Federated Learning for Mobile Traffic Forecasting

Machine Learning for Mobile Communication Systems Ph.D. Course (A.A. 23/24)

Riccardo Mazzieri
Department of Information Engineering
riccardo.mazzieri@phd.unipd.it

***Abstract*—This project report explores the application of Federated Learning to a multi-dimensional time series forecasting problem using real mobile traffic data from an LTE cell in Madrid, El Rastro neighborhood. The study involves training a Recurrent Neural Network model, referred to as FedRNN, within a Federated Learning framework. The performance of FedRNN was evaluated against models trained on local datasets in two modalities: first, by assessing the models on the union of all local test sets to evaluate generalization, and second, by comparing performance on each individual local test set. The results demonstrate that FedRNN significantly outperforms local models in both modalities. Additionally, this report investigates the impact of input sequence length on the quality of the predictions and briefly explores the use of the FedProx federated strategy as an alternative to the more common FedAvg, even though no substantial difference was observed. Experimental results are thoroughly discussed, together with aspects regarding the model's predictive limitations and future work ideas.**

I. Introduction and Problem Statement

As Artificial Intelligence (AI) technologies become increasingly pervasive, the burden on centralized servers responsible for running such AI models is quickly growing. This trend presents several challenges, including but not limited to high maintenance costs, environmental sustainability issues, privacy concerns and increased system latency.

In response, the edge computing paradigm is receiving considerable attention for its potential to alleviate these problems by shifting computation from centralized servers to the network edge. This paradigm entails a new set of challenges, not present in a classic, fully centralized setting. For example, as the AI models are supposed to run on computationally and battery constrained devices (such as smartphones, edge servers or IoT devices), the computational and memory complexity of the models will need to be tuned accordingly. Another problem arises from the scattered nature of the data. Indeed, if on one hand the edge computing paradigm greatly reduces communication costs required for data transfer to central servers, on the other hand the problem of how to aggregate the knowledge of all the local models, learning from a local set of data, arises.

Federated Learning (FL) [1] refers to a distributed learning paradigm where multiple clients work together to tackle machine learning problems under the guidance of a central aggregator. This framework enables efficient learning from a decentralized set of data, where each client involved in the training process effectively contributes to a unified model, by just sending model parameters or local gradient information. This paradigm allows the computation to take place at the network edge, on smaller devices with a much lower carbon footprint, potentially powered by batteries or renewable energy sources. Additionally, by removing the need to send private data to a central server, FL serves as an inherently privacy-preserving training paradigm.

This project explores the effectiveness of a simple FL strategy for solving a multi-dimensional time series forecasting problem involving real mobile traffic data. More in detail, a dataset D was provided, reporting the activity of one LTE cell in Madrid, El Rastro neighborhood. The observation period comprises the temporal window between the end of June and the end of July 2016 (06/29 - 07/29) including a total of approximately four weeks of information. Indexed by a time index t, each record of the dataset $\mathbf{x}[t]$ can be represented as a 18 dimensional feature vector containing information regarding communication patterns, at a time resolution of 1 second. Table I reports the meaning and name of all such features.

The problem statement for this project is the following:

1) Dataset D is first to be split in 4 portions (D_1, D_2, D_3, D_4), each of which corresponding to one week of measurements.
2) For each D_i, build one model based on artificial neural networks able to automatically predict `overallusers`, `overallratedw`, `overallrateup`, at time instant $t+1, t+2, \ldots, t+N$, with $N = 10$. Select the most appropriate inputs to get the outputs requested with the lowest error. You can use one of the artificial neural network architectures introduced in the lessons (e.g., Multilayer Perceptrons, Convolutional Neural Networks, Recurrent Neural Networks) or a combination of those.
3) Create a federated global model using the Federated Averaging (FedAvg) algorithm, which is merging the local knowledge of the 4 datasets (D_1, D_2, D_3, D_4), and is performing the same task.
4) Evaluate the accuracy of the 4 local models and compare it with that achieved by the global model. For this, carefully identify the test sets to be used for the evaluation.

II. Proposed Method

In this work, a Recurrent Neural Network (RNN) architecture for mobile traffic forecasting is presented. This model is designed to predict the next N time steps of downlink and

TABLE I
FEATURES CONTAINED IN DATASET D.

Feature name	Feature description
`overalltime`	time (seconds) starting from first day
`overallusers`	n. of transmitting users
`overallrbdw`	total n. of resource blocks in downlink
`overallrbdwmean`	mean n. of resource blocks in downlink
`overallrbdwstd`	std of resource blocks in downlink
`overallratedw`	rate (bit/s) in downlink
`overallratedwmean`	mean rate (bit/s) in downlink
`overallratedwstd`	std of rate in downlink
`overallmsgdw`	n. of messages in downlink
`overallretxdw`	n. of retransmitted packets in downlink
`overallrbup`	total n. of resource blocks in uplink
`overallrbupmean`	mean n. of resource blocks in uplink
`overallrbupstd`	std of resource blocks in uplink
`overallrateup`	rate (bit/s) in uplink
`overallrateupmean`	mean rate (bit/s) in uplink
`overallrateupstd`	std rate (bit/s) in downlink
`overallmsgup`	n. of messages in uplink
`overallretxup`	n. of retransmitted packets in uplink

uplink traffic, as well as the number of transmitting users, by observing a sequence of length L of feature vectors containing detailed information about the observed mobile traffic.

In this section, the proposed method is described. First, in Section II-A, the entire data pre-processing pipeline is outlined. Then, in Section II-B, the proposed neural network architecture is described in detail. Finally, section II-C reports the details of the training process.

A. Data Pre-Processing

The following data pre-processing pipeline was applied to the raw dataset D:

1) *Handling missing values and time gaps:* An initial analysis revealed significant issues with missing values and time indices, likely due to errors in the data recording process. Although neural networks can be somewhat robust to noise, leaving numerous gaps in the data would result in an inconsistent time scale (i.e., a time step in an input sequence would not consistently correspond to one second of measurements). To address this, all the time gaps of length less than 3 seconds were filled using linear interpolation. While this method introduces some bias, it ensures a consistent time scale across the dataset.
2) *Unit conversion:* Features representing downlink and uplink traffic, initially measured in bits per second (bit/s), were converted to megabits per second (Mb/s). This conversion was performed in order to avoid large discrepancies in magnitude compared to other features, which could potentially lead to instability in the training process.
3) *Inclusion of temporal features:* The hour of the day and the day of the week were derived from the `overalltime` variable and were added as two additional features to the dataset. This choice was motivated by the observation that mobile communication patterns significantly vary based on the day of the week and the time of day. Including these features allows the model to leverage temporal information that would otherwise be absent, potentially improving its predictive capabilities. The day of the week was encoded as an integer ranging from 0 to 6, and the hour of the day was similarly encoded as an integer ranging from 0 to 23
4) *Data standardization:* The dataset was standardized to mitigate issues like numerical instability and gradient saturation, which are common concerns in machine learning applications. Specifically, each feature was centered by subtracting its mean and then scaled by dividing by its standard deviation.
5) *Sequence extraction*: The next step in the pre-processing pipeline involved extracting sequences to serve as input to the neural network. This was done by slicing the dataset D into sequences of fixed length $L + N$, using a step size S. The first L elements serve as the input to the neural network, while the last N elements are used as the labels for the training process. To maintain the integrity of the data, any sequences containing time gaps were excluded from the analysis. This sequence extraction was performed independently for each subset D_i, ensuring that each client's data resulted in a unique set of sequences S_i, with $S_i \cap S_j = \varnothing,\ \forall i \neq j$. These sequences were then randomly divided into training, validation, and test sets, with the proportions 0.8, 0.1, and 0.1, respectively.
6) *Feature selection*: As a final step, a subset of features was selected for training the model. In machine learning, utilizing too many features during training can lead to overfitting on the training data, as well as result in models that are more computationally intensive and memory-demanding. Therefore, only a subset of the original features was chosen.
   In order to choose the subset of features to exclude, several training sessions were carried out, testing various combinations. Such experiments didn't reveal a big gap of performances between models trained with all the features and models trained with very restricted set of features (including just the three features the model is supposed to predict). In the final experiments, just the following features were excluded: `overallbdwmean`, `overallbdwstd`, `overallratedwmean`, `overallratedwstd`, `overallrateupmean` and `overallrateupstd`. The choice was motivated by observing that these features were probably derived by other existing features of the dataset, thus probably being unhelpful in increasing the predictive potential of the model.

B. Model Architecture

The proposed neural network architecture adopts a sequence-to-sequence framework, employing multilayer RNN for both the encoder and decoder components. At the output of the decoder, a final linear layer is used to generate the predictions for each timestep.

To mitigate the information bottleneck commonly encountered in early neural machine translation models, where the last hidden state of the encoder struggled to capture all the information from long input sequences, an attention mechanism is incorporated between the encoder and decoder. This attention layer ensures that the model can effectively focus on relevant parts of the input sequence during prediction.

It is worth noting that, in an hypothetical FL scenario, clients taking part to a federated training session would execute its local computations on devices constrained both from a computation and memory standpoint. For this reason, a good amount of attention was devoted to make the model as computationally and memory efficient as possible, while still retaining good predictive capabilities.

A more detailed description of each module of the network is reported below:

1) *Encoder*: The encoder comprises $K = 3$ stacked layers of Gated Recurent Unit (GRU) cells [2], each with $h = 64$ hidden units. GRUs were chosen for their efficiency, requiring fewer parameters compared to Long Short Term Memory (LSTM) cells [3]. The encoder processes the input sequence, computing one hidden state for each element of the sequence and for each layer, which will be denoted as $\mathbf{h}[i]_j,\ i = 1, \ldots, L,\ j = 1, \ldots, K$.
The hidden states from the final layer $\mathbf{h}[i]_K,\ i = 1, \ldots, L$ are then returned for later use by the attention mechanism (for ease of notation $\mathbf{h}[i]_K$ and $\mathbf{h}[i]$ will be later used interchangeably). The encoder also outputs the concatenation of all the hidden states from each layer at the final timestep $\mathbf{h}_{\text{last}} = \text{concat}[\{\mathbf{h}[L]_i\}_{i=1}^{k}]$. This serves as a comprehensive representation of the sequence to be used as the first hidden state for the decoding process.
2) *Attention Layer*: The attention mechanism, specifically the dot product attention [4], is employed to address the information bottleneck problem by allowing the decoder to focus on relevant parts of the input sequence. For each decoding step t, the attention weights $\alpha[t]_i$ are computed as the dot product between the decoder's current hidden state $\mathbf{h}^D[t]$ and each of the encoder's hidden states $\mathbf{h}[i]$. By referring to the terminology popularized by [5], the decoder's current hidden state $\mathbf{h}^D[t]$ acts as the query, while the encoder's hidden states $\mathbf{h}[i]$ serve as both the keys and values. Attention weights $\alpha[t]_i$ are therefore computed as:

$$\alpha[t]_i = \frac{\exp(\mathbf{h}^D[t]^\top \mathbf{h}[i])}{\sum_{j=1}^{L} \exp(\mathbf{h}^D[t]^\top \mathbf{h}[j])}. \tag{1}$$

The context vector $\mathbf{c}[t]$ for the current timestep is then computed as a weighted sum of the encoder's hidden states:

$$\mathbf{c}[t] = \sum_{i=1}^{L} \alpha[t]_i \mathbf{h}[i]. \tag{2}$$

This mechanism, also used in more modern architectures such as Transformers, is computationally efficient and requires fewer trainable parameters than other methods such as the additive attention mechanism [6].
3) *Decoder*: As the encoder, also the decoder consists of K GRU layers with h hidden units. At each timestep t, the decoder uses the context vector $\mathbf{c}[t]$ from the attention layer, concatenated with the previous output $\mathbf{y}[t-1]$, (or an initial token tensor $\mathbf{y}_0$ for the first decoding step) to update its hidden state $\mathbf{h}^D[t]$ and generate the next output $\mathbf{y}[t]$ by employing a simple linear fully connected layer with 3 output units (i.e. the `overallusers`, `overallratedw` and `overallrateup` output features).

A simplified diagram of the model architecture is provided in Figure 1, while a comprehensive list of all the model hyperparameters is reported in Table II.

TABLE II
SUMMARY OF ALL PROPOSED SYSTEM HYPERPARAMETERS

System hyperparameters		
N. of GRU hidden layers	K	3
N. of hidden units in GRU cells	h	64
Sequence length	L	60
Crop step size	S	10
Number of future time steps to predict	N	10
Batch size	B	128

C. Model Training

As previously described, the original dataset D was first preprocessed and then partitioned in 4 subsets (D_1, D_2, D_3, D_4). In this work, the previously described model was trained in two different modalities. The first follows a FL framework, where each subset D_i is treated as a local dataset for a client taking place in the training process. The second is local centalized training, which is performed for each subset D_i for later comparison.

All the training sessions were carried out using an Adam optimizer with a learning rate of 10^{-4}, using a batch size of $B = 128$, and the Mean Absolute Error (MAE) Loss function. During the decoding process, it was chosen not to employ the teacher forcing paradigm, meaning that at each decoding step the input was taken as the model's prediction at the previous time step.

1) *Federated Training*: One version of the model, which will be referred as FedRNN, was trained using a FL learning approach employing the FedAvg strategy [1]. The datasets (D_1, D_2, D_3, D_4) were treated as local datasets belonging to 4 different clients, participating in the FL training session. All clients were treated as permanently available to take part in the training process, and for each communication round of FedAvg each client was allowed to perform 20 local learning iterations, before sending local gradient information back to the central coordinator. The federated training process was carried out until convergence, for a total of 100 communication rounds. It's also worth puntualizing

that in general the FedAvg strategy contemplates the possibility of sampling only a fraction of clients to perform training at each communication round. Due to the very low number of clients, for the purposes of this project it was chosen to sample all the clients for each communication round.

2) *Single Client Training*: for each client i, the model was trained on its respective local dataset D_i for 20 epochs, after which all of them reached convergence.

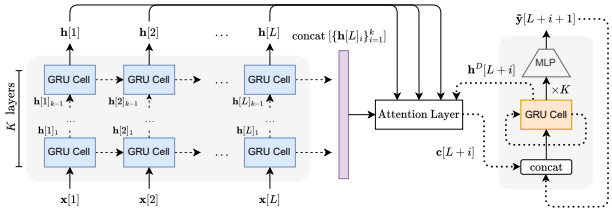

Fig. 1. Model Architecture. Dotted arrows in the decoder module represent recurrent connections.

The implementation of the models and training pipeline was carried out in Python, using the PyTorch [7] and Flower[1] frameworks. Training was carried out on a Linux machine equipped with a NVIDIA RTX3080 GPU. The code to reproduce the experiments is available online on GitHub [2].

III. Results

In this section, experimental results are presented and analyzed. The main objective reported in the problem statement of the project was to compare the performance of the FedRNN model with the one of the local models, trained on their local dataset in a centralized fashion. The Root Mean Squared Error (RMSE) was used as the primary metric for performance evaluation.

Comparing the performances of FedRNN with the ones of all the local models is not a trivial task, and different conclusions may be deduced based on the choice of the test set. As already introduced in Section II-A, each local dataset D_i was partitioned in training, validation and test split. For the purposes of this work, the federated and local models were evaluated in two different modalities, each of which considering different arrangements of the aforementioned test sets.

1) *Performance on the union of local test sets:* In this first modality, the performance of FedRNN was compared with the performance of each local model, and the test set was chosen as the union of all the local test sets. This modality was chosen in order to assess how much the FL strategy was effective in capturing the different characteristics of each different partition, and therefore also to evaluate the overall generalization capabilities of all the trained models. Figure 2 shows that the RMSE achieved by the FedRNN model on this aggregated test set is significantly lower than all the other RMSEs achieved by the local models. This result demonstrates that the federated training strategy effectively leveraged the diverse data distributions from all local datasets, resulting in better generalization. In contrast, local models, trained exclusively on their respective datasets, performed less effectively on the combined test set, highlighting the advantage of the federated approach in capturing a more comprehensive data distribution.

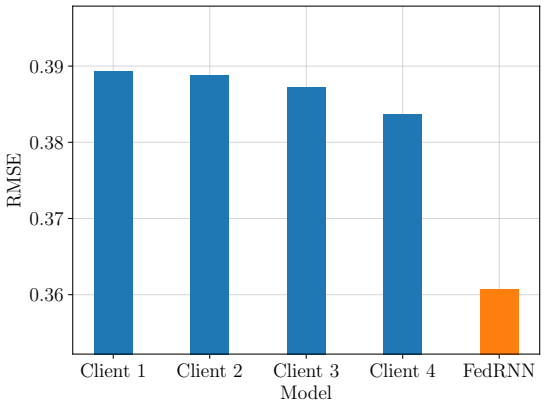

Fig. 2. Performannce comparison of FedRNN against the other four local models, on the union of all the local test sets.

2) *Performance on individual local test sets:* The performance of the FedRNN model was also compared with that of each other local model on its own local test set. Despite being trained on a broader and more diverse dataset, Figure 3 shows that the FedRNN model consistently achieved a lower RMSE on each local test set compared to the local model trained on that specific partition. This might suggests that the data distributions across different partitions are relatively similar and the FedRNN model's ability to learn from a larger and more diverse pool of training data allowed it to outperform the local models on their respective test sets.

The impact of varying input sequence lengths on the performance of the FedRNN model was also investigated. As it can be expected, too short input sequences may not provide enough information for accurate predictions, while too long sequences may hinder the model capabilities in modeling long-term dependencies in the input time sequences. To investigate this phenomenon, the model was trained with different input sequence lengths ($L = 10, 20, 30, 40, 50, 60$) and the final performance for each sequence length was evaluated as the weighted average of the RMSE across all local test sets, the weights being the number of samples in each local dataset. The results in Figure 4 and 5 indicate that the choice of different input sequence length significantly influences the model's performance, with the longest sequence length considered (60

[1]https://github.com/adap/flower

[2]https://github.com/rmazzier/MLCommSysProject

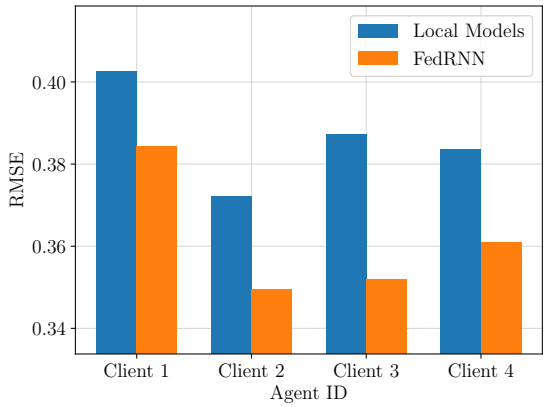

Fig. 3. Comparison of each local model with the federated model on the local test set.

seconds) providing the lowest RMSE.

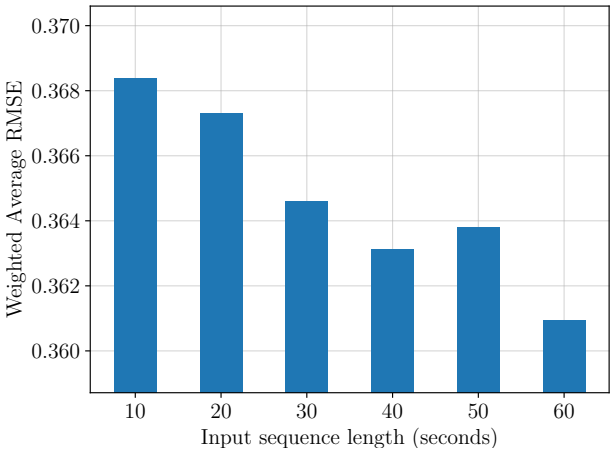

Fig. 4. Final weighted aveage RMSE over the different test sets, varying the input lengths

IV. Conclusion and Discussion

This project explored a simulated federated learning (FL) scenario using real mobile traffic data from an LTE cell in Madrid. The performance of the aggregated model, referred to as FedRNN , trained with the FedAvg strategy, was compared with models trained locally on each dataset partition D_i, in a classic, centralized fashion. The experimental results demonstrate that the FedRNN model not only excels in overall performance by effectively generalizing across all the local test sets, but also outperforms local models on individual test partitions. This outcome suggests a relative homogeneity in the data distributions across the different local datasets.

The observed similarity across local datasets is likely due to the simulated nature of this FL setup, where all local datasets, D_i, are actually part of the same dataset, captured from the same LTE cell. In a real-world FL scenario, clients are

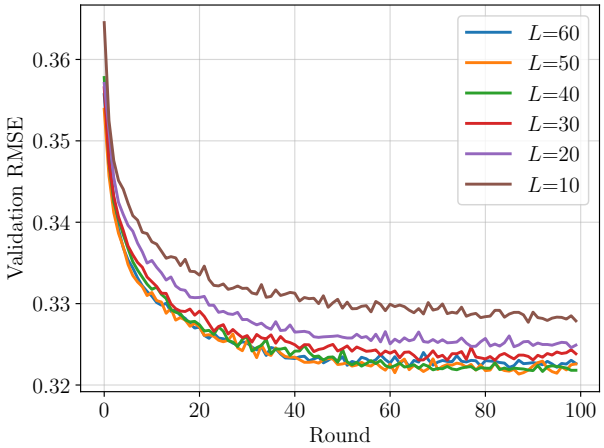

Fig. 5. Validation RMSE for the federated training using the FedAvg strategy, when considering different sequence lengths.

typically distributed across different geographical locations, leading to more heterogeneous and varied data distributions. In such cases, one would expect local models to perform better on their respective local test sets compared to the federated model, but to underperform on the union of all clients' test sets due to the broader generalization requirement.

Several ideas regarding other strategies and model architectures were considered but not pursued (or pursued in part) due to time constraints. For instance, integrating both Convolutional Neural Networks (CNNs) and RNNs could potentially enhance the model's efficiency, both in terms of parameter count and inference speed. Additionally, a comparison between the FedAvg and FedProx [8] strategies was conducted in the latest stages of the project. The network was trained using FedProx with varying proximal weight values μ. The results were essentially identical to those obtained with FedAvg, likely because FedProx typically only shows improvements in more challenging conditions with significant heterogeneity among local datasets, which our experimental results did not suggest. To give a hint of this behavior, the training curves for the model trained with the FedProx strategy is reported in Figure 6.

Regarding the model's predictions, some observations are worth discussing. As illustrated in Figure 7, the model's predictions for the number of users accurately follow the trends, apart from some high-frequency oscillations, and the uplink activity is similarly well-predicted. However, the model's performance deteriorates when predicting downlink traffic. While it can generally capture the trends and periods of increased activity, it struggles with extreme and sudden spikes. This type of explosive behavior, presenting many sudden peaks, is inherently challenging to predict, particularly when not preceded by any indicative activity, as is often the case with the observed patterns of downlink traffic.

Furthermore, additional work could enhance the feature selection process, which was not thoroughly optimized in this

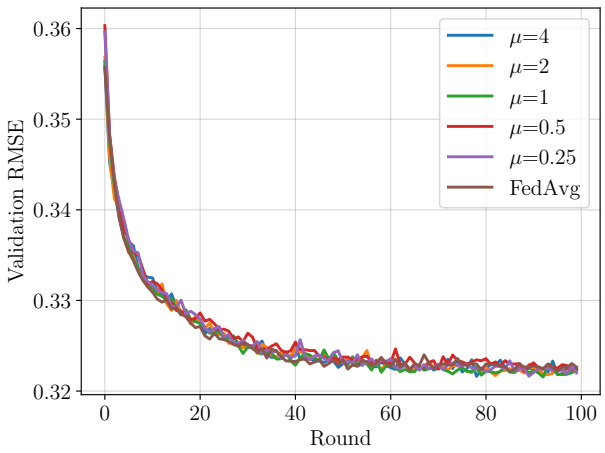

Fig. 6. Validation RMSE for the federated training, using the FedProx strategy for different values of the proximal weight μ. The resulting model is essentially unchanged on each variant, and ends up being equivalent to the model trained using the FedAvg strategy.

project. A more rigorous approach could consist in analyzing the correlation matrix of the data to identify pairs of highly correlated features, thereby better informing an effective selection of features for training.

In conclusion, while the FedRNN model demonstrated strong generalization capabilities and robust performance across local datasets, further improvements in model architecture and feature selection could yield even better results.

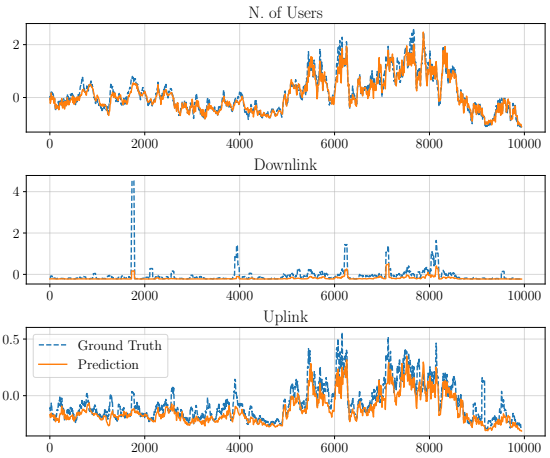

Fig. 7. Example of predictions of the federated model againts the ground truth, for a 10000 seconds section gathered from the union of all the test sets. The signals were slightly smoothed using a simple moving average filter with a window length of 1 minute for better visualization.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[2] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[8] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.