W_{s+1}^l . f_k, f'_k, f_l , and f'_l denote forward and backward propagation computations in clients k and l , respectively.

Without loss of generality, in Algorithm 1, we assume that $X_j = \{X_{j_s}^k, X_{j_{s+1}}^l\}$, i.e., client k contains the starting segment of X_j with no previous segments, and client l contains the last segment of X_j with no next segments. While the vanilla RNN is considered in Algorithm 1, the proposed SL method is applicable to other RNNs, including GRUs, LSTMs, etc. Likewise, it is expandable to multi-layer and bidirectional RNNs. In this way, we will be able to train RNNs on sequentially partitioned data across clients without compromising privacy. Instead of sharing raw data or model

parameters between clients, the only information being transmitted among clients is the activations of the hidden states and the gradient of the loss w.r.t. the activations of the hidden states.

Algorithm 1: Split Learning for Recurrent Neural Networks

while client k has new s^{th} segments and client l has new $(s + 1)^{\text{th}}$ segments of sequential training data **do**

Client k executes:

- 1) Initialize weights W_s^k randomly
- 2) Initialize hidden state $h_{\tau_k}^k$ randomly
- 3) Compute forward propagation to get the new hidden state value,
 $h_{\tau_k}^k \leftarrow f_k(X_{j_s}^k, h_{\tau_k}^k, W_s^k)$
- 4) Send $h_{\tau_k}^k$ to client l

Client l executes:

- 5) Initialize weights W_{s+1}^l randomly
- 6) Initialize hidden state $h_{\tau_{k+1}}^l$ with the received hidden state from client k , $h_{\tau_{k+1}}^l \leftarrow h_{\tau_k}^k$
- 7) Compute forward propagation to get the output at last time step,
 $\hat{y}_{\tau_k+\tau_l} \leftarrow f_l(h_{\tau_{k+1}}^l, X_{j_{s+1}}^l, W_{s+1}^l)$
- 8) Having the label, compute the value of loss at the last time step,
 $L_{\tau_k+\tau_l} = \text{loss}(\hat{y}_{\tau_k+\tau_l}, y_{\tau_k+\tau_l})$
- 9) Compute the gradient of loss w.r.t W_{s+1}^l , $\frac{\partial L_{\tau_k+\tau_l}}{\partial W_{s+1}^l} \leftarrow f'_l(h_{\tau_{k+1}}^l, X_{j_{s+1}}^l, W_{s+1}^l)$
- 10) Update $(s + 1)^{\text{th}}$ model parameters in client l , $W_{s+1}^l \leftarrow W_{s+1}^l - \eta \frac{\partial L_{\tau_k+\tau_l}}{\partial W_{s+1}^l}$ (η is the learning rate)
- 11) Compute the gradient of loss w.r.t the hidden state in client k , $\frac{\partial L_{\tau_k+\tau_l}}{\partial h_{\tau_k}^k}$
- 12) Send $\frac{\partial L_{\tau_k+\tau_l}}{\partial h_{\tau_k}^k}$ to client k

Client k executes:

- 13) Having $\frac{\partial L_{\tau_k+\tau_l}}{\partial h_{\tau_k}^k}$, compute the gradient of loss at last time step on client l w.r.t W_s^k ,
 $\frac{\partial L_{\tau_k+\tau_l}}{\partial W_s^k} \leftarrow f'_k(X_{j_s}^k, h_{\tau_k}^k, W_s^k)$
- 14) Update s^{th} model parameters in client k , $W_s^k \leftarrow W_s^k - \eta \frac{\partial L_{\tau_k+\tau_l}}{\partial W_s^k}$ (η is the learning rate)

end

3.2.1 Byproducts of SL in Recurrent Neural Networks. In addition to preserving privacy of raw data, splitting RNNs have the following benefits.

(i) It leads to speed-up in the learning process. In centralized learning, where all the segments of multiple segment sequences are gathered at one client, one (unsplit) RNN trains on a long sequence containing all the segments. While, in our proposed SL approach, each split RNN only trains on short single segments of the distributed sequences in collaboration with other split RNNs on other clients. This speedup, as a result of temporal partitioning in data, is not achieved in the vanilla SL [10], where a feed-forward neural network is split. Because all training data is available in one client.

(ii) Large number of time steps in a sequence is challenging to model for RNNs [20]. By splitting RNNs between clients, split RNNs deal with short segments of sequences, instead of long complete sequences

(iii) Proper hidden-state initialization is crucial in RNNs for successful sequence modeling [14, 22]. In the proposed SL approach, the hidden states of the RNNs train on next segments are initialized

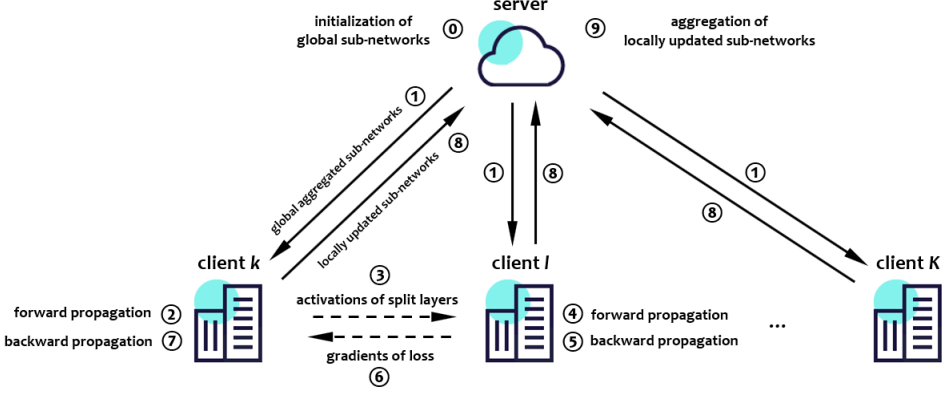


Fig. 4. Block diagram of the proposed federated split learning algorithm, FedSL, to train models on sequentially partitioned data. See Section 3.3. The numbers ①–⑨ are correspondent to the numbers in Algorithm 1.

with the hidden states of the RNNs train on previous segments. This initialization strategy works better than random initialization, since both previous and next RNNs are trained by the consecutive time steps of the same sequence.

3.3 Federated Split Learning on Distributed Sequential Data

In the previous subsections, we defined the problem of sequentially partitioned data among clients, and presented SL in RNNs. In this subsection, we present a distributed privacy-preserving ML framework utilizing FL along with SL in RNNs, FedSL, to train models on sequentially partitioned data.

There are K clients in the federated split learning environment. At each round t , $t = 0, \dots, T$, C_t fraction of clients are selected by the server to participate in federated split learning. $m_t = \max(C_t \cdot K, 1)$ is the number of clients participating in the round t of algorithm. bs , and ep are local mini-batch size and the number of local epochs on clients for local training. S is the maximum number of segments in multiple-segment sequential data.

$W_s^{k(t)}$, $s = 1, \dots, S$, $k = 1, \dots, K$, and $t = 0, \dots, T$ is the model trained locally at client k with s^{th} segments of sequential data at round t of FedSL. $W_s^{(t)}$ is the model generated on the server by aggregating locally trained models $W_s^{k(t)}$ at the t^{th} round of communication between clients and the server.

$X_j = \{\dots, X_{j_s}^k, X_{j_{s+1}}^l, \dots\}$ is the j^{th} multiple-segment sequential training sample, with segments distributed among clients. The label Y_j is available at the client containing the last segment of X_j . An ID bank is available on the server, containing a set of sample IDs $S_{\text{ID}} = \{0, 1, \dots, j, \dots\}$ and sets of segment IDs $S_{\text{segment}_j} = \{0, 1, \dots, k, \dots\}$. These sets of IDs are updated at each round of communication between clients and the server when the new generated training samples on clients are used in FL process (see Section 3.1).

Detailed steps of the proposed method are given in Algorithm 2 as well as in Figure 4. The steps of algorithm in Figure 4 are correspondent to the steps in Algorithm 2.

Algorithm 2: Federated Split Learning on Distributed Sequential Data. The steps ②-⑦ are split learning between clients (see Algorithm 1)

Input: multiple-segment sequential data distributed between clients, $\{X_j, Y_j\}$

Output: trained split RNNs, $W_s, s = 1, \dots, S$

① Server initializes models $W_s^{(t)}, s = 1, \dots, S, t = 0$

① Server sends $W_s^{(t)}, s = 1, \dots, S, t = 0$ to all m_t clients participating in the round t of federated split learning

② Each client k , containing the s^{th} segment of $X_j (X_{j_s}^k)$, performs forward propagation on the model $W_s^{(t)}$ up to its split layer

③ Each client k , performed forward propagation using s^{th} segment in ②, sends the activation of its split layer to client l , containing the $(s + 1)^{\text{th}}$ segment

④ Each client l , having $(s + 1)^{\text{th}}$ segment of $X_j (X_{j_{s+1}}^k)$, the label Y_j , and received the activation from the client k in ③, performs forward propagation on $W_{s+1}^{(t)}$ from its split layer to the final layer, and computes the output and the loss values

⑤ Each client l , performed forward propagation in ④, performs backward propagation on $W_{s+1}^{(t)}$, and updates its weights to generate split model $W_{s+1}^{l(t+1)}$

⑥ Each client l , performed backward propagation in ⑤, sends the gradient of its loss value, w.r.t to the activation received from the client k , to the client k

⑦ Each client k , received the gradient of the loss from the client l in ⑥, performs backward propagation on $W_s^{(t)}$, and updates its weights to generate split model $W_s^{k(t+1)}$

⑧ All clients send their locally-updated split models to the server

⑨ Sever aggregates all received split models to generate global models for each segment ID as follows

$$W_s^{(t+1)} = \sum_{k=1}^K \frac{n_s^k}{n_s} W_s^{k(t+1)}$$

where n_s^k is the number of s^{th} segments of training samples at client k , and n_s is the total number of s^{th} segments of training samples in all clients.

4 EXPERIMENTAL RESULTS

As per previous works in the area of FL and SL [5, 10, 12, 21, 27–29], we evaluate the performance of FedSL on image classification and electronic health record datasets. We perform experiments using different variations of RNNs, in different settings, and in comparison with previous related works.

4.1 Image Classification

The first dataset is sequential MNIST dataset [14] which is constructed from the original MNIST handwritten digit recognition dataset by converting 28×28 pixel gray-level images into 784 sequences, i.e. the models read one pixel at a time in scan-line order, starting at the top left corner of the image, and ending at the bottom right corner, [14]. This dataset contains 60,000 training and 10,000 test samples. There are equal number of samples in each of ten digit classes.

The second dataset is fashion MNIST dataset which has been used extensively in the literature for evaluating FL algorithms, [12, 28]. This dataset contains equal number of images of ten different types of clothes. There are 60,000 training and 10,000 test gray-level images of size 28×28 . The samples in this dataset are images. However, RNNs in the proposed method deal with distributed

sequential data. Therefore, in our experiments, each 28×28 pixel image is converted to a sequence of length 28 with 28-element feature vector in each time step, i.e. each 28-pixel row of the image is considered as one time step of a 28-length sequence.

The above datasets are not sequentially partitioned or distributed. As per the previous works in the area of FL and SL [10, 12], we will divide training samples into segments and distribute the segments among consecutive clients in a federated setting. The performance metrics for image classification datasets are training loss and test-set accuracy in communication rounds of FL algorithms.

4.1.1 Sequential MNIST. The sequential MNIST dataset is used to evaluate the performance of the proposed method in modeling long-range dependencies in long sequences, [3, 14]. Le et al. [14] proposed to use ReLU as activation function in the vanilla RNN and initialize the hidden states with the identity matrix, and called it IRNN. The IRNN achieved superior performance compared the vanilla RNN and LSTM in learning long-term dependencies. For the same reason, we use IRNN for sequential MNIST dataset classification.

In FedAvg [21] (described in Section 2.1), which cannot handle sequential data, complete training sequences are available at single clients, i.e. clients contain complete 784-length sequences. However, in the proposed FedSL, each 784-length sequence is distributed among two or three clients, i.e. each two or three consecutive clients contain consecutive segments of the complete sequences. In both the FedAvg and FedSL approaches, at each time step of sequences, a single-element feature vector is available corresponding to one pixel in the MNIST images.

In FedAvg, the IRNN has one unidirectional layer with 64 neurons, and after that a fully-connected layer with 64 neurons. The hidden states are initialized with the identity matrix. In FedSL, the IRNN is split into two or three sub-networks. The first sub-networks, trained on the clients containing the first segments of training sequences, are IRNNs having one unidirectional layer with 64 neurons. The hidden states are initialized with the identity matrix. The second sub-networks, trained on the clients containing the second segments of training sequences, are IRNNs with one unidirectional layer with 64 neurons. The hidden states of the second sub-networks are initialized by the first sub-networks during communications in SL (see Section 3.2). The third sub-networks, trained on the clients containing the third segments of training sequences and the labels, are IRNNs with one unidirectional layer with 64 neurons plus a fully-connected layer with 64 neurons. The hidden states of the third sub-networks are initialized by the second sub-networks during communications in SL (see Section 3.2). The learning rate in the both methods is 0.000001.

Local computation per client: Figure. 5 shows the learning curves of FedAvg and FedSL over 500 rounds of communication, when local batch size is 8, and 64. Train loss and test accuracy vs. the number of communication rounds is shown in Figure. 5 left and right, respectively. In this experiment, the distribution of data between clients is IID. The total number of clients $K = 100$ and $C_t = 0.1$, i.e. in each round of communication, $m_t = \max(0.1 \times 100, 1) = 10$ clients participate in federated averaging. As can be seen in Figure. 5, in all cases, the proposed FedSL outperforms FedAvg, while handling distributed sequential data among clients. The FedSL achieves higher accuracy in fewer rounds of communication. This superiority of the proposed method is due to the byproducts of SL in RNNs in addition to preserving privacy (see Section 3.2.1). In both methods, increasing local computation per client (by decreasing the local batch sizes) results in convergence in fewer rounds of communication.

Increasing parallelism in non-IID data: Figure. 6 compares FedSL and FedAvg in handling non-IID data, as described in [21], when different number of clients participate in FL, C_t is 0.1 and 1 ($K = 100$). In this experiments, local batch sizes, and local epochs equal 64, and 1, respectively. In all cases, FedSL achieves lower loss and higher accuracy in fewer rounds of communication compared

to FedAvg, while handling multiple-segment sequential data. In both the methods, increasing parallelism, participating more clients in FL results in achieving lower train loss and higher test accuracy in fewer rounds of communications.

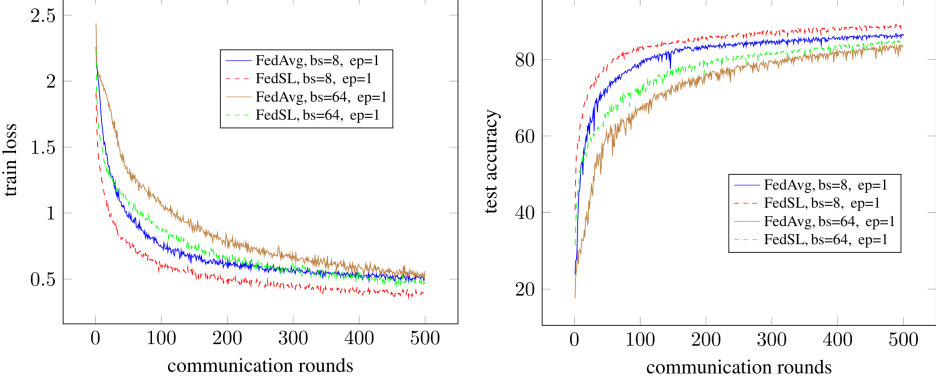


Fig. 5. Train loss (left) and test accuracy (right) versus communication rounds for FedAvg [21] and the proposed FedSL on the sequential MNIST dataset with different values of local batch sizes (bs). In all cases, the proposed FedSL achieves lower train loss and higher test accuracy in fewer rounds of communication.

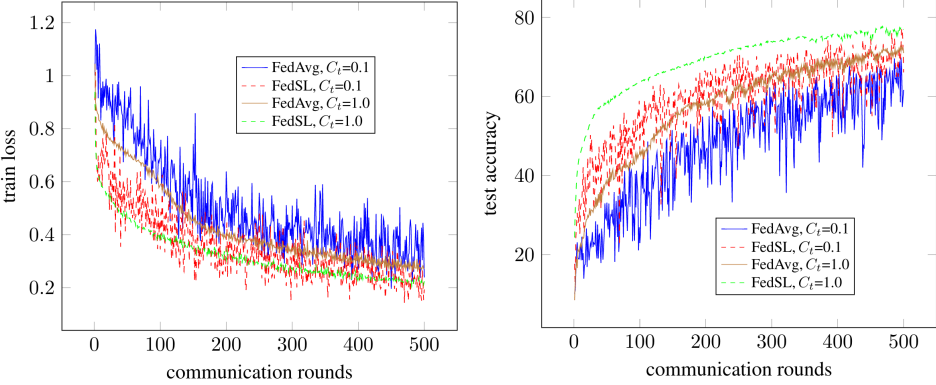


Fig. 6. Train loss (left) and test accuracy (right) versus communication rounds for FedAvg [21] and the proposed FedSL on the sequential MNIST dataset with non-IID distribution for different values of C_t , different number of participants in federated learning process. In all cases, the proposed FedSL achieves lower train loss and higher test accuracy in fewer rounds of communication.

Different number of distributed segments: In the two previous experiments on FedSL, two-segment sequential data were distributed among clients. In this experiment, we study the effect of the number of distributed segments among clients using sequential MNIST dataset. In undistributed case, the entire 784 time steps of sequences are available on single clients. In two-segment distribution case, two 392-length sequences are distributed among two consecutive clients, while, in three-segment distribution case, one 264-length sequence and two 260-length sequences are distributed among three consecutive clients. Figure. 7 depicts the learning curves of the above three cases in 500 rounds of communication using local batches of size 64 and 1 local epochs. In the first case, the FedAvg is used, because the data is not sequentially distributed. In the two-segment and

three-segment cases, the FedSL with two and three sub-networks are used to handle sequentially distributed sequences. As can be seen in Figure. 7, FedSL outperforms FedAvg in terms of achieving lower loss and higher accuracy in fewer rounds of communication. In addition, when three-segment sequences are distributed among clients (and three sub-networks are trained in SL), higher accuracy is achieved in comparison with two-segment sequence distribution where two sub-networks are trained in SL. This improvement from one-segment (using FedAvg) to three-segment sequences are due to the byproducts of SL in sequential neural networks in addition to preserving privacy (see Section 3.2.1).

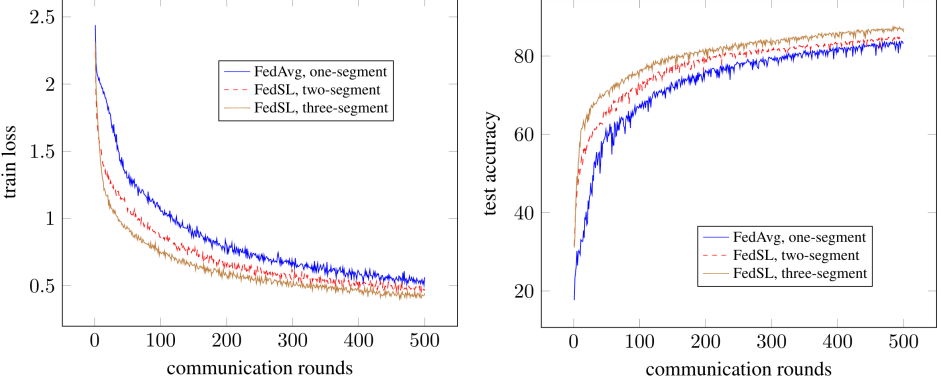


Fig. 7. Train loss (left) and test accuracy (right) versus communication rounds for FedAvg [21] (with one-segment undistributed data) and FedSL (with two and three-segment distributed data) using sequential MNIST dataset. There is an improvement in performance from one-segment sequential data (using FedAvg) to two and three-segment distributed sequential data (using FedSL).

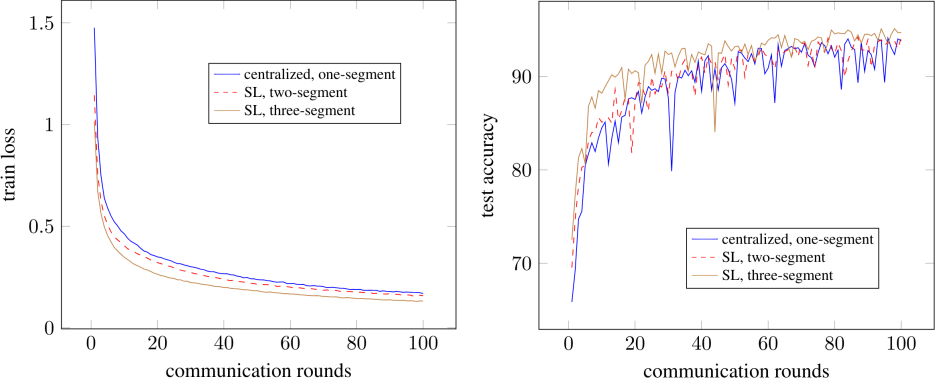


Fig. 8. Train loss (left) and test accuracy (right) of applying centralized learning and the proposed SL approach (with two and three-segment distributed sequential data) to sequential MNIST dataset. Distributed learning using the proposed SL method for RNNs converges faster compared to the centralized learning.

Comparing the proposed SL method for RNNs with centralized learning: we examine the performance of the proposed SL algorithm for IRNNs, compared to the centralized learning. In SL, two (or three) consecutive clients contain one half (or one third) of the time steps of the training sequences. These two (or three) clients communicate to train their sub-networks without data

sharing (see Section 3.2). Figure. 8 left and right depict the learning curves of the two methods when batch size in centralized learning and local batch sizes in clients in SL are 64. As can be observed in Figure. 8, the proposed SL method for IRNNs outperforms the centralized learning with less convergence time.

4.1.2 Fashion MNIST. We report the results of applying the proposed method to the fashion MNIST dataset, in comparison with FedAvg [21] (described in Section 2.1). For the experiments related to the FedAvg, being unable to handle sequentially distributed data, it is assumed that complete training sequences are available at single clients, i.e. clients contain complete 28-length sequences. However, in the proposed FedSL, each 28-length sequence is distributed among consecutive two clients, i.e. each two consecutive clients contain 14-length segments of the complete sequences, one related to the first half and another is related to the second half of each 28-length sequence. In both FedAvg and FedSL approaches, at each time step of sequences, a 28-element feature vector is available corresponding to one row of the 28×28 gray-level images. In this experiment, the GRU is used as sequence classifier. In FedAvg, the GRU has one unidirectional layer with 64 neurons, and after that a fully-connected layer with 64 neurons. The hidden states are initialized with zeros. In the FedSL, the GRU is split into two sub-networks. The first sub-networks, trained on the clients containing the first segments of training sequences, is a GRU having one unidirectional layer with 64 neurons whose hidden states are initialized with zeros. The second sub-networks, trained on the clients containing the second segments of training sequences and the label, is a GRU with one unidirectional layer with 64 neurons plus a fully-connected layer with 64 neurons. The hidden states of the second sub-networks are initialized by the first sub-networks during communications in SL (see Section 2.2). The learning rate in both methods is 0.1.

Local computation per client: Figure. 9 shows the learning curves of FedAvg and FedSL over 500 rounds of communication, when the local batch sizes are 8 and 64, and local epochs are 1 and 5 (for 64 local batch size). Train loss and test accuracy vs. the number of communication rounds is shown in Figure. 9 left and right, respectively. In this experiment, the distribution of data between clients is IID, i.e. clients contain equal number of training samples in different classes. The total number of clients $K = 100$ and $C_t = 0.1$, i.e. in each round of communication, $m_t = \max(0.1 \times 100, 1) = 10$ clients participate in FL. As can be seen in Figure. 9, the proposed FedSL successfully follows the FedAvg, while handling sequentially distributed data among clients. In both methods, increasing computation per client (by decreasing the local batch size or increasing local epochs) results in convergence in fewer rounds of communication. As can be seen in the legend of Figure. 9 (left) where the training times (in seconds) are described, the training time in FedSL is shorter than that of FedAvg. This is due to the more distributed processing in FedSL in comparison with FedAvg. In FedSL, each multiple-segment sequence is processed on two clients, while in FedAvg each multiple-segment sequence is processed on one single client.

Increasing parallelism in IID data: Figure. 10 depicts the learning curves of FedAvg and FedSL over 500 rounds of communication, when different number of clients participate in federated learning process, i.e. different values of C_t , 0.1, 0.5, and 1 ($K = 100$). In this experiments, local batch sizes, and local epochs equal 64, and 1, respectively. As can be seen in Figure. 10, increasing the number of participants does not reduce the needed number of communication rounds to converge in case of IID data. The performance of FedSL is comparable to FedAvg. In addition, as described in the legend of Figure. 10 (left), due to the more distributed processing in FedSL, the convergence time in FedSL is always shorter than FedAvg. In FedSL, sequences are distributed among clients, and single clients train on single segments of sequences, instead of complete sequences.

Comparing the proposed SL method for RNNs with centralized learning: we examine the performance of the proposed SL algorithm for RNNs (GRU in this experiment), compared to the

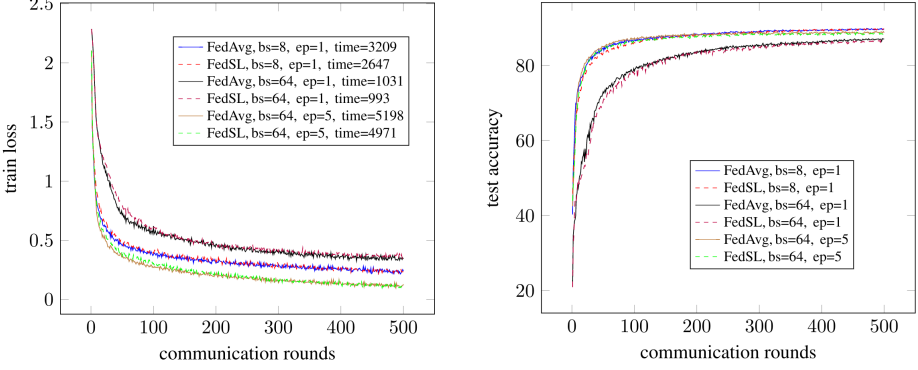


Fig. 9. Train loss (left) and test accuracy (right) versus communication rounds for FedAvg [21] and the proposed FedSL on the fashion MNIST dataset for different values of local batch sizes (bs) and local epochs (ep). The proposed FedSL successfully follows FedAvg in 500 rounds of communication. The training times (in seconds) are described in the left figure.

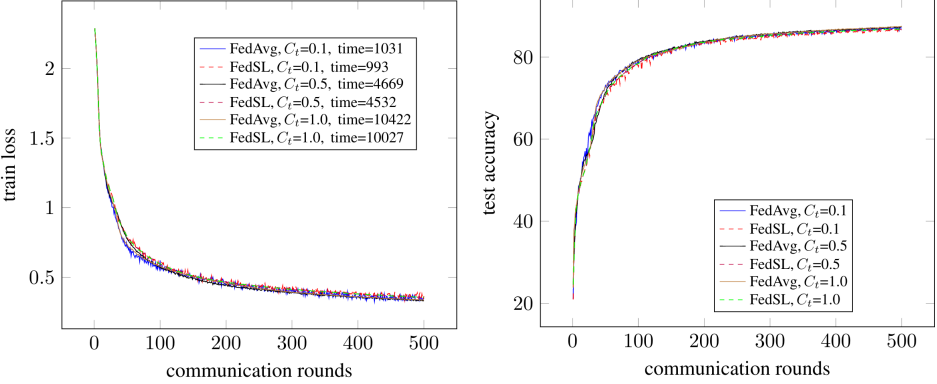


Fig. 10. Train loss (left) and test accuracy (right) of applying FedAvg [21] and the proposed FedSL to fashion MNIST dataset, for different values of C_t , different number of participants in federated learning. The training times (in seconds) are described in the left figure. In case of IID data, increasing parallelism does not increase the convergence rate.

centralized learning. In the proposed SL environment, two consecutive clients contain one half of the training sequences. These two clients communicate to train their sub-networks without data sharing (see Section 2.2). Figure. 11 depicts learning curves of the two methods when batch size in centralized learning and local batch sizes in clients in SL are 8 and 64. The number of epochs in both methods is considered as 1. As can be observed in Figure. 11, in all cases, the proposed SL method for RNNs outperforms centralized learning with much less convergence time and with smaller amount of fluctuations in the values of train loss.

4.2 eICU Dataset

The next dataset is eICU [23], a multi-center Intensive Care Unit (ICU) dataset containing data of 200,859 admissions to ICUs for 139,367 unique patients in 208 hospitals located throughout the United States. In this dataset, there are patients with multiple unit/hospital admissions. The dataset

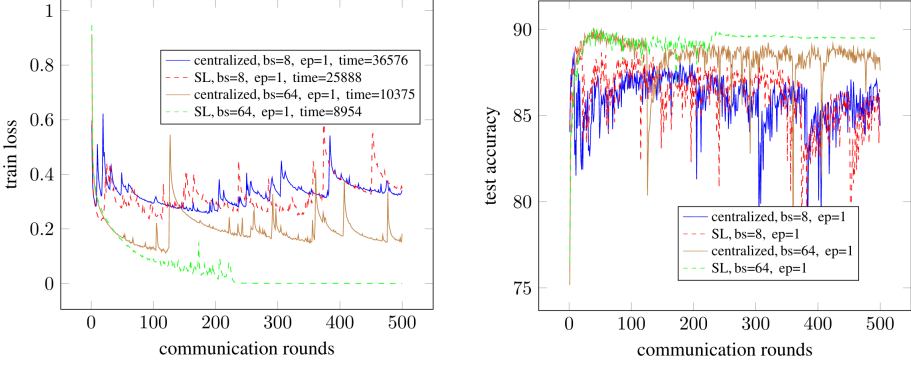


Fig. 11. Train loss (left) and test accuracy (right) of applying centralized learning and the proposed SL approach to the fashion MNIST dataset with different values of batch sizes. Distributed learning using the proposed SL method for RNNs converges faster compared to the centralized learning.

is deidentified, and includes vital sign measurements, care plan documentation, severity of illness measures, diagnosis information, treatment information, etc., [23].

Out of 31 available tables in the eICU dataset [23], we extract information from the following three tables, patient table (containing demographic and administrative information regarding the patient and their unit/hospital stay), lab table (containing laboratory measurements for patient derived specimens), and nurseCharting table (containing information charted at the bed side, including vital signs). As discussed in [25], 20 numerical and categorical variables are extracted as features describing the first 48 hours of ICU stays of patients. The 13 numerical features are scaled and transformed individually between -1 and 1, and the 7 categorical variables are one hot encoded, resulting in a 419-element feature vector for each hour of ICU stay. Therefore, we have a sequential data with 48 time steps with 419-element feature vector at each time step. In this paper, we work on in-hospital mortality prediction task which is a binary classification problem with labels 0 or 1 indicating survival or death of the patient after being discharged from the ICU. All patients with two admissions are extracted from all 208 hospitals resulting in 13,277 patients with two consecutive admissions with 11.57% mortality rate. The label (survival or death of the patient) is available at the hospital containing the data of the second admission of the patient. Since, there are different values of mortality rate in different hospitals, this dataset is in nature non-IID [13]. There are different number of samples in each of two classes in different clients. In our experiments, 80% and 20% of samples are used for training and test, respectively.

We report the results of applying the following methods to the eICU dataset: centralized learning, the proposed SL for RNNs (see Section 3.2), FedAvg [21], LoAdaBoost FedAvg [13], the proposed FedSL (see Section 3.3), and the combination of the proposed FedSL with LoAdaBoost FedAvg. In the centralized learning, all data is assumed to be available in one location and the RNN is trained using conventional centralized learning. In the proposed SL for RNNs (see Section 3.2), the segments of sequential data is assumed to be distributed among two consecutive clients. Two sub-networks are trained on two clients using consecutive segments of distributed sequential data. In FedAvg, all segments of each sequential training sample are assumed to be available in one client, full segment sequential data are distributed among clients. The LoAdaBoost FedAvg [13], described in Section 2.1, is an improved version of FedAvg that modifies the local updates at client side and can be incorporated with the proposed FedSL. In the combination of the FedSL with the

LoAdaBoost FedAvg [13], the training of each sub-network in each client is performed through LoAdaBoost FedAvg [13].

For the eICU dataset, the LSTM is used as sequence classifier. In centralized learning, FedAvg, and the LoAdaBoost FedAvg [13], the LSTM has one unidirectional layer with 64 neurons, and after that a fully-connected layer with 64 neurons. The hidden states are initialized with zeros. In the proposed SL, FedSL, and the combination of FedSL with the LoAdaBoost FedAvg [13], the LSTM is split into two sub-networks. The first sub-networks, trained on the clients containing the first segments of training sequences, is an LSTM having one unidirectional layer with 64 neurons whose hidden states are initialized with zeros. The second sub-networks, trained on the clients containing the second segments of training sequences and the label, is an LSTM with one unidirectional layer with 64 neurons plus a fully-connected layer with 64 neurons. The hidden states of the second sub-networks are initialized by the first sub-networks during communications in SL (see Section 3.2). The learning rate for centralized learning and the proposed SL is 0.01, and for other methods is 0.1.

Comparing the proposed SL method for RNNs with centralized learning: Figure. 12 shows the train loss and AUC (Area Under Curve) of ROC (Receiver Operating Characteristics) curves vs. 100 communication rounds of the proposed SL method and the centralized learning on the eICU dataset. We performed experiments for batches of size 8 and 64. As can be observed in the learning curves in Figure. 12, in all cases, the proposed SL method for RNNs successfully follows the centralized learning and achieves higher AUC-ROC, while handling sequentially distributed data and preserving privacy.

Local computation per client: In Figure. 13, we report the results of applying the proposed FedSL to the eICU dataset compared to the FedAvg [21], LoAdaBoost FedAvg [13], and also the combination of the proposed FedSL with LoAdaBoost FedAvg, for local batches of size 8 and 64, and local epochs of size 1 and 10. As can be observed in Figure. 13, in terms of AUC-ROC, the proposed FedSL outperforms FedAvg, and also the combination of FedSL with LoAdaBoost outperforms LoAdaBoost FedAvg.

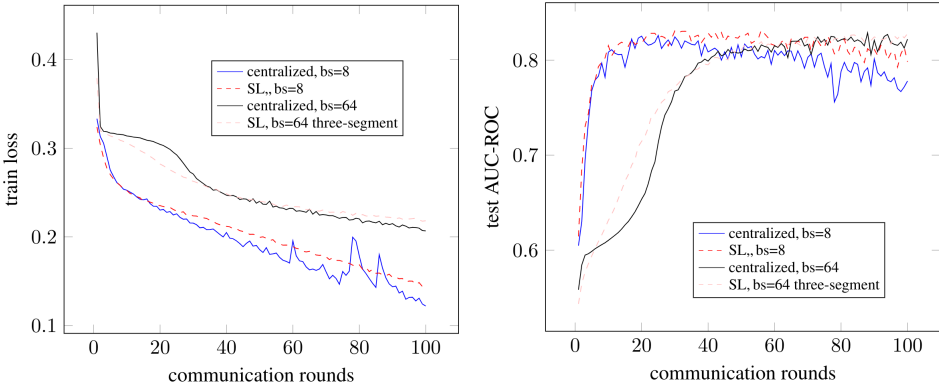


Fig. 12. Train loss (left) and test AUC-ROC (right) of applying centralized learning and the proposed SL approach to eICU dataset for different values of batch sizes (bs). The proposed SL method for RNNs successfully follows the centralized learning and achieves higher AUC-ROC, while handling sequentially distributed data and preserving privacy.

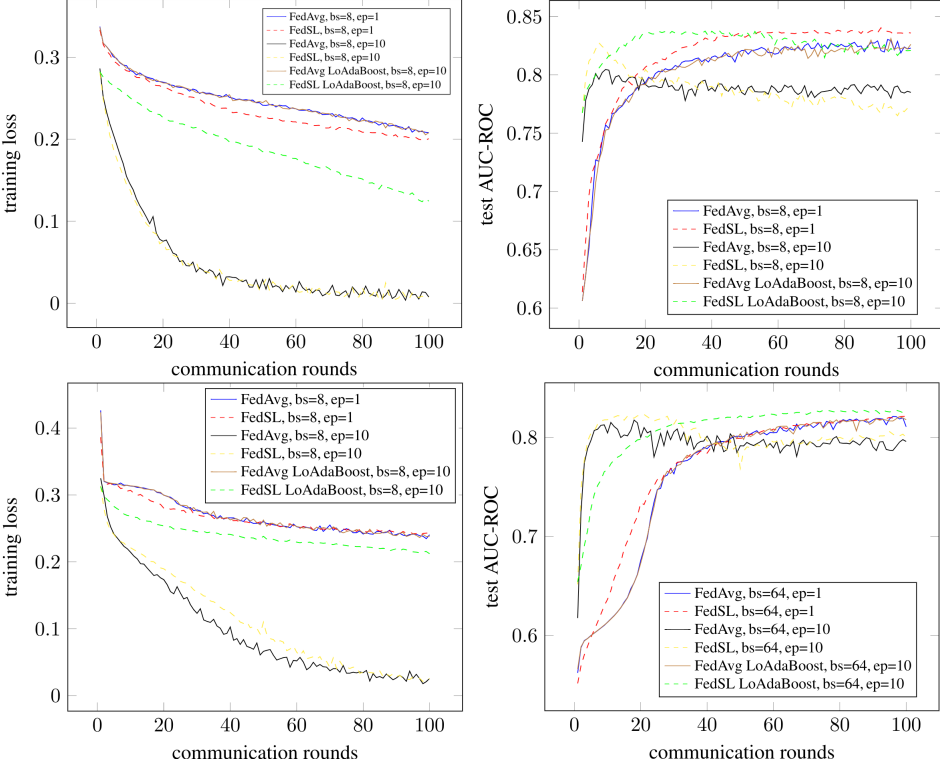


Fig. 13. Train loss (left column) and test AUC-ROC (right column) of applying different FL approaches to eICU dataset for different values of batch sizes (bs) and local epochs (ep).

5 CONCLUSIONS AND FUTURE WORK

In this paper, we defined and addressed the problem of training models in a federated privacy-preserving setting on distributed sequential data occurring frequently in practice. Sequentially partitioned data is different from the horizontal and vertical partitioned data studied in the previous FL and SL works, [12]. We presented a novel architecture that integrates FL and SL to address the problem of training models on sequentially partitioned data. In this architecture, none of the data, or label, or complete model parameters are shared between clients or between clients and the server. In order to analyze multiple segments of distributed sequential data, we introduced a novel SL approach tailored for RNNs. Consecutive splits of RNNs are trained on consecutive clients containing consecutive segments of multiple-segment sequential data distributed across clients. This SL approach for RNNs, is different from the previous SL approaches working on feed-forward neural networks.

The experimental results on simulated and real-world multiple-segment sequential data (with different values of local epochs and local batch sizes, with different number of participants, and on IID and non-IID data) demonstrate that proposed method successfully train models on distributed sequential data, while preserving privacy, and outperforms previous FL and centralized learning approaches in terms of achieving higher accuracy in fewer communication rounds.

In future research, we plan to work on split learning in novel sequential neural networks, including Temporal Convolutional Networks (TCNs), 3D CNNs, and use the split sequential models

in our proposed FedSL framework. In addition, we aim to apply modified versions of FedAvg [12] in the FedSL framework. Other directions of our further research are utilizing differential-privacy techniques in the FedSL framework, and working on techniques in SL to personalize split models for individual clients.

REFERENCES

- [1] Sharif Abuadbbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A. Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. 2020. Can We Use Split Learning on 1D CNN Models for Privacy Preserving Training? arXiv:2003.12365 [cs.CR]
- [2] Pragati Baheti, Mukul Sikka, KV Arya, and R Rajesh. 2020. Federated Learning on Distributed Medical Records for Detection of Lung Nodules.. In *VISIGRAPP (4: VISAPP)*. PMC, 10.1101/2020.08.11.20172809, 445–451.
- [3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. arXiv:1803.01271 [cs.LG]
- [4] Christopher Briggs, Zhong Fan, and Peter Andras. 2020. Federated learning with hierarchical clustering of local updates to improve training on non-IID data. arXiv:2004.11791 [cs.LG]
- [5] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. 2020. VAFL: a Method of Vertical Asynchronous Federated Learning. arXiv:2007.06081 [cs.LG]
- [6] Yiqiang Chen, Jindong Wang, Chaohui Yu, Wen Gao, and Xin Qin. 2019. FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare. arXiv:1907.09173 [cs.LG]
- [7] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. 2019. SecureBoost: A Lossless Federated Learning Framework. arXiv:1901.08755 [cs.LG]
- [8] Siwei Feng and Han Yu. 2020. Multi-Participant Multi-Class Vertical Federated Learning. arXiv:2001.11154 [cs.LG]
- [9] Yansong Gao, Minki Kim, Sharif Abuadbbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit A. Camtepe, Hyoungshick Kim, and Surya Nepal. 2020. End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things. arXiv:2003.13376 [cs.CR]
- [10] Otkrist Gupta and Ramesh Raskar. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 116 (2018), 1–8.
- [11] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2019. Federated Learning for Mobile Keyboard Prediction. arXiv:1811.03604 [cs.CL]
- [12] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Xinghua Zhu, Jianzong Wang, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annamaram, and Salman Avestimehr. 2020. FedML: A Research Library and Benchmark for Federated Machine Learning. arXiv:2007.13518 [cs.LG]
- [13] Li Huang, Yifeng Yin, Zeng Fu, Shifa Zhang, Hao Deng, and Dianbo Liu. 2020. LoAdaBoost: Loss-based AdaBoost federated machine learning with reduced computational complexity on IID and non-IID intensive care data. *Plos one* 15, 4 (2020), e0230706.
- [14] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. 2015. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. arXiv:1504.00941 [cs.NE]
- [15] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, and Bingsheng He. 2020. A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. arXiv:1907.09693 [cs.LG]
- [16] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.
- [17] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2020. On the Convergence of FedAvg on Non-IID Data. arXiv:1907.02189 [stat.ML]
- [18] Timothy P Lillicrap and Adam Santoro. 2019. Backpropagation through time and the brain. *Current opinion in neurobiology* 55 (2019), 82–89.
- [19] Yang Liu, Yan Kang, Xinwei Zhang, Liping Li, Yong Cheng, Tianjian Chen, Mingyi Hong, and Qiang Yang. 2020. A Communication Efficient Collaborative Learning Framework for Distributed Features. arXiv:1912.11187 [cs.LG]
- [20] Yi Luo, Zhuo Chen, and Takuya Yoshioka. 2020. Dual-path RNN: efficient long sequence modeling for time-domain single-channel speech separation. arXiv:1910.06379 [eess.AS]
- [21] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629 [cs.LG]
- [22] Nima Mohajerin and Steven L Waslander. 2019. Multistep prediction of dynamic systems with recurrent neural networks. *IEEE transactions on neural networks and learning systems* 30, 11 (2019), 3370–3383.

- [23] Tom J Pollard, Alistair EW Johnson, Jesse D Raffa, Leo A Celi, Roger G Mark, and Omar Badawi. 2018. The eICU Collaborative Research Database, a freely available multi-center database for critical care research. *Scientific data* 5 (2018), 180178.
- [24] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated Learning for Emoji Prediction in a Mobile Keyboard. arXiv:1906.04329 [cs.CL]
- [25] Seyedmostafa Sheikhalishahi, Vevake Balaraman, and Venet Osmani. 2020. Benchmarking machine learning models on multi-centre eICU critical care dataset. *Plos one* 15, 7 (2020), e0235424.
- [26] Alex Sherstinsky. 2020. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena* 404 (2020), 132306.
- [27] Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. 2019. Detailed comparison of communication efficiency of split learning and federated learning. arXiv:1909.09145 [cs.LG]
- [28] Chandra Thapa, M. A. P. Chamikara, and Seyit Camtepe. 2020. SplitFed: When Federated Learning Meets Split Learning. arXiv:2004.12088 [cs.LG]
- [29] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. arXiv:1812.00564 [cs.LG]
- [30] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.