POLITECNICO DI TORINO

Master's Degree in Communications and Computer Networks Engineering

Master's Degree Thesis

Federated learning for vehicle trajectory prediction

Supervisors

Prof. MARCO AJMONE MARSAN

Dr. GIANLUCA RIZZO

Candidate

MINA AGHAEI DINANI

July 2020

Summary

Time Series data has become present everywhere, thanks to affordable edge devices and sensors. Much of this data is valuable for decision making. To use this data for the forecasting task, the conventional centralized approach has shown deficiencies regarding extensive data communication and data privacy issues. Federated learning allows learning from decentralized data without the need to store it centrally. The data remains where it was generated, which guarantees privacy and reduces communication costs [1]. FL already naturally selects only a few nodes randomly at each round. They have non-iid data and also varying in amount. After some iteration of training, the central aggregator will generate a global model. That is, heavily learning depends on a coordinator, which causes scalability issues with large numbers of nodes, and besides, there is a single point of failure, which is not suitable for some applications.

An example of this is predicting the online trajectory of moving nodes to manage traffic, or proactive resource allocation in vehicular networks. In this work, to tackle these problems, we use personalized distributed Federated Learning which is online, peer-to-peer and provides asynchronous communications. Each node in this network is a client for other existing nodes and use its local dataset to improve their models. At the same time, it is like a coordinating server that merges received models and personalized the model for itself. There can be as many models as many as the number of clients. We present three practical algorithms called DFed Avg, DFed Pow and DFed Best for the serverless federated learning of deep networks based on iterative model averaging, and an empirical evaluation which considers time series datasets and an LSTM model. DFed Avg merges models based on the technique used in Federated Averaging, while DFed Best, and DFed Pow at every iteration rounds use different methods to merge models. They will be discussed in details in chapter four.

Our goal is to evaluate our optimization algorithms, not to achieve the best possible accuracy on these tasks. The experiments demonstrate that these approaches are learning when we have numerous clients with the dynamic, unbalanced

and non-IID data distributions. However, to evaluate their robustness we need to do more experiments.

Acknowledgements

First of all, I would like to pay my special regards to my supervisor Professor Marco Ajmone Marsan, who always encouraged and guided me throughout this period.I wish to show my gratitude to Dr Giannluca Rizzo, my industrial supervisor. He supported me with numerous meetings. He kept his door open for me, and without his persistent help, the goal of this thesis would not have been realized.

I would like to thank my friends Matteo Cognolato, Gaetano Manzo for boosting me up whenever I felt weak. I would also thank Greta Vallero for his insightful and interesting pieces of advice that made starting point easier to me.

Finally, I wish to express my deepest gratitude to my family, and Marie-Bernadette Rey. My husband, Masoud, who is always there for me and without his support, I would never be able to reach here. My sisters, grandmother, and uncle who promote me with their motivation and love.

Table of Contents

List of Tables

List of Figures

Acronyms

FL
Federated Learning

LSTM
Long Short Term Memory

AI
Artificial Intelligent

ML
Machine learning

NGSIM
Network Generator SIMulation

RMS
Root Mean Square

ANN
Artificial Neural Network

RNN
Recurent Neural Network

Chapter 1

Introduction

1.1 Background

Standard machine learning approaches entail the training data to be stored centrally on one machine or in a data centre [2] illustrated in figure 1.1. Storing such data in a central place has become more and more problematic in the past years due to data protection rules. Since May 2018, with the entry into the use of the General Data Protection Regulation, a set of data protection rules is presented for all companies operating in the EU, regardless of where they are based. Stricter rules on data protection mean people have more control over their data, and businesses benefit from a level playing field[3]. In general due to the increasing public awareness to issues related to data handling, increase in the volume and diversity of the data generated at the edge has presented the limitations of cloud computing[4]. For specific scenarios, it turns out to be impractical and inefficient to log large volumes of data to a data center in order to process it [5]. Likewise, uploading the raw data to the cloud is not possible when there is privacy concerns about data generated at the edge [6].These kinds of concerns lead to a new computing framework which moving processing closer to where data is generated which is edge computing[7].

One of the most encouraging applications of edge computing is due to the recent up rise of artificial deep learning. Traditional Machine learning techniques based on neural networks need massive datasets and their performance increases with the amount of available data available[8] . However, this dataset might not be in access centrally because of the inherently diffused type of data, for instance, when data is produced at the edge. According to the International Data Corporation (IDC), “ more than 40 per cent of IoT-created data is saved,processed, and acted upon close to, or at the network edge ” [9].

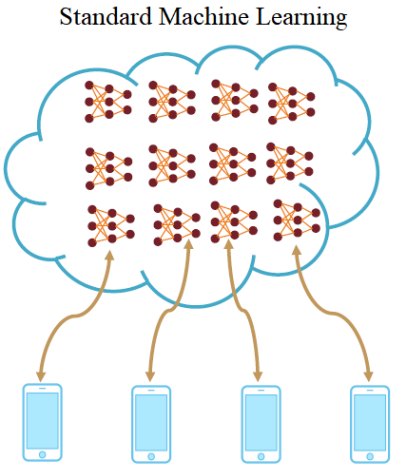

Figure 1.1: Standard Machine Learning

This growth in Mobile Edge Computing requires a new learning technique that can be used at the edge. Researchers in this field have presented different algorithms to overcome the limitations of general distributed deep learning. The most popular nowadays is Federated Learning introduced by Google [1]. It attempts to solve the problem described above. Thanks to the Federated Learning, mobile phones are able to learn a shared prediction model collaboratively while all the training data has been stored on a device,So there is no need to store data on the cloud. It goes beyond the use of local models to predict on mobile devices, by bringing model training to the device[2]. Distributed and federated learning operate on local datasets with different properties, as federated learning aims to learn over heterogeneous datasets, when datasets are non-iid, unbalanced and have different size [10].

The main advantage of Federated Learning approaches to traditional machine learning is to make sure about data privacy or data secrecy. No raw data is uploaded to the server or exchanged. Since the entire database is on the local machine, this makes it more difficult to hack into it. With federated learning, clients exchange machine learning parameters. Besides, such parameters can be encrypted before sharing between learning rounds to prolong privacy [11].

Federated Learning algorithms may operate based on a central server that organises the different steps of the algorithm and performs as a reference clock. It allows us to train a shared model without storing data centrally. In Federated Learning , there is a coordinating server that manages learning phases and generates meta-model. We have some clients, each has a local dataset which never share

with others. Server sends the latest version of meta-model to some clients, these clients update the received model by using their local dataset. Then clients only send back their updated models to the server. They will not communicate back any data. The central server combines the partially trained models to create a meta-model. It is also called federated model. Figure 1.2 represents a centralized federated learning.

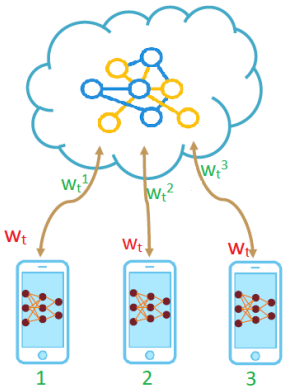

Figure 1.2: Centralized Federated Learning

Centralized FL heavily depends on a coordinator, which causes scalability issues when there are a large number of clients, and the existence of a server means there is a failure point in the system. So this method might not be ideal when there is a time limitation to guarantee a response within any time frame like real-time applications. Also, each device cares more about capturing the pattern in its local data and utilizing model parameters of some intentionally selected nodes by itself more than having a focus on randomly selected nodes by a server to improve its model. For example, with the online prediction of next a few seconds trajectory in vehicular networks, each edge device gathers different data with different patterns as they usually have different sources and destinations, and their patterns vary over time depending on the traffic jam, unexpected events, accidents, etc. In this case, old generated models may not help to estimate the future path. Nodes must update their model frequently. There is a high probability that one car follows the same path as one of its neighbours in the near future, not those nodes which are too far from. Hence, the training model by the help of neighbouring nodes might be more useful than select a node randomly.

In this kind of applications, it is crucial to guarantee a response time. To tackle these problems in the centralized Federated Learning, we use a serverless Federated

Learning approach. This approach is a peer-to-peer and asynchronous communications system where there is no single point of failure, and no scalability issue by having many devices as in the classic Federated Learning approach [12]. As it is shown in figure 1.3 Each node can behave like a server and aggregates models to generate an updated model and select some among existing nodes to transmit the updated shared model. Like before chosen nodes train the shared model on their local datasets and behave as a client.

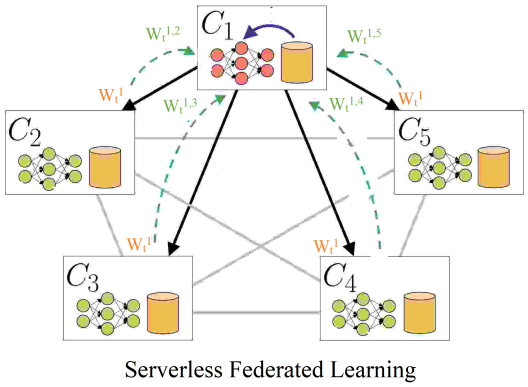

Figure 1.3: Serverless Federated Learning

1.2 Problem

Federated learning aims to create a global model with the ability to generalise it by the usage of clients' local datasets. The data at each node is generated by different distribution, so there is data heterogeneity. When the server aggregates the local model, it cares more about how well the meta-model makes a prediction on all the data, rather than nodes' data individually. In some cases, the performance of the global model on data samples from all nodes might be in tension with the performance of the global model on a node's dataset. Therefore, the idea of one-fits-all might not be useful when we have nodes with different structure present in their data. For example, If there is a global model trained on labelled images and an individual node is more probable to create nature images, that node cares more about the global model working well with nature images than with any other kind of images. Moreover, the existence of a server causes scalability issues with a large number of clients, and also there is a failure point in the system; so this method might not be ideal when there is a time limitation required to guarantee a response within any time frame like real-time applications. For example, in vehicular networks by using self-driving cars; due to the high number of

self-driving cars and the need for them to promptly respond to real-world situations.

Resolving these tensions is the primary goal of this thesis, which has been influenced by previous works. McMahan et al. represented two different algorithms for the federated learning of deep networks based on an iterative model averaging, and conducted an extensive empirical evaluation. Nevertheless, significantly, in this study, the algorithms were synchronous, and they worked with a coordinator server[1]. Virginia Smith et al. modelled the similarities amongst the nodes in the network via diffused multi-task learning, where each node's model is a task, but there exists a structure that associates to the tasks [13]. Others such as Bandara et al. presented two studies in which they group time series based on their similarities into subgroups before using LSTMs to learn across each subgroup of time series. In the first study, groups were made by extracting characteristics from the time series [14]; and in their second study, groups were focused on available domain knowledge[15]. Aymen Cherif and Hubert Cardot also divided the time series into subsets based on similar characteristics before using a recurrent neural network for prediction [16]. All these studies, except McMahan et al. which predominately demonstrated traditional federated learning, assumed that the data is centrally available and make use of traditional learning techniques.

A few attempts have been done to address the problem of decentralized federated learning. In [17], a gossip protocol is used in which local models are distributed over a peer-to-peer network. It considers an application of distributed learning for medical data centres where several servers cooperate to learn a model over a fully connected network. However, they did not take into account network scalability and connectivity issues. In [18], a segmented gossip aggregation is introduced. The global model is divided into non-overlapping subsets. There are some local learners which aggregate the segmentation sets from other learners. The proposed approach is application-dependent and not suitable for more general machine learning contexts. S.Savazzi et al. proposed a fully serverless federated learning approach. In [19] nodes receive a combined model from the neighbours, and each one independently performs training steps on their local dataset, then similar to gossip [17] nodes forward updated model to the one-hop neighbourhood for a new consensus step. Their goal is exploiting serverless consensus paradigm for federated learning and enhancing the speed of convergence. To the best of our knowledge, they are not personalizing the model for each node to increase local performance. Moreover, nodes are stationary, and datasets have fix size.

In this thesis there has been a decision to personalize the model for online trajectory prediction via decentralized collaborative method. So, it is also vital to consider

a few research in motion prediction domain. Many techniques for motion prediction have been suggested in the literature. Many approaches have already been examined for behavior prediction, such as hidden Markov models [20], [21], Kalman filtering[22], Support Vector Machines [23], [24]. However, implementation and the structure of these approaches might be too simple; they do not often perform well for long term prediction. To overcome this limitation recently, artificial neural network approaches have been proposed [25],[26]. Long Short Term Memory and it variations have been used to predict the vehicle trajectory [27]. In [28],The ranging sensor measurements is used to follow the position of the object by using the LSTM. B.Kim et al [29] use the LSTM to predict the position of the vehicle after x seconds using the past trajectory data. To the best of our knowledge, in the above mentioned cases prediction is based on learning on big datasets such as the Next Generation Simulation (NGSIM) dataset[30] by using traditional learning methods.

This study considers this gap worthy of further investigation. In this work, we propose 3 algorithms to personalize the model of each node via serverless federate learning; so there is no single point of failure and no scalability issue. In this way, instead of training a single global model, the idea is to train many models as number of clients in the network to generate the models which are personalized for each client. Clients' datasets are varied in size and pattern. We focus on time series data set which is dynamic. It means samples are added to the network during learning steps (nodes are collecting data samples as time is passing). There is a notion of similarity amongst nodes.

The key problem addressed in this thesis is trajectory prediction in a vehicular network by using represented algorithms. We assume that the personal learning task of each car is to predict where it will be in x time intervals from the present time. The goal of each user is therefore to learn a model which generalizes well to new trajectories drawn from its personal distribution. We therefore focus on a collaborative setting, in which each user minimizes a loss function and maximizes an accuracy metric over its local dataset by leveraging availability of other user's models to improve its personalized model. In chapter four, Decentralized Federated Learning algorithm is explained in details, one of the critical aspects of federated learning is how to build a meta-model. To do so, we have represented some methods to merge models. They are discussed in chapter four. From the results gathered we will show in chapter five how local performance is improved. We will compare these methods in the results to see which one is more superior.

1.3 Benefits

The main advantage of using the federated approaches to traditional machine learning is to guarantee data privacy. Local data is not uploaded externally, or exchanged. Entire dataset only is saved on local device[1]. The rapid growth in smart phone usage, IoT adoption, and big data analytic have led to a massive growth in data centers, but they come with a cost. The amount of energy utilised by data centers increases to double every four years, meaning they have the fastest-growing carbon footprint of any area within the IT sector [31].This technique is in line with what is referred to as 'green computing', as data is saved and processed locally instead of sending it to the cloud. So it reduces network traffic on data centers, in addition personal data is more secure.

Moving data processing closer to the edge decrease latency. It significantly improves application performance thus allowing for real-time data analysis. The goal of this thesis is to evaluate if there is any local performance gain in serverless federated learning to do real-time prediction. In this case there is no single point of failure and scalability issue. Serverless Federated learning can represent a solution for limiting volume of data transfer and accelerating learning processes while there are many nodes and response time is important. This method can be used in transportation system. Implementing proactive strategies for resource allocation, e.g. for MEC services. Self driving cars use many machine learning functions: computer vision for detecting obstacles, applications use in traffic management requires real time information and the processing of short but frequent packets from a multitude of vehicles. Due to the potential large number of self-driving cars, they need to quickly respond to real world situations, so serverless federated learning can be instrumental.

For some companies, like Uber, "forecasting enables to predict user supply and demand in a spatio-temporal fine granular fashion to direct driver-partners to high demand areas before they arise, thereby increasing their trip count and earnings" [32].Another example of the value of real-time processing is Envision, a power producer company who was able to cut its data analysis from minutes to seconds, enabling it to increase the wind turbines production by 15 percent [33]. Therefore, forecasting is of utmost importance for any business, since it guarantees efficient utilisation of capital and correctness of management decisions.

Generally, this method can be used in numerous applications, specifically forecasting applications on time series while data security is important as well as scalability and response time. Such as, Companies that wish to tailor the behavior of their systems; financial risk prediction; electronic health record mining, and smart manufacturing.

1.4 Challenges and issues

There are some significant challenges to be able to online predict the trajectory of a car by using server less Federated Learning when models are personalized. For instance, we do not have fixed number of clients at every time, clients join and leave the system periodically and this means the situation is constantly fluid and changing. In federated learning, learning process usually iterates until the model converges to the optimal model and when model converges; in this network, The number of iterations amongst nodes varies and depends on how long they stay in the city.

Furthermore, the size of datasets may vary significantly. For example, some drivers may be in the city for 10 minutes some more and some less.Their time series dataset is not stationary and they are accumulating data as time is passing. Given this, there is a temporal heterogeneity, each local dataset's distribution will vary with time.These above mentioned factors make splitting dataset to training, validation, and test set very challenging.

Another challenge is choosing the appropriate metrics. Generally in applied machine learning if we choose the wrong metric to evaluate our models, we are likely to choose a weak model, or in the worst case, be deceived about the anticipated performance of our model. There are some standard metrics that are extensively used for assessing and classifying predictive models, such as classification accuracy or classification error [34]. As this body of work is not an extensive study on trajectory prediction to reach the best accuracy, we are going to evaluate optimization of our methods when there is no server. We will choose some metrics because other authors used these metric in analyzing similar problems and it worked for them.

Moreover, the next few second trajectories of each vehicle depends strongly on the surrounding vehicles' behaviors and recent movements. Therefore, all the surrounding trajectories and their interactions should be taken into account. In fact, every car has various numbers of surrounding cars and also different types of dynamic datasets. The analysis of the interaction and the future movement is based on a short period of observations of all surrounding cars' movements and its recent movements. They cannot handle massive data entry and therefore efficiently take full advantage of the dataset. In this work, a Recurrent Neural Network Long Short Term Memory (LSTM) framework is proposed to tackle above mentioned difficulty.

In addition,the way of combining received models from surrounding nodes to build a personalized model for each user is also examined. Doing this will have a

high influence on performance. McMahan et al. in [1] use a weighted averaging method, in which weight of node's model is proportionally to the number of its samples. We suggest a Serverless federated algorithm with three ways of combining models. The effectiveness of the suggested methods will be explored on non stationary time series datasets in chapter four.

Finally, nodes communicate through a wireless channel, which is unreliable. Participators in the training rounds may not be able to send or receive model correctly, so parameters may confront failure, which imposes severe issues on the updated model. Here, we imagine there is no failure in communication among nodes.

The rest of this work is organised as follows; in chapter two, we introduce the relevant theories; in chapter three, machine learning methods are discussed. Chapter four is associated to explain about proposed algorithm, in chapter five, we consider different experiments and display their result, and finally, we have a conclusion and future works.

Chapter 2

Relevant theories

In this chapter, the relevant theory of our research is discussed. The primary goal of this thesis is to evaluate the effectiveness of Serverless Federated Learning by using the proposed algorithm when we are personalizing local model for each node through a collaborative method, while we are going to predict next a few seconds trajectory. The problem at discussion is strictly connected with multiple areas. First, since the main topic is serverless federated learning, it is compulsory to talk about the different type of federated learning as a learning mechanism. Then, the problem that we address in this work is a time series problem, so it is also needed to know about time series and variation of that.

2.1 Federated learning

In the machine learning algorithm, not only model can learn from centrally stored data but also thanks to federated learning, it is possible to learn from distributed data. Federated Learning (FL) makes training possible on a large amount of decentralized data belongs to multiple devices like mobile phones. Data on each client is non-IID and unbalanced.

- Non-IID: each data sample does not have the same probability distribution as the others and also they are not mutually independent.

- Unbalanced: It means in each local dataset, target value has more observations in one or a few specific classes rather than all other classes. In figure 2.1 unbalanced and balanced dataset is shown. In balance dataset female and male have almost the same number of data samples while in another dataset, number of samples for normal transactions is much larget than fraudulent transactions.

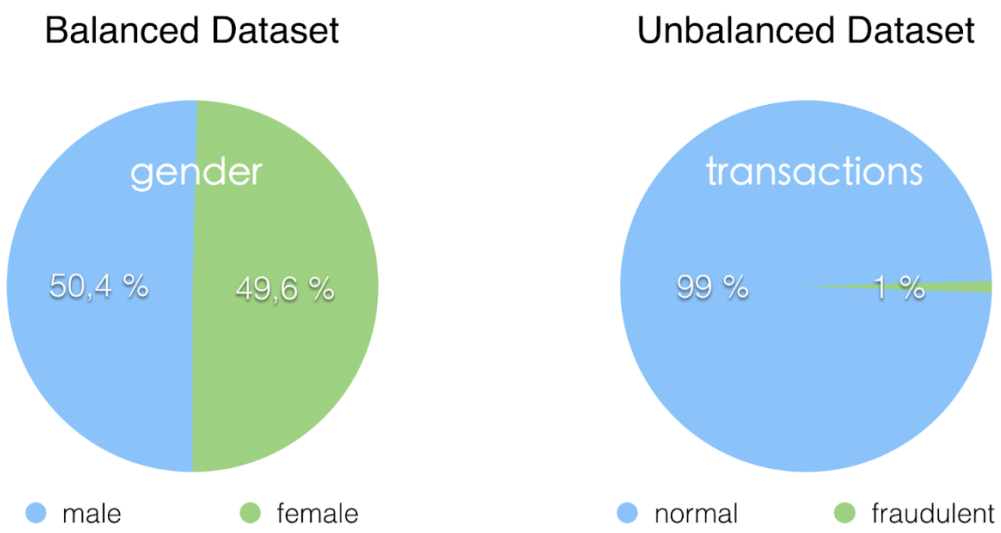

Figure 2.1: Balanced dataset vs. unbalance dataset

In federated learning, transferring data to the cloud is not necessary; Data is stored locally; as a result, we can be sure about data confidentially and decreasing the traffic of the network and diminishing communication costs. Federated learning depends on an iterative process. At every iteration, we have a set of client-server interactions. Every iteration is known as a federated learning round. Server at each round sends the current statistical model to a fraction of local nodes or clients. Clients start training on local data while other nodes do nothing and wait for next federated learning round. Each client sends the locally updated model back to the server. Clients only exchange models and machine learning parameters, and no raw data. The server aggregates all the updated models and creates a meta-model. Process of training by nodes and aggregation by the server is done frequently until a pre-specified termination condition has been happened (e.g. the total number of rounds or average accuracy upper than some targets). In the end, the server contains a robust model which was trained over several rounds on multiple heterogeneous datasets. Explained steps are clearly shown in the photo 2.2 by Jeromemetronome.

On the assumption of existing a central coordinating server, it is called centralized federated learning. A drawback of centralized federated learning is the dependency on a central server. It needs all clients to come to an agreement on one trusted central server. A failure in central point would disrupt the training process of all clients, and it makes our network unreliable. Moreover, heavy dependency on a coordinator causes scalability issues with large numbers of nodes. The way the local updated models are combined, and the methods that nodes communicate

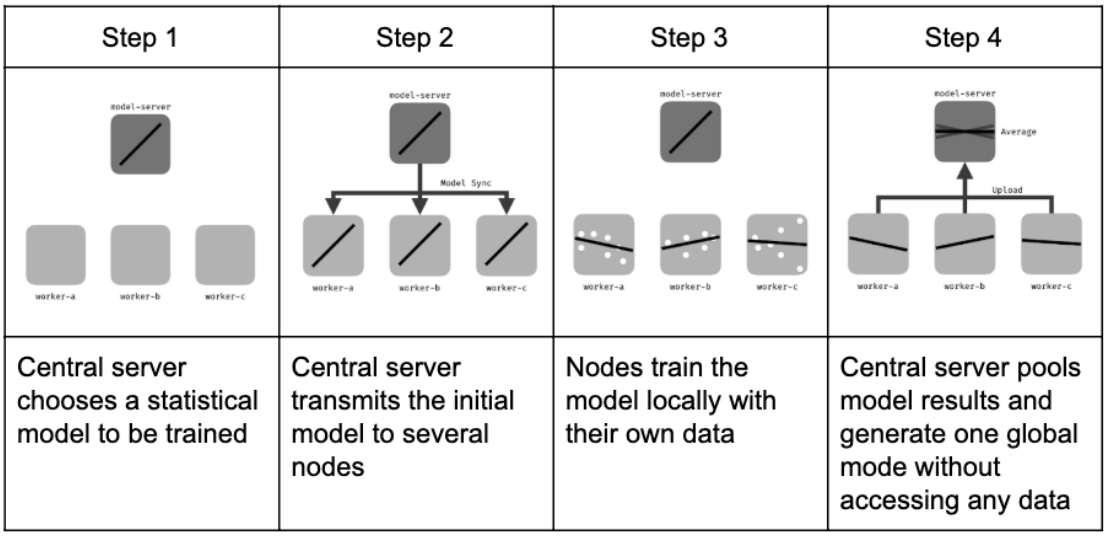

Figure 2.2: Centralized fedeareted learning steps - By Jeromemetronome - licensed under CC BY-SA 4.0
Source: https://commons.wikimedia.org

with each other lead to a variety of federated learning approaches. For instance, no central orchestrating server, or stochastic communication [35]. In particular, serverless federated learning is a prominent variation. In this case, there is no central point of failure and scalability issue. It is a peer-to-peer network. Each local node sends its updated model to several nodes that might be selected randomly, and aggregation of results is done locally. This approach sometimes reduces training and computation cost [36].

However, the core topic of this thesis is based on serverless federated learning; let us be more specific about the details of one proposed federated algorithm to understand this research better. In this part, we stay on the explanation of Federated Averaging of Communication-Efficient Learning of Deep Networks from Decentralized Data, H. Brendan McMahan et al. 2017[1].

2.1.1 The Federated Averaging algorithm

We first introduce some concepts which are main parameters of Federated Averaging algorithm and authors evaluate performance of algorithm by changing theses parameters.

- Epoch:the number of epoch indicates number of learning iteration over entire dataset that machine learning algorithm has to complete. One epoch means

each sample in the dataset only has one opportunity to update parameters. Usually, to generate a model with the high performance, we need to have a high number of epochs, hundreds or thousands. We increase number of epoch until the error from model has become minimized.

- Mini-batch: An epoch is comprised of one or more batches. If the batch size is greater than one and smaller than the size of the training set, it is called mini-batch Gradient Descent. In machine learning examples usually batch size is from 32 to 512. Small values for batch size lead to quick convergence at the cost of noise in the training process. On the other hand, large values give the training process to converge slowly with precise estimates of the error gradient. According to [37] [38] a good default value for batch size is 32.

- Learning rate: one of the key hyper-parameters to set in order to train a neural network is the learning rate. It determines the step size at each iteration while moving toward a minimum of a loss function. This parameter scales the magnitude of our weight updates in order to minimize the network's loss function. If our learning rate is set too low, training will progress very slowly as we are making very tiny updates to the weights in our network. However, undesirable divergent behavior in the loss function can occur if learning rate is too high, it can cause divergence, shown in photo 2.3. The optimal learning rate depends on both model architecture and dataset. Using a default learning rate (ie. the default value of learning rate sets by deep learning library) may provide decent results, we can often increase the performance or speed up training by tuning learning rate and searching for an optimal value.

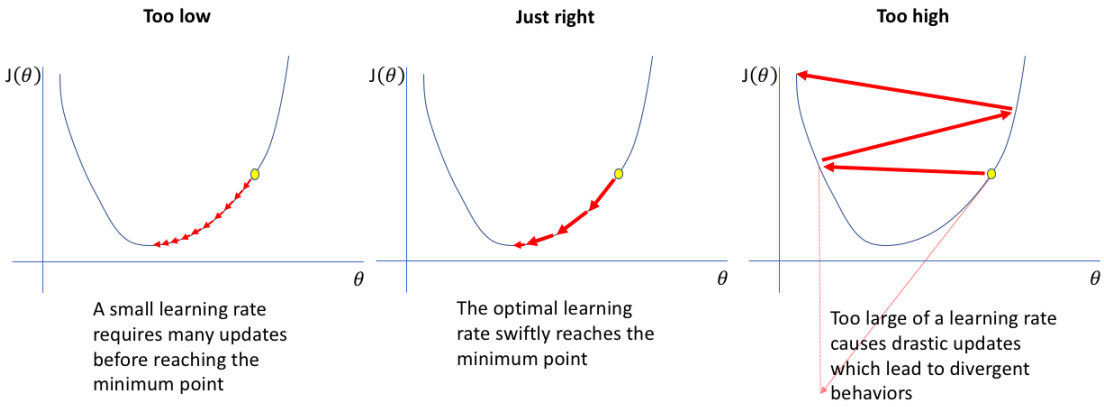

Figure 2.3: Effect of learning rate size on the learning process.

Federated Averaging (FedAvg) is a generalization form of Federated Stochastic Gradient Descent (FedSGD) which is synchronous and centralized federated learning. It is a server-client network. The Server or central coordinator coordinates training steps and generates a meta-model by combining updated local models of clients or local nodes.

In FedSGD, at every training round, a random fraction of all nodes is selected to be involved in the training phase. All the data on these nodes is used to make one step of the gradient descent. Local nodes send updated models back to the server. Then server averages the gradients proportionally to the number of training samples on each node[39].

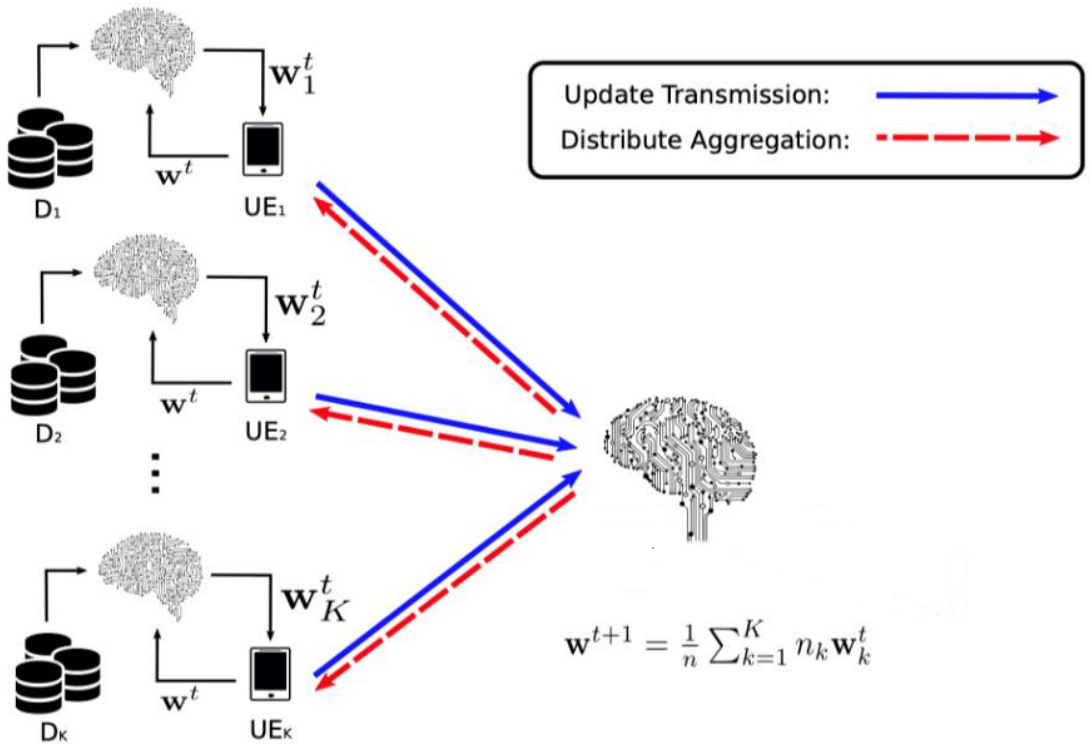

Figure 2.4: The Federated Averaging

While in FedAvg performing more than one batch update on the local data and transmitting the updated weights instead of the gradients is possible. Similar to FedSGD, first of all, the server selects a fraction of total clients. These clients, as mentioned earlier, compute a number of the batch update on their datasets, and local model can be updated for more than one epoch. If algorithm treats the

local dataset as a single mini-batch (batch size equal to infinite) and train samples once(only one epoch), this algorithm corresponds exactly to FedSGD. Then, all the local updates model are sent to the server. Finally, the server aggregates them by computing a weighted average. Each updated model is weighted according to the number of samples in each client's local dataset. For example, if 2 clients out of 10 clients are selected. And the first client has 100 samples in its dataset and the second client has 200; when the central server aggregates the updates from these two clients, it will give more importance to the model generated by the second client (2/3 to the second client and 1/3 to the first one). This algorithm is an iterative algorithm. These steps are done repeatedly until termination conditions happen. This algorithm is given in Algorithm 1 and communication steps are depicted in figure 2.4.

Algorithm 1 Federated Averaging algorithm. Clients are indexed by k, B is the local mini-batch size, E is the local number of epochs and η is the learning rate.

Server execute:
initialize weights w_0
for each round t=1,2,...,r **do**
  $m \leftarrow max$(C.K,1)
  $S_t \leftarrow$ (random set of m clients)
  for each client k $\in S_t$ **do**
    $w_{t+1}^k \leftarrow \text{CLIENTUPDATE}(k, w_t)$
  end for
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{N} w_{t+1}^k$
end for
procedure CLIENTUPDATE(k, w_t) ▷ Run on client k
  $\beta \leftarrow$ split ρ_k into batchsize of B
  for each local epoch i from 1 to E **do**
    for batch b $\in \beta$ **do**
      $w \leftarrow w - \eta \bigtriangledown (w; b)$
    end for
  end for
  return w to the server
end procedure

As presented by McMahan and et al. these four parameters: number of selected clients at each round, local mini-batch size, number of epochs, and learning rate, have an extreme effect on the convergence of learning algorithm. For example, increasing parallelism (increasing number of chosen clients) usually decreases the

number of communication rounds needed to converge. On the other hand, it also outcomes to higher communication costs, so it is necessary to trade-off to find out which one is more important in our problem. Moreover, for very large numbers of local epochs, FedAvg can plateau or diverge. This result suggests that for some models, especially in the later stages of convergence, it may be useful to decay the amount of local computation per round (moving to smaller number of epoch or larger batch size) in the same way decaying learning rates can be useful.

Many researchers have been attracted to the federated learning algorithms recently. Some have presented enhancements to the algorithm, for example, by enhancing the model aggregation [40], or by improving the communication efficiency [41]. It is expected by increasing edge computing devices federated learning, specifically serverless federated learning algorithms, play a vital role in the near future. In the following, we explain about time series dataset and its different kind.

2.2 Time series

A sequential set of observations, normally measured over consecutive times is called time series. Time series forecasting is all about estimating the future. A time series sample is denoted as a set of vectors x(t),t = 0,1,2,... where t illustrates the time elapsed [42]. x(t) is given as a random variable. Depending on the number of random variables, time series is divided to univariate and multivariate time series.

- Univariate time series: If a time series contains samples of only one single variable, it is called the univariate time series. For example, measuring the temperature of room hourly. We have only temperature as a variable.

- Multivariate time series: samples considering more that one variable is termed as multivariate time series. Like the problem in our hands we have a time series of the position of cars, each position is shown by X and Y which are lateral and longitudinal coordinates respectively.

In another classification, the time series can be divided into two categories, continuous and discrete.

- Continuous time series: in a continuous time-series, data is measured at every moment in time; It means there is a data value corresponding to every moment in time. All analogue signals are continuous naturally.

- Discrete time series: observations are measured in time intervals that are usually greater than one second and discrete points of time. Time intervals can be either infrequent (e.g., one data point per hour) or irregular (e.g., whenever

a user logs in). Usually, in discrete time series, the consecutive samples are calculated at equally spaced time intervals such as hourly, daily, weekly, monthly, etc. It shows the continuous time series can be easily converted to a discrete one only by pick samples up in larger time interval, or some cases by averaging data over a specific time interval[43].

The below table represented the first five rows of our dataset. Rows have represented the samples which are collected every five seconds. Columns give features. Each car has a unique id. The city has been divided into different partitions called area or cell represented by character c. X and Y are local coordinates. X, Y and area show the position of each car at every time step in the city. It is illustrated speed of cars in the unit of a meter per second as well.

Simulation time	Car id	speed(m/s)	X	Y	Area
0	1	20	6588.69	6074.39	21
0	2	32	7434.28	5733.13	25
5	1	20	6589.93	6073.25	21
5	2	30	7434.29	5733.17	26
5	3	20	5722.69	5777.38	14

Table 2.1: Examples of time series dataset

Since we have the position of cars every five seconds, by putting cars with the same id within one group, we can present the trace of each car. Understanding the time series pattern is critical to analysis and solve the problem, especially in the implementation task when data is needed to be rescaled. Typically, time series data exhibits different patterns or a combination of different patterns. It is essential to understand the concept of trend, seasonal , and cyclic to recognize a time series pattern.

- A trend exists when there is a long-term tendency in the data. It can be upward, or downward, linear or exponential.
- The seasonal pattern occurs when there are periodic fluctuations in the data always within a specified period or fixed frequency.
- Cyclic happens when there are periodic fluctuations in data but not in a fixed frequency. By comparison to the seasonal pattern not only duration of fluctuations is not fixed but also the duration is more considerable.

It is also possible that for some kinds of time series, like our scenario, data has no seasonality, trend, or cyclic behaviour. Random fluctuations and lack of strong

pattern make prediction very difficult and a model learned on the local dataset might not be useful to make a prediction; hence we use the neighbouring datasets to estimate the future.

Figure 2.5 shows four different patterns for time series data. On the top left the monthly housing sales chart shows seasonality within each year and also cyclic behaviour within a more substantial period about six years. The plot of US treasury bill contracts on the top right shows there is no seasonality, but instead, there is a clear downward trend. On the bottom left, it shows a strong increasing trend with seasonality and finally on the bottom right the daily change in the Google closing stock price, there is no evidence of having seasonality, trend or cyclic behaviour.

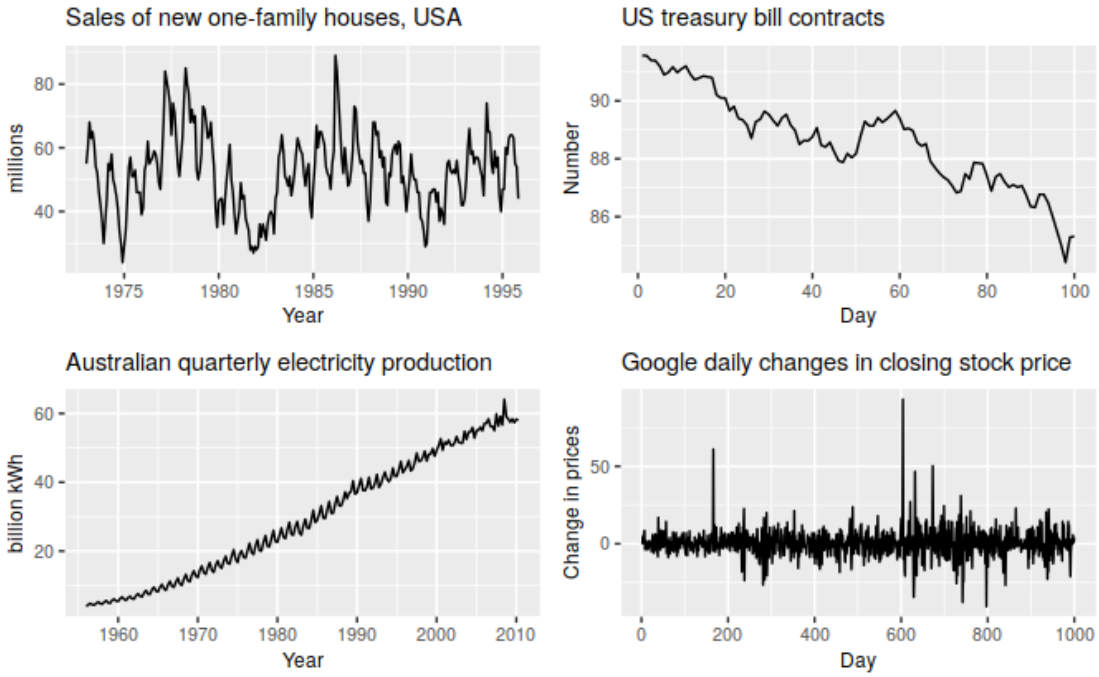

Figure 2.5: Four examples of time series showing different patterns https://otexts.com/fpp2/tspatterns.html

In the next chapter, machine learning methods based on neural networks that can be used to solve the forecasting problem will be discussed. Chapter four contains methodology and proposed algorithms in details. Chapter five covers different experiments and the results will be displayed, and finally, chapter six is about future works and conclusion of our thesis.

Chapter 3

Machine Learning methods

3.1 Background

Models for time series data can have many forms and represent different stochastic processes. AutoRegressive and Moving Average, and their combinations such as AutoRegressive Integrated Moving Average (ARIMA) models are widely used in literature. Due to their comparative simplicity in implementation and understanding, they have drawn much consideration [44]. The drawback of such techniques is the poor robustness when time series has rapid fluctuations[45]. Moreover, these methods work with homogeneous time series[46]; Therefore, they are not in our interest in this work. Support Vector Machine is used in many applications in a variety of domains such as regression estimation and time series prediction [47] [48]. Advantages of using SVM models: they work well when we are dealing with unstructured data, or semi-structured data like text, or image. Besides, the risk of overfitting is less, and they gives better results when compared to Artificial Neural Networks. On the other hand, it is not straightforward to comprehend and interpret the final model variable weights. Moreover,when the training size is large, it necessitates a massive amount of computation which enlarges the time complexity of the solution, also tuning the parameters is not easy [48]. In recent years , Artificial neural networks (ANNs) techniques gained huge reputation and has been recommended to apply in time series forecasting problems [49][50]. Following this chapter is assigned to explain about Artificial Neural Networks and its verities like Recurrent Neural Network and Long Short Term Memory that can be used to solve the time series forecasting problem.

3.2 Artificial Neural Networks

The fundamental objective of ANNs was to create a model for imitating the intelligence of the human brain into the machine [49] [50]. ANNs try to learn from input data and provide a generalized result based on all information that they have already received and known about it. It is similar to what the human brain is doing. In the beginning, the objective of ANNs was biologically motivated, but after a while They have been used in many domains, specifically classification algorithms, and forcasting problems[49] [50]. Below some prominent features of ANNs which make them preferred for time-series forecasting are mentioned.

- ANNs are self-adaptive and data-driven in nature [51].We do not need to specify a specific model form or making any former assumption about the statistical distribution of data. In ANNs, the model is adaptively created based on the data features. This characteristic makes ANNs models quite useful for many practical situations, where there is no information about the data generation process.
- ANNs are fundamentally non-linear, so they are more practical and accurate to construct a model for intricate data patterns, in contrast to several traditional linear approaches, such as ARIMA methods[50][51].
- ANNs' model learns from input data even if input data is fuzzy, erroneous, or it contains missing values [50].

The most general kind of artificial neural network entails of three layers of units: the input layer, hidden layer and output layer. In which input layer units (neurons) are connected to hidden layer units and hidden layer units in layer two are connected to output layer units. As it is shown in the figure 3.1 this network is fully connected.

Input data is fed to the network through the input units. This is also called the input layer.The second layer is called hidden layer. The activity of each unit in the hidden layer is determined by the activities of the input nodes and the weights on the connections between itself and previous layer units. The activity of the hidden units and also the weights between the two last layers determine the behaviour of the output layer.

When we use multiple layers in ANNs it is called Deep Neural Networks. In this network perceptrons stacked one after the other in a layerwise mode. The objective of the network is to approximate a function f^*. For example, by considering a regression algorithm, $y = f^*(x)$, It maps an input data x to an output y which is a real value. A feed forward neural network designates, a mapping $y = f(x,\theta)$, and learns the appropriate value for the weights θ that decrease the error

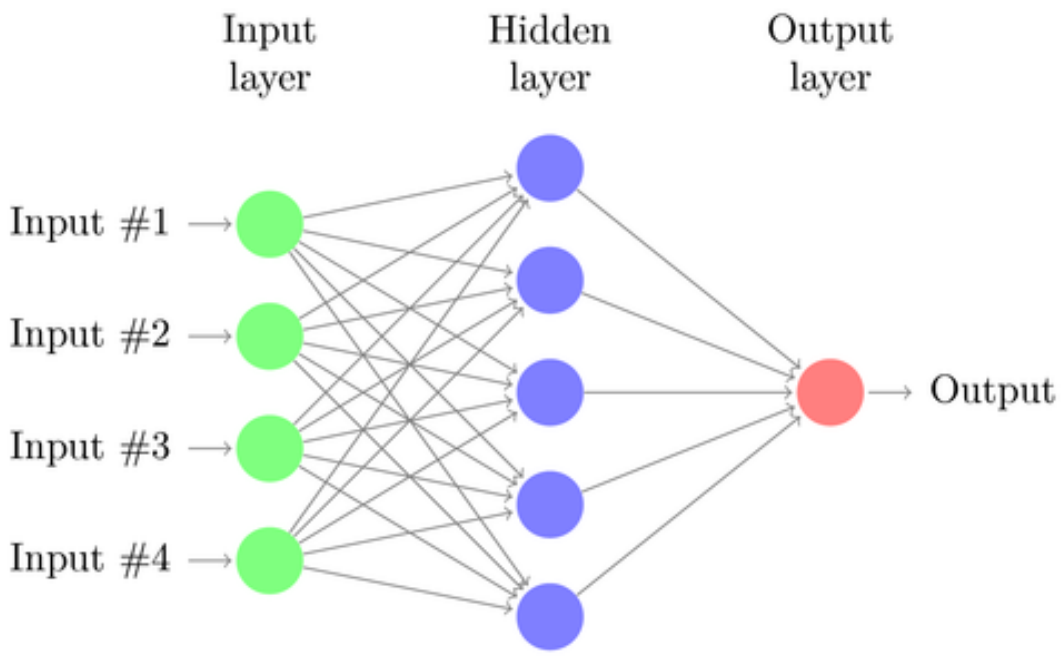

Figure 3.1: Artificial Neural Network with three layers
https://otexts.com/fpp2/nnetar.html

of the estimation. In this networks information always goes from input to the output; it never goes backwards. There are no feedback connections in such a way that outputs go backwards and are fed as input again. Feed forward neural networks do not have the memory to memorize the previous understanding of data [52]. They are not instrumental for time series predictions that we want to understand sequential data. In this work, we focus on an algorithm that have attracted substantial interest in the forecasting field by using Long Short Term Memory networks (LSTM) which is a kind of RNN network. In the following, LSTM and its variety are discussed in details.

3.2.1 LSTM

When we start reading a sentence to understand the concept of that, we do not consider every word separately. We understand each word based on an understanding of the previous words. Feed Forward Neural Networks can not learn through a sequence of data. Recurrent NNs address this problem by having a loop in their structure; it allows information to persist.

RNNs can use their memory (internal state) to process variable length sequences

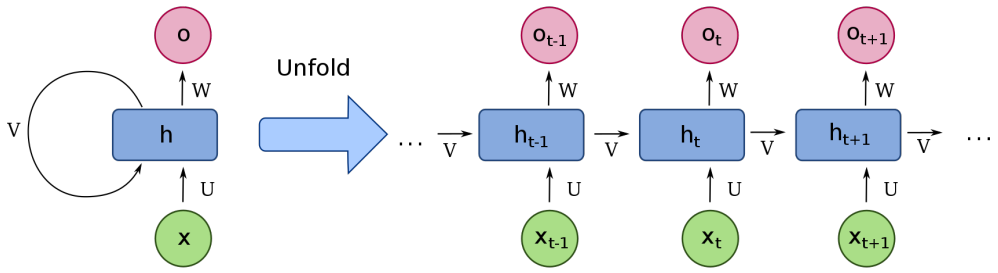

Figure 3.2: Recurent Neural Network.By fdeloche - Own work, CC BY-SA 4.0 https://commons.wikimedia.org/w/index.php?curid=60109157

of inputs [53]. In figure 3.2, we have a chunk of neural network, inputs are shown by x indexed by sampled time, and outputs denote by o, produced at every time step. A loop permits information to be traversed from one step of the network to the next. RNNs can use the past information to learn where the gap between the relevant information and the place that it's needed is small. Long Short Term Memory (LSTM) networks are a special kind of RNN, capable of learning long term dependencies. They were introduced by Hochreiter,and Schmidhuber (1997) [27], and were refined and popularized by many people in following work and were improved by many people. LSTMs work remarkably well on a large variety of problems. They are designed to overcome the problem of RNNs to avoid the long term dependency problem. Remembering information for long periods is practically their default behaviour, not something they try to learn.

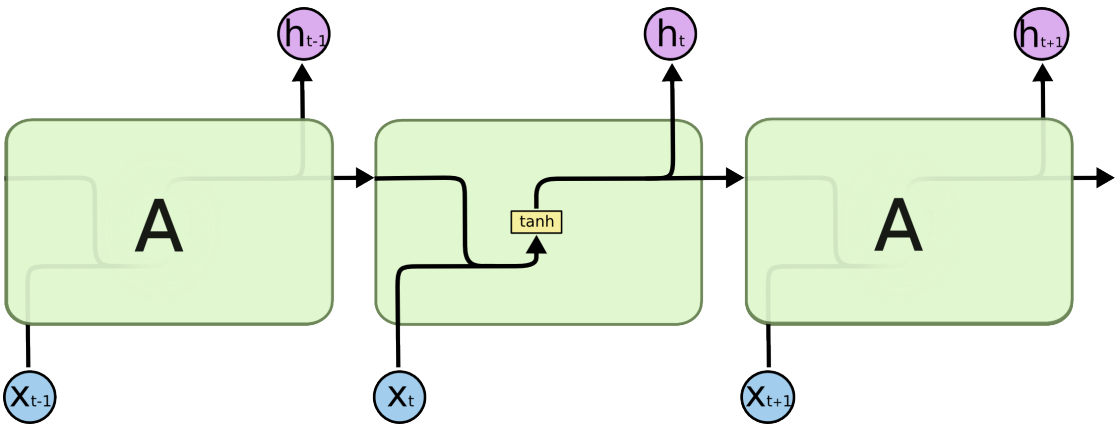

Figure 3.3: The repeating module in a standard RNN contains a single layer

All recurrent neural networks have the form of a chain of duplicating modules of a neural network. In standard RNNs, this repeating module will have a straightforward structure, such as a single tanh layer represented in figure 3.3.

LSTMs also have this chain but with a different structure. Instead of having a single layer, there are four layers. They are interacting in a very special way.

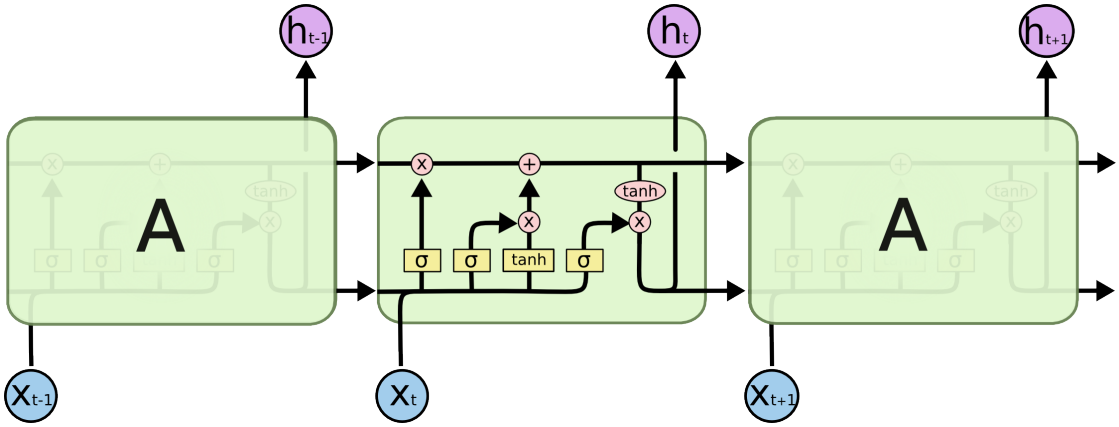

Figure 3.4: The repeating module in an LSTM contains four interacting layers https://towardsdatascience.com/understanding-lstm-networks-by-example-using-torch-c63dba7bbb3c

The main part of LSTMs is the cell state, in the figure 3.4 the horizontal line running through the top of the diagram. It goes through the entire chain, with only some minor linear computations. LSTM can eliminate or add the information to the cell state, adjusted carefully by structures called gates.

Forget Gate: output of the previous state is shown by h(t-1). Forget gate gets the output of the former state. By the help of sigmoid function, which generates a number between zero and one, It takes decisions about what must be eliminated from h(t-1) state and thus holding only relevant stuff.

Input Gate: LSTM decides how much data must be added from the present input to the current cell state. Sigmoid layer determines which values to be updated and tanh layer produces a vector for new candidates to appended to present cell state.

Output Gate: finally, LSTM generates the output of this state. Data in cell

state goes through a tanh to crush the value between -1 and 1. Its result is multiplied by the production of the sigmoid function. The input of the sigmoid function is the output of the previous layer concatenated by the present input[54].

Many types of LSTMs can be used for time series prediction problems. Depending on the number of inputs, and outputs (figure 3.5), and number of variables we must select the proper LSTM which is more suitable for the problem. In the following, we discussed about a few type of LSTMs.

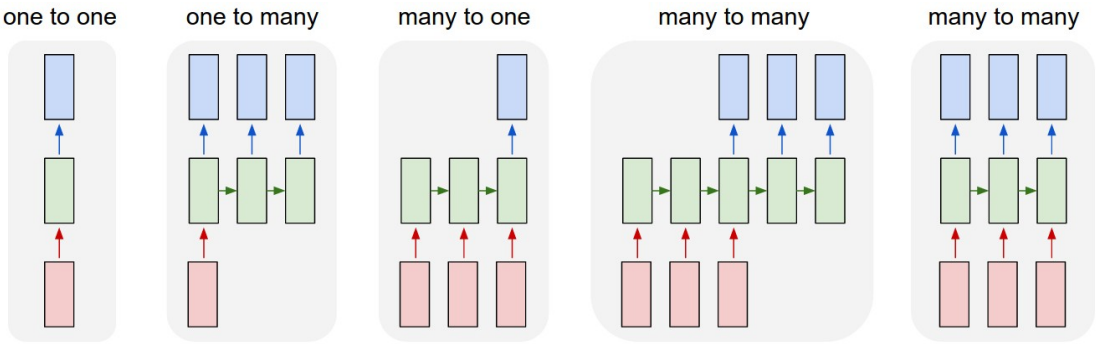

Figure 3.5: Recurrent Neural Network sequence

- Vanilla LSTM: It is an LSTM model that has one hidden layer of LSTM units, and a single output layer used in making prediction. When we have univariate time series forecasting a vanilla LSTM can be used[55].

- CNN-LSTM: A convolutional neural network is a type of neural network developed for working with two dimensional image data. A CNN model can be used in a combination model with an LSTM. This hybrid model is termed a CNN-LSTM. The CNN can be beneficial at automatically extracting and learning features from one dimensional sequence data such as univariate time series data[55].

- Encoder-decoder LSTM: This model specifically developed for multi-step forecasting with multivariate input and output data. As its name suggests, the model is composed of two layers (the encoder and the decoder). In this architecture, the encoder receives the input sequence step-by-step. The output of encoder is a fixed-length vector. This vector encapsulates the information for entire inputs to help the decoder makes accurate predictions. This vector is then given as an input to the decoder that interprets it as each step in the output sequence is made [55].

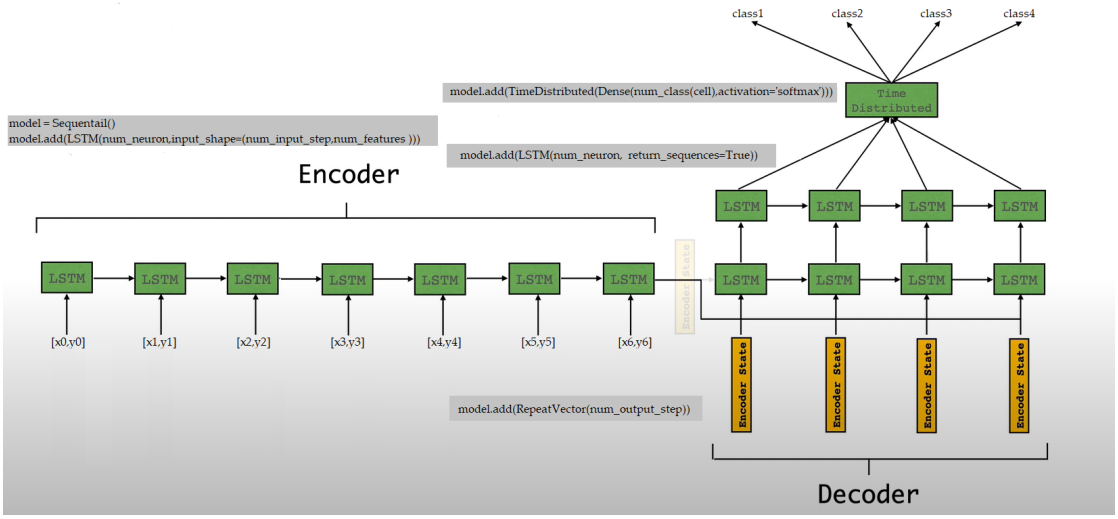

Figure 3.6: Encoder-decoder LSTM

In this work, we use encoder-decoder LSTM to train our data, when multivariate sequence of input is associated to univariate sequence output. Through an example in figure 3.6 how LSTM encoder-decoder is working in keras library is depicted. In this example we have multivariate, multi-step input, and univariate,multi-step output. Corresponding inputs are x and y at different time steps. In code instead of the num-input-step must be written 7(number of input steps), and num-features is replaced with two (x,y). Output of encoder is a fixed-length vector. Its size is equal to number of output steps. In this example, the number of output steps is four. The output of the fixed-length vector goes to the decoder. To implement the decoder model one or more LSTM layers can be used ; return-sequence = True allows us to have two LSTM layers. For the last step, according to the problem, we have to select a proper activation function. The softmax activation function is widely used in multi class classification problems, hence we use it as well.

The next important issue is implementation, i.e. to apply Long Short Term Memory for generating forecasts through a distributed learning algorithm. In the next chapter, our proposed algorithms will be discussed in details and the way of implementation is also discussed and then in chapter five results are displayed; finally, chapter six is assigned to conclusion part and future works.

Chapter 4

Design of serverless distributed FL algorithms for trajectory prediction

In previous chapters, we examined the machine learning methods and theories related to the problem. In this part, we formulate an optimization problem to decrease the loss value on the test set . Then, a computation-efficient algorithm is proposed to approach the optimal solution. The needed practical steps to tackle this problem has also been discussed.

4.1 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. We assume to observe the system over a finite time window. 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 region of the plane is partitioned into *cells*, and let c be the label of the generic element of this partition. Cells can be of any shape and size, but these aspects are typically determined by the specific application scenario. For instance, in a scenario with vehicles in which part of the computing tasks are to be offloaded via a Mobile Edge Computing (MEC) service, a cell might correspond to the coverage area of the roadside unit to which a specific MEC server is associated.

We assume that, from the beginning of the observation window, each vehicle samples its position in space at regular time intervals. Let i be the label of the $i-th$ time interval, and let $Delta_k$ be the duration of such time intervals for the $k-th$ vehicle. The resulting time series $x_1, x_2, ..., x_i$ constitutes the *local dataset* of each vehicle up to the $i-th$ time interval. With $n_{k,i}$ we denote the size of the local dataset of the $k-th$ car at the $i-th$ time interval.

The fundamental problem addressed in this thesis is online trajectory prediction in a vehicular network by using decentralized collaborative learning of personalized models.

4.2 Our DFL algorithms for trajectory prediction

As explained, we consider a scenario in which a MNO receives regularly predictions of car trajectory in order to implement proactive strategies for resource allocation, e.g. for MEC services.

We assume therefore that each user (vehicle) has a *personal* learning task, consisting in predicting its location in h time intervals from the present time, for the whole duration of the observation window. To this end, starting from its local dataset and the one from the other vehicles in the scenario, each user has to learn a model which generalizes well to new trajectories drawn from its distribution.

We focus on a collaborative setting, in which each user minimizes a loss function and maximizes accuracy metric over its local dataset by leveraging the availability of other user's models to improve its personalized model.

In general the outcome is a different model for each user, as we have as many learning tasks as are the users in the area.

Our algorithm is structured as follows. We assume the time spent by each user in the scenario, during the observation window, to be divided into two stages.

- **Initialization stage**. It starts from the time interval in which the node enters the considered area, denoted as i_k (or from the beginning of the observation window, for those nodes present in the scenario at that time), and it ends at time interval $i_k + V$. During this time, the node does not perform any prediction, but it just builds its local dataset. At the end of this stage, the local dataset constitutes the *validation set* for the next time interval. The validation set is used to evaluate the model generated at any time interval.

- **Exploitation stage**. At the end of time interval $i_k + V$, the node initializes the model by assigning random values to its parameters. Depending on the type of time series; if it does not have any strong pattern, the local information might not be useful to predict the future so that parameters can be initialized by random numbers. Otherwise, each node can initialize the model by training it on local data.

  In this phase, a key role is played by two time series: The validation set at time interval i, which is the portion of the local dataset constituted by those samples associated to the time. Based on the specific application scenario, validation set can have static fixed size, or expanding size rolling windows, or fixed size rolling window $(i - V, i)$. For example, in online learning methods which retraining model is done when new data becomes available. This determines whether a fixed or expanding size rolling validation set window will be applied.

  The test set is used to assess the performance (e.g. in terms of accuracy or loss) of the model. Like validation set, there are different ways to select the test set. It can have static fixed size, or a short rolling window size for data sampled in short intervals, and a larger size for data gathered in longer intervals [56].

  For a short rolling window size, the test set at time interval i, constituted by those samples associated to the time interval $(i - l, i + h)$ where l is time duration of input sequence, and h is the forcast horizon. Static fixed test set can be considered as an example: the last 30 per cent of dataset.

To each input we can associate as an output a *cell trajectory* $[c_{t_1}, c_{t_2}, ..., c_{t_l}]$, where c_{t_x} denotes the label of the cell containing the point (x_{t_x}, y_{t_x}).

We assume time to be partitioned into rounds, of equal duration and equal to one or more time intervals.

Nodes can have two roles: client and server. These roles are the same as in classical FL algorithms. That is, a node is a client when it receives a model from another node (called server), it updates model using its local dataset, and send back to the server the updated version. The server node, as in classical FL, combines models to have a federated model and controls learning steps. However, in our algorithm, during the exploitation stage, each node is a server for his personal learning task, for which it is building its model, while at the same time it is available as a client for the tasks of all other nodes in the scenario.

The model of node k is shown by w_k. To do such collaborative learning, each node after x time interval sends its model to surrounding nodes which are in its range. Neighbouring nodes will train received models on their local dataset (Depending on the application scenario, it is needed to determine whether the model will be

trained on all available dataset or only on the most recent observations. This leads to the use of whether a sliding or expanding window of training set), and then will send it back to the source. Each node has control of its data, and never share its dataset with others. Instead, they share their models; this makes sure data confidentiality.

Source node combines received shared model to build an updated model. Then they use the updated model for trajectory prediction. Every x time interval source node will repeat this process to improve its model; so at every round, it gets through the updated shared model w_{t+1} to make a prediction. The maximum number of rounds depends on the duration of time in which node is in the space. One of the crucial aspects of building meta models is how merging models. In the following, we are using three different methods to merge models and create a meta-model by using collaborative learning technique. One of them is the way of combing model that McMahan et al. have used in federated averaging and we discussed it in chapter two. Two new methods are given in the following. The collaborative learning algorithm while there is no server is depicted in algorithm 2.

Algorithm 2 DFed: Decentralized Federated Learning algorithm. The K Clients are indicated by k, E is the number of local epochs, B is the local mini-batch size, R shows the maximum number of round, η is the learning rate, and ρ denotes the local dataset.

Server execute:
do nothing until collect data for validation set
initialize weights w_1
for each round t=1,2,...,R every x time interval **do**
  $S_t \leftarrow$ (Clients in transmission radius)
  Send(w_t,k) ▷ **TX** to neighbours
  for each client k $\in S_t$ **do**
    $w_{t+1}^k \leftarrow$ CLIENTUPDATE(k, w_t) ▷ **RX** from neighbours
  end for
  $w_{t+1} \leftarrow$ MERGEMODELS(.)
  update validation set
end for
procedure CLIENTUPDATE(k, w_t) ▷ Run on client k
  $\beta \leftarrow$ split ρ_k into batchsize of B
  for each local epoch i from 1 to E **do**
    for b $\in \beta$ **do**
      $w \leftarrow w - \eta \bigtriangledown (w; b)$
    end for
  end for
  return w to the server
end procedure

4.2.1 Decentralized Federated Averaging (DFed Avg)

When server receives updated models from neighbours has to merge them to have a meta-model. In the following, we explain the method used in literature to combine models.

DFEDAVG:
procedure MERGEMODELS(.)
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{N} w_{t+1}^k$
end procedure

- w_{t+1}: updated model of server at round t.

- w_{t+1}^k: updated model of client k at round t.
- n_k: the number of samples involved in the training phase of client k
- N: Total number of samples of all clients involved in the training phase.
- K: all clients involved in training phase

w_{t+1} is computed through a weighted sum over all K clients. In which w_{t+1}^k is weighted by the proportion of the number of its samples involved in the training phase (n_k) over total number of samples in the training phase (N). This method gives more weight to a model which has more samples. It seems reasonable when we have more data to train, generation of a model with higher performance is more probable. Pseudo code is illustrated in procedure 1.

4.2.2 Decentralized Federated Powerloss (DFed Pow)

DFedPow:

procedure mergeModels(.)

$l_k \leftarrow$ evaluate loss of w_{t+1}^k on validation set

$p_k \leftarrow 10^{-l_k}$

$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{p_k}{P} w_{t+1}^k$

end procedure

We assume there are some similarities amongst nodes in dataset. Usually node cares more about models generated on more similar datasets. For example, when a vehicle goes from Torino to Milano, a well trained model on cars' trajectories of Rome city may not help to make a correct prediction; this node cares more about models generated by those nodes travel from Torino to Milano. Therefore, it would be rational to give more weight to a model that has more similar observations.

- w_{t+1}: see procedure1.
- w_{t+1}^k: see procedure 1.
- l_k: evaluated loss of updated model of node k on the local validation set of server
- p_k : 10^{-l_k}
- P: summation of all p_k

The preliminary assumption in the decentralized collaborative learning algorithm is to consider a portion of the local dataset as the validation set. Server at round t evaluates loss value of the updated model of node k, w_{t+1}^k, on its local validation set. We call it l_k . l_k may get less loss when server validation set and the training set of node k have more similar data points. To compute loss value we are using categorical cross-entropy formula described in section 4.3. In order to give more weight to a model with less loss, we inverse the result of this formula(l_k) by computing 10^{-l_k} which is represented by p_k. Then to generate w_{t+1} , we use a weighted sum in which w_{t+1}^k is weighted by taking the proportion of p_k over summation of all p_ks .

4.2.3 Decentralized Federated Best(DFed Best)

DFedBest:
procedure mergeModels(.)
$l_k \leftarrow$ evaluate loss of w_{t+1}^k on validation set
$w_{t+1} \leftarrow w_{t+1}^k$ where l_k has the smallest value
end procedure

In both DFed Avg, and DFed Pow, a server at every round receives updated models,w_{t+1}^k, and create a meta-model,w_{t+1}, from two different weighted sums. Sometimes newly merged model (w_{t+1}) does not perform as well as individual models(w_{t+1}^k). Therefore, instead of merging local models, we take the best model at every round and utilise it for next communication round of training. To do so, server evaluate local updated models of K client on the validation set and choose one with the lowest loss value as the best model to contribute in the next training step.

4.3 performance metrics

At each iteration of training, a loss function is used to calculate the error between the target value and the predicted one. The loss is backpropagated so that the network can update its weights and reduce loss, and as a result, generate a confident model. In purely local learning, a typical choice of loss function for multi class time series prediction is the categorical cross-entropy loss function $\mathcal{L}$. Output of deep learning network is a vector which its length is equal to the total number of classes. It contains probability values representing the probability of being in each class. True class is shown by a one-hot encoded vector, size of this vector is equal to the total number of classes. All values in this vector are zero except one which is set to "1" to show in which class we are. Categorical cross-entropy will compare

the output of machine learning which is a probability vector with one-hot encoded vector representing the true class, and by using below formula can compute the amount of loss. Loss value increases as the predicted probability diverges from the actual label, whereas a perfect model would have a loss value of "0". The loss function is given by

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} (y_{n,c}) \log(\hat{y}_{n,c}) \tag{4.1}$$

where:

- N number of total samples
- C number of total classes
- n iterates over N samples
- c iterates over C classes
- $y_{n,c}$ is target label for sample n for class c. It is a binary variable, equal to 1 if class c is visited, and zero otherwise;
- $\hat{y}_{n,c}$ is the probability predicted by the model for the nth observation belongs to the class c

Another mostly used performance metric in classification algorithms is accuracy. It is defined as the percentage of correctly classified outputs.

$$accuracy = \frac{Number \quad of \quad correct \quad predictions}{Total \quad number \quad of \quad predictions \quad made} \tag{4.2}$$

The rest of this work is organized to consider experiments and depicting results in chapter five, and conclusion and future works in chapter six.

Chapter 5

Experiments and results

In this section, we present the results of the experiments on practical datasets. We must emphasise again that, given the large number of parameters in NNs, we decided to choose particular value for some parameters, as it is suggested in the literature, and since our main concern is performance evaluation of the proposed algorithms which personalized model through a collaborative learning when there is no server.

5.1 general considerations

5.1.1 Data collection

Data collection is the first most crucial step in every machine learning algorithm. Because they are learning from data, and it is very crucial to feed them with correct data to avoid getting unpleasant outputs. We have to be sure that we are using a proper date with a useful scale, format, and meaningful features. To estimate the trajectory of 10 seconds ahead, essential features are X and Y. We need to sample X and Y at the same time intervals for all nodes; X and Y are the lateral and longitudinal position of each node, respectively. To have such a dataset, we use synthetic traffic simulation of Luxembourg city centre. It contains car trajectories from 12AM to 10 AM. It is imperative that nodes visit k neighbours in every training rounds(k » 1), so we consider simulation time between 6:30 and 7:30 am, which is a dense traffic time. After we collect data, some actions must be done on it. We have to select how we can use this data and in which form it must be to work. To do data pre-processing, we must do some steps. They are mentioned in the following.

5.1.2 Visualization and data analysis

The purpose of data analysis and visualization is to gain insight into the dataset and better understand the problem. One way of visualizing data is plotting the graphs that summarize the data. In figure 5.1 trajectories of three vehicles are illustrated. The common routs that cars have crossed are highlighted. It shows, there are some similarities amongst nodes' datasets, and time series has no pattern (seasonality, cyclic, or trend). In general, dataset contains information of 1948 vehicles; each has it own local dataset. Data over clients is unbalanced and non-IID. On average cars spend fifteen minutes in the plane. Maximum time which a car is in the plane is 43 minutes. 1201 cars stay in the space for less than 10 minutes as they might start their journey before 6:30 AM or continue it after 7:30 AM.

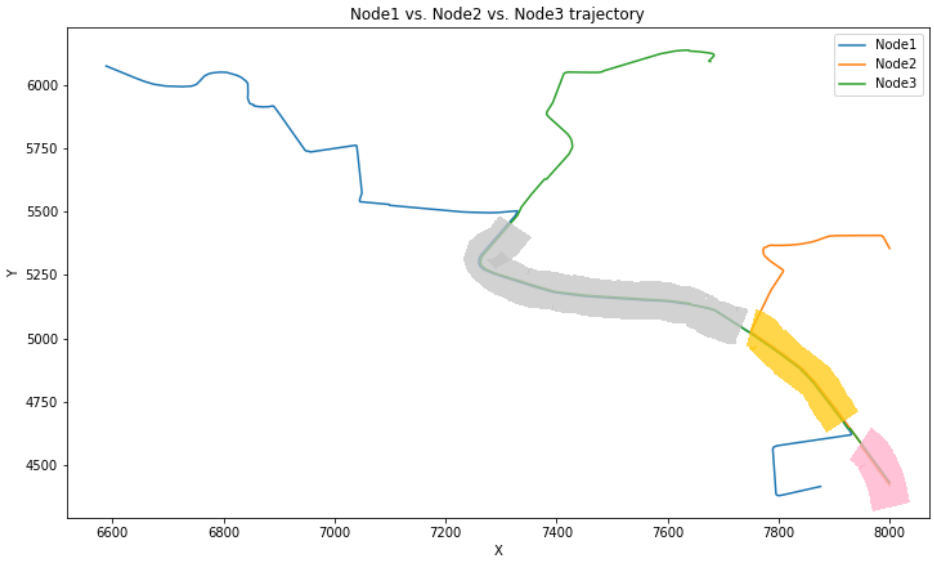

Figure 5.1: Car trajectory of 3 nodes.

We did data visualization not only to have a summary of data but also to explore data in order to identify if data cleaning operation is needed.

5.1.3 Re-scaling

Better performance will be achieved in some machine learning algorithms when data has been scaled or distributed consistently. Normalization and standardization are two techniques that we can utilize to rescale our time series data. In practice,

it is almost always an advantage to applying pre-processing rescaling to the input data before it is given to a network. In the same way, for the outputs of the network [57]. Having a large range of values may result in significant error values affecting weight values to alter drastically. It might make the learning process unstable. Hence, in deep learning neural networks scaling inputs and outputs is a critical step. As it is mentioned in subsection 5.1.2 our time series has no pattern, so we can not use transformation techniques for scaling. We used normalization for input values. X and Y are the lateral and longitudinal position of each node in the plane, respectively. For each of them, we have the maximum and minimum allowed values. We use the following formula for both X and Y to have values in range 0 and 1.

$$\mathcal{S} = \frac{(s - min)}{(max - min)} \tag{5.1}$$

where:

- S is scaled value
- s is the real value of variable X, or Y
- min is minimum value of X, or Y
- max is maximum value of X,or Y

5.1.4 partitioning

The input of the demonstrated network is the ordered set of examples $D = [(x_{t_1}, y_{t_1}), (x_{t_2}, y_{t_2}), ..., (x_{t_l}, y_{t_l})]$ of l coordinates in space. To each input we can associate as an output a *cell trajectory* $[c_{t_1}, c_{t_2}, ..., c_{t_l}]$, where c_{t_x} denotes the label of the cell containing the point (x_{t_x}, y_{t_x}). x_t , and y_t are provided in our dataset. We have calculated corresponding c_t by the use of below formulas.

$$K = math.ceil[\frac{(max(x) - min(x))}{segment \quad size}] \tag{5.2}$$

$$J = math.ceil[\frac{(max(y) - min(y))}{segment \quad size}] \tag{5.3}$$

$$XX = math.ceil[\frac{(x_t - min(x))}{segment \quad size}] \tag{5.4}$$

$$YY = math.ceil[\frac{(y_t - min(y))}{segment \quad size}] \tag{5.5}$$

$$c_t = XX * K + YY - J \tag{5.6}$$

To do data transformation of our output, which is categorical value, we use One-Hot Encoding. In this case, integer representation of variable will be removed and instead a new binary variable is added for each unique integer. In new binary vector, all bits are zero except a single bit which is one to identify the unique variable.

5.1.5 Model structure

Finally, by having all this information, we need to train our data to generate a model. On this dataset we train an encoder-decoder LSTM model. The model takes a series of car positions as input. It is associate to $D = [(x_{t_1}, y_{t_1}), (x_{t_2}, y_{t_2}), ..., (x_{t_l}, y_{t_l})]$ of l coordinates in space. D contains twenty four data points. Input passed through two LSTM layers, each has fifty neurons. Output of LSTM layer is associated to *cell trajectory* $[c_{t_1}, c_{t_2}, ..., c_{t_h}]$ with fixed size of h. We are using two samples of cell trajectory as output. Time interval between sampling for both input and output series is five seconds; therefore, by having last two minutes trajectory of car we predict cell position in the next ten seconds. At the end the output of the second LSTM layer is sent to a softmax output layer with one neuron per cell. By partitioning in subsection 5.1.4 fifty Square cells with a length of 200 meters have been generated. Size of cell corresponds to the coverage area of the roadside unit to which a specific MEC server is associated. Corresponding values for l,h, and number of neurons have been selected empirically. To do so, we modify one or more variables (cause), and controls and measures the change in other variables (effect).

A common problem we all face when working on deep learning projects is how optimizing the hyper-parameters. In deep leaning there are many parameters that needs to be set. The goal of this thesis is not reaching maximum accuracy or precision on the specific dataset. We are evaluating the efficiency of represented methods to anticipate next 10 seconds trajectory in vehicular networks. We choose particular value for some parameters, as it is suggested by some authors.The local batch size is 32, According to [37] [38] a good default value for batch size is 32. We can often use default value 0.001 for learning rate [58]. Adam optimizer is straightforward to implement, and computationally efficient. It requires little memory, and well suited for problems with large parameters. It is also appropriate for non-stationary objectives [59]. We consider number of epoch equal to one to avoid training time increase. Totally, the full model has 33350 parameters.

5.2 Server-Based Federated Learning

However, The main topic in this thesis is the performance evaluation of proposed serverless FL algorithms; we first evaluate the performance of centralized FL using

FedAvg algorithm explained in chapter two to evaluate loss and accuracy over rounds, and to see how this algorithm is behaving on non-IId and unbalanced dataset. For this particular case, we first eliminate cars' dataset which are for less than 3 minutes in the space, because in 3 minutes they collect few samples and this amount of data can not be useful in training phase. Then we use the first 2/3 of the dataset contains 3323240 time series over 632 cars for the training part, and the last 1/3 of dataset holds 911416 samples collected by 208 cars for the test set.

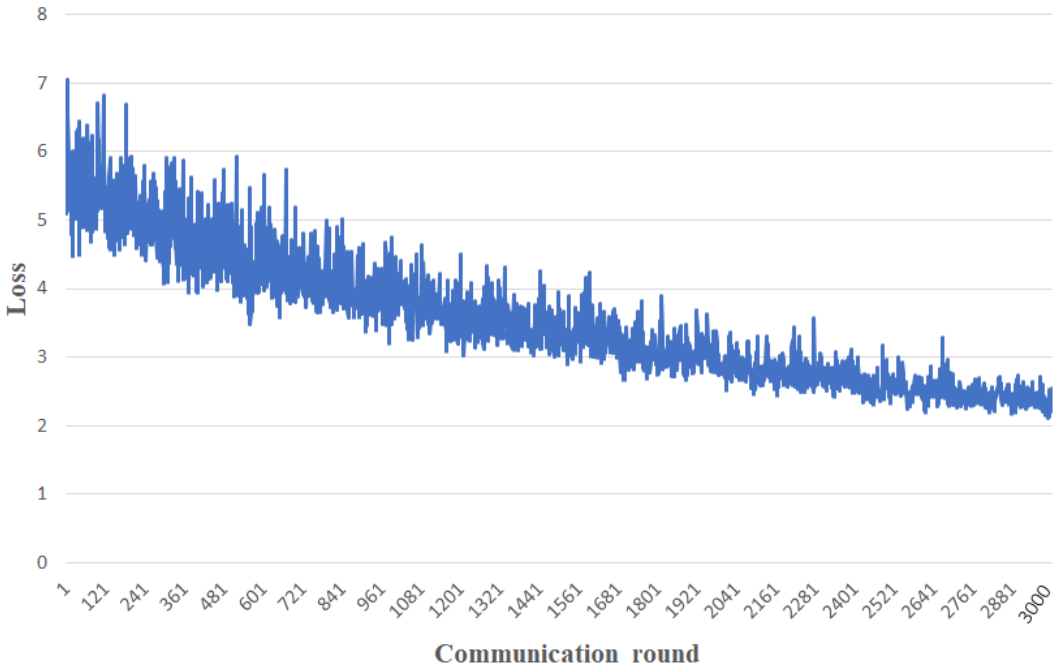

Figure 5.2: Loss evaluation on test set using Centralized Federated Averaging.

Regarding the configuration parameters of the federating averaging algorithm, we select ten clients randomly at every round to involve in the training phase, the configuration is as following:

- the local batch size is 32,
- learning rate is 0.001
- number of epoch equal to one
- optimaizer : Adam

The reason of selecting these values for hyper-perameters explained in subsection 5.1.5.

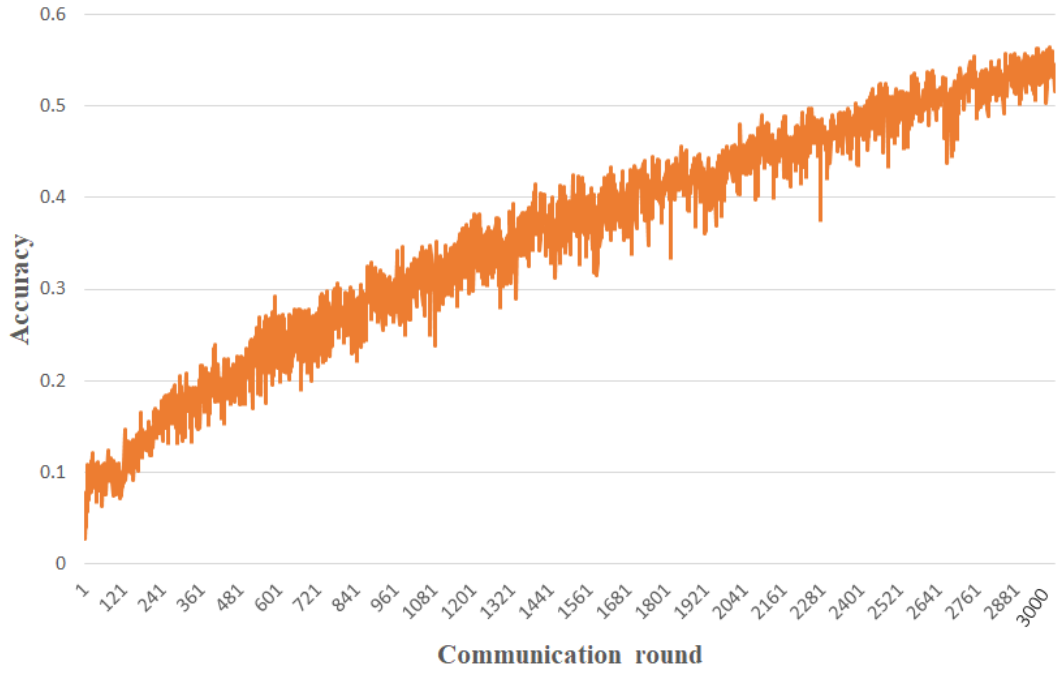

Figure 5.3: Accuracy evaluation on test set using centralized Federated Averaging.

Usually in ML hundreds or thousands learning round is needed to have a high performance model, so we select communication round equal to 700,1400, and 3000 to evaluate the performance on the test set. By increasing number of communication rounds, model learns more but very slowly. Figure 5.2 depicts when we train a model for 3000 communication rounds. It shows that loss value over the test set is slightly decreasing, although it has fluctuation. In addition, accuracy increases steadily to reach almost 52 per cent at the last round shown in figure 5.3. These fluctuations are because we select random nodes with different datsets at every communication round.

We would expect that additional rounds of communication produces further performance improvements; However, it is shown in figure 5.4 and 5.5 increasing communication rounds does not lead to improvement of clients' performance individually.

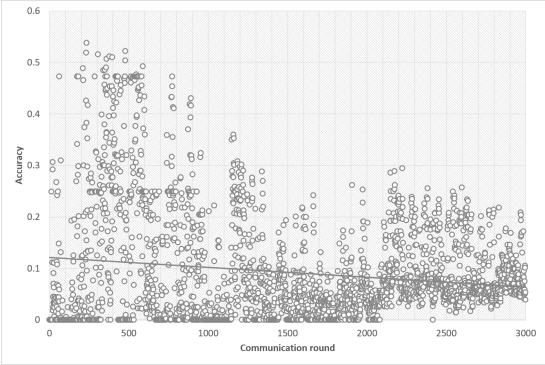

Figure 5.4: Test set performance of one car using centralized Federated Averaging

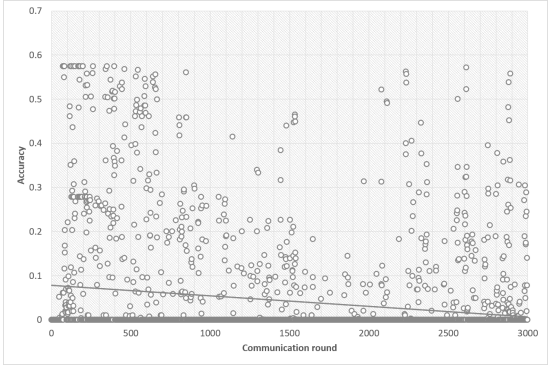

Figure 5.5: Test set performance of one car using centralized Federated Averaging

	Avg acc	Avg loss	Min acc	Min loss	Max acc	Max loss
FedAvg	0.52	2.54	0	0.34	0.88	9.64

Table 5.1: Test set performance of Fed Avg over 3000th communication rounds.

In summary, by using centralized federated averaging for 3000 communication round we achieve the average accuracy equal to 52 per cent and average loss equal to 2.54. In table 5.1 under minimum accuracy we observe value zero, and 9.64 for maximum loss. These values imply that generated model does not make any correct prediction on some cars' dataset. According to downward trend depicted in figure 5.4 and 5.5 we can not assure by increasing number of communication round the finalized model performs well on all individual cars' dataset. We propose serverless federated learning algorithms discussed in chapter 4, not only to tackle this problem but also to solve scalability issue and existence of single point of failure in centralized federated learning. In section 5.3 results of these algorithm is discussed.

5.3 Server-less Federated Learning algorithms

In this section, we evaluate the performance of the proposed algorithms (DFed Avg, DFed Pow, and DFed Best explained in chapter four) through an online learning method. We update models each time step when new data becomes available. We check the availability of data every five seconds. If we increase this time step, we

have less communication rounds; so it is harder to estimate performance of our model(because nodes stay for short time in the plane). On the other hand, by decreasing this value, It needs more time to run. In addition only a few samples are added to the scenario that may not help to improve model.

We evaluate the performance of algorithms on 31 vehicles which are for more than 20 minutes in the space (on average cars spend 20 minutes in the space). Every car has its own route related to different part of city. Only a few cars are evaluated as experiments are expensive in terms of time.

In the following experiments, when cars enter to the space collect samples for five minutes and consider it as validation set. If we consider this time less than 5 minutes, nodes have not collected enough samples to evaluate models. On the other hand, If we assign more than 5 minutes to collect data for validation set, number of communication round will decrease and we can not evaluate the performance of our model perfectly. Needed time to collect data for validation set is shown by v.

Then training phase starts. Cars communicate via a wireless technology. We assumed transmission radius equal to 250 meters. If we consider transmission radius larger we have more nodes involved in training phase, so it is computationally more expensive. By considering a less value lower than 250 meters, number of involved nodes in training phase is less and model needs more time (communication round) to learn.

Car sends its model to all neighbours in its transmission radius to train. If instead of sending models to neighbouring nodes, we select nodes randomly, DFed Avg and centralized Fed Avg will be equivalent.

All neighbours train the shared model and send it back to the source.

Depending on time spent in the space by each car, maximum number of learning round is different amongst cars. We limit number of learning round to 115. It is equal to almost 10 minutes.

Totally, we consider observed samples over first 5 minutes as validation set, next 10 minutes car communicates with others to build its model. Needed time to do learning phase is shown by lp .

In the following, we appraise the performance of three merging methods used by DFed algorithms (DFed Avg, DFed Pow, and DFed Best) while the test set is defined in two ways:

- short rolling test set window: Test set at time interval i constituted of those samples associated to the time interval $(i-l, i+h)$ where l is time duration of input sequence, and h is the forecast horizon. l represents a total of 120 seconds past samples, and h represents next ten seconds from the current moment.

- fixed test set window: it contains observations collected over m minutes after finishing learning phase (v+lp, v+lp+m). In all below experiments we considered m equal to 5 minutes. Like what we do for validation set, to have enough samples in test set to evaluate the performance of model.

Not only by changing hyper-parameters of LSTM model the results may change but also changing in size of validation set might lead to different results. We also consider the effect of validation set when we have :

- Fixed size rolling validation set : Validation set at time interval i contains observations collected over recent five minutes of nodes' trajectory.

- Expanding size validation set : In the server side observations over first 5 minutes are considered as validation set. By passing time new observations are added to the validation set so its size increases.

5.3.1 Experiment 1: short size rolling test set

In this experiment, We will evaluate the performance of DFed Avg, DFed Pow, DFed Best by considering a short rolling window for the test set and a fixed size rolling validation set window.

	Avg acc	Avg loss	Min acc	Min loss	Max acc	Max loss
DFed Pow	0.72	3.1	0.1	0.08	0.93	6.59
DFed Avg	0.41	3.5	0	0.73	0.79	15.3
DFed Best	0.65	4.7	0.07	0.76	0.9	14.9

Table 5.2: Performance evaluation of algorithms on short rolling test set window over 115 communication rounds.

Table 5.2 is demonstrating accuracy and loss corresponding to each algorithm we used in this thesis. We explain some elements of table:

- Average accuracy and loss: First we compute average accuracy and loss of each car's dataset over 115 communication rounds. Then we take average over all obtained averages.

- Minimum accuracy and loss: First we compute average accuracy and loss of each car's dataset over 115 communication rounds. Then we take minimum over all obtained averages.
- Maximum accuracy and loss: First we compute average accuracy and loss of each car's dataset over 115 communication rounds. Then we take maximum over all obtained averages.

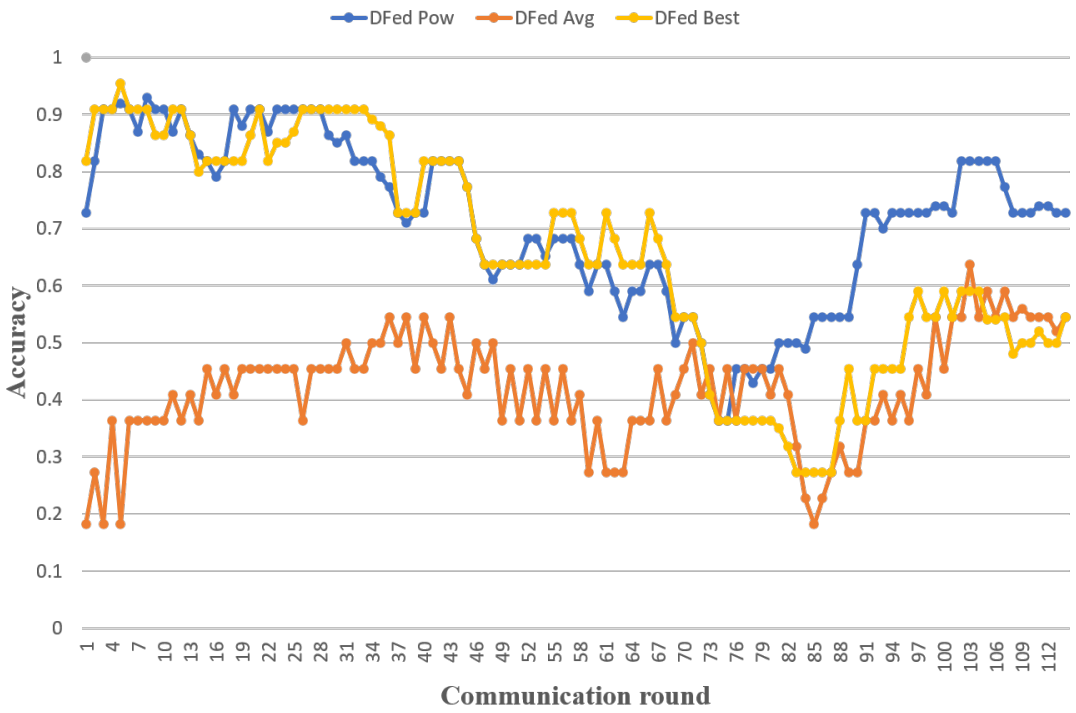

Figure 5.6: Accuracy evaluation on short rolling test set window.

It is shown in table 5.2 DFed Pow gives a pretty high average accuracy , the average loss is rather high as well; this means the generated model is not so confident. Instead DFed Avg does not obtain a good result in both accuracy and loss because it just dedicates weights to a model based on number of samples involved in training phase. The last line of table illustrates results came out from Dfed Best which is not the best in spite of its name and this tells us that choosing the best model according to criteria used in this algorithm is not necessarily correct. By looking at column minimum accuracy, we observe by using DFed Avg some cars are not able to do any prediction correctly like what we experienced in centralized Fed Avg after 3000 communication rounds (explained in section 5.2).

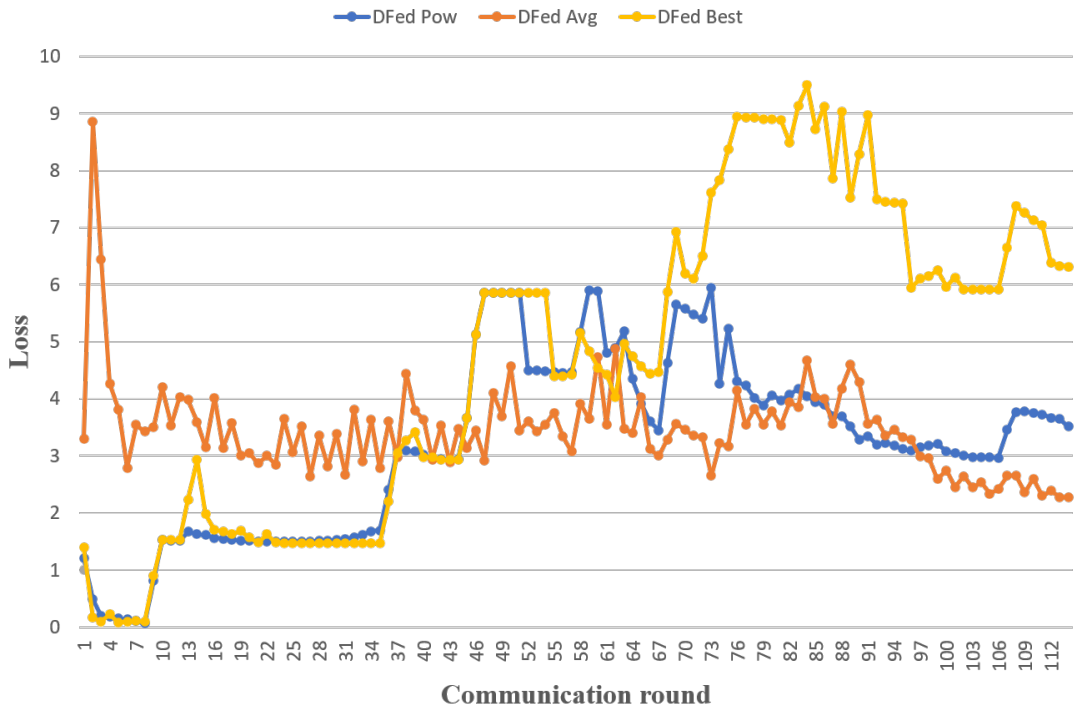

Figure 5.7: Loss evaluation on short rolling test set window.

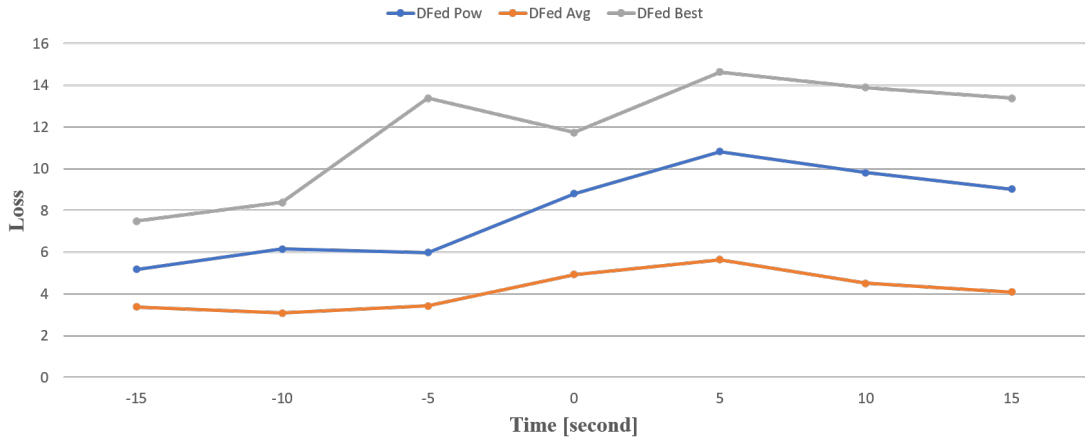

Figure 5.8: Loss evaluation when going to new cell.

In general, DFed Pow has privilege with respect to others. It gives higher average accuracy and lower average loss.

Figure 5.6 provides a detailed information on accuracy at every communication round. Except the DFed Avg the other two algorithms are following similar general trend. This contrasts when we look closely from communication round 34 to 76. In this interval we witness a continuous down fall in DFed Pow and DFed Best where as DFed Avg exposes a pretty constant behavior in this interval. By considering datasets we observed in this time interval many nodes are going to new cell, and these two algorithms do not perform well to predict new cell. From communication interval round 88 on all algorithms experience an analogous behavior. Among all three algorithms, DFed Pow produces higher accuracy in majority of rounds due to better decision making mechanism.

Loss evaluation in detail is exposed to see in figure 5.7. From the beginning up to round 34 all 3 algorithms are following an almost constant trend with some fluctuations in DFed Avg.Then DFed Pow and DFed Best starts to rise and increase their loss value from 1.5 to 3 sharply just after passing 4 rounds and from that round on DFed Pow stops elevating and keeps its stability loss between values 3 and 4 after round 79 to the end, while DFed Best rockets to loss 9 after experiencing a short time fluctuation between loss 4 and 6. DFed Avg remains waving almost without significant change after first round between loss 2 and loss 5. Like the case of centralized FedAvg it learns very slowly.

As we observed, DFed Pow and DFed Best do not perform well to predict new cell, they have delay to recognize in which cell they are. So we asses the performance of algorithms when cars go from one cell to another one. In figure 5.8 "0" represents time that car arrives from another cell to the new cell. Loss value of all three algorithms starts to grow before reaching to new cell and reaches its peak 5 seconds after arriving to new cell and then starts to decrease steadily. We observe DFed Avg has lower loss, in addition it can predict new cell with 20 per cent accuracy, while DFedPow with 10 per cent and DFed Best only 7 per cent.

To sum up, the generated model by DFed Pow predicts with lower loss and more accuracy, while to predict moving to new cell it has delay, and can not perform well. We have to tune parameters and try to decrease this delay. In addition, in this experiment as test set is changing at every round (because of changes in patterns of mobility over time) we can not estimate how well in future this generated models will work. To evaluate the performance of models after several communication rounds, we assign a portion of time to learning phase and then we evaluate the performance of generated model on the test set. In experiment 2 we will see the

results.

5.3.2 Experiment 2: Fixed test set window

Unlike previous experiment, we are not evaluating the performance of algorithms on short rolling test set window; instead we use a fixed test set as it is explained in section 5.3.

	Avg acc	Avg loss	Min acc	Min loss	Max acc	Max loss
DFed Pow	0.59	5.54	0	0	1.0	16.11
DFed Avg	0.66	2.05	0	0.0	1.0	15.1
DFed Best	0.39	8.6	0	0	1.0	16.11

Table 5.3: Performance evaluation of algorithms on static fixed test set window over 115th communication rounds.

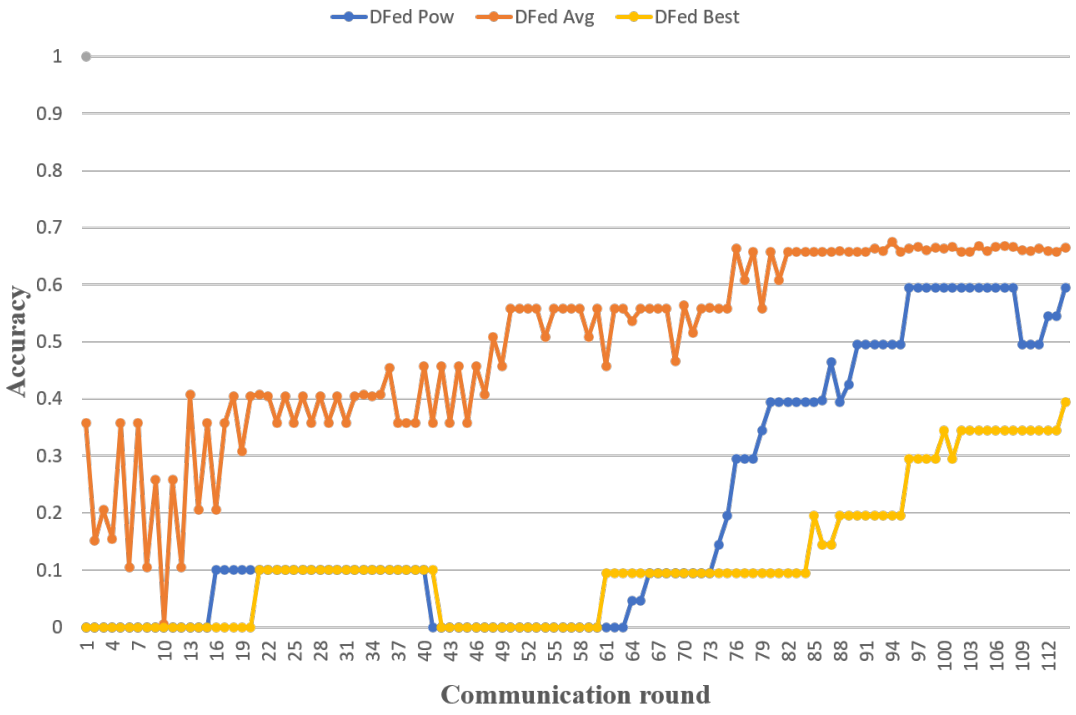

Figure 5.9: Accuracy evaluation on fixed test set window

It is shown in table 5.3, unlike previous experiment DFed Avg provides the best

average loss and accuracy amongst 3 algorithms. Like what we had in centralized Federated Averaging, and experiment 1 for DFed Avg, all three algorithms can not make any correct prediction on some datasets. By considering the performance of algorithms on each car dataset, we sometimes see, in one specific car model generated by DFed Avg diverges and is not able to learn anything to make a prediction on its test set. While models generated by DFed Pow and DFed Best are showing different behavior, and they converge on the same car's test set. For some other cars model generated by DFed Avg converges while DFed Pow, and DFed Best models diverge. DFed Pow and DFed Best are showing more or less the same behaviours

However, we evaluated performance of DFed Avg and centralized Federated Averaging on different test sets. we can roughly say by using DFed Avg we need less communication rounds to generate a high performance model, depending on the way of selecting involved nodes in the training phase. As usually node cares more about models generated on more similar datasets to improve their models. Similarities in datasets of nodes which are in each others' transmission radius coverage is more probable.

Evaluation of loss and accuracy over communication rounds are depicted in figure 5.10 and 5.9. In all rounds accuracy of DFed Avg is higher than others, and DFed Pow in almost all round reaches higher accuracy than DFed Best. Three algorithms have upward trend, although generated models by DFed Best and DFed Avg can not almost make any correct prediction in first 60 rounds and they can not improve their models. It means only 60 communication rounds is enough for them to reach to the same accuracy and loss. Then they have to predict next 5 minutes after 60 rounds (60 rounds is equal to 5 minutes). We also see from round 80 on DFed Avg sticks to accuracy around 65 percent.

By looking at figure 5.10 we observe 3 algorithms have downward trend. DFed Avg not only at first communication round is experiencing by far lower loss than others but also keep this distance until the end of communication round.

In general, as loss value of DFed Avg is decreasing steadily over communication rounds, it seems by increasing number of communication rounds we can experience lower loss and higher accuracy. While the performance of DFed Pow and DFed Best improves as we are getting close in time to the point in time where test set data is taken, it seems they need less communication round to converge the model.

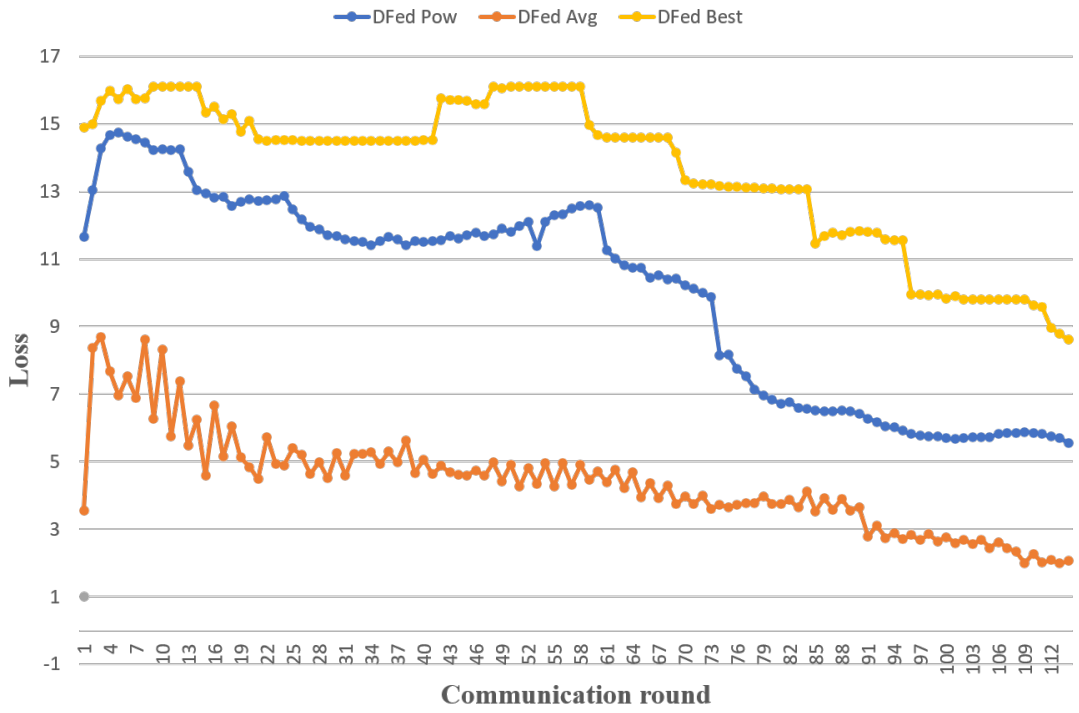

Figure 5.10: Loss evaluation on fixed test set window

5.3.3 Experiment 3: how validation set impacts on performance

In this experiment we consider the effect of size of validation set on DFed Pow algorithm while we have :

- Expanding size validation set: at every time step, new observed samples are added to validation set to increase the size of validation set (for more details see section 5.3).
- Fixed size validation set : it is a rolling validation set like what we considered in experiment 1 and 2.

This experiment is done only for 70 communication rounds equal to 6 minutes; since in the previous experiment(2) it seems DFed Pow needs less communication round to generate a high performance model. Like experiment 2 we have fixed test set contains samples collected over 5 minutes after finishing learning phase.

In figure 5.11 loss reaches at lower value by using expanding validation set, with respect to using fixed size rolling validation set over all communication rounds

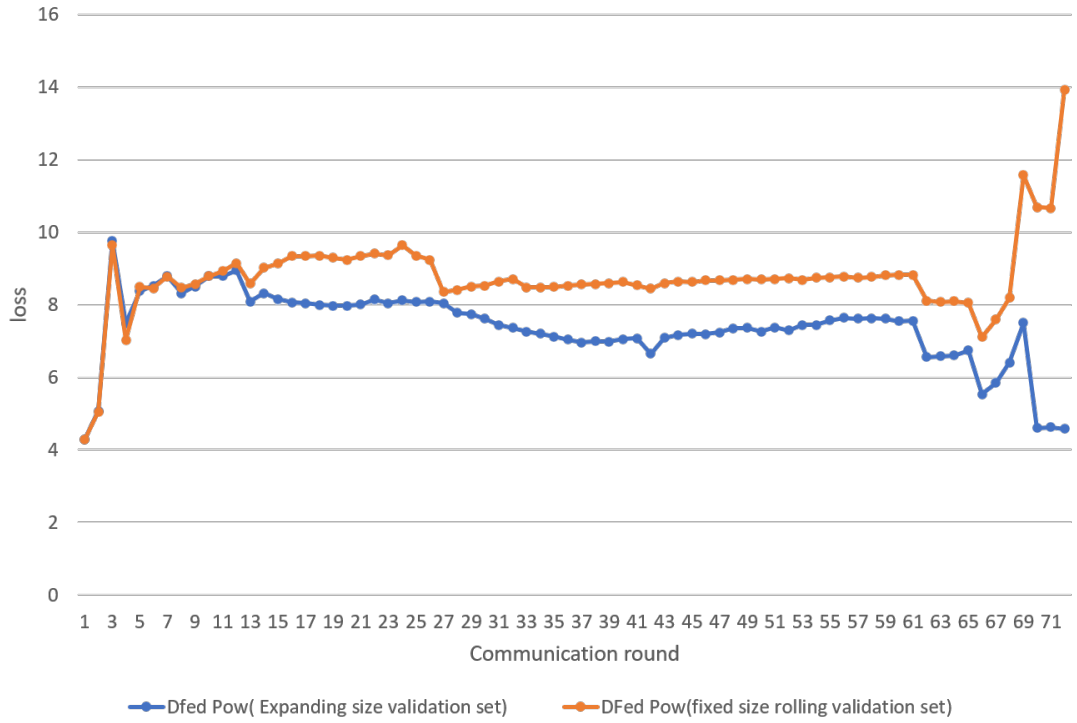

Figure 5.11: Evaluate the effect of size of validation set on the fixed test set

except first rounds that both fixed size and expanding validation set have more or less the same size. Both curves are following approximately the same pattern, but only at the ending rounds loss value generated by DFed Pow using fixed size rolling validation set increases sharply, in contrary another one decreases drastically.

DFed Pow is giving more weight to more similar models. When we have an expanding validation set, it means we have a validation set with more samples. When we have more samples, it is more probable to evaluate the loss of models much preciser. Hence, giving more weight correctly to a model which is more similar.
Moreover, as the size of validation set grows by passing time; it is more probable to have various similarities between clients' models and validation set of server. For example, if validation set contains 100 time series samples, similar points between server validation set and clients' models vary from 0 to 100. While when we have a fixed size validation set contains observed samples over less time, for examples it contains 20 time series samples, similar points between server validation set and clients' models vary from 0 to 20. The probability of having the same number of

similar samples between clients' model and validation set of server is higher. So server will give the same weight to these models to generate a meta-model. In this case we are not able to distinguish amongst received model and select the more similar one to give more weight.

Notice that in this scenario we could not take size of validation set so large as the size of datasets is limited. On average vehicles are for 20 minutes in the space. We need to assign a portion of time to collect data points for validation set, another portion of time to learning phase and the last portion for test set to evaluate our models. In the next chapter, weaknesses and strengths of each algorithm are discussed and the limitation of the assessment that we have. What should be done to overcome them is discussed in next chapter as well.

Chapter 6

Conclusion and future works

6.1 Conclusion and future works

In this work, we proposed three serverless federated learning approaches using three different techniques to combine models to tackle the learning problems of centralized federated learning. These algorithms are asynchronous and peer-to-peer, which personalize model for each node through online collaborative learning. The proposed distributed federated learning approaches were validated on a car trajectory prediction scenario where an LSTM model was trained to predict the next ten seconds cell trajectory of vehicles by using models updated by neighbouring nodes through a collaborative method.

Our experiments show, DFed Pow trains models which are performing efficiently almost on all provided datasets when we have a short rolling test set window. Instead, DFed Avg needs more communication rounds to make a correct prediction, so it is instrumental when we can train model for a while and then check its performance, or we can increase the size of transmission radius to increase the number of involved clients in the training phase. Increasing number of involved clients in the training phase usually decreases the number of communication rounds needed to converge. On the other hand, it also outcomes to higher communication costs, so it is necessary to trade-off to find out which one is more important in our problem.

Models generated by all algorithms can not predict well when a car is going from one cell to another cell. We evaluated the performance of models only on transmission time (cars go from one cell to another cell). By using DFed Avg model we reached accuracy equal to 20 per cent, and model generated by DFed Pow

algorithm, only predicts 10 per cent times correctly. DFed Best model performs the worst amongst all. As nodes' datasets are unbalanced, and non-iid, it seems we need to assign more time to tune parameters and do more experiments to reach higher accuracy. In future work, we consider the effect of Machine Learning parameters on models generated by DFed Avg and DFed Pow, and try to improve accuracy on transmission time. We do not evaluate the performance of DFed Best in future work, as it does not have any advantages to others in all experiments.

We observed, the way of merging models to create federated model is so crucial. As obtained results of DFed Avg, and DFed Pow on the same car were totally different. For some cars, DFed Avg diverges, while DFed Pow converges, and vise-versa. In future, we apply a merging method that implies both merging characteristic of DFed Avg and DFed Pow to be able to perform well on more cases.

In this work, we did not consider the case when there are malicious users and adversaries. In future, we will study the impact of having an unreliable network on DFed Pow and DFed Avg algorithm.

Finally, as revealed in the considered case study, nodes do not use shared personalized models of others to improve their model, or to make a prediction; Hence, having a few shared meta-model in the network is an exciting direction for future work.

Bibliography

[1] H. Brendan et al McMahan. «Communication-Efficient Learning of Deep Networks from Decentralized Data». In: *arXiv: 1602.05629* (Feb. 2016) (cit. on pp. ii, 2, 5, 7, 9, 12).

[2] Google AI Blog. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`. Federated Learning: Collaborative Machine Learning without Centralized Training Data-Si TFT. 2017. (accessed: 04.03.2020) (cit. on pp. 1, 2).

[3] `https://ec.europa.eu/info/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules/eu-data-protection-rules_en`. EU data protection rules. 2018. (accessed: 01.01.2020) (cit. on p. 1).

[4] M. Manzurul, S. Morshed, and P. Goswami. «Cloud Computing: A survey on its limitations and Potential Solutions. In: IJCSI International Journal of Computer Science 10.4». In: (July 2013), pp. 159–163 (cit. on p. 1).

[5] P. Hofmann and D. Woods. «Cloud Computing: The Limits of Public Clouds for Business Applications. In: IEEE Internet Computing 14.6». In: (Nov. 2010), pp. 90–93 (cit. on p. 1).

[6] Bay Arinze and Murugan. Anandarajan. «Factors that Determine the Adoption of Cloud Computing: A Global Perspective». In: (Oct. 2010), pp. 55–68 (cit. on p. 1).

[7] Weisong et al Shi. «Edge Computing: Vision and Challenges». In: *IEEE Internet of Things Journal 3.5* (Oct. 2016), pp. 637–646 (cit. on p. 1).

[8] Chen et al Sun. «Revisiting Unreasonable Effectiveness of Data in Deep Learning Era». In: *arXiv:1707.02968 [cs]* (July 2017), pp. 637–646 (cit. on p. 1).

[9] Carrie et al MacGillivray. *IDC FutureScape: Worldwide Internet of Things 2017 Predictions. en.* https://www.idc.com/research/viewtoc.jsp?containerId=US40755816 .accessed Dec 13th 2019. Nov. 2016 (cit. on p. 1).

[10] H. Brendan McMahan Jakub Konecny and Daniel Ramage. «Federated Optimization: Distributed Optimization Beyond the Datacente». In: (2015) (cit. on p. 2).

[11] Keith Bonawitz. «Practical Secure Aggregation for Privacy Preserving Machine Learning». In: (2018) (cit. on p. 2).

[12] Bellet Tommasi Vanhaesebrouck Guerraoui Taziki and Zantedeschi. «Graph signals: learning and optimization perspectives- Workshop». In: (May 2019) (cit. on p. 4).

[13] Virginia et al Smith. «Federated Multi-Task Learning». In: *arXiv:1705.10467* (May 2017) (cit. on p. 5).

[14] K.Bandara C.Bergmeir and Slawek Smyl. «Forecasting Across Time Series Databases using Recurrent Neural Networks on Groups of Similar Series: A Clustering Approach». In: *arXiv: 1710.03222* (Oct. 2017) (cit. on p. 5).

[15] Kasun et al Bandara. «Sales Demand Forecast in E-commerce using a Long Short-Term Memory Neural Network Methodology». In: *arXiv:1901.04028* (Jan. 2019) (cit. on p. 5).

[16] A.Cherif H.Cardot and R.Boné. «SOM time series clustering and prediction with recurrent neural networks». In: *Neurocomputing 74.11* (May 2011), pp. 1936–1944 (cit. on p. 5).

[17] et al M. Blot. «Gossip training for deep learning," 30th Conference on Neural Information Processing Systems (NIPS)». In: *Barcelona, Spain.Available: https://arxiv.org/abs/1611.09726* (2016) (cit. on p. 5).

[18] J. Jiang C. Hu and Z. Wang. «Decentralized Federated Learning: A Segmented Gossip Approach». In: *1st International Workshop on Federated Machine Learning for User Privacy and Data. Available: https://arxiv.org/abs/1908.07782* (Aug. 2019) (cit. on p. 5).

[19] Stefano Savazzi et al. «Federated Learning with Cooperating Devices: A Consensus Approach for Massive IoT Networks». In: *arXiv:1912.13163v1* (Dec. 2019) (cit. on p. 5).

[20] K. Mekhnacha C. Tay and C. Laugier. «Probabilistic Vehicle Motion Modeling and Risk Estimation». In: (2012), pp. 1479–1516 (cit. on p. 6).

[21] T. Streubel and K. H. Hoffmann. «Prediction of driver intended path at intersections, IEEE Intelligent Vehicles Symposium, Proceedings». In: (2014), pp. 134–139 (cit. on p. 6).

[22] S. Lefevre A. Carvalho Y. Gao and F. Borrelli. «Stochastic predictive control of autonomous vehicles in uncertain environments, in 12th International Symposium on Advanced Vehicle Control». In: (2014) (cit. on p. 6).

[23] S. Lefevre P. Kumar M. Perrollaz and C. Laugier. «Learning-based approach for online lane change intention prediction». In: (June 2013), pp. 797–802 (cit. on p. 6).

[24] H. M. Mandalia and M. D. D. Salvucci. «Using Support Vector Machines for Lane-Change Detection, Proceedings of the Human Factors and Ergonomics Society Annual Meeting». In: 49 (Sept. 2005), pp. 1965–1969 (cit. on p. 6).

[25] T. A. Wheeler D. J. Phillips and M. J. Kochenderfer. «Generalizable Intention Prediction of Human Drivers at Intersections». In: (2017), pp. 1665–1670 (cit. on p. 6).

[26] S. Yoon and D. Kum. «The multilayer perceptron approach to lateral motion prediction of surrounding vehicles for autonomous vehicles». In: (June 2016), pp. 1307–1312 (cit. on p. 6).

[27] Florent Altché and Arnaud de La Fortelle. «An LSTM Network for Highway Trajectory Prediction». In: (2017) (cit. on pp. 6, 22).

[28] P. Ondruska and I. Posner. «Deep tracking: Seeing beyond seeing using recurrent neural networks». In: *Proc. AAAI Conf. on Artificial Intelligence* (2016), pp. 3361–3367 (cit. on p. 6).

[29] B.Kim et al. «Deep tracking: Seeing beyond seeing using recurrent neural networksProbabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network». In: *arXiv:1704.07049* (2017), pp. 3361–3367 (cit. on p. 6).

[30] U.S. Federal Highway Administration. «US Highway 101 dataset». In: (2015) (cit. on p. 6).

[31] `https://www.datacenterknowledge.com/industry-perspectives/data-center-dilemma-our-data-destroying-environment`. (accessed: 8.01.2020) (cit. on p. 7).

[32] Franziska Bell and Slawek Smyl. *Forecasting at Uber*. accessed Jun 20th 2020. Dec. 2018 (cit. on p. 7).

[33] *Value of Real-Time Data Is Blowing in the Wind*. accessed Jun 20th 2020. Sept. 2015 (cit. on p. 7).

[34] Jason Brownlee. `https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/`. 2019. (accessed: 12.01.2020) (cit. on p. 8).

[35] Zhanhong Jiang et al. «Collaborative Deep Learning in Fixed Topology Networks». In: (2017) (cit. on p. 12).

[36] J.Daily A.Vishnu C.Siegel T.Warfel and V.Amatya. «GossipGraD: Scalable Deep Learning using Gossip Communication based Asynchronous Gradient Descent». In: (2018) (cit. on p. 12).

[37] Yoshua Bengio. «Practical Recommendations for Gradient-Based Training of Deep Architectures». In: *arXiv:1206.5533* 2 (Sept. 2012) (cit. on pp. 13, 37).

[38] Dominic Masters and Carlo Luschi. «Revisiting small batch training for deep neural networks». In: *arXiv:1804.07612* (Apr. 2018) (cit. on pp. 13, 37).

[39] R. Shokri and V. Shmatikov. «Privacy Preserving Deep Learning». In: (2015) (cit. on p. 14).

[40] Keith et al Bonawitz. «"Practical Secure Aggregation for Federated Learning on User-Held Data». In: *arXiv: 1611.04482* (Nov. 2016) (cit. on p. 16).

[41] Jakub et al Konený. «"Federated Learning: Strategies for Improving Communication Efficiency». In: *arXiv: 1610.05492* (Oct. 2016) (cit. on p. 16).

[42] John H. Cochrane. «Time Series for Macroeconomics and Finance». In: (1997) (cit. on p. 16).

[43] Ratnadip Adhikari and R. K. Agrawa. *An Introductory Study on Time Series Modeling and Forecasting* (cit. on p. 17).

[44] Vinay Kumar. «Time series modeling and forecasting using stochastic models». In: (Dec. 2016) (cit. on p. 19).

[45] J. Wang, J. Tang, Z. Xu, Y. Wang, X. Zhang G. Xue, and D. Yang. «Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach-Si TFT». In: (2017), pp. 1–9 (cit. on p. 19).

[46] Hoang Duy Trinh, Lorenza Giupponi, and Paolo Dini. «Mobile Traffic Prediction from Raw Data Using LSTM Networks-Si TFT». In: (2018) (cit. on p. 19).

[47] J.A.K. Suykens and J. Vandewalle. «Least squares support vector machines classifiers». In: *Neural Processing Letters, no.3* 9 (June 1999), pp. 293–300 (cit. on p. 19).

[48] L.J. Cao and Francis E.H. Tay. «Support Vector Machine with Adaptive Parameters in Financial Time Series Forecasting». In: *IEEE Transaction on Neural Networks* 14,no. 6 (Nov. 2003), pp. 1506–1518 (cit. on p. 19).

[49] R.Begg J.Kamruzzaman and R.Sarker. «Artificial Neural Networks in Finance and Manufacturing». In: *Idea Group Publishing, USA* () (cit. on pp. 19, 20).

[50] R.O. Otieno J.M. Kihoro and C. Wafula. «Seasonal Time Series Forecasting: A Comparative Study of ARIMA and ANN Models». In: *African Journal of Science and Technology (AJST) Science and Engineering Series, No.2* 5 (), pp. 41–49 (cit. on pp. 19, 20).

[51] B.E. Patuwo G. Zhang and M.Y. Hu. «Forecasting with artificial neural networks: The state of the art». In: *International Journal of Forecasting* (1998), pp. 35–62 (cit. on p. 20).

[52] MURAT H. SAZLI. «A brief review of feed-forward neural networks». In: (2006), pp. 11–17 (cit. on p. 21).

[53] Samuel Dupond. «A thorough review on the current advance of neural network structures». In: *Annual Reviews in Control. 14* (2019), pp. 200–230 (cit. on p. 22).

[54] Christopher Olah. *http://colah.github.io/posts/2015-08-Understanding-LSTMs/*. accessed Jan 13th 2020. Aug. 2015 (cit. on p. 24).

[55] Jason Brownlee. *https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/*. accessed Jan 13th 2020. Nov. 2018 (cit. on p. 24).

[56] E Zivot and J. Wang. «Modeling Financial Time Series with S$_{PLUS}$». In: *NY: Springer Science+Business Media* 2 (2006) (cit. on p. 28).

[57] Christopher M. Bishop. «Neural Networks for Pattern Recognition». In: (1995), p. 296 (cit. on p. 36).

[58] Aurelien Geron. «Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems». In: (Mar. 2017), p. 357 (cit. on p. 37).

[59] Diederik P. Kingma and Jimmy Lei Ba. «ADAM: a method for stochastic optimization». In: *arXiv:1412.6980v9* (Jan. 2017) (cit. on p. 37).