Miguel José Rodrigues Fernandes

FLEET LEARNING APPLIED TO AIRCRAFT MAINTENANCE

1 2 9 0 UNIVERSIDADE Đ COIMBRA

1 2 9 0

UNIVERSIDADE Đ
COIMBRA

Miguel José Rodrigues Fernandes

FLEET LEARNING APPLIED TO AIRCRAFT MAINTENANCE

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems, advised by Professor Joel Arrais, co-advised by Bernardete Ribeiro and presented to Faculty of Sciences and Technology / Department of Informatics Engineering.

June 2020

Acknowledgements

First and foremost, I would like to thank my supervisors, Prof. Joel Arrais and Prof. Bernardete Ribeiro, as well as the remaining professors of University of Coimbra involved in the project, for their guidance and dedication throughout the whole project and for providing constant feedback during this year.

Moreover, I would like to thank all my friends and family for their amazing support and for always being there for me.

Finally, I thank my girlfriend for her love and dedication in all the times we have been together.

This page is intentionally left blank.

Abstract

Aircraft systems are complex machines susceptible to equipment deterioration. When systems can no longer perform tasks they were design to do, they reach a failure state. If this happens unexpectedly, a failure in an aircraft system can cause potential danger to passengers or lead to flight delays. As a consequence, airlines are also affected financially.

Hence, Aircraft Maintenance (AM) is of major importance in the aircraft industry, ensuring actions and procedures are taken so that functionality and safety are guaranteed. In this field of work, Prognostics and Health Management (PHM) can be of extreme relevance, helping in the detection of the precursors of a system's failure and predicting its health.

Aircraft are equipped with sensors that are constantly generating data during their many flying routes. These data can be extremely valuable as it contains information which can be used in a PHM system in order to predict future failures. However, since airplanes generate gigabytes of data per flight, sending this information to a central server becomes an almost infeasible task due to the size of the available bandwidth. In addition, there are security constraints in transferring data through unsecured channels.

In order to attenuate these problems, this work proposes a PHM approach using a Fleet Learning methodology. Fleet Learning uses shared knowledge of all the parties in a fleet, which can help creating more powerful models. In a recent approach defined as Federated Learning (FL), a single model is shared between a server and the clients. When applying FL in a Fleet Learning setting, it can reduce the amount of data transferred and attenuate the privacy concerns since each model is computed locally by their respective client and only the model is shared, instead of the data itself.

Federated Learning is still a recent technology and, as such, much research is yet to be done. This work presents three main contributions, which, to the best of our knowledge are new to this research field: 1) application of FL to PHM, 2) proposal and implementation of two Federated Learning algorithms, namely, Federated Congruent Directional Learning (FedCong) and Federated Momentum (FedMom) and comparison with state of the art, 3) development of a tool (Fleet Learning Demonstrator Tool (FLDT)) which demonstrates the behaviour of commercial aircraft systems and predicts their health.

Keywords

Federated Learning, Machine Learning, Artificial Intelligence, Aircraft Maintenance, Prognostics and Health Management, Neural Networks

This research is supported by European Union's Horizon 2020 programme under the ReMAP project, grant No 769288.

ReMAP

This page is intentionally left blank.

Resumo

Os aviões são máquinas complexas suscetíveis à deterioração dos seus componentes. Quando estes deixam de ser capazes de executar as suas tarefas, os aviões atingem um estado de falha. Situações de falhas inesperadas num sistema de um avião podem ter consequências que afetam a segurança dos passageiros e originar atrasos nos voos. Como consequência, companhias aéreas são bastante afetadas em termos financeiros.

Portanto, Manutenção de Aviões (AM) é de extrema importância na indústria aeronáutica, garantindo que ações e procedimentos são tomados para garantir a funcionalidade e segurança. Nesta área, Prognostics and Health Management (PHM) poderá ajudar na detecção de precursores de falhas e previsão da vida de um sistema.

Os aviões estão equipados com sensores que estão constantemente a gerar dados durante as suas rotas aéreas. Estes dados são extremamente valiosos pois contêm informação que pode ser usada em sistemas de PHM para prever futuras falhas. No entanto, como os aviões podem gerar vários gigabytes de dados por voo, enviar esses dados para um servidor central torna-se uma tarefa quase inviável devido ao tamanho da largura de banda disponível. Existem também restrições de segurança no envio de dados por canais não seguros.

Para atenuar estes problemas, este trabalho propõe uma abordagem PHM usando uma metodologia Fleet Learning. Fleet Learning usa conhecimento compartilhado de todas as partes de uma frota, o que pode ajudar a criar modelos mais eficazes. Numa abordagem recente definida como Federated Learning (FL), um único modelo é compartilhado entre um servidor e os seus clientes. Ao aplicar os conceitos de FL em Fleet Learning, este pode reduzir a quantidade de dados transferidos e atenuar as preocupações com a privacidade, pois cada modelo é gerado localmente pelo respectivo cliente e apenas o modelo é compartilhado, em vez dos próprios dados.

Federated Learning ainda é uma tecnologia recente e, como tal, a sua investigação ainda está nos seus primórdios. Este trabalho apresenta três contribuições principais que, até onde sabemos, são novas neste campo de investigação: 1) aplicação de FL a PHM, 2) proposta e desenvolvimento de dois novos algoritmos de Federated Learning e comparação com o estado da arte: Federated Congruent Directional Learning (FedCong) e Federated Momentum (FedMom), 3) desenvolvimento de uma ferramenta intitulada Fleet Learning Demonstrator Tool (FLDT) que demonstra o comportamento dos sistemas de aviões comerciais e prevê a sua vida.

Palavras-Chave

Federated Learning, Machine Learning, Artificial Intelligence, Aircraft Maintenance, Prognostics and Health Management, Neural Networks

This research is supported by European Union's Horizon 2020 programme under the ReMAP project, grant No 769288.

ReMAP

This page is intentionally left blank.

Contents

Acronyms

AM Aircraft Maintenance. v, xii, 1, 5, 73

ANN Artificial Neural Networks. xii, 8–11, 24

CC Cloud Computing. xii, 12

CL Centralized Learning. xii

CNN Convolution Neural Networks. xii

CR Communication Round. xii, xvii–xix, 13, 18, 19, 24, 33–35, 37–56, 61, 72

DEI Department of Informatics Engineering. xii

EC Edge Computing. xii, 1, 5, 12

FE Flight Emulation. xii, 60, 61

FedAvg Federated Averaging. xii, xiv, xv, xvii, xviii, 2, 15, 18, 19, 23, 26, 29, 35, 37–39, 41–44, 46–49, 51–53, 56, 57

FedCong Federated Congruent Directional Learning. v, vii, xii, xiv, xv, xvii, xviii, 3, 4, 21, 23, 24, 29, 35, 39, 41, 44, 46, 49, 51, 54, 56–58, 71, 72

FedMom Federated Momentum. v, vii, xii, xiv, xv, xvii–xix, 3, 4, 21, 23, 26, 27, 29, 35, 40, 41, 45, 46, 50, 51, 55–58, 71, 72

FedProx Federated Proximal. xii, xiv, xv, xvii, xviii, 15, 19, 29, 35, 38, 41, 43, 46, 48, 51, 53, 56–58

FFNN Feed Forward Neural Networks. xii, xiv, 8–10, 23, 35

FL Federated Learning. v, vii, xii, xiv, 2–5, 12, 13, 15, 18, 22, 23, 33–35, 69, 72, 73

FLDT Fleet Learning Demonstrator Tool. v, vii, x, xii, 4, 59–61, 68, 69, 71–73

GD Gradient Descent. xii, 11, 24, 26, 35

HI Health Indicator. xii, xiv, 1, 5, 6, 15, 16, 21, 23, 30, 33, 35, 36, 60, 65, 69, 72, 73

IID Independent and Identically Distributed. xii, xiv, 18, 19

IT Information Technology. xii

KNN K-Nearest-Neighbours. ix, xii, 8, 16

MAE Mean Absolute Error. xii, xvii, xviii, 35, 37, 42, 47, 52

MGD Momentum Gradient Descent. xii, 11, 12

ML	Machine Learning. xii, 5–8
MSE	Mean Square Error. xii, xvii, 32, 34, 35, 61
MSGD	Mini-Batch Stochastic Gradient Descent. xii, 11
PHM	Prognostics and Health Management. v, vii, ix, xii, xiv, 1–6, 15, 21, 22, 71, 73
RBF	Radial Basis Function. xii, 16
RNN	Recurrent Neural Networks. xii, 9, 10, 15, 23
RUL	Remaining Useful Life. xii, xiv, 1, 5–7, 15, 17, 21, 22, 30, 33, 71, 73
SE	Server Emulation. xii, 60, 61
SGD	Stochastic Gradient Descent. xii, 11, 12, 18, 19

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

Aircraft are complex machines composed of interrelated equipment which has to be functional and reliable. Equipment deterioration is inevitable as accumulative usage of the systems leads to gradual degradation. When systems can no longer perform tasks they were design to do, they reach a failure state [25]. Systems are susceptible to different types of failures, which can have a wide range of impact levels depending on the system and its importance.

In order to maintain aircraft systems in a healthy condition, Aircraft Maintenance (AM) can restore an item to a viable condition [50]. Examples include servicing, repairing, overhauling, inspection and determination of condition. Maintenance checks have to be performed on each aircraft after reaching a certain number of flight hours [21]. These checks are considered as preventive maintenance techniques. Maintenance can also be reactive, that is, items are only repaired when they fail. It is clear that the latter procedure can bring high financial costs and can also cause flight delays.

Furthermore, techniques such as Prognostics and Health Management (PHM) can be used to detect the precursors of a failure. These can help predicting the health of the system, HI, which can be used to calculate how much time remains before a likely failure, which is referred as the Remaining Useful Life (RUL) [43]. As such, PHM methods can bring massive improvements to aircraft maintenance, reducing the number of failures.

Airplanes are constantly generating new data from sensors' readings, for example, a Boeing 787 can generate around 500 Gigabytes of data every flight [2]. Hence, new methods can be created and constantly updated, even during fight hours. These methods can be extremely valuable as the new data generated by the aircraft can aid PHM predicting the system's health. However, because of the large volume of the data, sending it to the server becomes an almost infeasible task due to the size of the available bandwidth. In addition, another issue that should be mitigated is the security constraints in sending data through unsecured channels.

In order to solve these problems, several companies such as Tesla are using Fleet Learning methodologies to improve the learning capability of their systems [46]. Fleet Learning uses shared knowledge of all the parties in a fleet, which can help creating more powerful models. For example, when one vehicle gathers data, all the other vehicles can use these data to improve their systems by communicating with a central server. In addition, the server's tasks can be reduced by performing part of the computation in each specific party. This technique, named Edge Computing (EC), can increase the system's scalability and operational efficiency.

In a recent approach defined as Federated Learning (FL) [17], a single model is shared between a server and the clients. When applying FL in a Fleet Learning setting, it can reduce the amount of data transferred between the stakeholders. Moreover, it attenuates the privacy concerns [17] since each model is computed locally by their respective client and only the model or its updates are shared, instead of the data itself.

Given that fleets are composed of many aircraft, Federated Learning methodologies can be used to predict an aircraft system's health without compromising the performance of the learning methods. This work presents some improvements in the field of Federated Learning, more specifically, in the area of PHM. The next sections outline the motivation, research questions and contributions of this work.

1.1 Motivation

Improvements in PHM methods can bring massive benefits to companies and passengers allowing monitoring, estimation and prediction of the system's health. These methods can make products more reliable and eliminate redundant inspections, reducing operational costs associated with equipment testing. In addition, by improving reliability prediction, new advances in warning of failures can be created which can avoid catastrophic failures [7].

Due to the high availability of data and its distribution across multiple clients, Federated Learning can be used to help accelerating the training of models that estimate an aircraft system's health. A method that can collectively obtain the benefits of shared models trained from these data can help prevent avoidable future failures. Furthermore, without the need send all the aircraft's data, FL is able to minimize the time and privacy constraints associated with data transferring [17].

Another key advantage of FL is that this method does not require that every client is permanently online [12, 16, 28]. Instead, it takes into account the fact that the communication between the server and the client is sometimes limited as some clients may be offline or with a reduced communication speed.

Since Federated Learning is still a recent technology, it has not had many application in real world scenarios. In particular, to the best of our knowledge, Fleet Learning has not yet been used to develop PHM methods for aircraft maintenance.

Furthermore, there are still improvements that can be made to the current Federated Learning algorithms. More specifically, `Federated Averaging (FedAvg)`, which is considered to be the state of the art of FL, converges slowly to a local minimum. As such, new methods should be developed which increase the model's convergence speed.

When using Federated Learning, as the data of each party is not identically distributed, each party might try to minimize different problems [60]. In order to reduce the error of the global problem, an algorithm can be created which maximizes the impact of models that are trying to minimize the same error and mitigates the impact of those who are trying to minimize different errors.

In addition, a FL algorithm which takes into consideration the global model's history by looking at the direction of the previous update should be created. This technique, named momentum, should be able to accelerate the converge speed [35, 56] in FL algorithms.

Finally, a tool that can collectively predict real time information of an aircraft system's health using Federated Learning methodologies should be developed.

1.2 Research Questions

The main objective of this work is to implement Federated Learning methodologies in aircraft maintenance. More specifically, the following research questions are investigated:

Research Question 1: Is it possible to maintain the same performance as a centralized trained model, using Federated Learning methodologies without the need to send all the data to a server?

This work makes the hypotheses that Federated Learning can reach the same performance as a centralized learning model by tuning certain hyperparameters such as the learning rate and the number of epochs.

Research Question 2: Can the current state of the art algorithms for Federated Learning be improved with regard to their convergence speed and error minimization?

It is investigated whether it is possible to increase the convergence speed of the Federated Learning models by applying new techniques such as the momentum [35, 56].

1.3 Contributions

Based on the research questions defined in the previous section, the following were achieved:

Contribution 1: Proposal and implementation of a PHM methodology using Federated Learning concepts and assessment of its performance.

Contribution 2: Proposal and implementation of two Federated Learning algorithms which outperform the state of the art: `Federated Directional Congruent Learning (FedCong)` and `Federated Momentum (FedMom)`. While the first is based on the models' updates' directions, the second algorithm is based on the momentum of the global model.

Contribution 3: Development of a tool which emulates the behavior of commercial aircraft's engine degradation using real aircraft routes in a Fleet Learning setting. This tool is complemented with a graphical interface which presents a visual representation of the tool.

To the best of our knowledge, this is the first work that uses Federated Learning methods in PHM for aircraft maintenance. In addition, it is also one of the first works with the objective of accelerating the convergence speed of the FL models. Finally, as far as we are aware, this work presents the first tool to be used for aircraft maintenance using Fleet Learning methodologies.

1.4 Documents Structure

The remainder of this document is structured as follows:

Chapter 2 explains specific concepts associated with the work.

Chapter 3 presents the state of the art regarding the work previously developed by different authors.

Chapter 4 is divided into two main parts: 1) performance assessment of a PHM Federated Learning model compared to a centralized learning model, 2) explanation of two proposed FL algorithms, namely `FedCong` and `FedMom`, and comparison with state of the art.

Chapter 5 presents the experiments and the results of the methodology;

Chapter 6 describes a tool named Fleet Learning Demonstrator Tool (FLDT) which emulates the aircraft's degradation in a Fleet Learning setting and predicts the system's health;

Chapter 7 presents the conclusion and future work.

Chapter 2

Background

This chapter introduces important concepts to this work. Firstly, the field of Prognostics and Health Management (PHM) will be presented, introducing the concepts of Health Indicator (HI) and Remaining Useful Life (RUL). Afterwards, the field of Machine Learning (ML) will be presented, explaining the different types of learning and types of problems in the area, as well as some algorithms. Finally, the areas of Edge Computing (EC) and Federated Learning (FL) will also be presented.

2.1 Prognostics and Health Management (PHM)

In this section, the area of Prognostics and Health Management (PHM) will be presented. PHM can be defined as the detection of the precursors of a failure and the prediction of how much time remains before failure [7]. In addition, PHM methods can enable the prediction of the future performance of a system by analysing the degree of deviation from its normal working conditions.

This field has gained popularity in recent years since it brings advantages in the global market by improving reliability and safety. In addition, since financial costs associated with system failures can be reduced when PHM methods are implemented, this area has shown signs of success and growing interest [1].

Given their many advantages, PHM methodologies can be applied in the field of Aircraft Maintenance (AM), bringing massive improvements to aircraft maintenance and allowing the generation of sophisticated models with high performance in the detection of failures [1].

The next subsections will describe different types of approaches in PHM and relevant metrics that can be used to measure a system's health.

2.1.1 Types of Approaches

In Prognostics and Health Management (PHM) there are two main approaches which will be presented: physics-based and data-driven approaches.

Physics-based approaches

In physics-based approaches, the systems use mathematical and physics driven models to show the system's performance and make predictions about the system's future state [54]. On one hand, these types of approaches have the benefit that they are not as dependent on historical data as data-driven approaches. On the other hand, these have the disadvantage of requiring sufficient system's understanding [3].

Data-driven approaches

Data driven approaches perform system's health predictions from data collected from sensors readings created by the system under consideration. The data collected is a run-to-failure raw time series data which represents the degradation of the system [26, 27]. One advantage of this type of approach is that it does not required a thorough understanding of the system. However, datasets with a specific generalization of the system's degradation are hard to obtain.

2.1.2 Metrics

There are different metrics in PHM which can be used to measure the health of a system: Health Indicator (HI) and Remaining Useful Life (RUL). In the context of aircraft maintenance, sensors' information can be used in order to extract both values.

Health Indicator (HI)

The Health Indicator (HI) is defined as a measure of a system's health conditions [24]. The value of the HI can be represented as a percentage or an absolute value [52]. By evaluating a system's current health conditions, health indicators can then be used to predict the Remaining Useful Life (RUL) [51].

Remaining Useful Life (RUL)

The Remaining Useful Life (RUL) is defined as the remaining life span between the system's current moment and its end of life moment [14]. In other words, it is the time remaining before a mechanical system requires repairs or replacements.

Figure 2.1 shows a generic diagram of a system's RUL, where the blue plot represents the system's degradation over time. In the plot, the x-axes represents the time, the y-axes represents the system's health condition, T_f is the system's failure timestamp and t is the current timestamp. As such, the RUL can be defined as:

$$RUL = T_f - t \quad where \quad T_f > t \tag{2.1}$$

2.2 Machine Learning

Machine Learning (ML) can be defined as an application where systems learn automatically through experience without specific instructions [4]. ML is considered to be a field of

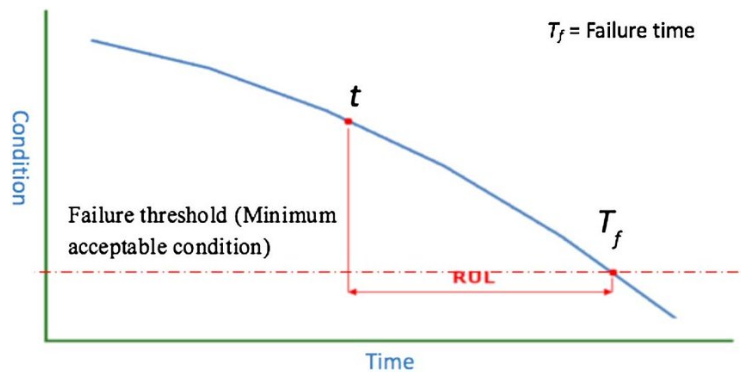


Figure 2.1: Generic diagram of the RUL [53].

Artificial Intelligence, which tries to mimic human intelligence in machines. The field of Machine Learning has experienced a significant growth in recent years due to the ability of ML techniques to handle high-dimensional problems [48, 58].

In this section, the area of Machine Learning will be explained in the scope of this work. Different types of ML concepts will be described, such as the types of learning and types of problems in this field. Moreover, some ML algorithms will be described, which are relevant to the understanding of this work, including: K-Nearest Neighbours, Feed Forward Neural Networks, Recurrent Neural Networks, Autoencoders and Neural Networks optimizers.

2.2.1 Types of Learning

This subsection will describe the types of learning in the field of ML. There are multiple types of learning paradigms and their usage depends on the type of data available and the problem which is being solved. The most common types of learning in this field are:

- **Supervised Learning** - In this type of learning, all of the data's labels are known. During training, it is required that the true target of a data instance is available [5].
- **Unsupervised Learning** - Contrary to supervised, in unsupervised learning the desired targets are unknown. Quite often, in this paradigm, the objective is to find distinct groups in a dataset based on similarity [5].
- **Semi-Supervised Learning** - It is a combination of supervised and unsupervised learning, where there is a small subset of labeled data and a large amount of unlabeled data. A special case of semi-supervised learning is Active Learning [22, 57].

2.2.2 Types of Problems

There are different types problems in Machine Learning, the most popular being Classification and Regression, which are associated with Supervised Learning, and Clustering, which is associated with Unsupervised Learning. These problems can be described as follows:

- **Classification Problem** - A classification has the objective of predicting a class, given a certain input. Quite often, for a given example, each output class will give a probability of that class belonging to it [23].

- **Regression Problem** - The main objective of a regression algorithm is to find a mapping function which transforms the input variables into a numerical output variable [49].

- **Clustering Problem** - A clustering problem has the objective of separating the data points in different areas, where data points which belong to the same area are more similar to each other than to points in other areas [5].

2.2.3 K-Nearest-Neighbours (KNN)

In this subsection, the K-Nearest-Neighbours (KNN) [10, 63] algorithm will be described. KNN is a simple non-parametric supervised ML algorithm which is used for classification or regression problems. This algorithm has the objective of finding the distances between the input and all the examples in the dataset, selecting the K closest elements to the input.

In a classification problem, in order to choose the class an instance belongs to, a voting method is used. In this setting, the input is assigned to the most common class based on its neighbours. In a regression problem, the most common methodology is to calculate the average of the neighbours' values.

2.2.4 Artificial Neural Networks

Artificial Neural Networks (ANN) [65] are algorithms based on biological neural networks, whose objective is to find patterns in data. This section describes different types of ANN algorithms relevant to this work, as well as different optimizers which have the objective of minimizing the ANN's error.

Feed Forward Neural Networks

Feed Forward Neural Networks (FFNN) are one of the simplest types of ANN [17, 42, 64], which contain at least three distinct layers: input layer, hidden layer and output layer. Neurons between the layers are usually fully connected and do not have internal loops. Figure 2.2 shows an example of a FFNN with an input layer, 2 hidden layers and an output layer.

In a FFNN, the output of each neuron is a weighted sum of the outputs of all the previous neurons plus a bias value. It is then followed by an activation function that can either be linear or non-linear. The formula of the output of a neuron is as follows:

$$\hat{y} = \varphi(\sum_{j=1}^{m} w_j \cdot x_j + b) = \varphi(w^T \cdot x + b) \tag{2.2}$$

where m is the number of inputs of a neuron, w are the weights that connect the neurons, x is the input data, b is the bias and φ is the activation function.

Assuming that this neuron belongs to the output layer, $\hat{y}$ is the predicted target. The loss function $l(\hat{y}, y)$ can be computed and it is usually denoted as the difference between the ground truth y and $\hat{y}$. The objective of the FFNN is to optimize w and b by minimizing

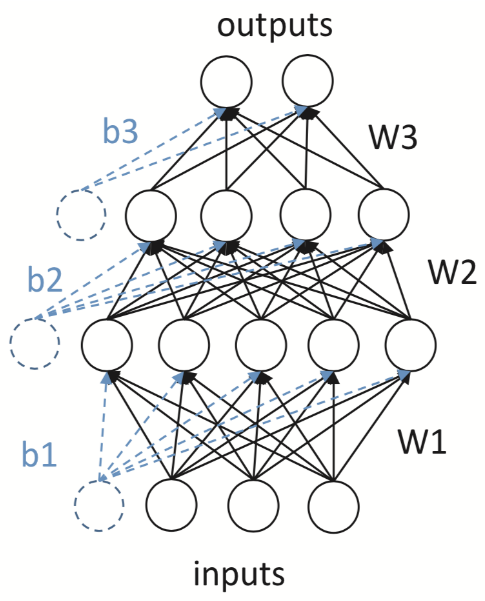


Figure 2.2: FFNN which contains an input, hidden and output layers. [64]

$l(\hat{y}, y)$:

$$\min \frac{1}{N} \sum_{i=1}^{N} l(\hat{y}_i, y_i) \tag{2.3}$$

where N is the size of the training data.

The most common method to train a FFNN is back-propagation, which allows the calculation of the loss function gradient with respect to each weight. In order to update w and minimize l, various optimizers can be used, which will later be described.

Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) [15, 20, 37], introduce a notion of time to the ANN. RNN build connections between direct data sequences since they have as its input not only the information from the previous layer, but also their output of the previous timestamp.

Figure 2.3 shows a generic representation of a RNN. As it can be observed, the output of a neuron is influenced not just by the w_k applied to the inputs, but also by a hidden state representing the history of previous inputs.

Mathematically, the output of recurrent neuron can be expressed as follows:

$$\hat{y} = \varphi(w_x^T \cdot x_t + w_{\hat{y}}^T \cdot \hat{y}_{t-1} + b) \tag{2.4}$$

where w_x and w_y are the weights, b is the bias value and φ the activation function. w_x connects the current input x to $\hat{y}$ and w_y connects the previous output $\hat{y}_{t-1}$ to $\hat{y}$. In a

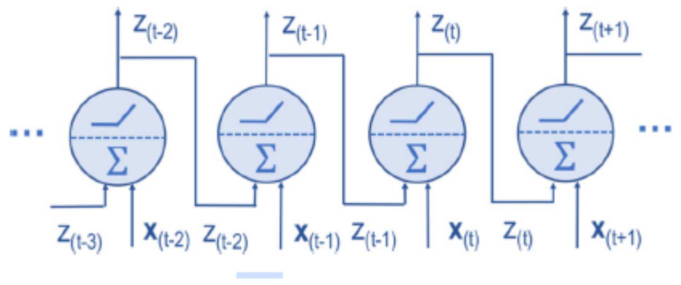


Figure 2.3: Generic representation of a RNN. [32]

RNN, quite often the activation function chosen is the hyperbolic tangent [20].

Similarly to FFNN, RNN's objective is to minimize the loss function, l, with a main difference: at each timestamp it is necessary to calculate each l_t and then aggregate them to form the total loss, L, given by:

$$L = \sum_{t=1}^{T} l_t(\hat{y}_t, y_t) \tag{2.5}$$

where T represents the total number of timestamps.

The method used to train RNN is named backpropagation through time [13]. A standard problem associated with RNN is the vanishing gradient problem. The problem is that the gradient can become extremely small, making the weight updates slow. As a consequence, it can be hard to capture long-term dependencies.

To solve this issue, Long-Short Term Memory (LSTM) [47] and Gated Recurrent Units (GRU) [9] have been proposed. These variants introduce gates that avoid long-term dependencies, mitigating the vanishing gradient problem [37].

Autoencoders

An Autoencoder is a type of ANN which learns data embeddings in an unsupervised manner [6, 34]. The objective of Autoencoders is to encode data and then reconstruct it back from the encoded form with the least possible distortion between the input and the output.

This type of ANN has three main constituents:

- **Encoder** - where the model learns how to reduce the input data dimension and create a new data embedding from the input data.
- **Bottleneck** - which contains the new data embedding of the input data.
- **Decoder** - where the model reconstructs the new data embedding into the original input data.

The architecture of the autoencoder may be different depending on the case, the simplest being a FFNN. However, RNNs [44] and Convolutional Neural Networks [29] can also be used. Figure 2.4 shows a generic representation of an autoencoder.

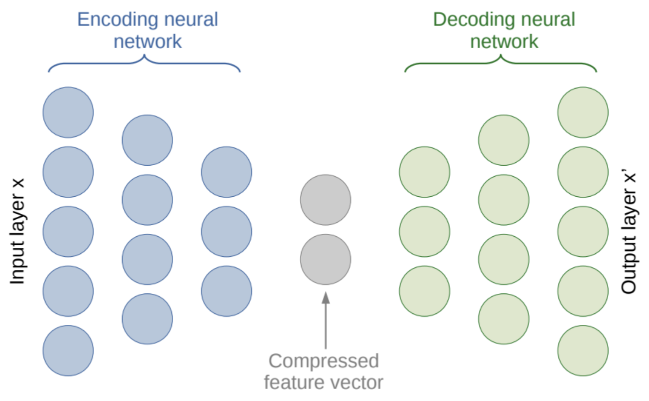


Figure 2.4: Generic representation of an autoencoder. [34]

The training of an autoencoder involves minimizing the network's reconstruction loss which is the measure of how close the output is to the input.

Autoencoders have many different applications but they are mainly used for dimensionality reduction and information retrieval.

Optimizers

Optimizers are algorithms used to update the w of ANN based on the loss function gradient with respect to w. These section describes some optimizers, namely Gradient Descent (GD), Stochastic Gradient Descent (SGD) and Momentum Gradient Descent (MGD).

GD Gradient Descent (GD) [36] is the most widely used method to find the local minimum of a function. In this algorithm, a specific parameter is moved in the direction of the steepest descend, which is the negative of the gradient. The weight update can be defined as follows:

$$w_{t+1} = w_t - \eta g_t \tag{2.6}$$

where η is the learning rate which controls how much the weights are adjusted and gt is the gradient of the error function with respect to w in the $t - th$ iteration. In GD, it is necessary to run through all the data samples to do a single gradient update. Hence, if the training data are big then the training process will take a long time.

SGD Stochastic Gradient Descent (SGD) [36] is a variation of the GD algorithm in which a random sample is chosen from the dataset. SGD only uses a single training sample in order to do a gradient update. As a consequence, this method usually also converges much faster than GD but the error is not as well minimized.

To find a balance between the two methods, Mini-Batch Stochastic Gradient Descent (MSGD) is an optimizer which selects a random mini-batch of training data for the gradient computation at each training iteration.

MGD Momentum Gradient Descent (MGD) [35, 56] is an optimization method which leads to faster convergence of the model by adding a fraction of the previous update to the current weight's update. This method is one of the most common optimization algorithms.

The following equation demonstrates how the momentum method works and how the weight's updates are conducted:

$$\begin{aligned} v_{t+1} &= \beta v_t + \eta g_t \\ w_{t+1} &= w_t - v_{t+1} \end{aligned} \tag{2.7}$$

where v_t is the momentum at timestamp t, v_{t+1} is the momentum at timestamp $t+1$, w_t is the weight at timestamp t, w_{t+1} is the weight at timestamp $t+1$, β is the momentum term, η is the learning rate and g_t is the error gradient with respect to w_t.

While in SGD the exact derivative of the loss function is estimated on small batches, MGD uses an exponential weighted average of the gradients of the loss function. This provides a closer estimate to the actual derivative value, comparing to SGD.

To conclude, using this method helps the gradient vector pointing to the right direction, damping oscillations and taking more straightforward paths to the local minimum.

2.3 Edge Computing

In this section, the Edge Computing (EC) paradigm will be described. Edge Computing is a distributed computing paradigm in which computing is as close as it can be to the data sources. EC can perform computing tasks such as: storage, processing and handling of requests from clients and the cloud.

On the other hand, in Cloud Computing (CC) data are gathered and processed in a centralized location (usually a data center) [40, 55], ensuring that those users can access that information from anywhere. Considering abundant data are being sent by the clients, it is clear that this is a slow process and is becoming an impracticable option [40].

As a solution, instead of placing all data computation in the cloud, Edge Computing brings computation and data storage closer to data sources, reducing the bandwidth. In this paradigm, the edge is defined as the technologies with any computing and network capabilities in the pathway between data sources and cloud [40, 55].

Fig 2.5 shows a simplified description of an EC infrastructure. It can be observed that the edge is responsible for tasks that would usually be done in a cloud environment. In addition, while EC operates on real-time data, CC operates above the edge.

2.4 Federated Learning

Federated Learning (FL) [17] is a rising decentralized learning technology which will be described in this section. While conventional Machine Learning methods require data to be centralized, FL allows each edge to learn a shared global model without needing to send its local data to a server. In addition, all of the model's training is done in the edge which is coordinated by a central server.

Firstly, an edged device receives a shared model from the server and trains it with data which is only accessible to it. Afterwards, it sends the updated gradients or weights back

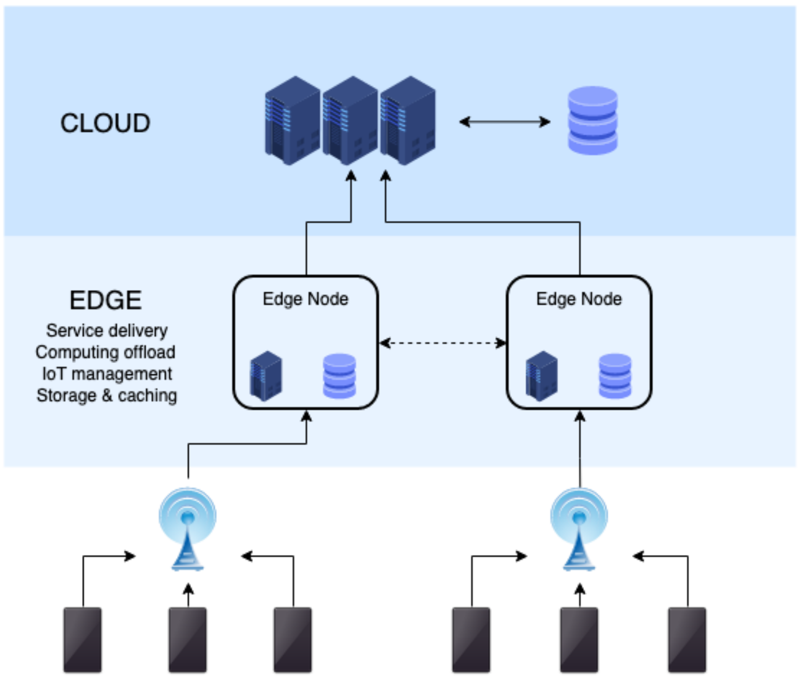


Figure 2.5: The Edge Computing infrastructure. [55]

to the server. In the server, the uploaded models are aggregated in order to form a new model. The way the models are aggregated will be explained in the next chapter.

Figure 2.6 shows the behavior of Federated Learning. Each client receives the parameters θ_t of the global model and trains it using their local data. Afterwards, each client sends the updated parameters θ_t^k to the server, where they are aggregated with the other client's updated models. Finally, the server generates the new global model θ_{t+1}, where t is the timestamp or Communication Round (CR).

Similar to the optimization problem of a multilayer perceptron, FL aims to minimize l but in a distributed paradigm, as follows:

$$\min \sum_{k=1}^{K} \frac{n_k}{n} F_k \quad \text{where} \quad F_k = \frac{1}{n_k} \sum_{i \in P_k} l(\hat{y}_i, y_i)) \tag{2.8}$$

where K is the number of clients, k is the index of the client, n_k is the size of the $k-th$ client's local data, n is the sum of all clients' local data and P_k is the set of data indexes whose length is n_k, i.e $n_k = |P_k|$.

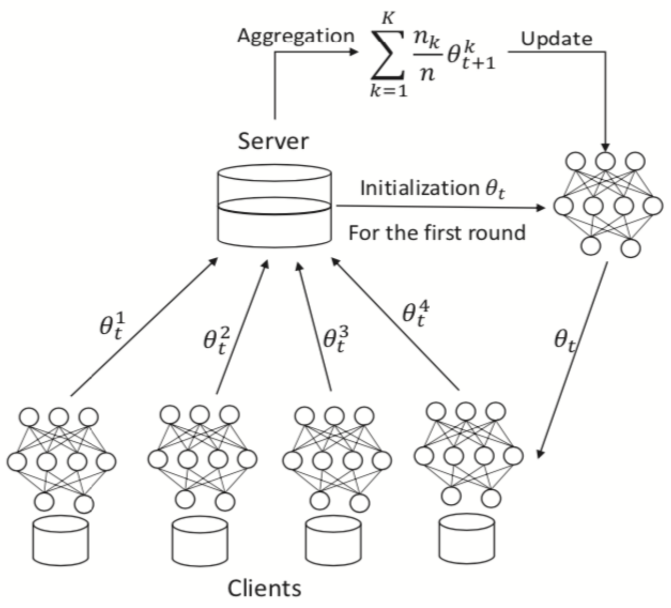


Figure 2.6: Diagram of Federated Learning, where n_k is the data size of client k, K is the total number of clients, t is the communication round and θ are the model's parameters. [64]

Chapter 3

State of the Art

In this chapter, the state of the art of an unsupervised method for Prognostics and Health Management (PHM) in aircraft maintenance will be presented. More specifically, this chapter introduces the work done by Shahid and Ghosh [44], whose objective is to predict an aircraft system's health in order to prevent future failures.

In addition, the state of the art of Federated Learning (FL) algorithms will be presented, namely: `Federated Averaging (FedAvg)` [17] and `Federated Proximal (FedProx)` [38].

3.1 TrajecNets

This section will present the work by Shahid and Ghosh [44], entitled *TrajecNets: Online Failure Evolution Analysis in 2D Space*. The objectives of this work are the following:

- Developing a set of stacked RNN Autoencoders in order to generate a 2D data embedding, which capture the system's evolution from healthy to failure.
- Predicting the RUL using an unsupervised methodology.

In the Autoencoder, both decoder and encoder are stacked RNNs. The bottleneck layer is a time distributed dense layer which outputs the data into a 2D space, extracting a new data dimensionality.

This work presents three different architecture types for the Autoencoders: TrajecNets-AE, TrajecNets-Siamese and TrajecNets-Super. TrajecNets-AE is an unsupervised stacked RNN Autoencoder which does not rely on class labels. TrajecNets-Siamese is a network build from the Autoencoder when there is a small number of samples. TrajecNets-Super is a supervised Autoencoder which relies on labeled failure classes. Figure 3.1 illustrates the three different architecture types.

The next subsection will describe the two steps the authors use to predict the Remaining Useful Life (RUL) of a system.

3.1.1 HI Computation

HI computation is defined as the health curve $h_t \in [0, 1]$ of a system. In order to calculate this curve, the authors propose a method to join what they refer as Local Health and

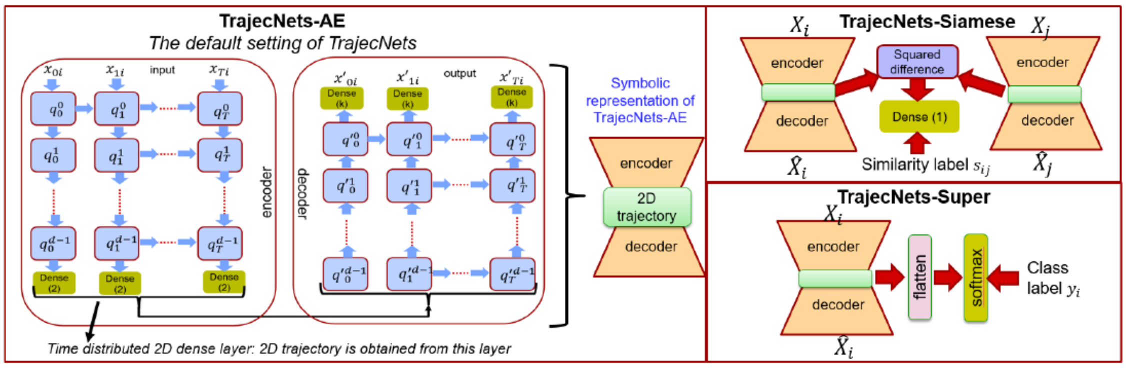


Figure 3.1: Different types of architectures for the TrajecNets' Autoencoders. [44]

Global Health, which will be explained in this section.

Considering the complete degradation trajectory of a time series, i.e the trajectory from the healthy state to a failure state, the Local Health is calculated as the distance from the evolving trajectory to the first trajectory point z_0. These trajectory points are the output of the bottleneck of the Autoencoder.

In order to minimize the noise of the trajectory, the authors calculate the mean of the 2D embedding with a defined window size w:

$$z_t = mean(z_{t-w}, z_{t-w+1}, ..., z_t) \tag{3.1}$$

This methodology defines a vector of euclidean distances: $d_t = [d_0, d_1, d_2, ..., d_T]$, such that each element is the distance of the $t-th$ trajectory point to the first one. Finally, d_t is transformed into a health vector h_t using a Radial Basis Function (RBF) kernel:

$$h_t = exp(\frac{-(d_t - d_{tmin})^2}{\sigma}) \tag{3.2}$$

where d_{tmin} is the minimum of d_t, σ is:

$$\sigma = -\frac{(d_{tmax} - d_{tmin})^2}{2}[\frac{1}{log_{10}\epsilon} + \frac{1}{log_{10}(\epsilon + \delta)}] \tag{3.3}$$

where d_{tmax} is the max of d_t. The σ ensures that the values of h_t range starts at 1 when h_0 and also that at a failure time $h_t \in [\epsilon, \epsilon + \delta]$.

The Global Health, contrary to the Local Health, is calculated as the distance from the mean of the KNN-Nearest-Neighbours healthy trajectory set to the evolving trajectory point:

$$d_t = ||z_t - mean(KNN(Z_{healthy}, z_t)|| \tag{3.4}$$

where $KNN(Z_{healthy}, z_t)$ are the KNN- Nearest Neighbours from the set of healthy trajectory points, $Z_{healthy}$. The health vector for the Global Health is then computed the same way as the Local Health.

The Local Health, L_t, and Global Health, G_t are fused to calculate the HI curve following one of two strategies. The first strategy uses the element wise product $h_t = hG_t \odot hL_t$, which applies an adaptive wear factor to each point of the Local Health. The second strategy considers $h_t = hG_0 \cdot h_t$, assuming a fixed wear factor.

Figure 3.2 shows all the steps taken to compute the health vector.

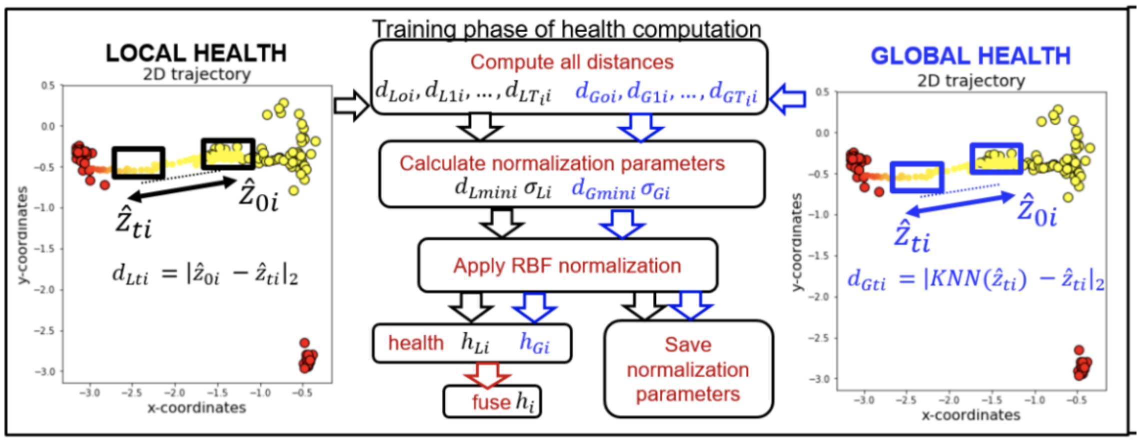

Training phase of health computation
LOCAL HEALTH
2D trajectory
$\hat{z}_{0i}$
$\hat{z}_{ti}$
$d_{Lti} = |\hat{z}_{0i} - \hat{z}_{ti}|_2$
y-coordinates
x-coordinates
Compute all distances
$d_{L0i}, d_{L1i}, \dots, d_{LT_ii}$
$d_{G0i}, d_{G1i}, \dots, d_{GT_ii}$
Calculate normalization parameters
d_{Lmini} σ_{Li} d_{Gmini} σ_{Gi}
Apply RBF normalization
health h_{Li} h_{Gi}
fuse h_i
Save normalization parameters
GLOBAL HEALTH
2D trajectory
$d_{Gti} = |KNN(\hat{z}_{ti}) - \hat{z}_{ti}|_2$

Figure 3.2: Computation of the Health Curve. [44]

3.1.2 RUL Estimation

After calculating the Health Curve, two different matching approaches are used in order to estimate the RUL: Global Health Matching approach and Local Health Curve Matching approach.

The Global Health Matching approach is similar to the approach presented by Wang et al. [52]. Each training unit health vector is compared to each testing unit using the euclidean distance. The training trajectories with the smaller distances comprise the possible RULs. The RUL is then computed as the weighted sum of the possible RULs. In the formulation suggested by Wang et al. [52], a bigger weight is given to smaller RULs. In this method, the entire health vector is taken into account.

The Local Health Curve Matching approach is similar to the above method with the main difference that, instead of using the entire health curve, only the most recent time window w is used. As a consequence, the training set is divided into smaller curves of size w. Afterwards, the testing window distance is computed for all the training windows.

Finally, the two approaches are combined by choosing the minimum value of the RUL of the two approaches at each timestamp.

3.2 Federated Learning with Autoencoders

Federated Learning is still a relatively recent technology and in most cases this methodology is used for classification problems [17, 45, 64]. As a consequence, not much work has been done using these two combined methods.

In [59], the authors propose a Proactive Content Caching algorithm which aims for the model to learn context-specific information in order to cache the most popular files. To find this context's specific features, the authors used Autoencoders. In order for the data to stay in the client side and not be sent to the server, Federated Learning is used to generate a global model. This algorithm outperformed other reference algorithms.

In another work by [34], the authors use Autoencoders for anomaly detection. The Autoencoders' training is also done using Federated Learning. However, in this scenario, the authors presented a limited performance.

3.3 Federated Averaging

`Federated Averaging (FedAvg)` is a FL algorithm presented by [17] which shows a way to generate the global model by periodically averaging the clients locally trained models [30].

The algorithm starts by initializing a global model θ_t. Afterwards, at the t Communication Round (CR), the server selects a random subset of clients S_t, $S_t < K$ and uploads the current global model to the clients in S_t. The chosen clients then perform SGD locally for E epochs to optimize the local objective. Lastly, the clients upload the resulting model updates to the server where they are aggregated to form the new global model θ_{t+1}. Algorithm 1 shows the pseudo-code of the `FedAvg` algorithm.

Algorithm 1 `Federated Averaging` where K represents the number of clients, E is the number of epochs, B is the batch size, η the learning rate and C is a random number between 0 and 1 that represents the fraction of clients chosen for training.

```
Server:
initialize $\theta_t$
for each round t = 1,2,... do
    Select $S_t = C \cdot K$ clients, where $C \in (0,1)$
    upload $\theta_t$ to each client
    for each client $k \in S_t$ do
        $\theta^k_{t+1} \leftarrow ClientUpdate(\theta_t)$
    end for
    $\theta_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \theta^k_{t+1}$
end for
ClientUpdate($\theta$) :
for each local epoch i from 1 to $E$ do
    for batch $b \in B$ do
        $\theta \leftarrow \theta - \eta g_t$
    end for
end for
return $\theta$
```

It is empirically shown in the work by H.Brendan McMahan [17] that the tuning of the number of local updates is of major importance for `FedAvg` to converge. It is clear that more local updates cause the model to be fitter for the local optimization problem and move further away from the initial model, possibly causing divergence.

In a recent work by Yue Zhao and Chandra [60], it is demonstrated that the convergence curves of `FedAvg` for Independent and Identically Distributed (IID) data mostly overlap with the ones resulting from local SGD. However, a significant drop in the test accuracy is noticed when using non-IID data. The work shows that the global weight convergence can be affected by the data distribution.

Figure 3.3 shows the global weight convergence problem. It is clear that in the IID setting, the difference between $w_{mT}^{(f)}$ `FedAvg` and $w_{mT}^{(c)}$ SGD is small. However, the same cannot be said for the non-IID data, where the data distribution makes the divergence between $w_{mT}^{(f)}$ and $w_{mT}^{(c)}$ much larger.

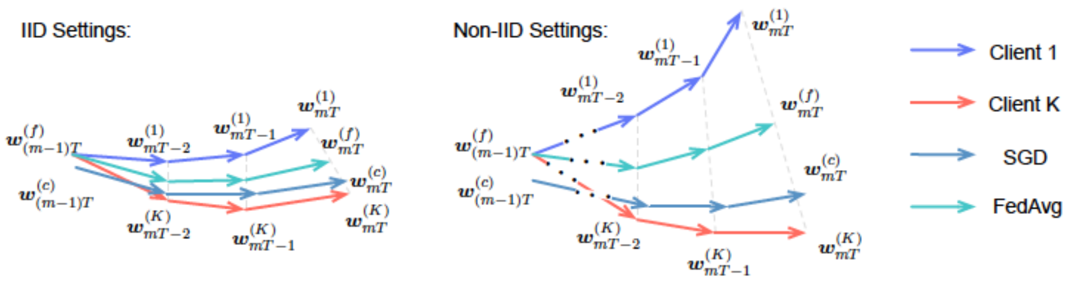

IID Settings:
Non-IID Settings:
$\boldsymbol{w}_{(m-1)T}^{(f)}$
$\boldsymbol{w}_{(m-1)T}^{(c)}$
$\boldsymbol{w}_{mT-2}^{(1)}$
$\boldsymbol{w}_{mT-1}^{(1)}$
$\boldsymbol{w}_{mT}^{(1)}$
$\boldsymbol{w}_{mT}^{(f)}$
$\boldsymbol{w}_{mT}^{(c)}$
$\boldsymbol{w}_{mT}^{(K)}$
$\boldsymbol{w}_{mT-2}^{(K)}$
$\boldsymbol{w}_{mT-1}^{(K)}$
Client 1
Client K
SGD
FedAvg

Figure 3.3: Weight divergence for `FedAvg` with IID and non-IID data, where m is the number of communication rounds. [60]

3.4 Federated Proximal

`Federated Proximal (FedProx)` [38] was developed with the purpose of restricting the amount of divergence of the local model with regard to the global model, removing the need for heuristically limiting the number of local updates.

Similarly to `FedAvg`, this algorithm starts by initializing a global model θ_t. Afterwards, at the t Communication Round (CR) the server selects a random subset of clients S_t, $S_t < K$ and uploads the current global model to the clients in S_t. The chosen clients not only minimize the local model's error, but also minimize the distance between the current model and the local model. Each client local optimizer minimizes the objective given by:

$$\min h_k \quad \text{where} \quad h_k = F_k + \frac{\mu}{2}||\theta - \theta^t||^2 \tag{3.5}$$

, where $\frac{\mu}{2}||\theta - \theta^t||^2$ corresponds to the *proximal term*, which reduces the effect of local updates by making the local model θ closer to the global model θ_t. A cautious reader will note that if $\mu = 0$, then this algorithm is the same as the `FedAvg` algorithm.

After the local optimization, the local updates are sent to the server where they are aggregated to form the new global model, θ_{t+1}, similar to the procedure implemented in `FedAvg`. Algorithm 2 show the pseudo-code of the `FedProx` algorithm.

Algorithm 2 Federated Proximal where K represents the number of clients, E is the number of epochs, C is a random number between 0 and 1 that represents the fraction of clients chosen for training and $\frac{\mu}{2}||\theta - \theta^t||^2$ corresponds to the *proximal term.*

```
Server:
$initialize\ \theta_t$
for each round t = 1,2,... do
    $Select\ S_t\ =\ C\ \cdot K\ clients,\ where\ C\ \in (0,1)$
    $upload\ \theta_t\ to\ each\ client$
    for each client $k \in S_t$ do
        $\theta^k_{t+1} \leftarrow \quad inexact\ minimizer\ of:\ arg\ min\ \mathrm{h}_k(\theta, \theta_t) = F_k + \frac{\mu}{2}||\theta - \theta^t||^2$
    end for
$\theta_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \theta^k_{t+1}$
end for
```

Chapter 4

Methodology

In this chapter, the methodology used to predict the system's health in a Fleet Learning paradigm will be described. The work can be divided into two sections: Federated Learning with TrajecNets and Federated Learning Optimization.

In the first section, the PHM approach which combines Federated Learning with TrajecNets will be described. This has the objective of evaluating whether Federated Learning can be used in a real life scenario, minimizing the time associated with data transferring while keeping a similar performance.

In the second section, several Federated Learning optimizers which were developed in this work will be explained, namely: `Federated Congruent Directional Learning (FedCong)` and `Federated Momentum (FedMom)`.

4.1 PHM Approach

This section will describe the steps to develop a PHM approach in a Fleet Learning paradigm. The system should be able to create a Remaining Useful Life (RUL) or Health Indicator (HI) prediction model by using the computing power available in the aircraft.

Figure 4.1 shows the common steps to formulate a RUL prediction methodology. First of all, the data are captured using the sensors available in the aircraft's systems, generating the dataset. In the second stage, the data are preprocessed in order to select the features that yield the most relevant information and transform the data into a suitable scale. Afterwards, the model is trained using different techniques that will depend on the available system's data. Finally, there is a prediction model that will predict the RUL or the HI.

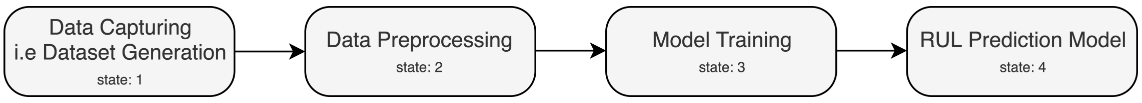


Figure 4.1: PHM pipeline.

It is clear that the data capturing (first stage) is done in the aircraft themselves. However, all the other steps are performed in a centralized environment. By transferring these stages to the edge, it is possible to speed up the process for the RUL or HI computation, minimizing the time constraints associated with data transferring.

4.1.1 Federated Learning

When applying PHM in a FL setting, the training of the model is coordinated by a central server and the model's training is done in the edge. In this work, each aircraft will function as an edge which starts by receiving a global randomly initialized model from the server.

During flight hours, aircraft's sensors capture data which show the continuous degradation of the system. The local model is then trained using all the available data in the aircraft.

Afterwards, the aircraft sends the updated gradients or weights back to the server. In the server, the uploaded models are aggregated in order to form a new global model. This process is repeated every time a model is generated.

When the plane lands in an airport, if a model has been trained then it is sent to the server to be aggregated with other updated models. Furthermore, if a new global model is available, the aircraft downloads it to be used as the new local model.

Figure 4.2 is a representation of a FL setting in Aircraft Maintenance.

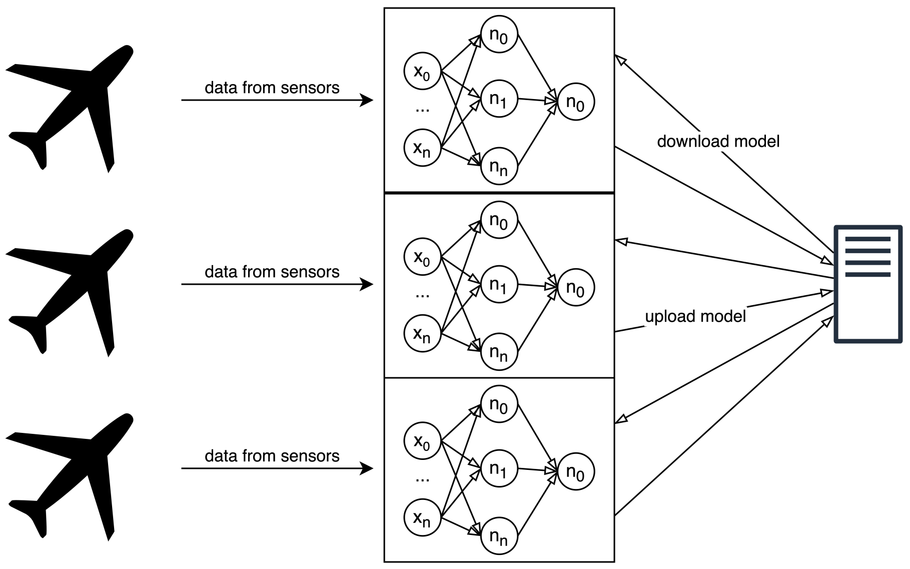


Figure 4.2: Representation of a FL setting in aircraft PHM.

4.1.2 Federated Learning with TrajecNets

This PHM approach was an exploratory work with the objective of verifying whether Federated Learning could be applied to aircraft's PHM systems, reducing the time associated with data transferring while keeping the same performance.

This study will be performed using the work done by Shahid and Ghosh [44], where the authors use TrajecNets to predict a system's RUL. TrajecNets are an important ReMAP project and, hence, was chosen to be analysed in a Federated setting. This work is a perfect example to implement FL since predicting the system's health involves a large amount of data to be transferred to a server to update the model. In addition, TrajecNets use parametrical models where FL can, hence, be applied.

As explained in the previous chapter, these models are Autoencoders used to generate a new feature embedding which will predict the RUL. In this project, the model used is a

Feedforward Autoencoder similar to the TrajecNet's Autoencoder. The main difference is that in this work a FFNN is used for the sake of simplicity, instead of stacked RNNs. In addition, the FL algorithm used is `FedAvg`.

Each aircraft will periodically train a model using its local data. Afterwards, the updated model is sent to a server to later be used to generate a global model. If the error difference between the two models is minimal, it can be concluded that the Federated TrajecNet's Autoencoder has similar results in the RUL prediction compared to the local TrajecNet's Autoencoder.

4.2 Federated Learning Optimization

This section contains the main contributions to this work, where several Federated Learning optimizers were explored. In this work, feedforward regression model was used with the objective of predicting the HI. In this particular case, the HI is equivalent to the life percentage of the system.

The next sections contain the new Federated Learning algorithms proposed and implemented in this work, namely `FedCong` and `FedMom`.

4.2.1 Federated Congruent Directional Learning

In this section, the `FedCong` algorithm will be presented. This algorithm tries to mitigate a problem in `FedAvg`. As it was previously explained, in `FedAvg`, the bigger the number of updates, the more fitted the model is to the local optimization problem, potentially causing divergence. This divergence can lead to a decay in the model's convergence speed.

In the following, the set of clients that minimize the loss function on all the clients will be called congruent and incongruent, in case they do not. Figure 4.3 shows the optimization paths of Federated Learning in two different situations: incongruent, in the left, and congruent, in the right.

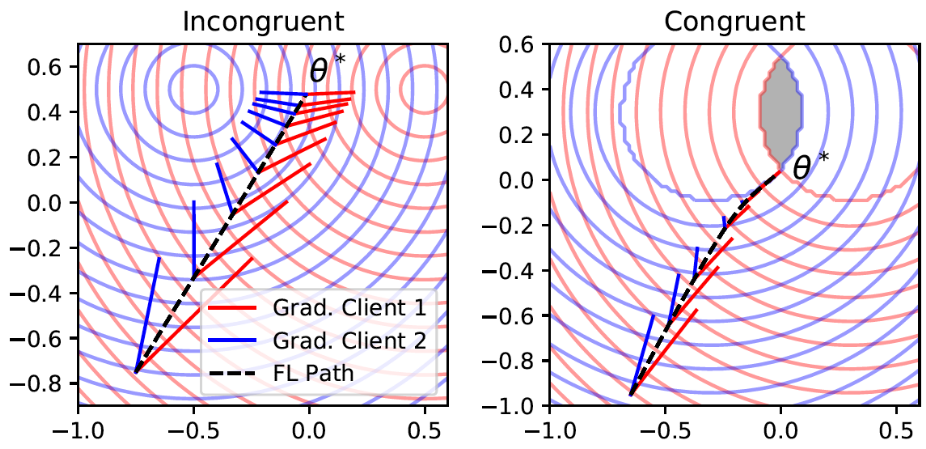


Figure 4.3: Congruent and Incongruent clients. [39]

In the incongruent clients case, the Federated Learning algorithm converges to a stationary point due to the fact that the two clients (red and blue) are converging in opposing directions, meaning that they are minimizing two different optimization problems. In the congruent case, there is a grey space where both clients' loss functions are minimized. The objective of this algorithm is to maximize the impact of congruent clients and mitigate the impact of incongruent clients.

Algorithm Description

It is known that Artificial Neural Networks (ANN) are initialized with random weights, meaning that each time an ANN is initialized there will be different starting points for the training process. In addition, if the network is using a mini-batch type training then the data will be randomly divided. As a consequence, during the error propagation phase, the networks take the input/output batches in a random manner.

Let us analyse a situation where two models that have similar initialization processes and are trained with the same data using a full batch training, meaning the input/output pairs are similar between the two networks. It is clear that these will converge in a similar fashion, represented by the following equation:

$$w_{k1} - w_{k2} = 0 \tag{4.1}$$

where w_{k1} and w_{k2} represent the same weight in the two different models, under the restrictions explained before.

The `FedCong` algorithm was developed taking these facts into consideration. Similar to the other methods, it starts by initializing a global model, θ_t. Afterwards, at a t Communication Round (CR) the server selects the available clients S_t and uploads the current global model to the clients in S_t. The chosen clients then perform GD locally for E epochs in order to optimize the local objective. Lastly, the clients upload the resulting model updates to the server.

At the server, for each local model θ^k_{t+1} received, each weight w^k_{t+1} update for the local problem is analysed. On the one hand, in case $w^k_{t+1} < w_t$, then it can be concluded that w^k_{t+1} had a negative update. On the other hand, in case $w^k_{t+1} > w_t$, then it can be concluded that w^k_{t+1} had a positive update.

After this process in completed, for each weight this algorithm calculates the number of positive and negative updates of all the local models. Afterwards, one of three possible situations will occur:

$$\begin{cases} \text{if} \quad P \geq K * \alpha, & \text{then average the positive clients} \\ \text{if} \quad N \geq K * \alpha, & \text{then average the negative clients} \\ \text{otherwise,} & \text{then average all the clients} \end{cases} \tag{4.2}$$

where K is the number of available clients, P and N are the number of positive and negative updates for a specific weight, respectively, and α(0,1) is a control parameter which specifies the minimum number of positive or negative updates that are necessary to average a weight using only the positive or negative updates. Algorithm 3 shows the pseudo-code for the `FedCong` algorithm.

Algorithm 3 Federated Congruent Directional Learning where K represents the number of clients, E is the number of epochs, B is the batch size, η the learning rate and α is the control term.

```
Server:
initialize $\theta_t$
for each round t = 1,2,... do
  $(L_P,\ L_N) \leftarrow (empty\ list,\ empty\ list)$
  upload $\theta_t$ to each client $\in K$
  for each client $k \in K$ do
    $\theta^k_{t+1} \leftarrow ClientUpdate(\theta_t)$
    if $\theta^k_{t+1} < \theta_t$ then
      $L_N.insert(\theta^k_{t+1})$
    else
      $L_P.insert(\theta^k_{t+1})$
    end if
  end for
  if $L_N >= K \cdot \alpha$ then
    $C \leftarrow L_N$
  else if $L_P >= K \cdot \alpha$ then
    $C \leftarrow L_P$
  else
    $C \leftarrow K$
  end if
  $\theta_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \theta^k_{t+1}$
end for
ClientUpdate($\theta$) :
for each local epoch i from 1 to $E$ do
  for batch $b \in B$ do
    $\theta \leftarrow \theta - \eta g_t$
  end for
end for
return $\theta$
```

4.2.2 Federated Momentum Learning

The FedMom algorithm was inspired by the Momentum optimizer, having the objective of maximizing the training speed of the FedAvg algorithm. A Momentum optimizer update is given in Equation 2.7 and it can be reformulated as follows:

$$\begin{aligned} v_{t+1} &= \beta v_t + \eta g_t \\ w_{t+1} &= w_t - v_{t+1} \end{aligned} \equiv \begin{aligned} v_{t+1} &= \beta v_t - \eta g_t \\ w_{t+1} &= w_t + v_{t+1} \end{aligned}$$

$$\begin{aligned} v_{t_0} &= -\eta g_0 \\ v_1 &= \beta(-\eta g_0) - \eta g_1 \\ v_2 &= \beta(\beta(-\eta g_0) - \eta g_1) - \eta g_2 \\ w_2 &= w_t + (\beta(\beta(-\eta g_0) - \eta g_1) - \eta g_2) \end{aligned} \equiv \begin{aligned} v_{t_0} &= \eta g_{t_0} \\ v_{t_1} &= \beta(\eta g_{t_0}) + \eta g_{t_1} \\ v_{t_2} &= \beta(\beta(\eta g_{t_0}) + \eta g_{t_1}) + \eta g_{t_2} \\ w_{t+1} &= w_t - (\beta(\beta(\eta g_{t_0}) + \eta g_{t_1}) + \eta g_{t_2}) \end{aligned} \tag{4.3}$$

This work purposes a new method named FedMom. The following equations (4.4, 4.5, 4.6) show how the momentum update can be reformulated into a Federated Learning setting.

Firstly, FedMom starts by initializing a global model, θ_t. Afterwards, similarly to FedAvg, the server selects the k available clients and uploads the current global model, θ_t. Subsequently, the clients selected take one or multiple GD steps locally as follows:

$$\theta^k_{t+1} = \theta_t - \eta a_k \quad where \quad a_k = (g^k_e + g^k_{e-1} + ... + g^k_0) \tag{4.4}$$

where θ^k_t is the local model of the $k-th$ client, η is the learning rate, t represents the global model timestamp, e represents the local model timestamp, g^k_e are the error gradients of θ^k_e, and a_k is the sum of all the gradient updates for the local model of the $k-th$ client.

Afterwards, the clients upload the resulting model θ^k_t to the server. In the server, for every θ^k_{t+1} the server calculates the local update. Then, it calculates the global update of the model using each local update as follows:

$$\begin{aligned} -\eta a_k &= \theta^k_{t+1} - \theta_t \\ \alpha &= \sum_{k=1}^{K} \frac{n_k}{n}(-\eta a_k) \end{aligned} \tag{4.5}$$

where α is the global model update, n represents the total number of data points and n_k represents the total number of data points of the $k-th$ client.

After this process is completed, the server stores the momentum variable and updates the global model as follows:

$$\begin{aligned} v_{t+1} &= \delta v_t + \alpha \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{4.6}$$

where δ is the momentum term. As it can be observed, the momentum update (v_{t+1}) is calculated by summing a fraction of the previous update and the global model update. Afterwards, the global model is updated by summing the previous model with the momentum update. Algorithm 4 represents the FedMom pseudo-code.

To sum up, besides making the model converge faster, the `FedMom` algorithm can minimize the noise of a local model if it is not representative of the global model. Algorithm 4 shows the pseudo-code for the `FedMom` algorithm.

Algorithm 4 Federated Momentum Algorithm where K represents the number of clients, E is the number of epochs, B is the batch size, η the learning rate, C is a random number between 0 and 1 that represents the fraction of clients chosen for training and β is the momentum term.

```
Server:
initialize $\theta_t$
for each round t = 1,2,... do
    Select $S_t = C \cdot K$ clients, where $C \in (0,1)$
    upload $\theta_t$ to each client
    for each client $k \in S_t$ do
        $\theta_{t+1}^k \leftarrow ClientUpdate(\theta_t)$
    end for
    $-\eta a_k = \theta_{t+1}^k - \theta_t$
    $v_{t+1} = \beta v_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} a_k$
    $\theta_{t+1} = \theta_t + v_{t+1}$
end for
ClientUpdate($\theta$) :
for each local epoch i from 1 to $E$ do
    for batch $b \in B$ do
        $\theta \leftarrow \theta - \eta g_t$
    end for
end for
return $\theta$
```

This page is intentionally left blank.

Chapter 5

Experimental Results

In this chapter, experimental results will be presented on two main works that have been developed: Federated Learning with TrajecNets and Federated Learning Optimization.

The work on Federated Learning with TrajecNets was developed in order to understand if a Federated Learning model can achieve the same performance of a centralized model.

The work on Federated Learning Optimization is the main contribution of this work and aims to analyse different Federated Learning optimizers. While some of these have been already created by other authors (`FedAvg` [17], `FedProx` [38]), this work introduces two new original optimizers, namely `FedCong` and `FedMom`, where the latter significantly outperforms the previous ones.

Both works were performed using the Turbofan Dataset [41], which will be described in the first section of this chapter section.

5.1 Turbofan Dataset

The Turbofan Dataset is a dataset which has information about aircraft's sensors. The data are not extracted from real airplanes' systems. Instead, it is generated through an engine degradation simulation software using Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) [11], which is a tool that simulates a turbofan engine. Nevertheless, the data simulated is very similar to real life sensors' readings. Figure 5.1 shows the main elements of the system emulated by C-MAPPS.

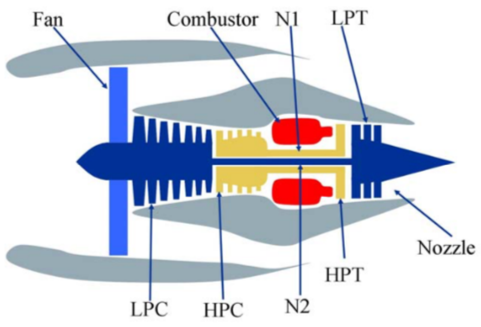


Figure 5.1: Simplified diagram of the turbofan engine. [41]

5.1.1 Structure

The Turbofan dataset is composed of four sub-datasets. Each dataset has a time series readings of 26 features, such as Operational Settings, unit number, time indicator and sensor's values regarding the turbofan engine components. The four different datasets are identical in structure but differ in the number of operational settings and failure modes.

The Operational Settings are represented by three main features: Altitude, Mach Number and Throttle Resolver Angle [41]. The Altitude can be defined as the plane's altitude in each flight cycle. The Mach Number refers to the ratio between the speed of the aircraft and the speed of sound. The Throttle Resolver Angle represents the angular deflection of the pilot's power lever, used to control the pressure applied to the engine. There are six different combinations of Operational Settings which have different impacts on the system's degradation.

The unit number feature acts as an identifier of a time series. At the start of each series, the system operates in a healthy condition until some point in time where the it enters a failure state and can no longer function.

The time indicator feature represents the current time in hours. At the start of the time series, when the system is still in a healthy state, the value is 1. The time eventually reaches time T, where T is the time when the system can no longer operate.

Table 5.1 shows the structure of the four different datasets and their respective properties.

Dataset	Unit Number - Train	Unit Number - Test	Operational Settings	Number of Failures
FD001	100	100	1	1
FD002	260	259	6	1
FD003	100	100	1	2
FD004	248	248	6	2

Table 5.1: Dataset's Structure.

Each dataset has a training and a testing set. The training sets are run to failure, i.e complete lifetime, whereas the testing sets are pruned some time prior to the failure. In addition, the testing set has information containing the value of the Remaining Useful Life (RUL) at the time that the unit was pruned prior to a failure.

The testing dataset was not used since it is supposed to be used to predict the RUL, which is not within the scope of this work. The training set was divided into 80% for training and 20% testing.

5.1.2 Data Preprocessing

In order to prepare the dataset to be used for this work, the data was preprocessed using different techniques. Firstly, in order to calculate the life percentage (HI), each time series X_i was converted in the following manner:

$$y_j = \frac{max_{X_i} - y_j}{max_{X_i}} \tag{5.1}$$

where y_j represents a time indicator of X_i and max_{X_i} is the maximum of X_i (i.e. the timestamp when the engine could no longer function).

The sensors' data are normalized to be within a range of [0, 1] using the min-max normalization:

$$x_{j,n} = \frac{x_{j,n} - min_{X_n}}{max_{X_n} - min_{X_n}} \tag{5.2}$$

where $x_{j,n}$ is the $i - th$ data point of the $j - th$ sensor and max_{X_n} and min_{X_n} are the maximum and minimum of $j - th$ sensor, respectively.

After some literature review, it was concluded that some sensors' values did not provide valuable information. According to [18, 26, 62], only 14 out of the 21 sensors presented valuable information to be used as input features, whose indexes are: 2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20 and 21. Hence, the other features where not used in this work to train the models.

5.2 Federated Learning with TrajecNets

In this section, the experimental results of Federated Learning using TrajecNets will be presented. The chosen dataset was the `FD001` dataset which has one operational setting and one failure type. In order to have a fair comparison, both centralized and Federated Autoencoders use the same training and testing sets.

The centralized Autoencoder (baseline) and the Federated Autoencoder have the same architecture which is represented in Figure 5.2. The encoder has two hidden layers with 100 and 20 neurons, respectively, and *tanh* activation functions. The bottleneck has two neurons and uses a linear activation function.

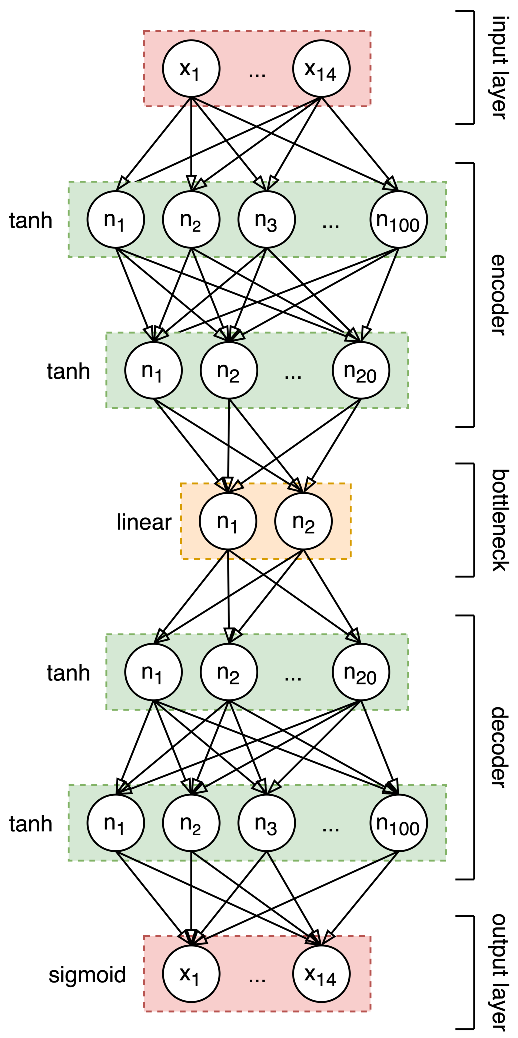


Figure 5.2: Representation of the Autoencoder's architecture, where n_i represents the $i-th$ neuron. The function on the left side of each layer represents that layer's activation function.

Both models were trained using their reconstruction error, which represents the proximity between the input and the output. In this case, the reconstruction error metric used was the Mean Square Error (MSE), given by:

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(\hat{y}_i - y_i)^2 \tag{5.3}$$

where n is the number of training samples, $\hat{y}_i$ is the predicted output and y_i is the actual output.

5.2.1 Baseline model

This section will present the results of the baseline model of this work, which is a centralized Autoencoder. The model is trained using the configurations used in [44]: a batch size of 16 and 50 epochs. For this work, the optimizer chosen was Adadelta [61] due to its fast convergence rate.

The Autoencoder bottleneck outputs a 2D data embedding that is then used to compute the RUL. Figure 5.3 shows a representation of the output of the Autoencoder's bottleneck, where $x0$ and $x1$ represent the output of the neurons on the bottleneck layer for the testing set.

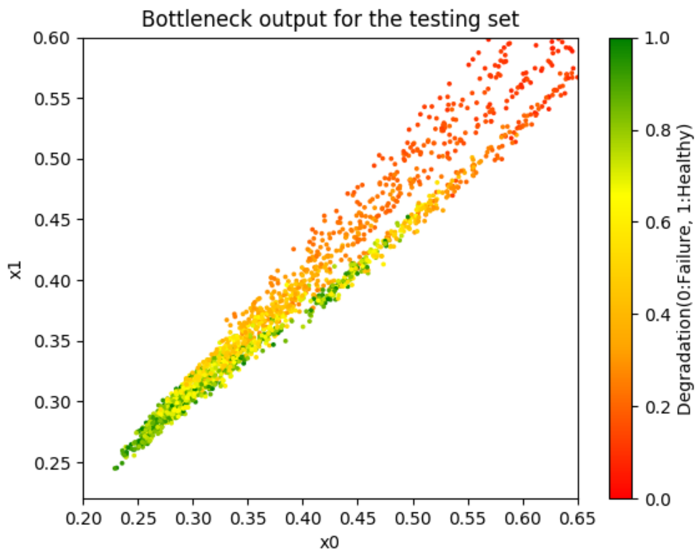


Figure 5.3: Output of the bottleneck layer for the testing set.

Each point in the plot represents an unit at a certain time step. In addition, a mapping was made between each point in the plot and its respective HI calculated in the data preprocessing stage. The color represents the unit's HI. When the points are green (1.0), the unit is in a healthy state. Contrarily, when the points are red (0.0), it means that the unit can no longer function.

Looking at the figure, it is clear that the Autoencoder finds a deterioration pattern in the data. On the one hand, if $x0$ and $x1$ present low values, it can be observed that the units have high health percentages. On the other hand, as xo and $x1$ increase, the life percentage of the units decreases.

5.2.2 Federated Autoencoder

The main objective of the Federated Autoencoder is achieve the same performance of the centralized Autoencoder (baseline) using FL methodologies.

A new global model is generated at each CR. For this experiment the number of Communication Rounds (CR) was set to 20. When a new global model is generated, the reconstruction error is calculated using the testing set. The error presents corresponds to the error collected when the model stabilized.

Different combinations of parameters (E, K and B) were experimented in order to maximize the performance of the Federated Autoencoder. It is expected that if the value of K is high, it may lead to drop in performance due to the fact that the data distribution for each client will not be representative of the population distribution.

5.2.3 Performance Comparison

In this section, a comparison between the centralized and Federated Autoencoder will be made in order to ascertain if the FL model achieves a similar performance. The MSE of the baseline model was 8.78×10^{-3}.

Table 5.2 shows the performance under different conditions for the `FD001` dataset.

K	E	B	MSE	improvements
2	30	10	8.49×10^{-3}	3%
2	40	15	7.77×10^{-3}	11%
5	30	10	8.89×10^{-3}	-1%
5	40	15	8.68×10^{-3}	1.1%
10	30	10	9.93×10^{-3}	-13%
10	40	15	9.95×10^{-3}	-13.3%

Table 5.2: Performance of the Federated Autoencoder for the `FD001` dataset, where K is the number of clients, E is the number of local epochs, B is the local batch size, MSE is the Mean Squared Error and improvements are the improvements over the baseline.

Firstly, the best overall performance was achieved when $K = 2$, $E = 40$ and $B = 15$, with an improvement of 11% over the baseline. This improvement might have to do with the fact that the constant averaging of the models might have caused it to not get stuck in sub-optimal solutions, which might have been the case in the centralized Autoencoder. This should be further investigated.

In addition, the number of local updates (E and B) prove to have a significant impact on the models convergence. When the number of clients 10 and there is a big number of local updates, the models diverge more from the global model.

Furthermore, with the increase of the number of clients, it can be observed that the MSE also increases. This is related to the fact that by dividing the data into multiple clients, each client will have a worst representation of the global data and will be trying to minimize a different optimization problem. As such, the bigger the number of clients, the worst the data representation is.

To conclude, this experiment shows that a Federated Autoencoder can achieve a similar performance to a centrally trained Autoencoder under specific conditions. However, when the number of clients is high, improvements to the Federated Learning algorithm should be made. In addition, it can be concluded that the data distribution is of major importance in the convergence of the Federated model.

5.3 Federated Learning Optimization

In this section, the experimental results of the Federated Learning optimizers will be presented. The optimizers tested were: `FedAvg`, `FedProx`, `FedCong` and `FedMom`.

In the scope of this work, the baseline model considered is the `FedAvg`, since this model was already implemented by H.Brendan McMahan [17]. In addition, `FedProx` by Sahu et al. [38] is also evaluated since it is one of the state of the art algorithms. The primary objective of this work is that `FedCong` and `FedMom` outperform `FedAvg`.

For each dataset, each of the optimizers was run five times and the results presented are the average values of the Mean Absolute Error (MAE) of the testing set and CR of stabilization of those runs. As the main objective of the optimizers developed is to increase the convergence speed of the models while maintaining the MAE, the best algorithm is the one whose CR of stabilization is the lowest without increasing the MAE.

In order to achieve this, for each dataset the `FedAvg` algorithm (baseline) was run until the error had stabilized and the corresponding CR was registered. With regard to the other algorithms, the CR when that algorithm achieves a similar performance in terms of error to the `FedAvg` algorithm is registered.

For each optimizer, different parameters will be varied in order to retrieve the best model in terms of Communication Round (CR). Finally, the best models of each optimizer will be compared for each of the four datasets which were previously described, namely: `FD001`, `FD002`, `FD003` and `FD004`.

Differently from the previous work on Federated Learning with TrajecNets, for each dataset J clients were created so that each client has exactly 2 units:

$$J = \frac{n_{ux}}{2} \tag{5.4}$$

where nu_x represents the number of units in the the `FD00x` training dataset. As such, each client only has a small portion of the data, which is usually the reality in a FL setting. In the following experimentations, for each CR K clients are randomly selected from J.

5.3.1 Local Model

Each client of the federated setting is represented by a FFNN that has the objective of predicting the system's HI. This neural network takes as input the preprocessed sensors' values. The network architecture is represented in Figure 5.4.

Looking at the architecture, it can be observed that it has three hidden layers with 20, 30 and 20 neurons, respectively, with *tanh* activation functions. The output layer is a single neuron with a *sigmoid* activation function, which represents the predicted system's HI. The local optimizer used was GD and the error function used was the MSE.

Figure 5.5 shows four different unit predictions of the HI of the testing set of the dataset. It can be observed that the closer the predicted value is to the ground truth, the smaller the error is.

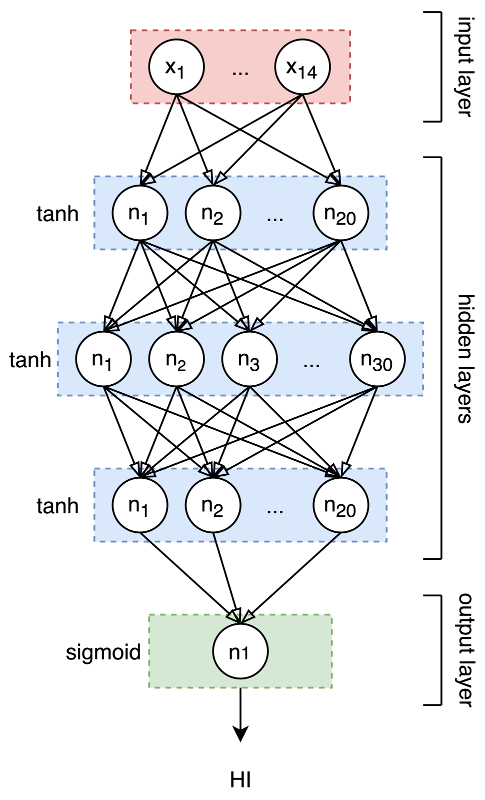
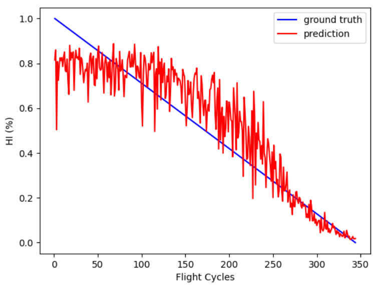
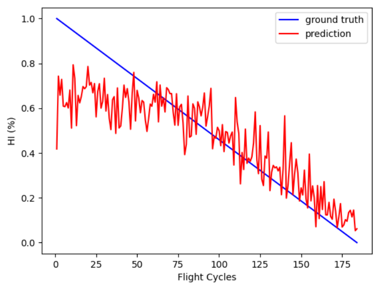


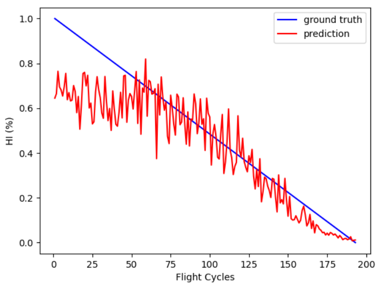
Figure 5.4: Representation of the network architecture where n_i represents neuron $i-th$. The function on the left side of each layer represents that layer's activation function.



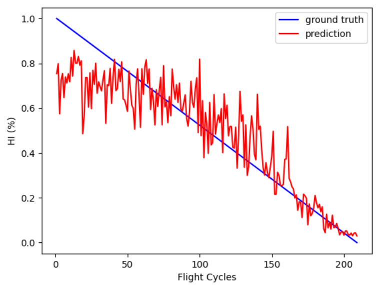
(a) HI prediction for unit 24.



(b) HI prediction for unit 30.



(c) HI prediction for unit 36.



(d) HI prediction for unit 47.

Figure 5.5: HI prediction for four different units

5.3.2 Dataset FD001

For the first dataset, the configuration of the optimizer chosen is presented in the table bellow. These configurations are fixed for all the tested optimizers.

E	CR	B	K	η
30	300	∞	10 or 20	0.01

Table 5.3: Overall configuration of the optimizers, where E represents the number of epochs, CR represents the number of communication rounds, B represents the local batch size, η the learning rate and K represents the number of clients.

Federated Averaging (FedAvg)

Table 5.4 and Figure 5.6 present the results of the `FedAvg` algorithm. Firstly, it can be observed that for both cases, $K = 10$ and $K = 20$, the error converges at the same CR, 250.

K	MAE	CR of stabilization
10	1.32×10^{-1}	250
20	1.31×10^{-1}	250

Table 5.4: `FedAvg` performance, where K represents the number of clients, MAE represents the mean absolute error and CR of stabilization represents the communication round when the error stabilized.

In addition, it can be observed that the MAE of the model for this CR using 10 clients is $1.32 * 10^{-1}$, which is the around same as the one using 20 clients, $1.31 * 10^{-1}$. The difference between the two, although being small, might have to do with the fact that the model using 20 clients uses a bigger batch, causing it to have a better representation of the global optimization problem.

This algorithm will serve as the baseline to the other algorithms in order to compare their performance and convergence speed.

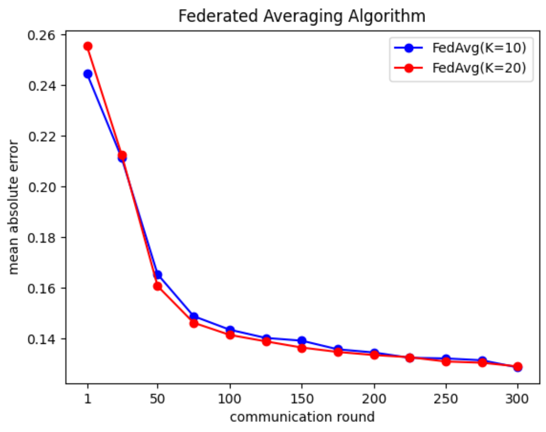


Figure 5.6: `FedAvg` for $K = 10$ and $K = 20$

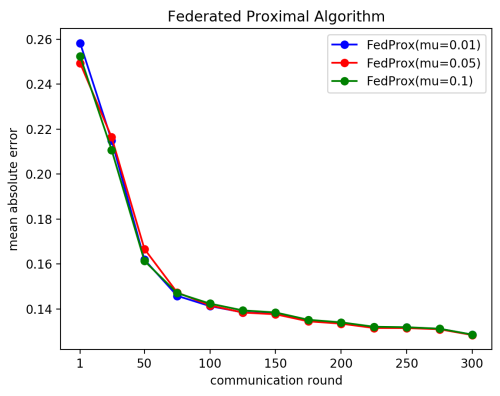
Federated Proximal (FedProx)

Table 5.5 and Figure 5.7 show the results for the `FedProx` algorithm for $K = 10$ and $K = 20$. Looking at the results, it can be concluded that all the models presented a similar or slower convergence speed comparing to the baseline model, `FedAvg`.

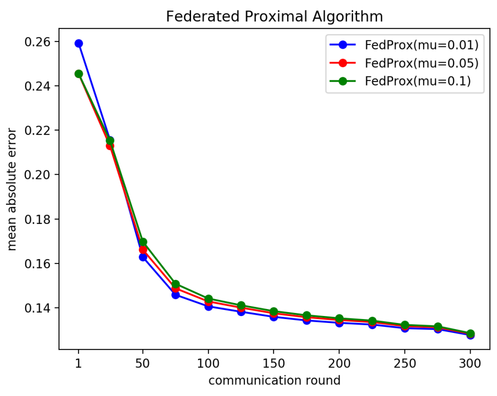
K	μ	CR of stabilization	improvements
10	0.01	250	0%
10	0.05	250	0%
10	0.1	260	-4%
20	0.01	250	0%
20	0.05	250	0%
20	0.1	275	-10%

Table 5.5: `FedProx` performance, where K represents the number of clients, μ represents the control term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

Moreover, when $\mu = 0.1$ for both $K = 10$ and $K = 20$ the model had slower convergence rate with decreases of 4% and 10% over the baseline, respectively. This overall drop in convergence speed is natural since the objective of `FedProx` is to restrict the divergence between the local model and the global model.



(a) K=10



(b) K=20

Figure 5.7: `FedProx` for different combinations of μ values.

Federated Directional Congruent Learning (FedCong)

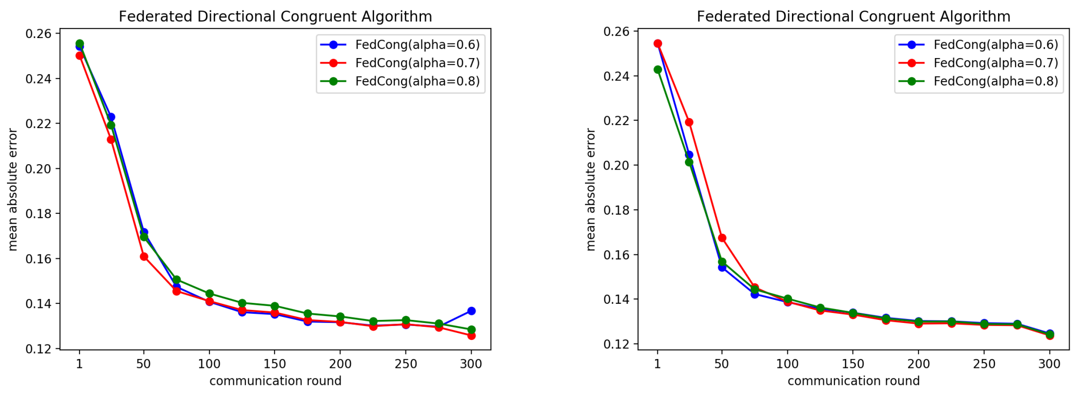
Table 5.6 and Figure 5.8 present the results of the `FedCong` algorithm for $K = 10$ and $K = 20$. Looking at the results, it can be concluded that all the models presented a faster or equal convergence comparing to the baseline model, `FedAvg`.

K	α	CR of stabilization	Improvements
10	0.6	160	36%
10	0.7	200	20%
10	0.8	250	0%
20	0.6	150	40%
20	0.7	140	44%
20	0.8	175	30%

Table 5.6: `FedCong` performance, where K represents the number of clients, α represents the control term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

Moreover, when the number of clients is 10, the best values of the control term are 0.6 and 0.7, with a substantial improvement of 36% and 40% over the baseline. Figure 5.23a compares the different α values for when $K = 10$.

Furthermore, looking at the results of the models using 20 clients, it can be concluded that the best model is the one where the control term is equal to 0.7, having an improvement of 44% over the baseline. Comparing to the model using 10 clients with the same control term, this model seems to converge better as well. Figure 5.23b compares the different α values for when $K = 20$.



(a) K=10 (b) K=20

Figure 5.8: `FedCong` for different combinations of α values.

Federated Momentum (FedMom)

Table 5.7 and Figure 5.9 present the results of the `FedMom` algorithm for $K = 10$ and $K = 20$. Looking at the table, it can be observed that this algorithm converged faster than the baseline for almost every δ value, regardless of the number of clients, and presenting its best performance when $\delta = 0.9$.

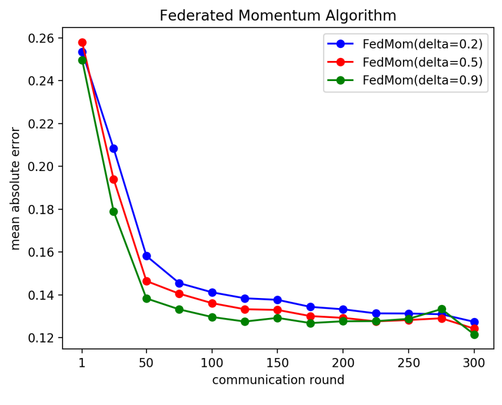
K	δ	CR of stabilization	Improvements
10	0.2	225	10%
10	0.5	200	20%
10	0.9	75	70%
20	0.2	250	0%
20	0.5	150	40%
20	0.9	70	72%

Table 5.7: `FedMom` performance, where K represents the number of clients, δ represents the momentum term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

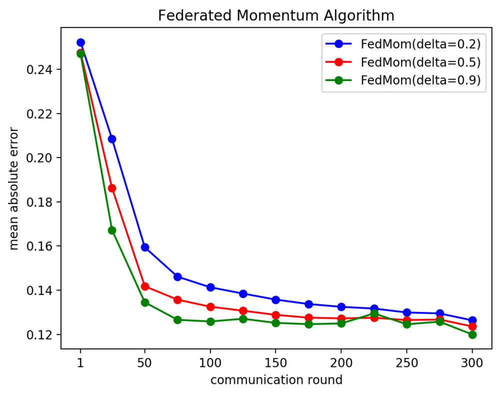
When the number of clients is 10, the best value of the momentum term is 0.9, with a substantial improvement of 70% over the baseline. The models with $\delta = 0.5$ and $\delta = 0.2$ also presented good improvements.

Plot 5.9a shows the error of the `FedMom` algorithm when $K = 10$. The convergence curve for $\delta = 0.5$ and $\delta = 0.2$ seems to be more robust comparing to the model where $\delta = 0.9$. The reason for this is most likely due to the fact that a high δ value causes the model to give high importance to momentum values that may have a bad representation of the global problem.

Plot 5.9b shows the error of `FedMom` algorithm when $K = 20$. Firstly, it can be observed that the results are similar to when $K = 10$. Similarly to before, the convergence curve seems more robust when $\delta = 0.5$ and $\delta = 0.2$, due to same reason explained before for when the number of clients is 10.



(a) $K = 10$.



(b) $K = 20$.

Figure 5.9: `FedMom` Algorithm for different combinations of δ values.

Summary

Table 5.8 and Figure 5.10 presents the summary of the results of the different optimizers for $K = 10$ and $K = 20$.

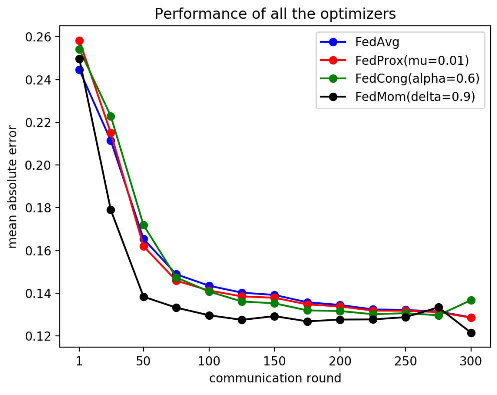
Optimizer	K	CR of stabilization	Improvements
`FedAvg`	10	250	—
`FedProx`	10	250	0%
`FedCong`	10	160	36%
`FedMom`	10	75	70%
`FedAvg`	20	250	—
`FedProx`	20	250	0%
`FedCong`	20	140	44%
`FedMom`	20	70	72%

Table 5.8: Optimizers performance, where K represents the number of clients, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

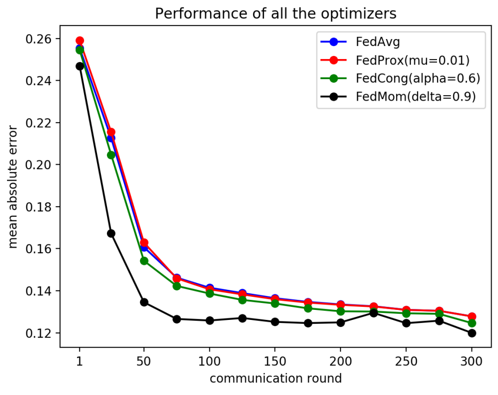
It is clear that both `FedCong` and `FedMom` improved the model convergence speed over the baseline with the later registering overall improvements of 70% and 72% for $K = 10$ and $K = 20$, respectively.

In addition, the `FedCong` registered its best performance when $\alpha = 0.6$, whereas `FedMom` registered its best performance when $\delta = 0.9$.

To sum up, looking at Figure 5.10 it can be observed that the `FedCong` and `FedMom` performance curves are steeper in comparison to `FedAvg` and `FedProx`, with `FedMom` being more noticeable. This means that the two new developed algorithms converge faster than the others while maintaining the same performance, which was the objective of this experiment.



(a) $K = 10$.



(b) $K = 20$.

Figure 5.10: Performance of the different optimizers.

5.3.3 Dataset FD002

For the second dataset, the configuration of the optimizer chosen is presented in the table bellow. These configurations are fixed for all the tested optimizers.

E	CR	B	K	η
120	300	∞	10 or 20	0.05

Table 5.9: Overall configuration of the optimizers, where E represents the number of epochs, CR represents the number of communication rounds, B represents the local batch size, η the learning rate and K represents the number of clients.

Federated Averaging (FedAvg)

Table 5.10 and Figure 5.11 present the results of the `FedAvg` algorithm. It can be observed that for both cases, $K = 10$ and $K = 20$ the error converges at the same CR, 150.

K	MAE	CR of stabilization
10	1.26×10^{-1}	150
20	1.25×10^{-1}	150

Table 5.10: `FedAvg` performance, where K represents the number of clients, MAE represents the mean absolute error and CR of stabilization represents the communication round when the error stabilized.

In addition, it can be observed that the MAE of the model when $K = 10$ is $1.26 * 10^{-1}$, which is the around same as the one when when $K = 20$, $1.25 * 10^{-1}$. Furthermore, the converge curve of the model when $K = 20$ looks to be more robust than when $K = 10$. The reason for this might have to do with the fact that the first has a better representation of the global error problem, causing it to have a faster convergence rate with less oscillations.

This algorithm will serve as the baseline to the other algorithms for the `FD002` dataset in order to compare their performance and convergence speed.

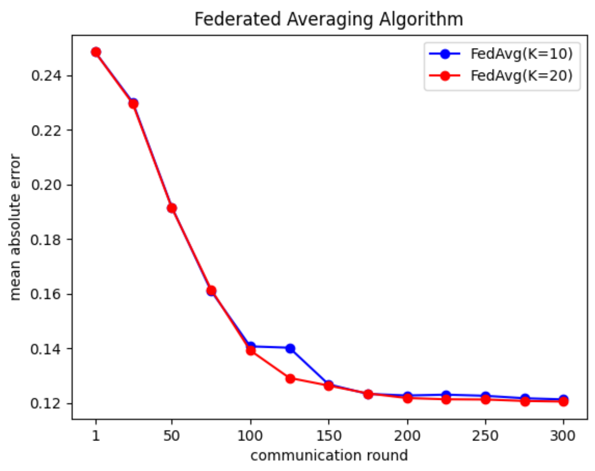


Figure 5.11: `FedAvg` for $K = 10$ and $K = 20$

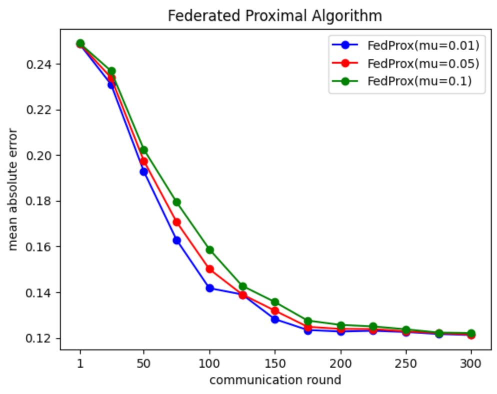
Federated Proximal (FedProx)

Table 5.11 and Figure 5.12 present the results of the FedProx algorithm for $K = 10$ and $K = 20$. Looking at the results, it can be concluded that all the models presented a similar or slower convergence comparing to the baseline model, FedAvg.

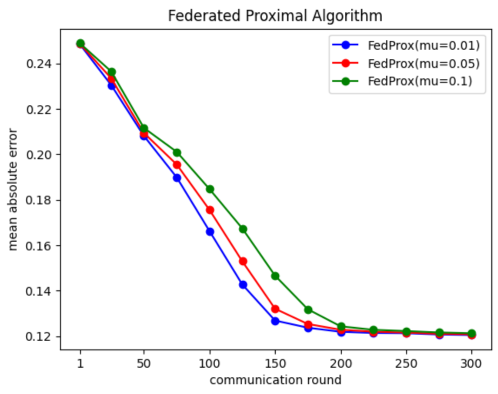
K	μ	CR of stabilization	improvements
10	0.01	150	0%
10	0.05	160	-6%
10	0.1	175	-16.7%
20	0.01	150	0%
20	0.05	175	-16.7%
20	0.1	200	-33.3%

Table 5.11: FedProx performance, where K represents the number of clients, μ represents the control term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

Moreover, when $\mu = 0.1$, for both $K = 10$ and $K = 20$, the model model took more time to converge, with decreases of -16.7% and -33.3%, respectively. This performance drop in convergence speed is normal since this algorithm tries to restrict the divergence between the local model and the global model.



(a) K=10



(b) K=20

Figure 5.12: FedProx for different combinations of μ values.

Federated Directional Congruent Learning (FedCong)

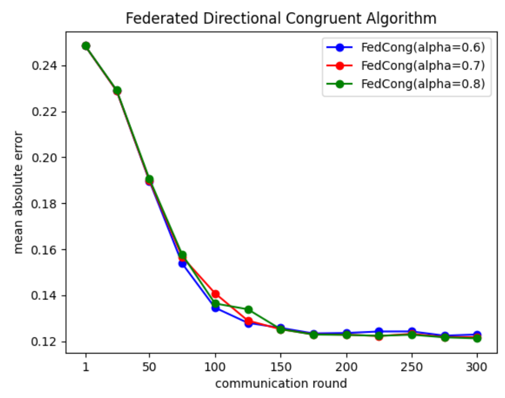
Table 5.12 and Figure 5.13 present the results of the `FedCong` for $K = 10$ and $K = 20$. Looking at the results, it can be concluded that most of the models presented faster convergence rate comparing to the baseline model, `FedAvg`.

K	α	CR of stabilization	Improvements
10	0.6	125	16.7%
10	0.7	125	16.7%
10	0.8	150	0%
20	0.6	125	16.7%
20	0.7	125	16.7%
20	0.8	125	16.7%

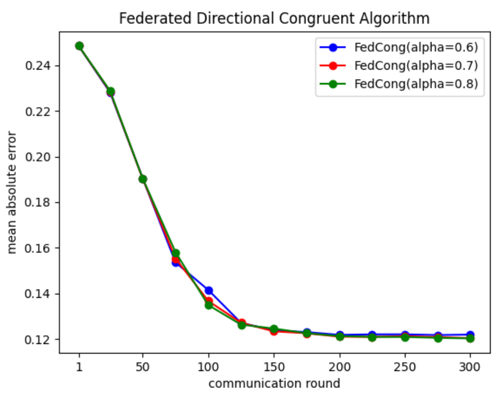
Table 5.12: `FedCong` performance, performance, where K represents the number of clients, α represents the control term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

Moreover, when the number of clients is 10, the best values of the control term are 0.6 and 0.7, with improvements of 16.7% and 16.7% over the baseline, respectively. Figure 5.13a compares the different α values for when $K = 10$. The spike in the baseline model between the 100 and 125 CR is less noticeable when $\alpha = 0.7$ and it completely disappears when $\alpha = 0.6$.

Furthermore, looking at the results of the models when the number of clients is 20, it can be concluded that the models convergence is similar, having an improvement of 16.7% over the baseline. Comparing to the model when $K = 10$ with the same control term, this model seems to have a more robust convergence with less fluctuation.



(a) K=10



(b) K=20

Figure 5.13: `FedCong` for different combinations of α values.

Federated Momentum (FedMom)

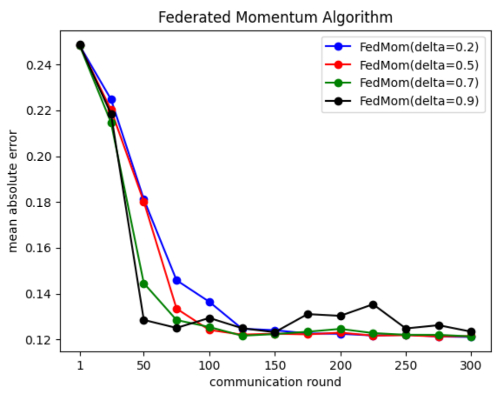
Table 5.13 and Figure 5.14 present the results of the `FedMom` algorithm for $K = 10$ and $K = 20$. Looking at the table, it can be observed that this algorithm converged faster than the baseline for every δ value, regardless of the number of clients.

K	δ	CR of stabilization	Improvements
10	0.2	125	16.7%
10	0.5	100	33%
10	0.7	75	50%
10	0.9	—	—
20	0.2	125	16.7%
20	0.5	100	33%
20	0.7	75	50%
20	0.9	75	50%

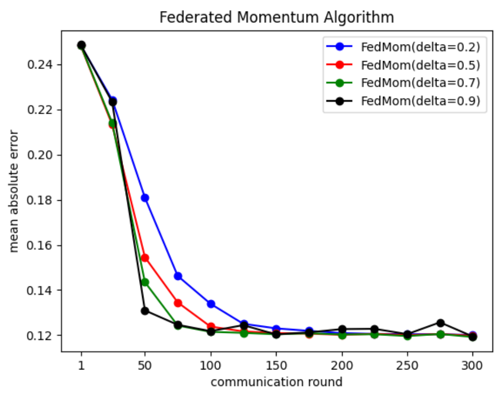
Table 5.13: `FedMom` performance, where K represents the number of clients, δ represents the momentum term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

When the number of clients is 10, the best value of the momentum term is 0.7, with a substantial improvement of 50% over the baseline. The models where $\delta = 0.5$ and $\delta = 0.2$ also presented good improvements. When $\delta = 0.9$, it can be observed that the convergence curve is not stable so it is impossible to say its CR of stabilization. The reason for this is most likely due to the fact that a high δ value causes the model to give higher importance to momentum values that may have a bad representation of the global problem.

When the number of clients is 20, the models present similar results to $K = 10$ with the exception of when the δ value is 0.9, which presents a more robust convergence. Similarly to before, the convergence curve seems more robust when $\delta = 0.5$ and $\delta = 0.2$, due to same reason explained before.



(a) $K = 10$.



(b) $K = 20$.

Figure 5.14: `FedMom` for different combinations of δ values.

Summary

Table 5.14 and Figure 5.15 present the summary of the results of the different optimizers for $K = 10$ and $K = 20$.

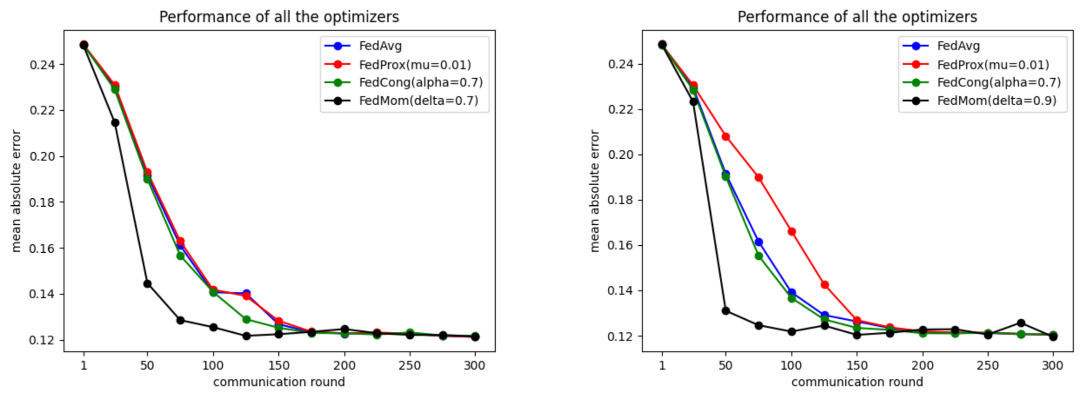
Optimizer	K	CR of stabilization	Improvements
`FedAvg`	10	150	—
`FedProx`	10	150	0%
`FedCong`	10	125	16.7%
`FedMom`	10	75	50%
`FedAvg`	20	150	—
`FedProx`	20	150	0%
`FedCong`	20	125	16.7%
`FedMom`	20	75	50%

Table 5.14: Optimizers performance, where K represents the number of clients, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

It is clear that both `FedCong` and `FedMom` improved the model convergence speed compared to the baseline with overall improvements of 16.7% and 50% for $K = 10$ and $K = 20$, respectively.

The `FedCong` registered its best performance when $\alpha = 0.7$. The value was chosen over $\alpha = 0.6$ due to the fact that it presented a more robust converge curve. `FedMom` registered its best performance when $\delta = 0.9$ for $K = 20$, and $\delta = 0.7$ when $K = 10$.

To conclude, looking at Figure 5.15 it can be observed that the `FedCong` and `FedMom` performance curves are steeper in comparison to `FedAvg` and `FedProx`, with `FedMom` standing out. This means that the two new proposed algorithms converge faster than the others while maintaining the same performance, which was the objective of this experiment.



(a) $K = 10$. (b) $K = 20$.

Figure 5.15: Performance of the different optimizers.

5.3.4 Dataset FD003

For the first dataset, the configuration of the optimizer chosen is presented in the table bellow. These configurations are fixed for all the tested optimizers.

E	CR	B	K	η
120	300	∞	10 or 20	0.002

Table 5.15: Overall configuration of the optimizers, where E represents the number of epochs, CR represents the number of communication rounds, B represents the local batch size, η the learning rate and K represents the number of clients.

Federated Averaging (FedAvg)

Table 5.16 and Figure 5.16 present the results of the `FedAvg` algorithm. Firstly, it can be observed that for both cases, $K = 10$ and $K = 20$ the error converges to the same CR, 250.

K	MAE	CR of stabilization
10	1.53×10^{-1}	250
20	1.52×10^{-1}	250

Table 5.16: `FedAvg` Performance, where K represents the number of clients, MAE represents the mean squared error and CR of stabilization represents the communication round when the error stabilized

In addition, it can be observed that the MAE of the model when the number of clients is 10 is $1.53 * 10^{-1}$, which is a similar value to when $K = 20$, $1.52 * 10^{-1}$. For this dataset, the data distribution influence is more noticeable. Figure 5.16 clearly shows that when $K = 10$ the model has problems converging. This is due to the fact that using a smaller sample $K = 10$ may not be representative of the global optimization problem.

This algorithm will serve as the baseline to the other algorithms in order to compare their performance and convergence speed.

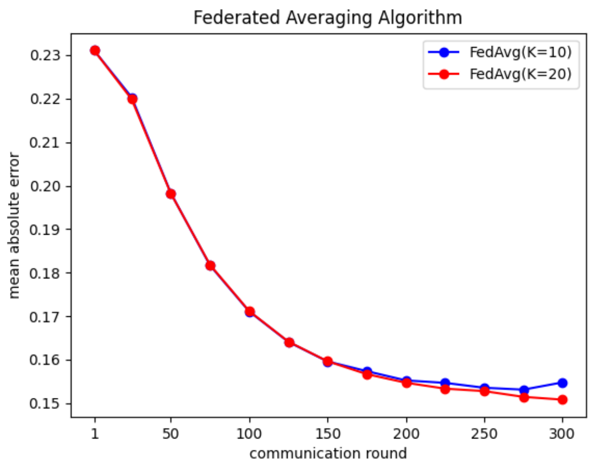


Figure 5.16: `FedAvg` for $K = 10$ and $K = 20$

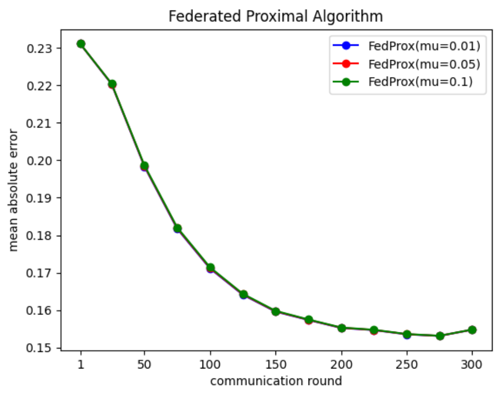
Federated Proximal (FedProx)

Table 5.17 and Figure 5.17 present the results of the `FedProx` algorithm for $K = 10$ and $K = 20$. Looking at the results, it can be concluded that all the models presented a similar convergence comparing to the baseline model, `FedAvg`.

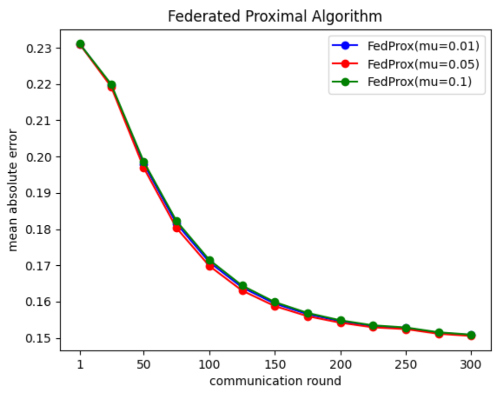
K	μ	CR of stabilization	improvements
10	0.01	250	0%
10	0.05	250	0%
10	0.1	250	0%
20	0.01	250	0%
20	0.05	250	0%
20	0.1	250	0%

Table 5.17: `FedProx` performance, where K represents the number of clients, μ represents the control term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

For this particular dataset, the results of FedProx are very similar to the baseline. In addition, the μ value does not seem to have influence since the error values do not vary between each other, making it hard to reach conclusions.



(a) K=10



(b) K=20

Figure 5.17: `FedProx` for different combinations of μ values.

Federated Directional Congruent Learning (FedCong)

Table 5.18 and Figure 5.18 present the results of the `FedCong` for $K = 10$ and $K = 20$. Looking at the results, it can be concluded that all the models presented faster or equal convergence rate comparing to the baseline model, `FedAvg`.

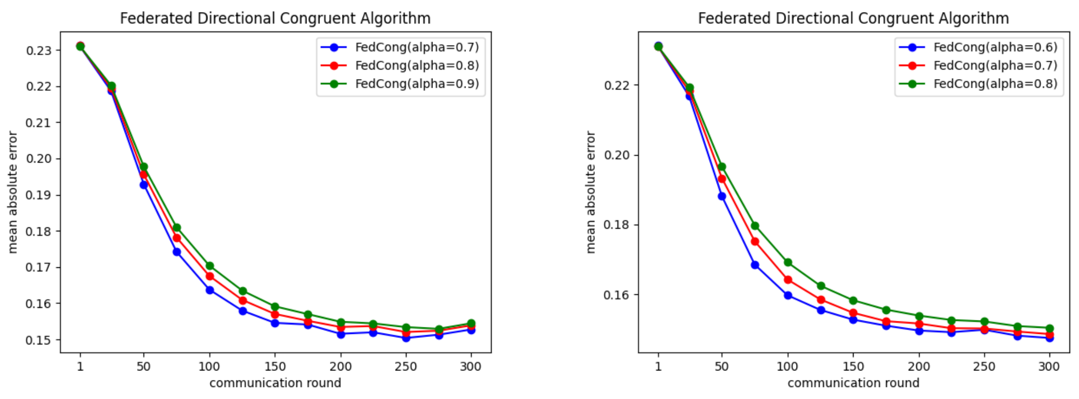
K	α	CR of stabilization	Improvements
10	0.7	175	30%
10	0.8	200	20%
10	0.9	250	0%
20	0.6	150	40%
20	0.7	175	20%
20	0.8	225	10%

Table 5.18: `FedCong` performance, where K represents the number of clients, α represents the control term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

In this dataset, for $K = 10$, the data distribution of the clients was causing difficulties in the model's convergence for lower α values. As such, the α values tested in this dataset are bigger, in order to use a bigger sample which allows to have a better error representation. Moreover, when $K = 10$, the best value of the control term is 0.7, with a substantial improvement of 30% over the baseline.

Furthermore, looking at the results of the models using 20 clients, it can be concluded that the best model is the one where the control term is equal to 0.6, having an improvement of 40% over the baseline. In addition, when the number of clients is 20 the sample size is bigger and, as a consequence, it has a better representation of the global problem.

To sum up, in this dataset, `FedCong` had a good performance by eliminating the updates that did not represent the global error problem.



(a) K=10 (b) K=20

Figure 5.18: `FedCong` for different combinations of α values.

Federated Momentum (FedMom)

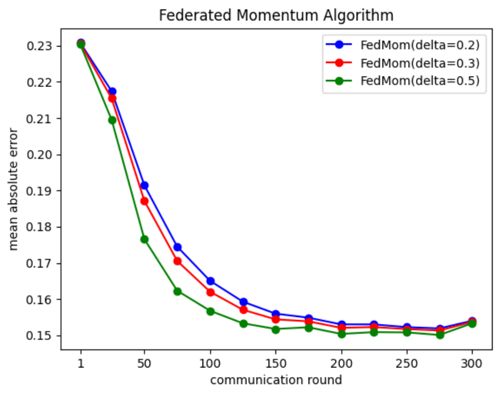
Table 5.19 and Figure 5.19 present the results of the `FedMom` algorithm for $K = 10$ and $K = 20$. Looking at the table, it can be observed that this algorithm converged faster than the baseline for every δ value, regardless of the number of clients and presenting its best performance when $\delta = 0.9$.

K	δ	CR of stabilization	Improvements
10	0.2	225	10%
10	0.3	175	30%
10	0.5	125	50%
20	0.2	200	20%
20	0.5	125	50%
20	0.7	100	60%

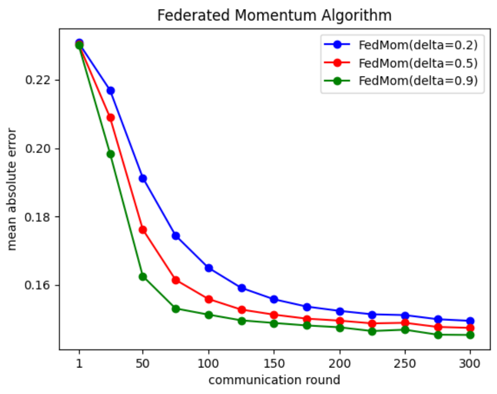
Table 5.19: `FedMom` performance, where K represents the number of clients, δ represents the momentum term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

When $K = 10$, the data distribution of the clients was causing difficulties in the model's convergence. As such, the δ value in this situation for this dataset were set to smaller values so that the model was able to converge. When the number of clients is 10, the best value of the momentum term is 0.5, with a substantial improvement of 50% over the baseline. The models where $\delta = 0.3$ and $\delta = 0.2$ also presented good improvements.

When $K = 20$, the sample size is bigger and, since it has a better representation of the global problem so the values of δ can be bigger. When the number of clients is 20, the best value of the momentum term is 0.7 with an improvement of 60% over the baseline. The models where $\delta = 0.2$ and $\delta = 0.5$ also presented good improvements.



(a) $K = 10$.



(b) $K = 20$.

Figure 5.19: `FedMom` for different combinations of δ values.

Summary

Table 5.20 and Figure 5.20 present summary of the results of the different optimizers for $K = 10$ and $K = 20$.

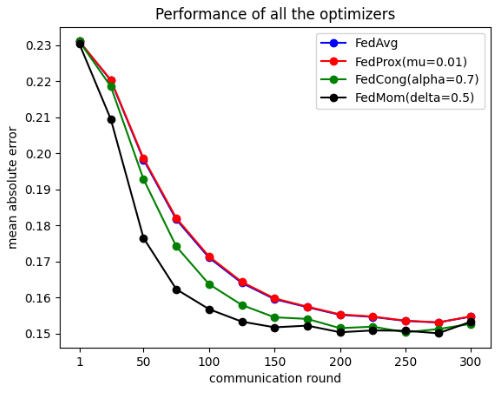
Optimizer	K	CR of stabilization	Improvements
`FedAvg`	10	250	—
`FedProx`	10	250	0%
`FedCong`	10	175	30%
`FedMom`	10	125	50%
`FedAvg`	20	250	—
`FedProx`	20	250	0%
`FedCong`	20	150	40%
`FedMom`	20	100	60%

Table 5.20: Optimizers performance, where K represents the number of clients, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

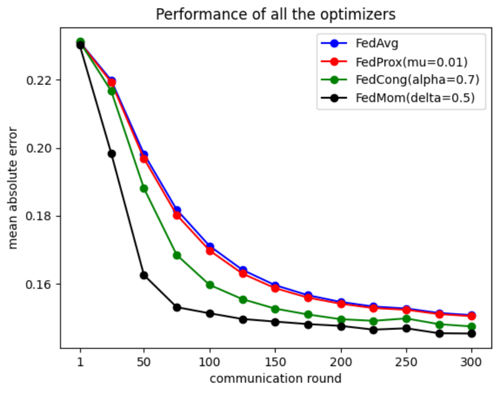
It is clear that both `FedCong` and `FedMom` improved the model's convergence speed over the baseline for $K = 20$, registering improvements of 40% and 60%, respectively.

Furthermore, the `FedCong` registered its best performance when $\alpha = 0.6$ for $K = 20$ and $\alpha = 0.7$ for $K = 10$, whereas `FedMom` registered its best performance when $\delta = 0.7$ for $K = 20$, and $\delta = 0.5$ for $K = 10$.

To conclude, it can be observed that the `FedCong` and `FedMom` performance curves are steeper in comparison to `FedAvg` and `FedProx`, with `FedMom` standing out. This means that the two new proposed algorithms converge faster than the others while maintaining the same performance, which was the objective of this experiment.



(a) $K = 10$.



(b) $K = 20$.

Figure 5.20: Performance of the different optimizers.

5.3.5 Dataset FD004

For the fourth and last dataset, the configuration of the optimizer chosen is presented in the table bellow. These configurations are fixed for all the tested optimizers.

E	CR	B	K	η
120	300	∞	10 or 20	0.01

Table 5.21: Overall configuration of the optimizers, where E represents the number of epochs, CR represents the number of communication rounds, B represents the local batch size, K represents the number of clients and η the learning rate.

Federated Averaging (FedAvg)

Table 5.22 and Figure 5.21 present the results of the `FedAvg` algorithm. Firstly, it can be observed that for both cases, $K = 10$ and $K = 20$, the error converges at the same CR, 225.

K	MAE	CR of stabilization
10	1.56×10^{-1}	225
20	1.55×10^{-1}	225

Table 5.22: `FedAvg` performance, where K represents the number of clients, MAE represents the mean absolute error and CR of stabilization represents the communication round when the error stabilized.

In addition, it can be observed that the MAE of the model for the 225 CR when the number of clients is 10 is $1.56 * 10^{-1}$, which is the around same as the one when $K = 20$, $1.55 * 10^{-1}$. The difference between the two, although small, as to do with the fact that the model when the number of clients is 20 uses a bigger batch, causing it to have a better representation of the global optimization problem.

Furthermore, it can be observed that the error value in this dataset is bigger compared to the others datasets. This might have to do with the fact that this dataset has more failures modes and operational settings, making it more difficult to recognize patterns in the data.

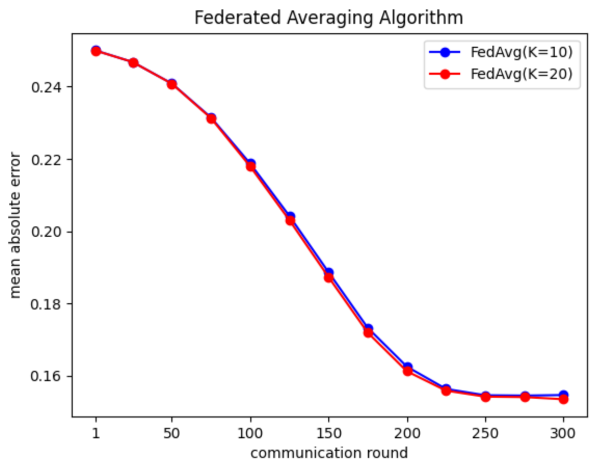


Figure 5.21: `FedAvg` for $K = 10$ and $K = 20$

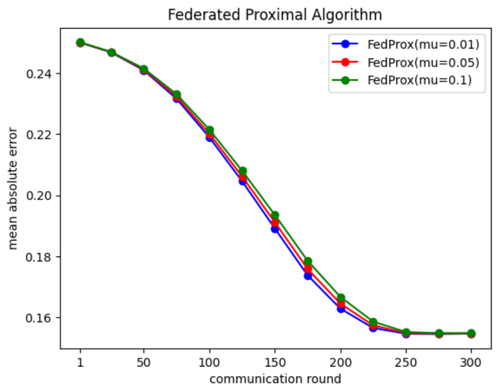
Federated Proximal (FedProx)

Table 5.23 and Figure 5.22 present the results of the `FedProx` algorithm for $K = 10$ and $K = 20$. Looking at the results, it can be concluded that all the models presented a similar or slower convergence comparing to the baseline model `FedAvg`.

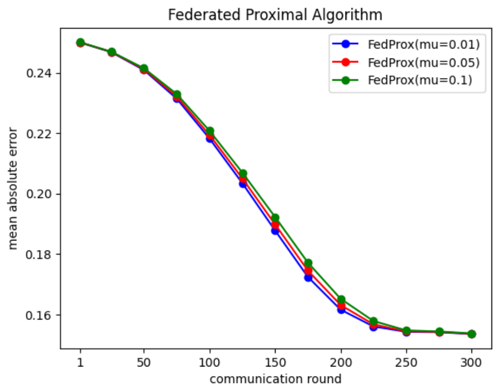
K	μ	CR of stabilization	improvements
10	0.01	225	0%
10	0.05	250	-11%
10	0.1	250	-11%
20	0.01	225	0%
20	0.05	225	0%
20	0.1	250	-11%

Table 5.23: `FedProx` performance, where K represents the number of clients, μ represents the control term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

Moreover, when $\mu = 0.1$ for both $K = 10$ and $K = 20$, the model took more time to converge, with decreases of 11% and 11%, respectively. This overall drop in convergence speed was expected since the objective of this method is to restrict the divergence between the local model and the global model.



(a) K=10



(b) K=20

Figure 5.22: `FedProx` for different combinations of μ values.

Federated Directional Congruent Learning (FedCong)

Table 5.24 and Figure 5.23 present the results of the `FedCong` algorithm for $K = 10$ and $K = 20$. Looking at the results, it can be concluded that there were improvements over the baseline.

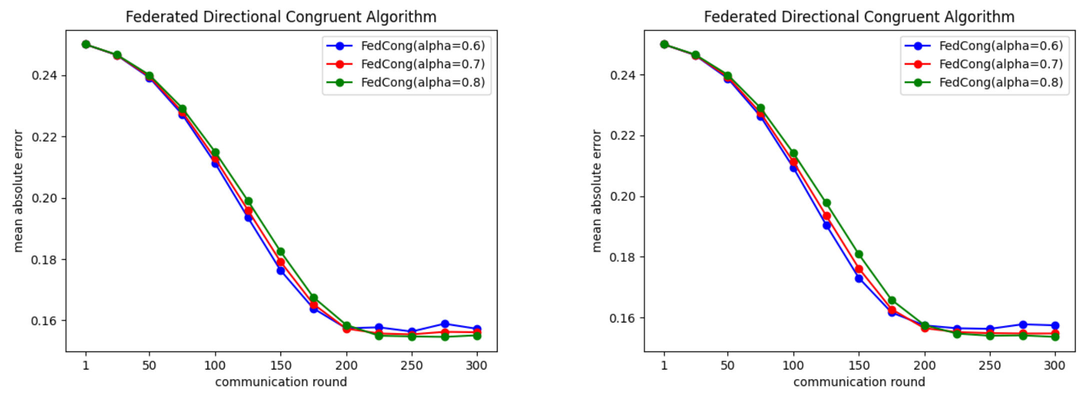
K	α	CR of stabilization	Improvements
10	0.6	—	—%
10	0.7	200	11%
10	0.8	200	11%
20	0.6	—	—%
20	0.7	200	11%
20	0.8	200	11%

Table 5.24: `FedCong` performance, where K represents the number of clients, α represents the control term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

Moreover, when the number of clients is 10, the best models are the ones where the control term is 0.7 and 0.8, having an improvement of 11%. In one case, the model did not achieve the same error as the baseline model.

Furthermore, looking at the results of the models when the number of clients is 20, it can be concluded that the best models are the ones where the control term is 0.7 and 0.8, having an improvement of 11% over the baseline. Similarly to when $K = 10$, when $\alpha = 0.6$ the model did not achieve the same error as the baseline model.

Finally, when the control term is 0.6, it is probable that the update is not representative of the global error optimization. In other words, in this case the best direction is not always the most common direction.



(a) K=10 (b) K=20

Figure 5.23: `FedCong` for different combinations of α values.

Federated Momentum (FedMom)

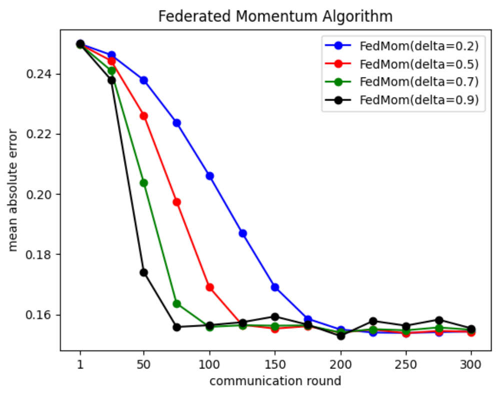
Table 5.25 and Figure 5.24 present the results of the `FedMom` algorithm for $K = 10$ and $K = 20$. It can be observed that this algorithm converged faster than the baseline for every δ value.

K	δ	CR of stabilization	Improvements
10	0.2	200	11%
10	0.5	150	33%
10	0.7	100	56%
10	0.9	75	67%
20	0.2	200	11%
20	0.5	125	44%
20	0.7	100	56%
20	0.9	75	67%

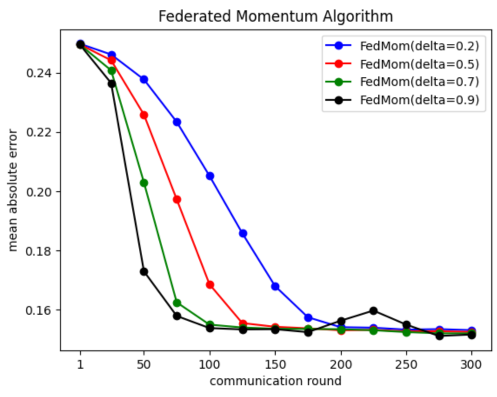
Table 5.25: `FedMom` performance, where K represents the number of clients, δ represents the momentum term, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

When the number of clients is 10, the best value of the momentum term is 0.9, although there were some fluctuations. This model had a substantial improvement of 67% over the baseline. The models where $\delta = 0.2$, $\delta = 0.5$ and $\delta = 0.7$ also presented good improvements and had less fluctuations. The reason for this is most likely due to the fact that a high δ value causes the model to diverge, giving more importance to momentum values that may have a poor representation of the global optimization problem.

When $K = 20$, it can be observed that the results are similar to when $K = 10$. In addition, when $\delta = 0.9$ the model diverged for 75 communication rounds (175 to 250). However, it still presented a more stable convergence in comparison to the model when $K = 10$. Similarly to before, the convergence curve seems more robust when $\delta = 0.2$, $\delta = 0.5$ and $\delta = 0.7$.



(a) $K = 10$.



(b) $K = 20$.

Figure 5.24: `FedMom` for different combinations of δ values.

Summary

Table 5.26 and Figure 5.25 presents the summary of the results of the different optimizers for $K = 10$ and $K = 20$.

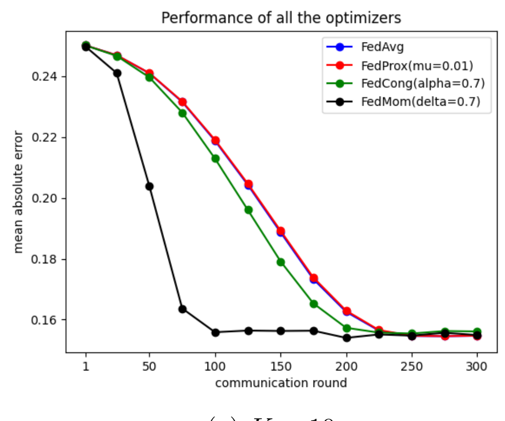
Optimizer	K	CR of stabilization	Improvements
`glsfedavg`	10	225	—
`FedProx`	10	225	0%
`FedCong`	10	200	11%
`FedMom`	10	75	67%
`FedAvg`	20	225	—
`FedProx`	20	225	0%
`FedCong`	20	200	11%
`FedMom`	20	75	67%

Table 5.26: Optimizers performance, where K represents the number of clients, CR of stabilization represents the communication round when the error reached the stabilization error of the baseline and improvements are the improvements in terms of CR over the baseline.

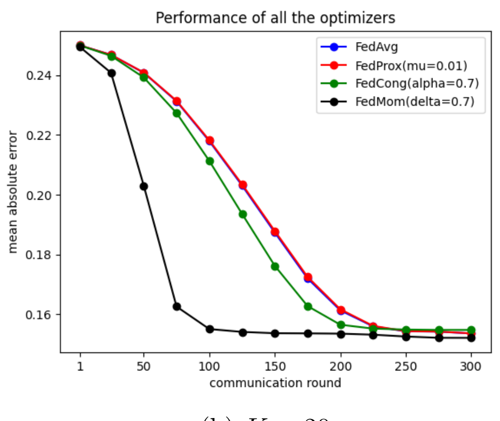
For the `FD004` dataset, it is clear that `FedMom` improved the model's convergence over the baseline, registering an improvement of 67% for $K = 10$ and $K = 20$. `FedMom` registered its best performance when $\delta = 0.9$, although it presented an error curve with less fluctuations when $\delta = 0.7$.

The `FedCong` algorithm did not perform as well as in the previous datasets. It registered improvements of 11% for $K = 10$ and $K = 20$.

To conclude, looking at Figure 5.25 it can be observed that the `FedCong` and `FedMom` performance curves are steeper in comparison to `FedAvg` and `FedProx`, with `FedMom` standing out. This means that the two new proposed algorithms converge faster than the others while maintaining the same performance, which was the objective of this experiment.



(a) $K = 10$.



(b) $K = 20$.

Figure 5.25: Performance of the different optimizers.

5.4 Summary

In this section, a summary of the experimental results of the Federated Learning optimizers will be described, namely `FedAvg`, `FedProx`, `FedCong` and `FedMom`. Table 5.27 presents the best optimizers average improvements in terms of performance compared to the baseline for $K = 10$ and $K = 20$.

Federated Averaging `FedAvg` is the current state of art algorithm in Federated Learning. This algorithm served as the baseline for the other algorithms in order to make comparisons in terms of performance and convergence rate. In all the experimentations, this algorithm converged slower than the two algorithms created in this work, namely `FedCong` and `FedMom`.

Federated Proximal `FedProx` has the objective of restricting the amount of divergence of the local model with regards to the global model. In all the experiments, this algorithm converged slower than all the others. However, `FedProx` showed less error fluctuations. In addition, the μ hyperparameter needs to be tuned in an optimal way. Otherwise, if the value is too big, then the model will converge very slowly; if it is too small, then it will not be able to restrict the divergence between the global and local models.

Federated Directional Congruent `FedCong` is one of the proposed and implemented algorithms in this work. In most of the experiments, it had a better convergence rate than all the other algorithms, with the exception of `FedMom`. In addition, when the α value is too low it sometimes caused the model to have more fluctuations, which might have to due with the fact that it has a bad error update representation. On the other hand, if the value of α is big, the model convergence rate is slower and has less fluctuations. Finally, it can be concluded that `FedCong` seems to be a good alternative to `FedAvg`, but a good choice of the α value will impact greatly the models convergence. Finally, looking at the table it can be observed that the average improvements of `FedCong` compared to the baseline for $K = 10$ and $K = 20$ were **23.4%** and **27.9%**, respectively.

Federated Momentum `FedMom` is one of the proposed and implemented algorithms in this work. In all the experiments, it outperformed all the other models by a big margin (sometimes over 50%), proving to have a better convergence rate than the other algorithms. Secondly, when the value of δ is too big, in certain situations it can cause the model to be unstable and fluctuate between error values. On the other hand, if the value of δ is too small, the model convergence rate is slower, although the fluctuations are less noticeable. Lastly, it can be concluded that `FedMom` looks to be a better alternative to `FedAvg`, although the choice of the proper hyperparameter δ is of major importance, since it will greatly influence the models' convergence. Finally, looking at the table it can be observed that the average improvements of `FedCong` compared to the baseline for $K = 10$ and $K = 20$ were **59.3%** and **62.3%**, respectively.

Optimizer	K	Average of Improvements
`FedProx`	10	0%
`FedCong`	10	23.4%
`FedMom`	10	59.3%
`FedProx`	20	0%
`FedCong`	20	27.9%
`FedMom`	20	62.25%

Table 5.27: Best optimizers performance for each K, where K is the number of clients and the Average of Improvements is the average improvements of all the datasets.

Chapter 6

Fleet Learning Demonstrator Tool

In this chapter, a software tool named Fleet Learning Demonstrator Tool (FLDT) will be described. This tool has the objective of showing the behaviour of commercial aircraft's system degradation in a real life scenario using a Fleet Learning paradigm, in this case, Federated Learning.

6.1 Description

Figure 6.1 illustrates a simplified diagram of the FLDT in which there are three main stakeholders: aircraft, airport and server.

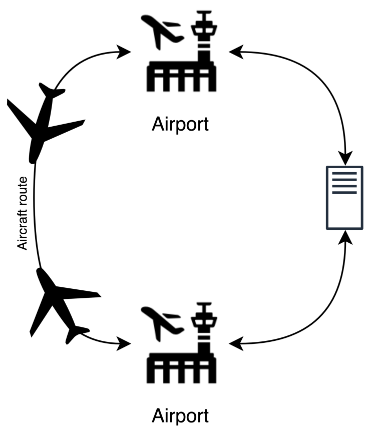


Figure 6.1: Exemplification of the aircraft tool.

The aircraft emulates the behavior of a commercial aircraft which has different systems that will degrade over time. The airport acts as a transfer point between the aircraft and the system's server. Each system is managed by a specific server that manages the models sent by the aircraft in order to generate a new updated global model.

In order for the emulation to begin, the user needs to define the number of planes that will partake in the emulation, as well as the number of aircraft systems. Each aircraft system is an equipment with multiple sensors' values (e.g. temperature, pressure) that is represented by a dataset whose structure will be described in the following section. In addition, the user can also define which local and Federated optimizer to use during the emulation.

6.2 Back-end

In this section, the FLDT back-end will be described in detail. The tool is composed of two separate working components: Flight Emulation (FE) and Server Emulation (SE). While the FE component emulates the behavior of an aircraft, the SE component emulates the behaviour of the server in a Federated Learning setting.

In the following sections the FE, SE and the data distribution will be described.

6.2.1 Data Distribution

Specific data has to be assigned to each aircraft in order to emulate each system's deterioration through the many flight hours the aircraft is subjected to.

Data has to have a discrete and sequential time frame which shows the degradation evolution. Each row of the dataset contains: 1) an unit number, u, which represents an aircraft's system identifier, 2) a life percentage, c, which indicates the life percentage of u at a given time frame, 3) N, which represent the sensors' values. Each u has multiple rows which correspond to the degradation of the aircraft's system over time.

The system's dataset units are distributed to each aircraft. In the worst case scenario, some aircraft may have an unit difference from their counterparts. In addition, the aircraft can have multiple systems.

During the emulation, each aircraft randomly chooses an unit to install. Afterwards, the aircraft will emulate a set of real aircraft routes extracted from the Bureau of Transportation Statistics [31] using the unit until it fails.

6.2.2 Flight Emulation

The main objective of the FE is to emulate the degradation of its associated systems. After the data are distributed to all of the aircraft, the FE component initiates its work.

Each aircraft has a set of real aircraft routes, with predefined departure and destination locations, as well as flight duration (expressed in cycles). The duration of each flight varies from one to eights cycles. The default value for a cycle is one second, although this value can be adjusted in order to speed up or slow down the emulation.

Every time a cycle passes, the aircraft systems update their sensors' values which are retrieved from the system's degradation dataset. Each new cycle corresponds to the next row in the unit degradation dataset. The system degrates over time until, eventually, a failure occurs. At that point, the system fails and the aircraft goes offline until it is repaired.

For each system of the Flight Emulation, there is a prediction model with a specific optimizer that is used to predict the HI at each cycle using the sensors' values. This model is defined by the user in the settings page by selecting a Python script that contains the configuration of the model and the optimizer.

Lastly, the FE component only communicates with the SE component in two different situations. Firstly, the Flight Emulation can receive a message from the Server Emulation when it receives a new model from the server. Secondly, the FE can receive a message from the SE component in order to train a model. Afterwards, when the aircraft has landed in an authorized airport, the FE sends the updated model back to the server.

6.2.3 Server Emulation

The Server Emulation works independently from the Flight Emulation and is responsible for generating the Federated model of each system using a specific optimizer and sending back the newly generated model. Note that for each aircraft system, there is a specific server emulation which handles that system's Federated model.

For each system of the Server Emulation, there is a specific optimizer that is used to generate the global model. This optimizer is defined by the user in the settings page by selecting a Python script that contains the configuration of the optimizer.

This component has two main options of operating: manual mode or automated mode. In the manual mode, the SE is at stand-by, waiting for the user's command to generate the Federated model. In the automated mode, the SE generates a model each time there is an n number of failures in each aircraft of the aircraft's system.

Furthermore, the SE also has the role of maintaining the historical data of the Federated models. This information includes data of each local model such as: the MSE and the minimum and maximum MSE of all the CR. The Server Emulation also saves information about the global model, including: the MSE and its corresponding standard deviation and the minimum and maximum MSE for each of the CRs.

Finally, the Server Emulation also calculates and saves the percentage of saved data using the Federated Learning algorithm, given by:

$$d_s = \frac{d_e}{d_t} \times 100 \tag{6.1}$$

where d_s is the data reduction percentage, d_t the data which was actually transferred and d_e is the data expected to be transferred, given by:

$$d_e = \sum_{k=1}^{K} n_k \tag{6.2}$$

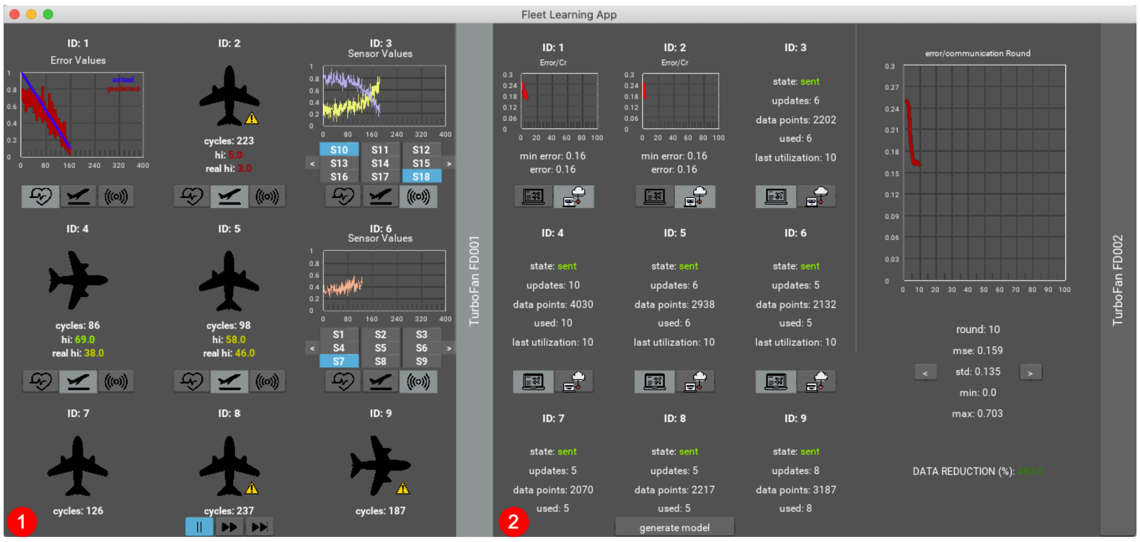
where K is the number of clients, k is the client index and n_k is the size of $k-th$ client local data used to generate the model in the current

6.3 Front-end

In this section, the front-end of the FLDT will be described in detail. The front-end allows the user of the tool to have a precise description of the Federated Learning process.

Figure 6.2 shows the layout of the tool's interface which displays real time information about the aircraft and the servers. It is divided into two grids: the Aircraft Grid, represented by the number one, in red, and the server grid, represented by the number two.

In addition, the highlighted vertical blue rectangle represents the selected system (server). In this particular case, the turbofan system was selected for visualization. The darker green vertical rectangle represents another system (server) that can be chosen to be visualized.



ID: 1
Error Values

ID: 2
cycles: 223
hi: 5.0
real hi: 3.0

ID: 3
Sensor Values

ID: 4
cycles: 86
hi: 69.0
real hi: 38.0

ID: 5
cycles: 98
hi: 58.0
real hi: 46.0

ID: 6
Sensor Values

ID: 7
cycles: 126

ID: 8
cycles: 237

ID: 9
cycles: 187

ID: 1
Error/Cr
min error: 0.16
error: 0.16

ID: 2
Error/Cr
min error: 0.16
error: 0.16

ID: 3
state: sent
updates: 6
data points: 2202
used: 6
last utilization: 10

ID: 4
state: sent
updates: 10
data points: 4030
used: 10
last utilization: 10

ID: 5
state: sent
updates: 6
data points: 2938
used: 6
last utilization: 10

ID: 6
state: sent
updates: 5
data points: 2132
used: 5
last utilization: 10

ID: 7
state: sent
updates: 5
data points: 2070
used: 5

ID: 8
state: sent
updates: 5
data points: 2217
used: 5

ID: 9
state: sent
updates: 8
data points: 3187
used: 8

error/communication Round
round: 10
mse: 0.159
std: 0.135
min: 0.0
max: 0.703

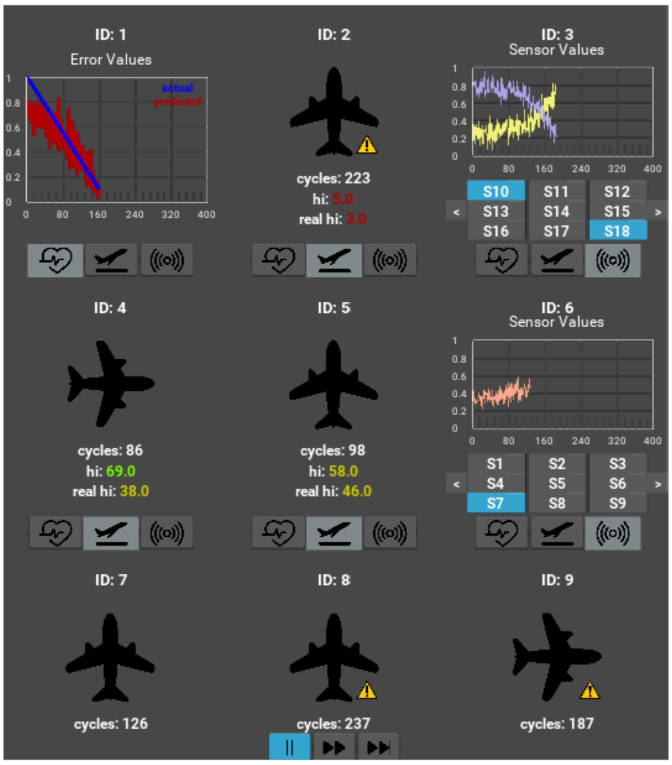
DATA REDUCTION (%):

Figure 6.2: Tool's Interface, (1) - Aircraft Grid; (2) - Server Grid.

6.3.1 Aircraft Grid

The Aircraft Grid corresponds to the first half of the front-end (1). This component has the objective of displaying detailed information about the aircraft's conditions.

Figure 6.3 shows the Aircraft Grid. In this emulation, ten different aircraft were defined where each aircraft has an associated ID number.



ID: 1
Error Values

ID: 2
cycles: 223
hi: 5.0
real hi: 3.0

ID: 3
Sensor Values

ID: 4
cycles: 86
hi: 69.0
real hi: 38.0

ID: 5
cycles: 98
hi: 58.0
real hi: 46.0

ID: 6
Sensor Values

ID: 7
cycles: 126

ID: 8
cycles: 237

ID: 9
cycles: 187

Figure 6.3: Aircraft Grid.

For each aircraft, three main windows can be chosen: Sensors Window, Aircraft Status Window and Health Indicator Window. These windows will be described in the next sections.

Sensors window

The main purpose of the Sensors Window is for the user to visualize the sensor evolution through the multiple flights routes the aircraft is subjected to. By pressing the button on the right (sensors' button), the user can choose which sensors' values are displayed.

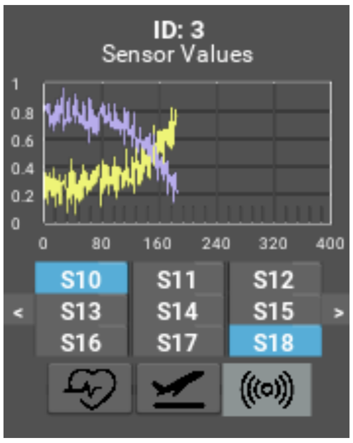


Figure 6.4: Sensors Window.

Figure 6.4 represents the sensors window for a specific airplane. The selected sensors are highlighted in blue and each one has a different color in the plot associated with it. The sensor's colors are the same between different aircraft to allow for a better comparison between the plots. This way, the user is able to better analyse the degradation of each airplane's system. Note that the sensors' values are scaled so that they can fit in the plot.

Furthermore, if the user wants to have a better visualization of the sensors' values plot, there is the possibility of pressing the plot and a new window appears which zooms the plot. Figure 6.5 presents the zoomed plot.

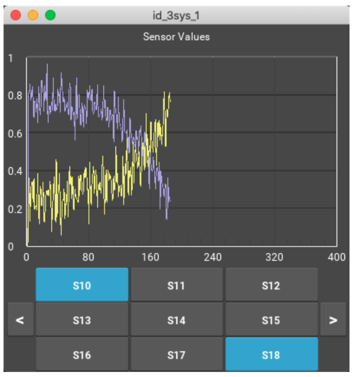
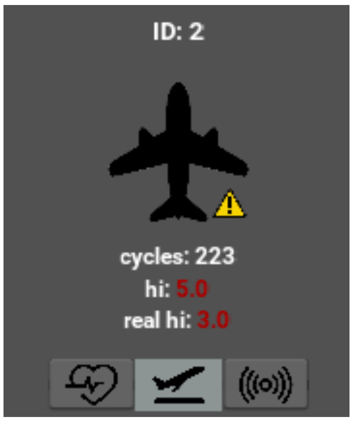


Figure 6.5: Zoomed Sensors' Window.

Aircraft Status window

The Aircraft Status Window has the objective of showing information about the aircraft's state. This includes information about the number of flight cycles without any aircraft's repair. Moreover, the health indicator's prediction and actual value is presented to the user. Figure 6.6 shows the aircraft's status window.


ID: 2

cycles: 223
hi: 5.0
real hi: 3.0

Figure 6.6: Aircraft Status Window.

Figure 6.7 shows the aircraft's flight mode. In the image on the left (a), the aircraft has a vertical orientation meaning it is in a traveling state (i.e. going from one airport to another). In the image on the right (b), the aircraft with a horizontal orientation meaning that it has landed in an airport.

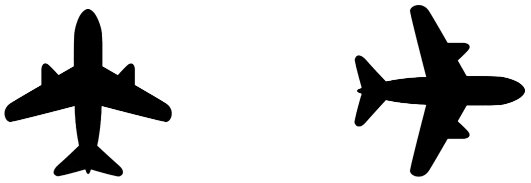

(a) Flying Plane (b) Landed Plane

Figure 6.7: Aircraft's Flight Modes.

Moreover, each aircraft has three status types that indicate the health state of that aircraft for a specific selected system: healthy state, danger state or failure statue. Figure 6.8 shows the different types of aircraft's status.

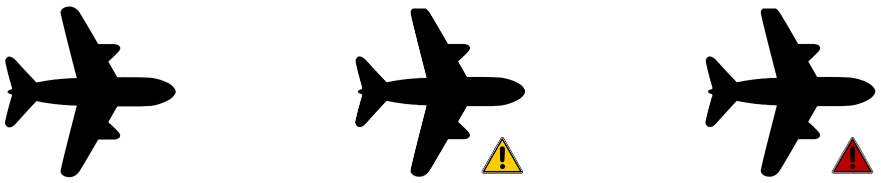

(a) Healthy state (b) Danger state (c) Failure state

Figure 6.8: Aircraft's Status Types.

Firstly, if an aircraft is at a Healthy State, it means that the current system's health indicator is above 33%. Secondly, if an aircraft is at a Danger State, it means that the system's heath indicator is above 0%. Finally, if the aircraft is at a Failure state, it means that the system's health indicator is at 0%, meaning that the system has failed.

Health Indicator Window

The Health Indicator Window has the objective of comparing the actual system's health (HI), with the predicted system's health. Figure 6.9 presents the Health Indicator Window.

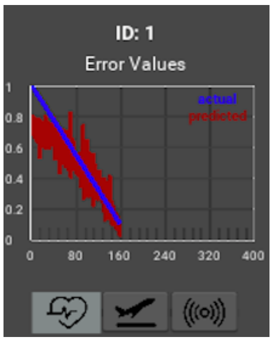


Figure 6.9: Health Indicator Window.

This window presents a plot where the blue curve represents the actual system's health and the red curve represents the predicted HI. This is specially useful to observe the performance of the local model.

Toggle Button

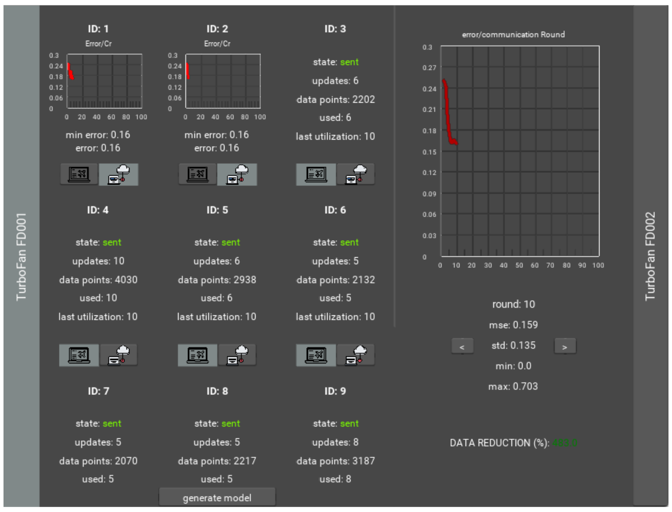
The Aircraft Grid also presents a Toggle Button in which the user can pause or speed up the emulation. When the speed is increased, the time is cut in half. In the lowest setting, one cycle corresponds to one second. In addition, when the aircraft lands it remains in the airport for one cycle.

Figure 6.10: Toggle button.

6.3.2 Server Grid

The server grid has the objective of presenting information about the models sent from the aircraft, as well as the error associated with the new global model at each communication round. Figure 6.11 shows the server grid of an aircraft's system.

The server grid can be divided into two main components: the Model Information Grid and the Communication Round Grid. These components will be described in the next sections.


TurboFan FD001

ID: 1
Error/Cr
min error: 0.16
error: 0.16

ID: 2
Error/Cr
min error: 0.16
error: 0.16

ID: 3
state: sent
updates: 6
data points: 2202
used: 6
last utilization: 10

ID: 4
state: sent
updates: 10
data points: 4030
used: 10
last utilization: 10

ID: 5
state: sent
updates: 6
data points: 2938
used: 6
last utilization: 10

ID: 6
state: sent
updates: 5
data points: 2132
used: 5
last utilization: 10

ID: 7
state: sent
updates: 5
data points: 2070
used: 5

ID: 8
state: sent
updates: 5
data points: 2217
used: 5

ID: 9
state: sent
updates: 8
data points: 3187
used: 8

generate model

error/communication Round

round: 10
mse: 0.159
std: 0.135
min: 0.0
max: 0.703

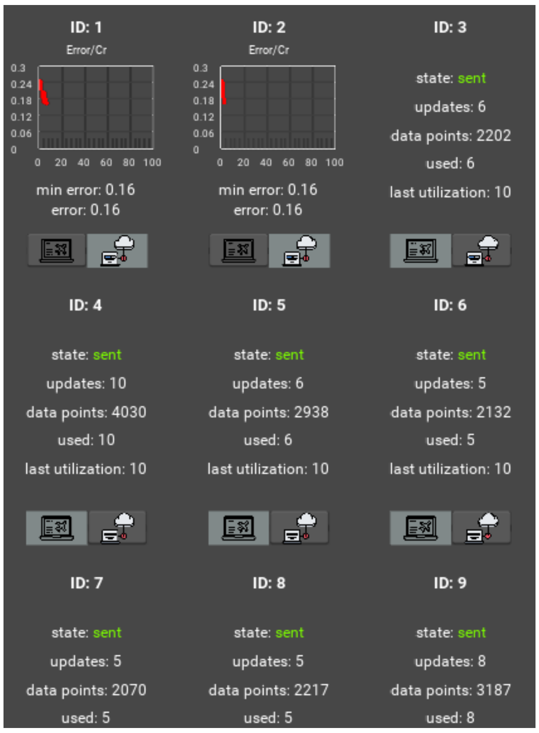
DATA REDUCTION (%): 483.0

TurboFan FD002

Figure 6.11: Server Grid.

Model Information Grid

For each aircraft system, the Model Information Grid displays information about all the aircraft's models in real time. Figure 6.12 shows the Model Information Grid of three different aircraft.


ID: 1
Error/Cr
min error: 0.16
error: 0.16

ID: 2
Error/Cr
min error: 0.16
error: 0.16

ID: 3
state: sent
updates: 6
data points: 2202
used: 6
last utilization: 10

ID: 4
state: sent
updates: 10
data points: 4030
used: 10
last utilization: 10

ID: 5
state: sent
updates: 6
data points: 2938
used: 6
last utilization: 10

ID: 6
state: sent
updates: 5
data points: 2132
used: 5
last utilization: 10

ID: 7
state: sent
updates: 5
data points: 2070
used: 5

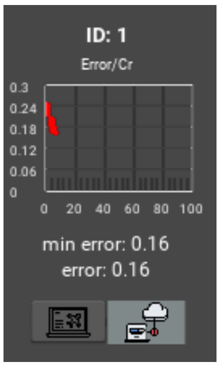
ID: 8
state: sent
updates: 5
data points: 2217
used: 5

ID: 9
state: sent
updates: 8
data points: 3187
used: 8

Figure 6.12: Model Information Grid

The Model Information Grid can be divided into two different windows: the Error Window and the Model State Window.

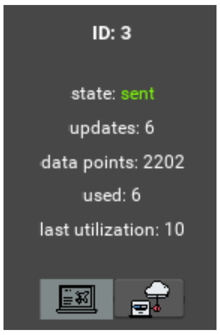
The Error Window presents information concerning the error's information about each local model. These data include the error value of the last communication round, as well as the minimum error up until that point in time. In addition, a plot is also presented which shows the error's history of that specific aircraft. Figure 6.13 shows the Error Window for a specific aircraft.



min error: 0.16
error: 0.16

Figure 6.13: Error Window

The Model State Window presents information concerning: 1) the model's state (i.e. if a specific aircraft uploaded a model), 2) the number of updates (i.e the number of times the aircraft sent information to the server), 3) the number of instances the model was trained with, 4) the number of times a model trained by an aircraft was used for the creation of the new model, 5) information concerning the last time a model was used to generate the global model. Figure 6.14 shows the Model State Window for a specific aircraft.


ID: 3

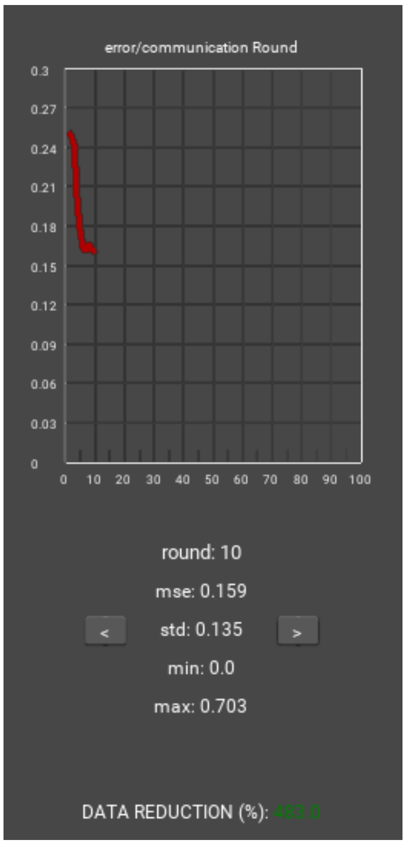
state: sent
updates: 6
data points: 2202
used: 6
last utilization: 10

Figure 6.14: Model State Window

Communication Round Grid

The Communication Round Grid, represented in Figure 6.15, presents a plot of the error of the global model. In addition, the history at each communication round's global model is also presented, including: the maximum error, the minimum error, the standard deviation and, finally, the mean squared error. Furthermore, by pressing the left arrow or the right arrow buttons, the user is able to change between communication rounds.

Finally, information about the data reduction transference is also presented. In this case, the ratio between the data sent by the model compared to the data that would be sent without using the federated algorithm is around 4.8.



round: 10

mse: 0.159

std: 0.135

min: 0.0

max: 0.703

DATA REDUCTION (%): 483.0

Figure 6.15: Servers' History

6.3.3 Settings Grid

In this section, the system's grid of the FLDT will be described. This grid is represented in Figure 6.16 bellow. In this component of the tool, three options may be chosen from the main menu: model, system and save.

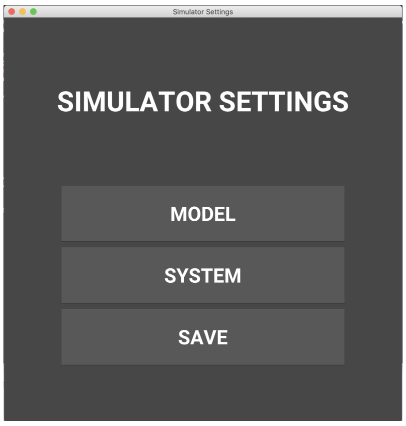
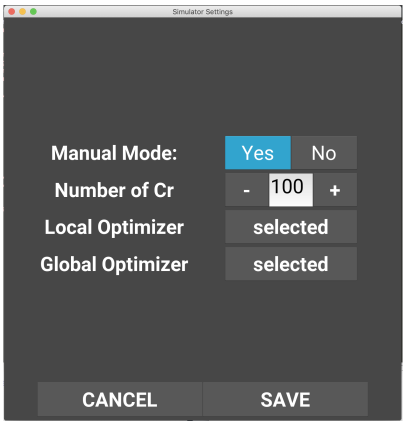


Figure 6.16: Settings Grid Main Menu

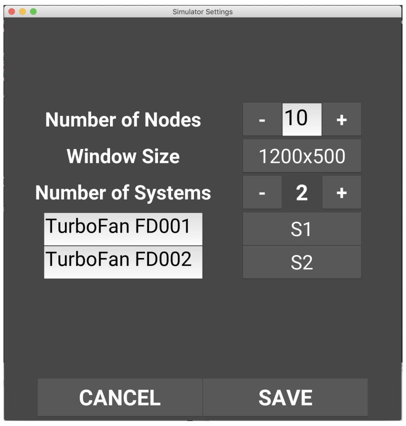
If the user clicks in the model option, the grid presented in 6.17 is shown. This allows the user to choose between the two models of flight (manual or automatic). In addition, the user can also select the number of communication rounds. Finally, the local and global optimizers can also be selected in this menu by uploading a Python class that has the model's configuration.



Manual Mode: [✓ Yes] No
Number of Cr - 100 +
Local Optimizer selected
Global Optimizer selected

Figure 6.17: Settings Grid Model Menu

By clicking the system option, the grid in Figure 6.18 is presented to the user. This window allows the configuration of the number of nodes which correspond to the number of aircraft. In addition, the window size can also be changed according to the user's preference. Furthermore, the number of systems can be configured, as well as their respective datasets.



Number of Nodes - 10 +
Window Size 1200x500
Number of Systems - 2 +
TurboFan FD001 S1
TurboFan FD002 S2

Figure 6.18: Settings Grid System Menu

Finally, by clicking the save option, the settings are saved and loaded into the Fleet Learning Demonstrator Tool.

6.4 Applications

This section describes the application of the Fleet Learning Demonstrator Tool (FLDT). The main application of this tool is to evaluate the impact of using Fleet Learning to calculate a system's HI. In particular, FLDT can be used to show the time saved by predicting the HI in mid-flight.

Furthermore, it shows how the use of Federated Learning (FL) can reduce the amount of data that needs to be sent to a server. It accomplishes this by displaying the systems current state, simplifying the system's analysis task.

Finally, FLDT the use of different FL prediction algorithms in the tool in order to evaluate their performance in terms of failure prediction and data transfer reduction.

This page is intentionally left blank.

Chapter 7

Conclusion and Future Work

The work developed can be divided into three main sections: Federated Learning with TrajecNets, Federated Learning Optimization and Fleet Learning Demonstrator Tool (FLDT). The next sections outline the conclusions of the work presented, as well as future work which can be further developed. Finally, the planning of the work is also presented.

7.1 Federated Learning with TrajecNets

This work proposed a new a PHM methodology which aims to minimize the time constraints associated with data transferring. More specifically, Autoencoders were trained in a Federated Learning setting in order to predict the RUL of an aircraft system using the work proposed by Shahid and Ghosh [44].

It was concluded that the proposed technique had an overall good performance in the training of the autoencoder. Furthermore, this method was able to minimize the time constraints associated with data transferring.

To the best of our knowledge, this is the first work which uses a Fleet Learning paradigm in an PHM methodology for Aircraft Maintenance by using Federated Learning in the training of the models.

For future work it is suggested:

Early Stopping In order to stop the models from diverging from the global model, it is suggested early stopping [8, 33]. This technique ensures that the model will stop the training process if there are no significant improvements in the global model.

7.2 Federated Learning Optimization

The second contribution of this work is the proposal and development of two new Federated Learning algorithms, namely `FedCong` and `FedMom`, and comparison with state of the art. These methods have the objective of improving the convergence speed of the Federated Learning models.

The `FedCong` algorithm tries to mitigate the impact of bad local error representations by finding the congruent clients and discarding the incongruent. It was concluded that the average improvements of `FedCong` compared to the baseline for $K = 10$ and $K = 20$ were **23.4%** and **27.9%**, respectively.

In `FedMom`, the model uses a fraction of the last update to calculate the new update, accelerating the gradient vectors in the right directions. It was concluded that the average improvements of `FedMom` compared to the baseline for $K = 10$ and $K = 20$ were **59.3%** and **62.3%**, respectively.

Although `FedCong` and `FedMom` greatly increased the convergence speed of the models, some limitations should be considered. Firstly, in `FedCong`, it is of most importance to tune the control term, α, with respect to the data distribution. If the α value is low and the current CR has a bad error representation, the global model will have difficulties in converging.

As for `FedMom`, the momentum parameter δ has to be tuned in order for the model to converge without divergence. During a model's update that has a poor representation of the global error, a large momentum term can cause the model to diverge even further and have constant fluctuations. This can cause difficulties in the convergence of the model.

For future work the following is suggested:

FedCong In `FedCong` it is possible to improve the algorithm by taking into consideration more than just the direction of the gradient descend step. Taking into consideration also the size of the step may reduce the fluctuation of the global model.

FedMom During the development of `FedMom` a similar method was published which is based on the same idea of momentum [19]. This method is similar to `FedMom`, the main difference being that it only uses the current model update to estimate the new weight value, instead of using an exponentially weighted average. We believe that if a comparison was made between the two, `FedMom` would outperform this method since the latter is more sensible recent changes which can make the model prune to more errors. For future work, we suggest making a rigorous evaluation of the method, comparing it to `FedMom`.

7.3 Fleet Learning Demonstrator Tool (FLDT)

The final contribution of this work is an emulation tool named FLDT which was developed with objective of emulating an aircraft system's degradation. This tool is able to emulate different types of Federated Learning algorithms with the objective of predicting a system's HI. Moreover, FLDT is able to calculate the improvements of Federated Learning compared to Centralized Learning.

The tool can be used to visualize the sensors' values and their fluctuation throughout the time series, giving the user a grasp of the evolving system degradation. FLDT is also able to evaluate the health condition of multiple systems simultaneously. Furthermore, FLDT can be used to test different FL algorithms.

The tool simplifies developers work due to the ability of emulating a Fleet Learning aircraft setting. Developers can make their own Federated algorithms and local models and then and apply them to FLDT. The tool handles the rest of the process by emulating the aircraft systems' degradation and the models' update.

For future work we suggest the following:

Front End FLDT was not developed following a strict design guideline. As such, the tool might not ready to be used by eventual users. It is suggested to design mockups of the tool and perform some usability tests with real potential users.

Back End FLDT does not allow the user to access some of its configurations from the front-end, such as the number of units used by each aircraft during training. In addiction, FLDT can only handle methods which predict the system's HI. Since there are several methods whose main objective is to predict the RUL or even if a failure is going to occur, it is important that FLDT handles this type of situations. Finally, it is suggested that software tests which test the functionality of the tool are developed.

7.4 Planning

Figure 7.1 shows the planned activities of all the work developed. Compared to the original planned activities, there was only one change which was the redesign of the FLDT app in May. All the other tasks were developed within the schedule.

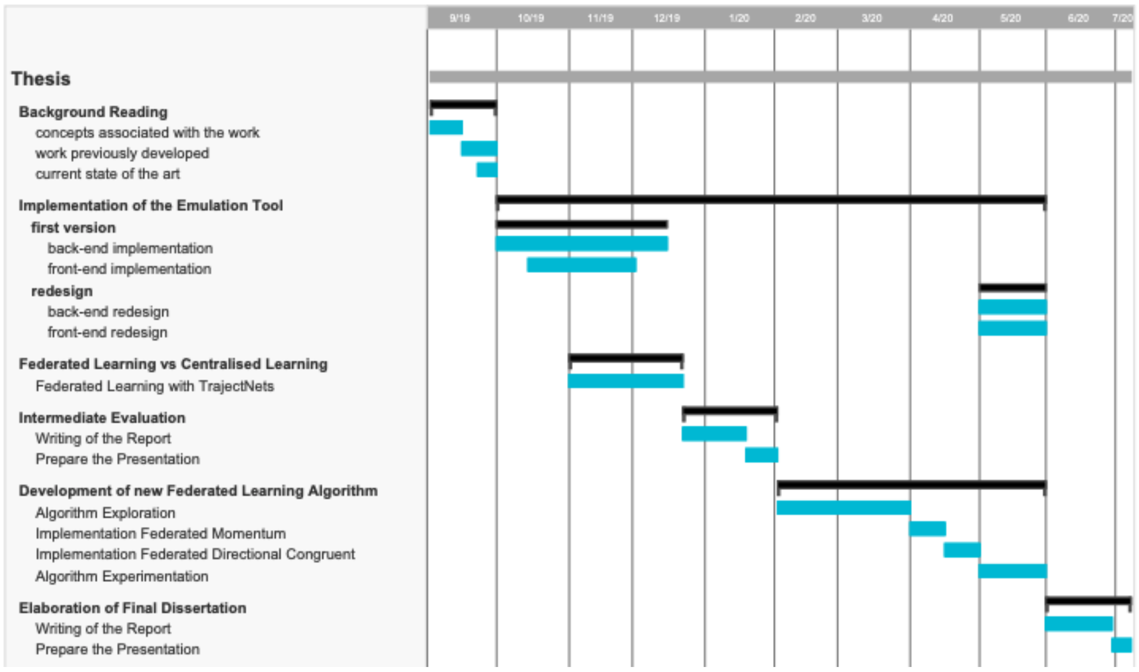


Figure 7.1: Gantt Chart

We hope that this work inspires the community to develop more Federated Learning solutions, especially in the field of PHM in AM, where massive improvements can be made in order to prevent failures. FL methods are able to train models locally and use them to generate increase the training speed of models, while maintaining the same performance. In addition, there is no need to share personal data with a server, which is beneficial in terms of security.

FL is still in the early stages of investigation and is not yet available to be deployed in large scale ventures. However, this work shows that there are improvements to be made in the FL research field which can improve its reliability.

References

[1] Prognostics and Health Management: A Review on Data Driven Approaches. *Mathematical Problems in Engineering*, 2015:793161, 2015. ISSN 1024-123X. doi: 10.1155/2015/793161. URL `https://doi.org/10.1155/2015/793161`.

[2] Boeing 787s to create half a terabyte of data per flight, says virgin atlantic., 2016. URL `https://www.computerworld.com/article/3417915/`.

[3] Farzaneh Ahmadzadeh and Jan Lundberg. Remaining useful life estimation : Review. *International Journal of Systems Assurance Engineering and Management*, 5(4):461–474, 2014. doi: 10.1007/s13198-013-0195-0.

[4] Ethem Alpaydin. *Introduction to Machine Learning.* The MIT Press, 2nd edition, 2010. ISBN 026201243X.

[5] Taiwo Oladipupo Ayodele and Yagang Zhang. *Machine Learning Overview, New Advances in Machine Learning.* 2 2010. doi: 10.5772/9374.

[6] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. Irvine, CA 92697-3435, 2012.

[7] Rui Kang Bo Sun, Shengkui Zeng and Michael Pecht. Benefits analysis of prognostics in systems. 2010.

[8] Rich Caruana, Steve Lawrence, and C. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. volume 13, pages 402–408, 01 2000.

[9] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

[10] Zhenyun Deng, Xiaoshu Zhu, Debo Cheng, Ming Zong, and Shichao Zhang. Efficient knn classification algorithm for big data. *Neurocomputing*, 195:143 – 148, 2016. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2015.08.112. URL `http://www.sciencedirect.com/science/article/pii/S0925231216001132`. Learning for Medical Imaging.

[11] Dean Frederick, Jonathan DeCastro, and Jonathan Litt. User's guide for the commercial modular aero-propulsion system simulation (c-mapss). *NASA Technical Manuscript*, 2007–215026, 01 2007.

[12] Mehryar Mohri Nathan Silberman Gideon Mann, Ryan McDonald and Daniel D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. 2009.

[13] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, May 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.393.

[14] Kai Goebel, Jose Celaya, Shankar Sankararaman, Indranil Roychoudhury, Matthew Daigle, and Abhinav Saxena. *Prognostics: The Science of Making Predictions.* 04 2017. ISBN ISBN-10: 1539074838 ISBN-13: 978-1539074830.

[15] Liang Guo, Naipeng Li, Feng Jia, Yaguo Lei, and Jing Lin. A recurrent neural network based health indicator for remaining useful life prediction of bearings. volume 240, 02 2017. doi: 10.1016/j.neucom.2017.02.045.

[16] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. 77 Massachusetts Ave, Cambridge MA 02139, USA, 2018.

[17] Daniel Ramage Seth Hampson Blaise Aguera y Arcass H.Brendan McMahan, Eider Moore. Communication-efficient learning of deep networks from decentralized data. In *In AAAI Fall Symposium*, Google, Inc., 651 N 34th St., Seattle, WA 98103 USA, 2017.

[18] F. O. Heimes. Recurrent neural networks for remaining useful life estimation. In *2008 International Conference on Prognostics and Health Management*, pages 1–6, Oct 2008. doi: 10.1109/PHM.2008.4711422.

[19] Zhouyuan Huo, Qian Yang, Bin Gu, and Lawrence Carin. Heng Huang. Faster on-device training using new federated momentum algorithm, 2020.

[20] Ruqiang Yan Jianjing Zhanga, Peng Wang and Robert X. Gao. Long short-term memory for machine remaining life prediction. 2018.

[21] Jeroen Beliën Jonas Peeters Jorne Van den Bergh, Philippe De Bruecker. Aircraft maintenance operations: state of the art. 2013.

[22] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from data, 2017.

[23] Sotiris Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica (Ljubljana)*, 31, 10 2007.

[24] Yaguo Lei, Naipeng Li, Liang Guo, Ningbo Li, Tao Yan, and Jing Lin. Machinery health prognostics: A systematic review from data acquisition to RUL prediction. *Mechanical Systems and Signal Processing*, 104:799–834, May 2018. doi: 10.1016/j.ymssp.2017.11.016.

[25] Alex Richman Leonard MacLean and Mark Hudak. Failure rates for aging aircraft. 2018.

[26] Xiang Li, Qian Ding, and Jian-Qiao Sun. Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering System Safety*, 172:1 – 11, 2018. ISSN 0951-8320. doi: https://doi.org/10.1016/j.ress.2017.11.021. URL `http://www.sciencedirect.com/science/article/pii/S0951832017307779`.

[27] Abd Kadir Mahamad, Sharifah Saon, and Takashi Hiyama. Predicting remaining useful life of rotating machinery based artificial neural network. *Computers Mathematics with Applications*, 60(4):1078 – 1087, 2010. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2010.03.065. URL `http://www.sciencedirect.com/science/article/pii/S0898122110002555`. PCO' 2010.

[28] Alex Smola Martin A. Zinkevich, Markus Weimer and Lihong Li. Parallelized stochastic gradient descent. Sunnyvale, CA 94089, 2010.

[29] Yu Zhao Athanasios V. Vasilakos Milad Makkie, Heng Huang and Tianming Liu. Fast and scalable distributed deep convolutional autoencoder for fmri big data analytics. 2019.

[30] Adrian Nilsson and Simon Smith. Evaluating the performance of federated learning. Arizona State University, Tempe, AZ 85287-5406, USA, 2018.

[31] Bureau of Transportation Statistics. `https://https://www.bts.gov/`.

[32] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do Internet services fail, and what can be done about it? In *4th USENIX Symp. on Internet Technologies and Systems (USITS'03)*, Seattle WA, USA, 2003.

[33] Lutz Prechelt. *Early Stopping - But When?*, pages 55–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_3. URL `https://doi.org/10.1007/3-540-49430-8_3`.

[34] Davy Preuveneers, Vera Rimmer, Ilias Tsingenopoulos, Jan Spooren, Wouter Joosen, and Elisabeth Ilie-Zudor. Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences*, 8:2663, 12 2018. doi: 10.3390/app8122663.

[35] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL `http://www.sciencedirect.com/science/article/pii/S0893608098001166`.

[36] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

[37] Zhenghua Chen Kezhi Mao Peng Wang Rui Zhao, Ruqiang Yan and Robert X. Gao. Deep learning and its applications to machine health monitoring: A survey. 2015.

[38] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *CoRR*, abs/1812.06127, 2018. URL `http://arxiv.org/abs/1812.06127`.

[39] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multi-task optimization under privacy constraints. *ArXiv*, abs/1910.01991, 2019.

[40] Mahadev Satyanarayanan. The emergence of edge computing. Carnegie Mellon University, 2017.

[41] A. Saxena, K. Goebel, D. Simon, and N. Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management*, pages 1–9, Oct 2008. doi: 10.1109/PHM.2008.4711414.

[42] Jurgen Schmidhuber. Deep learning in neural networks: An overview. 2014.

[43] Mark Schwabacher and Kai Goebel. A survey of artificial intelligence for prognostics. In *In AAAI Fall Symposium*, 2007.

[44] Nauman Shahid and Anarta Ghosh. Trajecnets: Online failure evolution analysis in 2d space. United Technologies Research Center, Penrose Wharf, Penrose Business Center, Cork, Ireland, 2019.

[45] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning, 2017.

[46] Jack Stilgoe. Seeing like a tesla: How can we anticipate self-driving worlds? *Glocalism*, 3, 2017.

[47] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. 09 2012.

[48] Christopher Irgens Thorsten Wuest, Daniel Weimer and Klaus-Dieter Thoben. Machine learning in manufacturing: advantages, challenges, and applications. 2016.

[49] Bogdan Trawiński, Magdalena Smętek, Zbigniew Telec, and Tadeusz Lasota. Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms. *International Journal of Applied Mathematics and Computer Science*, 22 (4):867 – 881, 2012. doi: https://doi.org/10.2478/v10006-012-0064-z.

[50] S. Thilakan U. PeriyarSelvam, T. Tamilselvan and M. Shanmugaraja. Analysis on costs for aircraft maintenance. In *Advances in Aerospace Science and Applications*, 2013.

[51] D. Wang, K. Tsui, and Q. Miao. Prognostics and health management: A review of vibration based bearing and gear health indicators. *IEEE Access*, 6:665–676, 2018.

[52] T. Wang, Jianbo Yu, D. Siegel, and J. Lee. A similarity-based prognostics approach for remaining useful life estimation of engineered systems. In *2008 International Conference on Prognostics and Health Management*, pages 1–6, Oct 2008. doi: 10.1109/PHM.2008.4711421.

[53] Weixin Wang, Qing He, Yu Cui, and Zhiguo Li. Joint prediction of remaining useful life and failure type of train wheelsets: Multitask learning approach. *Journal of Transportation Engineering Part A: Systems*, 144, 06 2018. doi: 10.1061/JTEPBS.0000113.

[54] Yiwei WANG, Christian GOGU, Nicolas BINAUD, Christian BES, and Jian FU. A model-based prognostics method for fatigue crack growth in fuselage panels. *Chinese Journal of Aeronautics*, 32(2):396 – 408, 2019. ISSN 1000-9361. doi: https://doi.org/10.1016/j.cja.2018.11.010. URL `http://www.sciencedirect.com/science/article/pii/S100093611930038X`.

[55] Quan Zhang Youhuizi Li Weisong Shi, Jie Cao and Lanyu Xu. Edge computing: Vision and challenges. In *IEEE INTERNET OF THINGS JOURNAL, VOL. 3, NO. 5*, Carnegie Mellon University, 2016.

[56] Tianbao Yang, Qihang Lin, and Zhe Li. Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization. *arXiv: Optimization and Control*, 2016.

[57] Donggeun Yoo and In So Kweon. Learning loss for active learning, 2019.

[58] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. Arizona State University, Tempe, AZ 85287-5406, USA, 2003.

[59] Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas. Federated learning based proactive content caching in edge computing. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2018. doi: 10.1109/GLOCOM.2018.8647616.

[60] Liangzhen Lai Naveen Suda Damon Civin Yue Zhao, Meng Li and Vikas Chandra. Federated learning with non-iid data. Arm, San Jose, CA, 2018.

[61] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.

[62] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan. Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2306–2318, Oct 2017. ISSN 2162-2388. doi: 10.1109/TNNLS.2016.2582798.

[63] Min-Ling Zhang and Zhi-Hua Zhou. A review on multi-label learning algorithms. *Knowledge and Data Engineering, IEEE Transactions on*, 26:1819–1837, 08 2014. doi: 10.1109/TKDE.2013.39.

[64] Hangyu Zhu and Yaochu Jin. Multi-objective evolutionary federated learning. 2019.

[65] Jinming Zou, Yi Han, and Sung-Sau So. *Overview of Artificial Neural Networks*, pages 14–22. Humana Press, Totowa, NJ, 2009. ISBN 978-1-60327-101-1. doi: 10.1007/978-1-60327-101-1_2. URL https://doi.org/10.1007/978-1-60327-101-1_2.