Vehicle Position Nowcasting with Gossip Learning

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*—Nowcasting, i.e., short-term forecasting, of end user location is becoming increasingly important for anticipatory resource management in radio access networks (RAN). In this paper we look at the case of vehicles moving in dense urban environments, and we tackle the location nowcasting problem with a particular class of machine learning (ML) algorithms that go under the name Gossip Learning (GL). GL is a peer-to-peer machine learning approach based on direct, opportunistic exchange of models among nodes via wireless device-to-device (D2D) communications, and on collaborative model training. It has recently proven to scale efficiently to large numbers of static nodes, and to offer better privacy guarantees than traditional centralized learning architectures. We present new decentralized algorithms for GL, suitable for setups with dynamic nodes. 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. Our results show that the proposed GL algorithms are capable of providing accurate vehicle position predictions for time horizons of a few seconds, which are sufficient to implement effective anticipatory radio resource management.**

I. Introduction

Vehicle position nowcasting (i.e., short-term forecasting) is attracting increasing attention, since it allows on the one hand to improve transport services and reduce traffic congestion [1], and on the other to implement anticipatory radio resource management in RANs. For instance, the allocation of processors in multi-access edge computing (MEC) facilities, or transmission resources in narrow beam millimeter wave base stations, can be implemented with increased effectiveness if the position of user terminals can be forecasted a few seconds in advance.

Nowcasting of user terminal positions is an obvious candidate for the exploitation of ML techniques. 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. However, 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, since their transfer can quickly clog the available bandwidth. Several distributed ML approaches have been proposed to overcome these limitations [2], [3]. Among these, Federated Learning (FL), first introduced by Google [4], 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. Decentralized 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) [5], 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 plays the role of a coordinating server that merges received models. GL has been applied to a wide spectrum of ML problems. In [6] 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. In [7], 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 it is unclear whether it may generalize to other applications and ML architectures. [8] proposes a fully serverless FL approach, in which nodes receive a combined model from their neighbours, and each one independently performs training on its local dataset. Then, similarly to [6], 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 speed of convergence. However, most of these 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 cannot be extended to dynamic setups.

In our previous work [9] we presented a distributed scheme applied to a classification problem, to derive a forecast about the section of the considered urban region in which a vehicle will be at a given time in the future. Each node implemented a distributed version of FL, periodically sending its model to all neighbors, who train it on their local database and send it back to be merged with those sent by all other neighbors. The main drawback is that only a small fraction of nodes are able to train models which achieve acceptable performance, while those in regions with low node density will consistently experience unacceptably poor performance.

In this paper we aim at addressing the above mentioned open issues. We consider the problem of nowcasting in an urban region, and we propose a gossip learning approach which allows high performing nodes to quickly disseminate their models to those which are not able to build a good one, and to

let their model persist probabilistically in the given area even after they left it. This allows their models to be constantly improved and adapted to changes in mobility patterns even by those nodes which do not possess enough neighbors or data to be able to train a high performing model by themselves. The main contributions of this paper are as follows:

- We propose a novel, fully distributed GL approach for highly dynamical settings, based on node cooperation, on the combination of local model training and fusion of models received via opportunistic interactions, and on the probabilistic persistence of node models in the region.
- We present two practical model aggregation strategies based on iterative model averaging and on two different estimators of the impact of each contribution on the accuracy of the model resulting from aggregation.
- We evaluate our approach over measurement-based mobility traces in several urban settings and traffic configurations. Result over different scenarios suggest that our approach is quickly effective, and able to converge and adapt even in presence of large variations in vehicle density and traffic patterns.

II. System model

We consider a set of vehicles moving on a road grid according to an arbitrary mobility model. We assume vehicles know their position at any point in time, and they communicate in a peer-to-peer fashion, using a wireless technology (e.g. WiFi, or Cellular D2D). When two vehicles are in range of each other (and thus able to exchange information directly) we say that they are *in contact*.

Without loss of generality, let us assume time to be divided into slots, and let Δ be the slot duration. Let $v \in \mathcal{N}$ be the unique identifier of a vehicle. Starting from the slot at which the $v-th$ vehicle enters the considered urban region (which we denote as slot 1), with $(x_t, y_t)^{(v)}$ we denote the vehicle position at the beginning of the t-th slot since ingress time. We consider a scenario in which at any slot each vehicle tries to predict its own location h slots in the future, where h is thus its *forecast horizon*, equal for all vehicles. We assume that vehicles entering the region possess a *local dataset*, composed by time series pertaining to their past trajectories in the region.

III. A distributed Gossip Learning architecture

A. Model architecture

We assume that each node employs a Long Short Term Memory (LSTM) network, a special kind of Recurrent Neural Network (RNN) widely used for vehicle trajectories prediction [10]. For time series forecasting, when a sequence of input and output multi-variant data is available, encoder-decoder LSTM models have shown to outperform other approaches for motion prediction, such as Kalman filtering or support vector machines [11], [12].

The LSTM model of each node is given by the concatenation of two LSTM layers (encoder and decoder). The encoder accepts as input a time series $\{(x_t, y_t)\}_{t=t_1,\dots,t_1+\alpha}$ of size α by 2, representing vehicle trajectories over α consecutive time slots. The output of the encoder layer 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 a sequence of h coordinate labels, describing the position in which the vehicle is predicted to be, from time slot $t_1+\alpha+1$ up to $t_1+\alpha+h$. 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. Finally, we employ the Adam optimizer [13] because of its computational efficiency and simplicity, as it requires little memory, and it is well suited for problems with many parameters. It is also appropriate to cope with non-stationary objectives [13].

Once the model is trained, at every time slot each node feeds the LSTM with the last α samples of the time series representing the node's own trajectory, and receives a prediction on where the node will be in the h future time slots.

B. Gossip Learning algorithm

In what follows we detail the GL algorithm run by each node, which aims at generating more accurate models than those resulting from training exclusively on the local dataset of each node. At the moment on which a node enters the region, it is endowed with a local dataset, as well as with a default instance of the encoder-decoder LSTM model, possibly incorporating a priori data on vehicular trajectories in the region. On entering the region, the node trains the default model instance on its local dataset. Then, at each time slot spent within the region, each node uses its local LSTM model instance to issue a prediction about its position in the future h slots. In addition, it keeps expanding its local dataset, by adding in real time data about its current trajectory.

Each node partitions the time spent into the region into *rounds*, all of the same duration, equal to one or more slots. Duration and timings of rounds are generally different among nodes. Each round is split into three phases. In the first one, a node sends the coefficients of its local model instance to all nodes with which it comes in contact. At the same time, from each of these nodes it receives their local model instances. In the second phase, similarly to traditional FL, a node combines the model instances received during the round in order to produce an updated version of the local model instance (denoted as *meta-model*). Finally, every m rounds, the node enters a third phase, in which it trains the meta-model on the local dataset, to include data about the most recent part of the node trajectory. Note that each node never shares its own data with others. Instead, nodes share their model instances, thus providing support for protection of data confidentiality. 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. The total number of epochs for each training step 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 to combine the model instances received through opportunistic exchanges. More precisely, model merging determines the coefficients of the meta-model as a linear combination of the corresponding coefficients of the models to be merged. Clearly, the strategy for choosing weights is the key factor influencing the performance of the meta-models and thus of the GL algorithm.

In these settings, static implementations of GL schemes, oriented at characterizing the correlation between the problems solved by neighbor nodes, are not fit for scenarios in which the neighbors of each node (and the specific prediction problem they are trying to solve, and thus their local model instances) change continuously over time. Thus, differently from what done in existing works, in the considered scenario any measure of quality of the models to be merged needs to be based mainly on properties of the models themselves. Following this idea, we propose two different approaches to meta-model building: *Decentralized Averaging* (DA) and *Decentralized Powerloss* (DP). They are both based on associating with each model instance a parameter ξ, representing an estimate of the quality of the model itself. They however differ on how ξ is derived and updated.

C. Decentralized Averaging

This merging strategy is outlined in Algorithm 1. At the beginning of the first round, when the local model instance is trained on the local dataset, ξ is chosen as the number of data points in the local dataset. When a set of model instances are merged, the meta-model is computed through a weighted sum over all merged model instances, in which the contribution of the $k-th$ merged instance is weighted by its associated parameter ξ_k, normalized over the sum of all such parameters from all merged instances. In addition, with the new meta-model is associated a parameter ξ computed as a weighted sum of the ξ_k of all the merged model instances, with the same weights as those used for the derivation of the meta-model itself. When instead a model instance is trained over the local dataset, the new value of ξ is the sum of the old value of the parameter, plus the number of data points used in the training process.

Hence, in DA the parameter ξ of a model instance is a loose measure of the amount of data points included in it by training or merging. Such a choice for ξ is based on the intuition that to a larger amount of data points used for training corresponds a more accurate model. The intuition behind this merging strategy is thus to give a larger weight in merging to those model instances which include the largest number of samples, either directly (through training on local dataset) or indirectly (by merging). In settings where the majority of merged models has a low value for ξ, this might still result in a relatively high contribution of these model instances to the resulting meta-model, at the expense of its performance. For this reason, the linear combination at the basis of the DA algorithm considers only the $\beta\%$ largest contributions (in terms of weight). The parameter β can be tuned for the algorithm to work well also in contexts where nodes with high performing models are relatively rare and sparsely distributed over the region.

Algorithm 1 Decentralized Averaging

ξ_k is the figure of merit for the k-th neighbor of node v at round j

$K'(\beta) \subseteq K_j^v \cup v$ is the smallest subset of $K_j^v \cup v$ composed by those nodes whose weights make up at least a fraction β of the total, where K_j^v is the set of v's neighbors at round j.

function MERGEMODELS($(w_j^v, \xi_j^v), \{(w_j^k, \xi_j^k)\}_{k \in K_j^v}$)

$$w^v \leftarrow \sum_{k \in K'(\beta)} \frac{\xi_j^k}{\sum_{h \in K'(\beta)} \xi_j^h} w_j^k$$

$$\xi_v \leftarrow \sum_{k \in K'(\beta)} \frac{(\xi_j^k)^2}{\sum_{h \in K'(\beta)} \xi_j^h}$$

Return (w^v, ξ_v)

end function

Algorithm 2 Decentralized Powerloss

l_k is the loss of model w_j^k, evaluated over the validation set of node v.

function MERGEMODELS($\{w_j^k\}_{k \in K_j^v \cup v}$)

for every node $k \in K'(\beta)$ **do**

Compute l_k

$p_k = |\log_{10} l_k|$

end for

$$P \leftarrow \sum_{k \in K'(\beta)} p_k$$

$$w^v \leftarrow \sum_{k \in K'(\beta)} \frac{p_k}{P} w_j^k$$

Return w^v

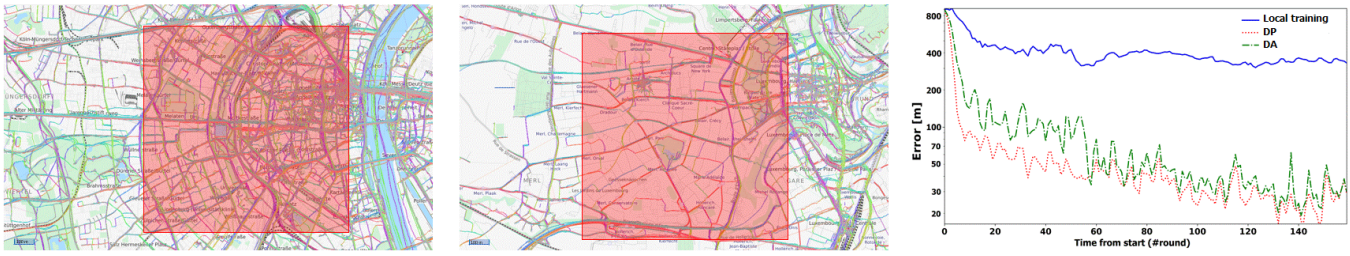
end function

D. Decentralized Powerloss

The DP algorithm differs from DA in the way in which the performance estimator ξ is computed. In DA, parameter ξ is independent from which node will use it to derive a merged model. In DP instead, the node which computes the merged model evaluates the performance estimator for each merging model *just before merging*. It does so by assessing each merged model on its local validation set (consisting in the last V samples of its trajectory), and by associating a loss value with each of these models. Thus in DP the performance parameter ξ is tightly related to the node which performs the merging operation, and to its most recent trajectory. As a loss function we consider mean squared error, where the error is the distance between the actual position of a vehicle and its predicted position. The weight associated with each merged model is then computed as a logarithmic function of the loss, as shown in Algorithm 2, in order to increase the weight of those models with small losses. Note anyway that our approach is very general, and it can be extended to any other type of loss functions.

IV. NUMERICAL EVALUATION

In order to assess numerically our algorithms, we considered scenarios from different cities and times of the day. A first setup exploited the LuST dataset [14], consisting in measurement-based vehicular traces over 24 hours from the



(a) (b) (c)

Fig. 1: Map and road grid of Cologne (Innenstadt) (a) and of Luxembourg City (b), with the considered region in red. c) Mean error versus time from bootstrap for the DA and DP algorithms, as well as for the local training scheme, for the Luxembourg scenario.

city of Luxembourg. The second scenario was based on the TAPAS Cologne dataset [15], again a measurement-based set of vehicular traces covering the greater urban area of the city of Cologne for a whole day. Fig. 1 (a) and (b) show the areas of Cologne and Luxembourg within which our framework was assessed. In both scenarios, the region is a square of side 1 km, covering a large fraction of the city center in both cities. We used Keras [16] to implement our GL algorithms, SUMO for vehicular mobility simulation and the Omnet++ framework for opportunistic communications among vehicles. We considered a slot duration of one second (typical of many present day car fleet management applications). We chose a duration of 15 s for each round, equal for all nodes. Indeed, as vehicles have an average speed of 11 m/s in urban settings, this choice on average ensures that the set of a node's neighbors significantly differs among consecutive rounds. At the same time, such a round duration is short enough to allow the node to capture in its local model changes in the context due to its mobility. We considered a forecast horizon of 5 s, compatible with such applications as predictive collision avoidance systems, MEC processor reservation strategies, and 5G beam resource allocation strategies. The input of the LSTM model is composed of 12 steps, with a gap of 5 s between two consecutive steps. Thus, our LSTM model required at least the last minute of the trajectory of a car in order to issue a trajectory prediction. For local training, we adopted a mini-batch Gradient Descent approach with batches of size 32 (as indicated in e.g. [17]), and a 10^{-3} learning rate, as suggested in [18]. Unless otherwise specified, we assumed $\beta = 1$. The number of neurons (50), the batch size, the number of input time steps as well as the other hyperparameters were tuned through extensive simulations.

To determine the local dataset of each vehicle when entering the region, for each scenario (and for each time interval) we have built a *scenario database*, consisting of all the trajectories in the given region outside of the considered time interval.

In each scenario, for each vehicle entering the given region, we have drawn at random a subset of the respective scenario database, corresponding to 5 minutes of trajectories. We have assumed that the tasks of local model training, of meta-model computation, and of model exchange among nodes can all be completed within a single time slot. In order to evaluate the impact which model merging has on the performance of our training algorithms, we have considered also the case in which each node trains the model only on the local database, thus performing no model exchange.

In a first set of experiments, we aimed at assessing the performance of our schemes in conditions which are as close as possible to stationary, in order to evaluate the convergence speed of the training process and the accuracy of the models in a way which is not significantly affected by underlying changes in mobility patterns. To this end we considered a one-hour time interval, from 6:30 to 7:30 AM, in the in the Luxembourg scenario. Such duration is, on one side, long enough to allow our training framework to progress. On the other side, we have verified that it is short enough for the average pattern of vehicular traffic flows not to vary significantly within it. Moreover, this time interval corresponds to one of the daily traffic peaks in the city, and it is thus appropriate for evaluating the impact of high vehicular densities on our algorithms. In the given region, 300 vehicles are present on average at every time instant in the time interval considered. In this setup, we have assumed a transmission radius of 150 m. As a result, on average every vehicle is in the range of 60 other vehicles.

One of the main performance metrics of our algorithms is the *mean error* at the t-th time slot, defined as the mean distance between the forecasted position and the actual one, averaged across all vehicles present in the scenario during that slot. As Fig. 1(c) shows, both our asynchronous GL algorithms manage to constantly improve model performance over time. In addition, our algorithms converge quickly, reaching a mean error inferior to 20 m in less than 40 minutes of operation. This plot shows also how both our algorithms perform significantly better than mere local training, proving the effectiveness of our asynchronous model merging based on opportunistic model exchanges. Indeed, the strategy based exclusively on local training does not improve its performance over time (except for the first $10 - 20$ rounds), reaching values of mean error more than one order of magnitude larger than through collaborative training schemes.

A key characteristic of our GL approaches is the exploitation of direct peer-to-peer opportunistic exchanges to collect those model instances which will be merged. Fig. 2 shows the evolution over time of the mean error from the start of the scheme for different values of the transmission radius. As the transmission radius affects the mean number of neighboring nodes which at any time instant are in range of a given node, it also directly influences the mean number of models which are merged at each round. From the figure it emerges that a larger amount of models merged at each round not only decreases

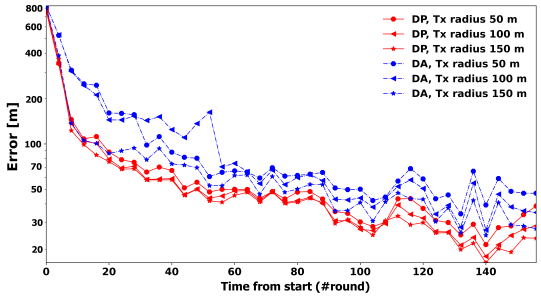


Fig. 2: Mean error versus time from bootstrap (first 40 minutes), for different values of transmission radius, for the DA and DP algorithms, in the Luxembourg scenario.

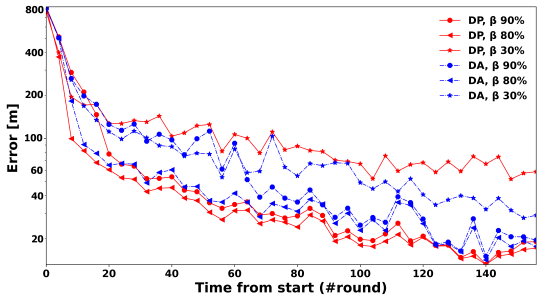


Fig. 3: Mean error versus time from bootstrap (first 40 minutes), for different values of $\beta\%$, for the DA and DP algorithms, Luxembourg scenario.

the mean error, but also helps both our schemes to converge more quickly with the number of iterations.

However, it is expected that what ultimately determines the performance of the meta-model is not the amount of merging models, but their potential to generate a meta-model which performs better than any of them. Nonetheless, when the share of low-performance merging models is preponderant, both DP and DA might still assign too high a weight to them. For this reason, in a new set of experiments we have analyzed the impact of the percentage of model contributions considered in weight computation in our algorithms. Fig. 3 shows that a value of β inferior to 100% may indeed positively affect both accuracy and convergence speed. Of course the optimal value of β (80% in the considered setting) is highly dependent on node density and mobility, on the mean sojourn time in the region, and it thus varies from one setup to the other.

Another key aspect of the performance of our scheme is how quickly a vehicle entering the region is able to improve its accuracy. Fig. 4 shows the mean error as a function of vehicle sojourn time, averaged across the whole time during which our algorithms have been active, for different scenarios and time window duration. As a reference, in Fig. 4 (b) and (c) we have also plotted the mean error resulting from a naive baseline algorithm, by which at any time the future location forecasted by each vehicle coincides with the present vehicle position. In both scenarios, and despite the short time from bootstrap, nodes are able to achieve satisfactory levels of accuracy after only few minutes in the given region. In the Cologne scenario, we witness a convergence which is initially faster than in the Luxembourg scenario. This is likely due to the larger transmission radius considered in Cologne (450 m versus 150 m). At the same time, the shorter mean sojourn time in Cologne (4 min 4 s) with respect to Luxembourg (11 min 12 s) brings the accuracy experimented by a single node to stabilize after the initial rapid improvement. In Luxembourg instead, longer sojourn times allow accuracy to keep improving throughout the whole trajectory of the vehicle in the region, achieving a mean error of 6 m within 20 minutes of sojourn time, and after only 40 minutes from the start of the scheme. Here we note that the absolute values of accuracy reachable by each algorithm have been evaluated on a very conservative worst case scenario, in which each node has only a limited initial database, and in which our scheme has been running for less than one hour. The good performance of our schemes in these conditions suggests that in more realistic settings, where our scheme has been running and training for long periods, the performance of our distributed algorithms is likely to be better than predicted by our experiments.

Another consideration emerging from these results is that DP performs better than DA in terms of both convergence speed and mean error in a consistent fashion, in all experiments. Indeed, while DA's model quality estimate is not directly related to a model's performance at a given point of the region, DP's is instead a function of each merging model's performance over the most recent part of the given node's trajectory, and it is thus close to the performance which the merging model would deliver on the trajectory of the given node in the near future. Finally, Fig. 4 shows how the distribution of the error is heavily skewed towards values lower than the mean

A. *Impact of nonstationarity in vehicular mobility*

One clear feature arising from the numerical assessment of our algorithms is the high impact which node density and mobility patterns have on accuracy and on convergence rate. As in real scenarios, these features change substantially over time, we have performed a set of experiments over time windows during which the configuration of vehicular traffic changes substantially. This corresponded to a time window of three hours, from 4:00 PM to 7:00 PM in the Luxembourg scenario, and from 2:00 PM to 5:00 PM in the Cologne scenario. As Fig. 5 and 6 show, notwithstanding the substantial differences in road grid configuration, time of the day, and mobility patterns, in both scenarios the mean error decreases steadily over time from the start of our schemes, despite variations of up to one order of magnitude in the number of vehicular nodes in the region. These features suggest that, except for minor fluctuations in the mean error, in realistic scenarios our GL schemes are able not only to adapt in a timely manner to changes in patterns of mobility, but also to keep on improving the model performance over time.

V. Conclusions and Future Work

In this paper we have discussed the application of gossip learning for nowcasting in dense urban environments, for the implementation of anticipatory strategies in radio access network management.

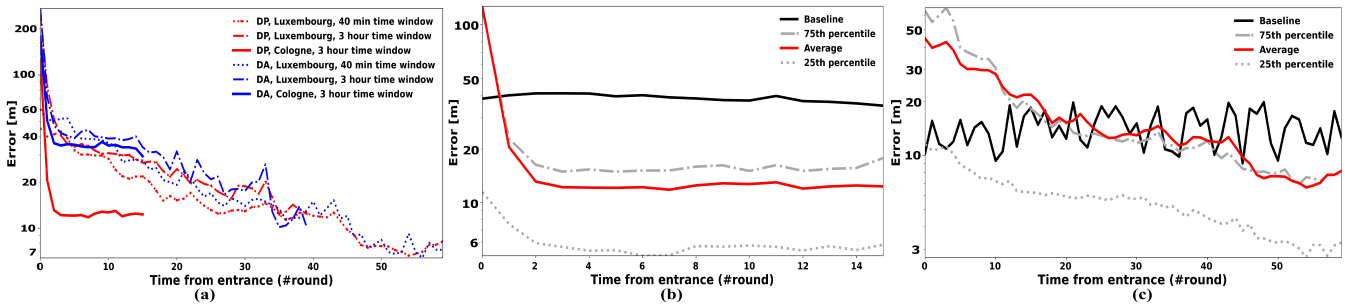


Fig. 4: a) Mean error versus vehicle sojourn time for the DA and DP algorithms. b) Mean error and quartiles versus vehicle sojourn time, for the DP algorithm as well as for the baseline scheme in the Cologne scenario with a 3 hours time window, and c) in the Luxembourg scenario with a 40 minutes time window.

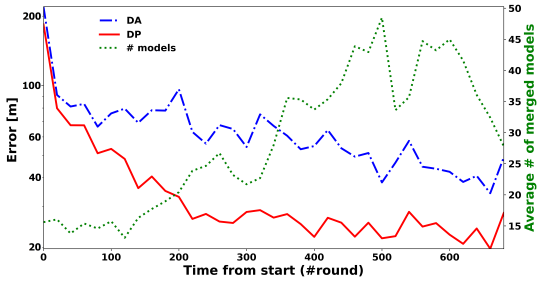


Fig. 5: Mean error versus time for the DA and DP algorithms, from 2:00 PM to 5:00 PM in the Cologne scenario.

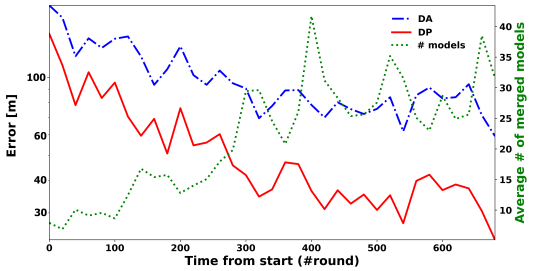


Fig. 6: Mean error versus time for the DA and DP algorithms, from 4:00 PM to 7:00 PM in the Luxembourg scenario.

We have proposed new algorithms for model merging, and we have shown that very good performance can be achieved in realistic dynamic environments within only a few hours from bootstrap, and with very small initial datasets for each node. The future steps of our work will include, among others, the investigation of the effect of heterogeneity in node initial dataset size, computing capacity, and wireless communication technology on the convergence of our schemes and on model accuracy.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] M. A. Dinani, A. Holzer, H. Nguyen, M. A. Marsan, and G. Rizzo, "Gossip learning of personalized models for vehicle trajectory prediction," in *2021 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2021, pp. 1–7.

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

[11] 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.

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

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

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

[15] S. Uppoor, O. Trullols-Cruces, M. Fiore, and J. M. Barcelo-Ordinas, "Generation and analysis of a large-scale urban vehicular mobility dataset," *IEEE Transactions on Mobile Computing*, vol. 13, no. 5, pp. 1061–1075, 2013.

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

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

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