Gossip learning of personalized models for vehicle trajectory prediction

Gossip Learning of Personalized Models for Vehicle Trajectory Prediction

Mina Aghaei Dinani, Adrian Holzer
University of Neuchatel, Switzerland
name.surname@unine.ch

Hung Nguyen
The University of Adelaide, Australia
hung.nguyen@adelaide.edu.au

Marco Ajmone Marsan
Politecnico di Torino, Italy and Institute IMDEA Networks, Spain
ajmone@polito.it

Gianluca Rizzo
HES-SO Valais, Switzerland, and University of Foggia, Italy
gianluca.rizzo@hevs.ch

***Abstract*—Gossip Learning (GL) is a peer-to-peer machine learning protocol based on direct, opportunistic exchange of models among nodes via wireless D2D communications, and on collaborative model training, which has recently proven to scale efficiently to large numbers of nodes, and to offer better privacy guarantees than traditional centralized learning architectures. Existing approaches to GL are however limited to scenarios in which nodes are static, or in which the node connectivity graph is fully connected, and they are fragile to node churn as well as to any change in network configuration. To overcome this limitation, we present a new decentralized architecture for GL suitable for setups with dynamic nodes, which benefits from node mobility instead of being hampered by it. In our approach, nodes improve their personalized model instance by sharing it with neighbors, and by weighting neighbors' contributions according to an estimate of their marginal utility. We apply our GL algorithm to short-term vehicular trajectory estimation in realistic urban scenarios. We propose a new strategy for the estimation of the neighbors' instances marginal utility, which yields satisfactory trajectory estimation accuracy for nodes with long enough sojourn times.**

I. Introduction

The amount of data produced by affordable edge devices and sensors has been increasing exponentially over the recent past, thanks to the increasing pervasiveness of the Internet of Things (IoT) paradigm, the growing interest in Smart City applications, and the gradual deployment of 5G networks [1], [2], [3]. The amount of data traffic is poised to grow faster than the number of connections because of the increased use of data hungry applications, such as telemedicine and smart driving systems [4]. Much of this data is key for enabling decision making so as to improve the behavior of complex systems. For example, vehicle trajectory prediction creates new opportunities for improving transport services and reducing traffic congestion [5]. An increasing number of services and applications in the vehicular domain rely on trajectory data and AI/ML (artificial intelligence, machine learning) techniques to improve their effectiveness and efficiency.

Standard ML algorithms rely on training over datasets built from a large number of sources, and typically stored centrally, on one machine or in a data center. Storing such data in a central place has become more and more problematic because of data protection rules, and customer privacy concerns. In addition, collecting data from different devices can be inefficient and their transfer can quickly clog the available bandwidth. Several distributed ML approaches have been proposed to overcome these limitations [6], [7]. Among these, Federated Learning (FL), first introduced by Google [8], is based on a synchronous coordinator-client (server-client) architecture. FL collaboratively learns a single consensus model for all clients from decentralized data without the need to store data centrally. Data remains where it was generated, which guarantees privacy and reduces communication cost. However, learning heavily depends on the coordinating server, which causes scalability issues with large numbers of nodes.

Decentalized ML algorithms have been proposed to tackle scalability issues. In these algorithms, learning is implemented collaboratively amongst all nodes with no central server, aggregator or coordinator. One of the state-of-the-art approaches in this field is Gossip Learning (GL) [9], which implements a decentralized version of FL. Each node in this distributed algorithm acts as a client for other nodes. At the same time, it can act as a coordinating server that merges received models. GL has been applied to different ML problems. In [10], a gossip protocol is used in which local models are distributed over a logically fully connected peer-to-peer network serving an application of distributed learning for medical data centres. However, the solution has scalability and connectivity issues of its own. In [11], a segmented gossip aggregation is introduced. The global model is divided into non-overlapping subsets. Local learners aggregate the segmentation sets from other learners. The proposed approach is application-dependent and not suitable for more general machine learning contexts. Savazzi et al.[12] proposed a fully serverless FL approach, in which nodes receive a combined model from their neighbours, and each one independently performs training steps on its local dataset. Then, similarly to [10], nodes forward updated models to their one-hop neighbourhood for a new consensus step. Their goal is exploiting a serverless consensus paradigm for FL and enhancing the speed of convergence. Individual model instances are not personalized to increase local performance. Moreover, the majority of existing GL approaches consider scenarios in which either each node communicates with all other nodes, or the connectivity graph is static. By not accounting for churn and mobility, these approaches are not applicable in dynamic setups such as vehicular ad hoc networks (VANETs).

In this work we consider a scenario in which the learning

agents are vehicles moving in an urban setting, and exchanging information directly in an ad-hoc mode, e.g. via cellular D2D, WiFi direct, or DSRC [13], [14]. We consider the case in which each vehicle has to train an LSTM model in order to predict in an online manner its own trajectory, for purposes of vehicular traffic management, or for the implementation of coordinated driving, or for enabling proactive resource allocation algorithms, e.g. in Mobile Edge Computing (MEC) schemes [15].

We propose a decentralized GL scheme which is online, peer-to-peer and based on asynchronous communications. Each node in this network uses its local dataset to improve the model instances of nodes it meets opportunistically. At the same time, each node acts as a coordinating server that merges received models in order to improve the performance of its own personalized model instance.

We present three practical algorithms, called DFed Avg, DFed Pow and DFed MinLoss, to personalize the model instance of each node, based on iterative model averaging. We evaluate our approach considering a dynamic time series data set. We perform our numerical experiments over measurement-based mobility traces in an urban setting. Result over a clean-slate scenario suggest that these approaches already perform well when vehicles spend at least about 20 minutes in the considered area, even with dynamic, unbalanced and non-IID data distributions.

The rest of this paper is organized as follows: Section II describes the system model. In Section III, GL algorithms are described in details. Section IV is dedicated to numerical evaluation and results, and finally future work is discussed in Section V.

II. System model

We consider a set of wireless nodes, moving on a finite region of the plane according to an arbitrary *stationary* mobility model. We assume nodes know exactly their position at any point in time. Nodes communicate among them using a wireless technology (e.g. WiFi, DSLR, Cellular D2D, among others). We say that two nodes are *in contact* when they are able to exchange information directly.

We assume that the given region of the plane is partitioned into *cells*. Location, size and shape of the region, as well as of each cell are typically determined by the specific application scenario requiring trajectory prediction. For instance, in a setup where vehicles offload part of their computing tasks to a MEC service, a cell of our system may correspond to the coverage area of the roadside unit(s) to which a specific MEC server is associated. The choice of the region instead depends on the spatial range of the specific service requiring trajectory predictions.

In addition to these cells, we define an *outside cell*, consisting in the area of the plane lying out of the given region. Without loss of generality, let us assume time to be divided into slots, and let Δ be the slot duration. Let v be the unique identifier of a vehicle in the given scenario. Starting from the slot at which the $v-th$ vehicle enters the given region (which we denote as slot 1), each vehicle samples its position in space at each time slot. If t is the label of the $t-th$ slot since ingress time, with (x_t, y_t) we denote the vehicle position at the beginning of that slot. The resulting time series $\{(x_t, y_t)\}_{t=1,\ldots,t_0}$ constitutes the *local dataset* of the vehicle up to the $t_0 - th$ slot from node ingress.

As explained, we consider a scenario in which at any slot each user (vehicle) tries to predict its own location h slots ahead in the future. The outside cell is used to predict whether a node will get out of the given region in h slots. In order to avoid trivial prediction tasks (due to e.g. regular mobility patterns of vehicles) in what follows we adopt a *clean-slate model*, by which we assume that nodes entering the given region possess an empty local dataset. This models the worst-case condition in which vehicles in the scenario are new to the area considered, and thus they cannot rely on data collected before entering the given region for elaborating a prediction of their trajectory. Though far from ordinary operating conditions in a realistic setting, the clean-slate assumption allows a first conservative assessment of our approach, in a way which is independent from any context-dependent assumption on the composition or the size of the initial dataset. As in machine learning techniques more data imply (at least the opportunity of) better performance for the model trained on that data, results obtained in a clean slate scenario may be seen as lower bounds on the actual performance achievable with our approach in a realistic scenario. Of course, note that our approach can be easily extended and adapted to the case in which such assumption does not hold, though in that case the performance (as well as the necessary adaptations) would be a function of the specific assumptions made on the composition of the initial dataset of each node.

III. A distributed architecture for Gossip Learning

In this section, we outline the architecture of our new GL protocol, whose goal is to endow each node in the scenario with a machine learning based model capable of accurately predicting its trajectory.

A. A LSTM architecture for trajectory prediction

In order to implement our learning architecture, we assume that each nodes employs a Long Short Term Memory (LSTM) network, a special kind of Recurrent Neural Network (RNN) already proposed in the literature to predict vehicle trajectories [16]. For the task of time series forecasting, when a sequence of input and output multi-variant data is available [17][18], encoder-decoder LSTM has been shown to outperform other approaches for motion prediction, such as Kalman filtering [19] or Support Vector Machines [20], as their lack of depth does not allow satisfactory prediction performance.

Thus, we assume that each node that traverses the considered region uses his local dataset to train an encoder-decoder LSTM model. Once the model is trained, at every time slot each node feeds the LSTM model with the last α samples of the time serie representing the vehicle's own trajectory in the last α time slots, and it outputs a prediction on where the node will be in h time slots in the future.

The detailed architecture of the encoder-decoder LSTM model is as follows. This model is the concatenation of two LSTM layers (encoder and decoder), each with 50 neurons. The LSTM encoder accepts as input a time series $\{(x_t, y_t)\}_{t=t_1,...,t_1+\alpha}$ of size α by 2, representing vehicle trajectories over α consecutive time slots. The output of the first LSTM layer (encoder) is a fixed-length vector which captures the temporal structure of the past trajectory. The second LSTM layer (decoder) maps the vector representation back to a variable-length target sequence. The target sequence is the *cell trajectory* $c_1, ..., c_h$ of length h, i.e. a sequence of h cell labels describing the cell in which the vehicle is predicted to be, from time slot $t_1 + \alpha + 1$ up to time slot $t_1 + \alpha + h$. The resulting LSTM network is trained over a sequence of epochs, i.e. of learning iterations performed over the entire dataset. In each epoch, a node trains once the local model instance using its local dataset, and it updates the model parameters. We adopt a mini-batch Gradient Descent training approach, in which the training during one epoch is partitioned in two or more batches of size 32. Finally, we have adopted the Adam optimizer [21] in our model because of its computational efficiency and simplicity, as it requires little memory, and is well suited for problems with many parameters. It is also appropriate to cope with non-stationary objectives. Please note that the number of neurons, the batch size, the number of input time steps as well as the other hyperparameters of our LSTM network and of the training process have been tuned through extensive simulations and testing.

B. A framework for collaborative training in dynamic settings

In this section we describe a decentralized, collaborative algorithm for training the LSTM network, in a way which maximizes the accuracy of the prediction for each user. An outline of our proposed strategy for decentralized GL is presented in Algorithm 1.

Our algorithm is structured as follows. We assume the time spent by each node in the region to be divided into two stages.

Initialization stage. It starts from the time slot in which the node enters the considered area (or from the beginning of the scheme if the node is already present in the given region at that point in time), and it lasts V slots. During this stage, the node does not possess yet a sufficiently large dataset to train its local model. Thus the node does not perform any prediction, but it collects data and it builds its local dataset.

Finally, at the end of the initialization stage, each node initializes the model by training it on its local dataset.

Exploitation stage. This second stage starts at slot $V+1$, and terminates when the node exits the given region.

Three are the main activities of each node in this stage. Firstly, the node keeps expanding its local dataset, by adding in real time data about its current trajectory. In the exploitation stage, the *validation set*, which is used to evaluate the performance of the training process, is constituted by the last V samples of the trajectory of each node.

Moreover, in the exploitation phase each nodes uses its LSTM model to issue a prediction about the cell in which it will be in h slots.

Algorithm 1 Decentralized GL algorithm.
K_j^v is the set of nodes in their exploitation phase which come in contact with node v during the $j-th$ round.

for every node v **do**
  for slot 1 to V **do** ▷ **Initialization stage**
    Update local dataset
  end for
  Train the initial model instance w_0^v over local dataset.
  w_1^v =CLIENTUPDATE(w_0^v)
  for Each round j **do** ▷ **Exploitation stage**
    for Every node $k \in K_j^v$ **do**
      Send model instance w_j^v
      Receive model update $w_j^{v,k}$
      Receive model w_j^k
      ▷ Train model w_j^k on local dataset
      $w_j^{k,v}$ =CLIENTUPDATE(w_j^k)
      Send model update $w_j^{k,v}$ to node k
    end for
    ▷ Merge all received model updates
    $w_j^v \leftarrow$MERGEMODELS($\{w_j^{v,k}\}_{k \in K_j^v}$)
    Update local dataset
    Train w_j^v over local dataset.
  end for
end for

function CLIENTUPDATE(w_j^v) ▷ Run on client k
  Split local dataset into B batches
  Let $w \leftarrow w_j^v$
  for batch $b \in 1, ..., B$ **do**
    $w \leftarrow w - \eta \bigtriangledown (w; b)$
    ▷ η is the learning rate
  end for
  return $w_j^{v,k} \leftarrow w$
end function

The third and foremost activity of the exploitation stage is the training of the local model of the node, through a collaborative training, supported by all the nodes with which the given node comes in contact. The goal is to improve the accuracy of the local model over that which can be achieved by relying exclusively on local training. To this end, from the beginning of this stage, each node partitions time into *rounds*, of duration equal to one or more slots.

From the beginning of a round, and for all its duration, the given node (which we denote as *server node*) sends the coefficients of its local instance of the LSTM model to all nodes which are within its transmission range (which we denote as *client nodes*). Every client node that receives the model instance trains it using a subset of its own dataset.

Once each client node has completed the training of the received instance, it sends it back to the server node if it is still in range and if the round has not passed, and discards it otherwise. At the end of the round, and similarly to traditional Federated Learning, the server node combines the model instances received from clients during the given round, to build a meta-model which is used for issuing trajectory predictions, and for initiating a new training round. Note that each node has control of its data, and never shares its dataset with others.

Instead, nodes share their model instances, thus providing support for protection of data confidentiality. Note also that the frequency of the training of the meta-model over the local dataset is a parameter which can be tuned and adapted to the specific setup considered. In particular, it depends on the duration of a round (in terms of number of slots), and it can be performed at the end of every round, or every n rounds.

We observe that the scheme we just described bears a close resemblance to classical FL algorithms. However, differently than classical FL, every node in the exploitation stage is at the same time playing the role of the *server* (or coordinator) for his own personal learning task, for which it is building its own local model, and of client node for other nodes' learning tasks. That is, each node in the exploitation stage (and thus with a substantial dataset) is available for training the models of all other nodes which come in contact. In this, they play a similar role as client nodes in FL, receiving a model instance from another node, and sending back to it an updated version of the received model instance. Differently than FL however, all client nodes participating in the training process in a round are all those nodes which the server node meets during the round. Moreover, each client is not required to be in contact with the server node for the whole round, as in FL. Instead, it may initiate the exchange with the server node at any time within the round, provided that the contact duration is long enough to allow for model exchange and training.

Among the many hyperparameters of our scheme, a key role is played by the total number of epochs for each training step. Our scheme adopts two different values for this parameter. When acting as client, and training other nodes' models, we chose a maximum number of epochs equal to one. Indeed, though our scheme differs in important ways from traditional, centralized FL, they both share the same client-coordinator structure, for which it has been shown [8] that choosing a single epoch maximizes accuracy while minimizing training time.

When a node trains its local model on its own local dataset instead, the total number of epochs is chosen in such a way as to avoid overfitting while maximizing accuracy.

A key aspect of our GL approach is represented by the strategy used by each server node to combine the model instances received by client nodes in order to produce an updated version of its local model instance (also denoted as *meta-model*). In what follows, we propose three different approaches to merge model instances received from neighbours and create a local model instance by using collaborative learning techniques. The performance of the three approaches will be then discussed in the numerical evaluation section.

It is important to stress that, in the considered setting, the strategies for the merging of model instances are the key factor influencing the system performance, due to the continuously changing set of neighbour nodes. For this reason, in this paper we concentrate on such strategies and on the corresponding model instance weighting approaches.

C. Decentralized Averaging (DFed Avg)

The *DFed Avg* approach is based on the way of combining models which is most frequently used in the literature for FL algorithms. With this approach, w^v is computed through a weighted sum over all clients model instances, in which the contribution of each client node is weighted by the proportion of the number of its samples involved in the training phase (n_k) over the total number of samples in the training phase (N). This approach gives more weight to a model instance which has a larger number of samples. The underlying idea is that to a larger amount of data points used for training it corresponds a more accurate model.

Algorithm 2 DFedAvg

n_k is the number of samples in the local database of client k which have been used for training

function MERGEMODELS($\{w^{v,k}\}_k$, n_k)
  $N \leftarrow \sum_{k \in K_j^v} n_k$
  $w^v \leftarrow \sum_{k \in K_j^v} \frac{n_k}{N} w^{v,k}$
  Return w^v
end function

D. Decentralized Powerloss (DFed Pow)

The *DFed Pow* approach weights each neighbor contribution to the meta-model according to a notion of similarity (in terms of roads and regions of the city covered) among the datasets of the server and of the client nodes. Specifically, the server node uses its local validation set to evaluate the loss value l_k of each of the updates of the model received by client nodes. As a loss function, we considered the well known categorical cross-entropy [22], which compares the probability vector given as the output of the model with a one-hot encoded vector representing the true class, and through a logarithmic formula computes the amount of loss. The result is a loss value which increases as the predicted probability diverges from the actual label.In computing the meta-model, each client contribution is proportional to the inverse of a function which grows exponentially with loss. Specifically, to the model update of each user k we assigned a weight given by $\frac{10^{-l_k}}{\sum_{k'} 10^{-l_{k'}}}$, where the normalization term at the denominator is a sum over all clients which returned a model in the given round.

E. Decentralized Minimum Loss (DFed MinLoss)

Finally, in the *DFed MinLoss* approach, the server selects among the received model updates the one with the lowest loss value (computed as in DFed Pow), and it takes it as its local model instance. This approach originates from the observation that in DFed Pow the meta-model does not always perform better than the single model updates which compose it.

IV. Numerical Evaluation

In this section we present the results of the numerical assessment of our framework for decentralized Gossip Learning.

Algorithm 3 DFed Pow

l_k is the loss of model update from node k (i.e. of $w^{v,k}$) over validation set of node v.

function MERGEMODELS($\{w^{v,k}\}_k$)
  for every node $k \in K_j^v$ **do**
    Compute l_k
    $P \leftarrow \sum_{k \in K_j^v} 10^{-l_k}$
    $w^v \leftarrow \sum_{k \in K_j^v} \frac{10^{-l_k}}{P} w^{v,k}$
  end for
  Return w^v
end function

Algorithm 4 DFedMinLoss

function MERGEMODELS($\{w^{v,k}\}_k$)
  for every node $k \in K_j^v$ **do**
    Compute l_k
  end for
  Compute $k' = \arg\min_{k \in K_j^v} l_k$
  Return $w^{v,k'}$
end function

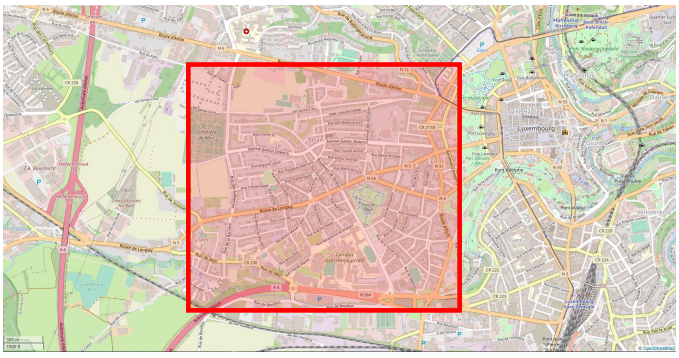

Fig. 1: Map and road grid of Luxembourg City center. The area within the red square corresponds to the region considered in our simulations.

In order to implement a realistic scenario in our simulation experiments, we adopted the dataset of the Luxembourg SUMO Traffic (LuST) Scenario [23], consisting in measurement-based vehicular traces over 24 hours from the city of Luxembourg, with a typical topology common in mid-size European cities, and with realistic traffic demand and mobility patterns. In particular, we considered a square region of side 1050 m (Fig. 1) in the city center, within which we assumed that predictions of vehicle trajectories are required. We partitioned such region into 49 square cells of side 150 m, compatibly with the case in which each cell corresponds to the coverage area of a MEC-enabled small cell base station. As in many real scenarios, vehicular traffic flows exhibit non-stationary properties which might affect the accuracy of our GL trained models. In order to limit such effects of the variations of traffic flows over time, we observed the performance of our scheme over a time interval of one hour (specifically, from 6 : 30 AM to 7 : 30 AM, where vehicular traffic is sufficiently intense and stationary). Such time interval has been chosen to be, on one side, long enough to allow our training framework to progress substantially, but short enough for the average pattern of vehicular traffic flows not to vary significantly. In the given region, on average about 300 vehicles are present at every time instant in the time interval considered, and every vehicle is in the range of about 60 other vehicles, of which on average only half are in the exploitation phase and are thus able to act as clients.

For implementing the simulations of opportunistic communications among vehicles, we adopted the Omnet++ [24] framework, while Keras [25] was used for implementing our GL algorithm. We assumed vehicles sample their position in space every 5 s, i.e. at rate comparable to that common in many present day car fleet management applications. Moreover, we assumed to require a prediction about a vehicle's position in 10 seconds in the future. Such forecast horizon is of the same order of magnitude of those required in, e.g., predictive collision avoidance systems, or in MEC resource preallocation strategies. Our LSTM model input being composed by 24 steps, it requires at least the last two minutes of the trajectory of a car in order to issue a trajectory prediction. We considered each node performs model training over its entire local dataset, adopting a local batch size of 32 data points (coherently with the indications in [26] [27], and a 10^{-3} learning rate, as suggested in [28].

Given that in the scenario considered the average sojourn time of vehicles in the given region has been around 20 minutes, after several experiments for each node we have chosen for the initialization phase a duration of about half of this time, i.e. 9 m 30 s. Indeed in the given setting such a duration is, on one side, short enough to have a substantial amount of nodes in the exploitation stage at each point in time (about 50%), and thus contributing to our collaborative training. On the other, it is long enough to have a substantial training set at each node, and thus to allow the collaborative training to progress at a reasonable rate within the residual sojourn time of each vehicle. Given the relatively short duration of the exploitation phase, we have experimentally verified that the periodic training of the local model instance on the local database during the exploitation phase has no significant impact on model accuracy. Note however that these considerations are strictly related to the characteristics of the chosen scenario, and specifically, to the size and shape of the given region, as well as the mean node speed. In order to perform a first evaluation of our approach, we have assumed that the tasks of model training, of computing a meta-model, and of exchanging a model are instantaneous, thus modeling the case in which the limiting factor of our training framework is given by the way in which training and meta-model computing task are implemented, rather than by their duration.

In order to assess the performance of the training process, the test set consisted in the data points of the trajectory of each vehicle within the last 5 minutes of the vehicle's sojourn time within the given region. With such a choice for the test set (henceforth denoted as *static*), at each point in time during the exploitation stage the evaluation is done on data points about

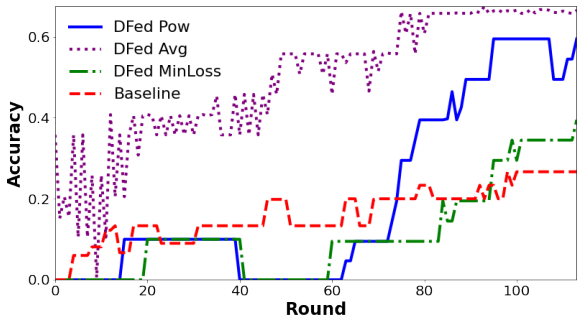

Fig. 2: Mean accuracy versus iterations for our GL algorithm, for the three model merging schemes, and baseline model, with static test set, in the Luxembourg scenario.

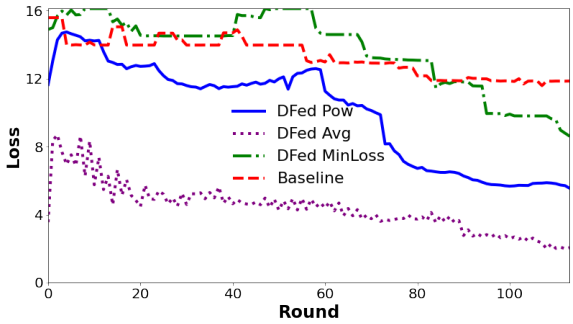

Fig. 3: Mean loss versus iterations for our GL algorithm, for the three model merging schemes and baseline model with static test set, in the Luxembourg scenario.

parts of the map which are generally far from those in which the node is located. This holds also, on average, for the dataset of the client nodes. Finally, in order to better appreciate in what measure our collaborative training improves over purely local training, we have considered this latter approach as as a baseline, and we have characterized its performance.

Fig. 2 and Fig. 3 show accuracy and loss of our GL schemes, averaged over the 30 cars in the area with longest sojourn time, for the static test set approach. As these figures show, one consequence of choosing a static test set is that on average the initial loss is high (and the accuracy low) for all of the three meta-model building approaches, and sometimes even lower than the baseline. The figures also show that these parameters keep on improving steadily across the iterations of our GL training scheme.

These results show also that despite every node enters the scenario with an empty dataset, a substantial improvement in model accuracy can be achieved within a relatively small amount of rounds. The two plots indicate that, when vehicles spend a sufficiently large amount of time in the scenario, GL schemes can collaboratively train models which achieve a substantially better performance with respect to their initial, locally trained models, and over relatively short timespans. Moreover, the plots Fig. 2 and Fig. 3 seem to suggest that, as expected, the main potential factor contributing to such improvement is the progress in model training, i.e. the gradual

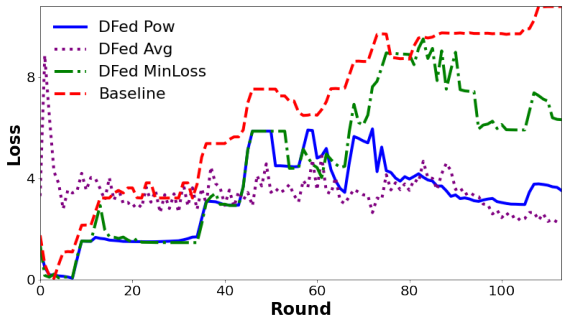

Fig. 4: Mean loss versus iterations for our GL algorithm with a short size rolling test set, for the three model merging schemes, in the Luxembourg scenario.

inclusion over time in the trained model of an ever larger amount of data about trajectories in the given region.

As for the relative performance of the three approaches to meta-model elaboration, Fig. 2 and 3 show that weighting each contribution to the meta-model based on the relative size of the local training set, outperforms approaches based on loss estimation. This might be due to the fact that while the validation set (over which loss is evaluated in the *DFed Pow* and *DFed MinLoss* approaches) consists in the last V slots of a user's trajectory, and thus to a region of space very close to where the user will be in h slots, the test set is relative to a portion of the user trajectory which is (generally, except for the final part of the vehicle's trajectory) far enough from the current vehicle position (and from where it will be) to make the performance over the validation set not indicative of the actual prediction accuracy of the model. I.e., as the model evolves constantly to adapt its performance to the specific prediction task and to the context in which the prediction is formulated, it would make more sense to take as test set data points which are as much as possible related to that same spatio-temporal context.

In order to address this issue, in a new set of experiments, at each time slot t we have adopted as test set the data points relative to the time interval $(i-l, i+h)$ where l is the length of the LSTM input (24 slots), and h is the forecast horizon (equal to 2 slots in our experiments). This should allow a more relevant assessment of model performance, as it is performed over the same segment of a vehicle trajectory over which the prediction is performed. As Fig. 4 and 5 show, when assessed with a rolling test set the performance (both in terms of accuracy and of loss) of the models trained with our GL approach is generally less pessimistic than with a static test set. In addition, it remains approximately constant as the number of rounds increases, if not for fluctuations which are due to spatial in homogeneities in node density, in the structure of the road grid, and to bursts of nodes arriving and leaving the considered region. Moreover, in this setting the loss-based meta model building approaches perform generally better than the strategy based on the relative size of the local dataset of client nodes, with the DFed Pow performing slightly better than the DFed MinLoss. This is likely due to the fact that test set and

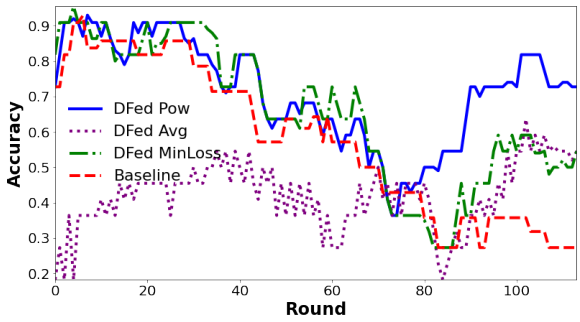

Fig. 5: Mean accuracy versus iterations for our GL algorithm with a short size rolling test set, for the three model merging schemes, in the Luxembourg scenario.

validation set are now relative to contiguous regions of space and time interval, making loss measured over the validation set a more relevant indicator of the performance over the (rolling) test set. Note that improvement over the baseline becomes noticeable only after that a substantial amount of rounds has passed. This is due to the fact that the baseline algorithm has been obtained via training over a dataset which typically contains only a very limited number of cell transitions. Thus, it performs poorly whenever the node moves to another cell, an event whose likelihood grows with time, thus steadily worsening the performance of the purely locally trained model over time.

V. Conclusions and Future Work

The preliminary results presented in this paper suggest that the performance of nodes with too small/poor datasets, or short sojourn times might be improved by having nodes with better models spread them opportunistically, and let other nodes use such received models as starting point for the GL algorithm. We thus plan on expanding the approach discussed in this paper with other forms of model exchanges among vehicles, which can better enable vehicles with short sojourn times and/or small datasets to contribute significatively to the model sharing scheme. Moreover, we plan on performing a thorough assessment of our algorithms on a variety of other vehicular scenarios, and to characterize their convergence properties.

References

[1] H. B. Sta, "Quality and the efficiency of data in "Smart-Cities"," *Future Generation Computer Systems*, vol. 74, pp. 409–416, 2017.

[2] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs, "Building a big data platform for smart cities: Experience and lessons from santander," in *2015 IEEE International Congress on Big Data*. IEEE, 2015, pp. 592–599.

[3] N. Javaid, A. Sher, H. Nasir, and N. Guizani, "Intelligence in IoT-Based 5G Networks: Opportunities and Challenges," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 94–100, 2018.

[4] U. Cisco, "Cisco annual internet report (2018–2023) white paper," 2020.

[5] Z. Xiao, P. Li, V. Havyarimana, G. M. Hassana, D. Wang, and K. Li, "GOI: A Novel Design for Vehicle Positioning and Trajectory Prediction Under Urban Environments," *IEEE Sensors Journal*, vol. 18, no. 13, pp. 5586–5594, 2018.

[6] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," *Advances in neural information processing systems*, vol. 30, pp. 4424–4434, 2017.

[7] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "MLbase: A Distributed Machine-learning System." in *Cidr*, vol. 1, 2013, pp. 2–1.

[8] 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*. PMLR, 2017, pp. 1273–1282.

[9] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[10] M. Blot, D. Picard, M. Cord, and N. Thome, "Gossip training for deep learning," *arXiv preprint arXiv:1611.09726*, 2016.

[11] C. Hu, J. Jiang, and Z. Wang, "Decentralized federated learning: A segmented gossip approach," *arXiv preprint arXiv:1908.07782*, 2019.

[12] S. Savazzi, M. Nicoli, and V. Rampa, "Federated learning with cooperating devices: A consensus approach for massive IoT networks," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4641–4654, 2020.

[13] A. Asadi, P. Jacko, and V. Mancuso, "Modeling D2D communications with LTE and WiFi," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 2, pp. 55–57, 2014.

[14] K. Abboud, H. A. Omar, and W. Zhuang, "Interworking of DSRC and cellular network technologies for V2X communications: A survey," *IEEE transactions on vehicular technology*, vol. 65, no. 12, pp. 9457–9470, 2016.

[15] Z. Sun and M. R. Nakhai, "An Online Mirror-Prox Optimization Approach to Proactive Resource Allocation in MEC," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[16] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction," in *2017 ITSC*. IEEE, 2017, pp. 353–359.

[17] B. Kim, C. M. Kang, J. Kim, S. H. Lee, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *2017 IEEE ITSC*. IEEE, 2017, pp. 399–404.

[18] P. Ondrúška and I. Posner, "Deep tracking: Seeing beyond seeing using recurrent neural networks," in *Proceedings of AAAI*, 2016, pp. 3361–3367.

[19] A. Carvalho, Y. Gao, S. Lefevre, and F. Borrelli, "Stochastic predictive control of autonomous vehicles in uncertain environments," in *12th International Symposium on Advanced Vehicle Control*, 2014, pp. 712–719.

[20] P. Kumar, M. Perrollaz, S. Lefevre, and C. Laugier, "Learning-based approach for online lane change intention prediction," in *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2013, pp. 797–802.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 215–232, 1958.

[23] L. Codeca, R. Frank, and T. Engel, "Luxembourg SUMO Traffic (LuST) Scenario," in *IEEE VNC*, Dec 2015, pp. 1–8.

[24] A. Varga, *OMNeT++*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 35–59.

[25] F. Chollet, *Deep Learning with Python and Keras: The practical manual from the developer of the Keras library*. MITP-Verlags GmbH & Co., 2018.

[26] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[27] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612*, 2018.

[28] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.