DataWise Project Master's Thesis Report

Adam El Youmi

Supervised by Dimitrios Panagiotakopoulaos and Weisi Guo

2020-2021

Outline

Introduction

Initial project definition

Here is the initial project definition, as formulated by NATS:

Project scope/description: The air-ground data communication links have been handling increasing amounts of data and increasing traffic in the skies communicating with the ground. This can potentially result in loss of data due to limitations of the communication channel usage capacity. Currently, the compression ratio is too small, and what needs to be explored is the optimal distribution of intelligence (federated, onboard, edge, cloud), and associated data communication channels. Therefore, research into machine learning algorithms of large/distributed datasets between the aircraft and the ground would be beneficial to manage potential operational risks. Through this project we aim to identify the best machine learning architectures to facilitate distributed data and intelligence, with the aim to reduce the overall data transfer across the air waves in an increasingly congested environment.

Objectives/outcomes: Investigate suitable machine learning algorithms which could be applied to effectively handle increasing data volumes transmitted between the aircraft and the ground to use the existing communication infrastructure optimally. The project can produce a baseline toy framework which can be used for further research in this area in an air traffic management context.

Towards a more concrete problematic

There are a lot of ways to answer such a broad problem. Subjects such as efficient database indexing and management, or reducing errors in air-ground Mode-S data transfers have been contemplated.
In this project, we want to reduce broadband usage by implementing a statistical 4D trajectory prediction method so that ground stations can infer the aircraft's position, instead of knowing it precisely through air-ground communication. We also want to try out federated learning for the training phase of the model, to see if this framework improves the quality of the training, and how it impacts broadband usage as compared to classical training.
We take inspiration from a recent work on trajectory prediction ([1]), which proposes the use of a recurrent neural network to predict 4D data points, with the help of weather data for more accurate predictions. We seek to make practical use of this method.

Acknowledgements

I would like to thank Cranfield University supervisors, Prof. Weisi Guo and Dr. Dimitrios Panagiotakopoulos, for assisting and guiding me in my research.
The feedback provided by the NATS stakeholders (Aditya Gaur, Preetam Heeramun, Mohammed Zakariyya, Sarah Dow) has also been of precious help, as it helped identifying grey areas in my approach, and guided the project in the right direction.

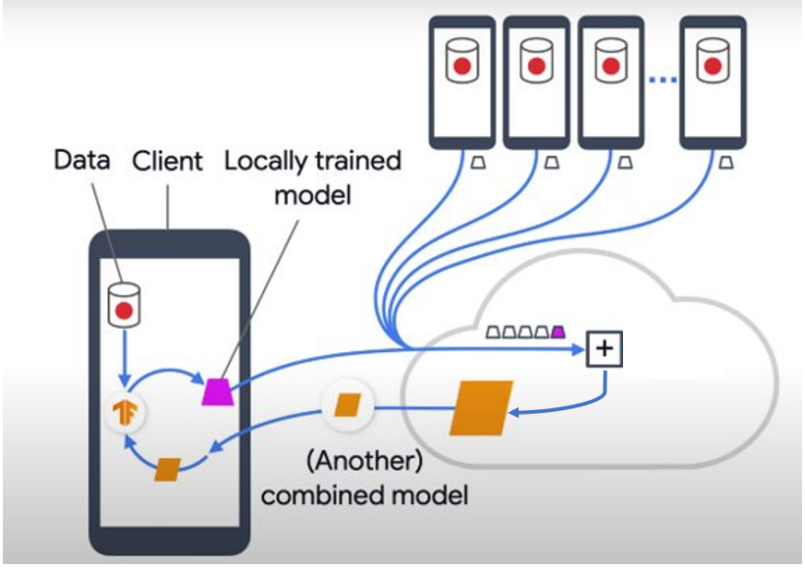
I – Literature Review

Federated Learning

Federated learning is a machine learning framework that trains an statistical model across multiple decentralized edge devices called clients holding local data samples, without exchanging them directly. This approach differs from traditional centralized machine learning techniques where all the local datasets are uploaded to one server. ([2])

Federated learning enables multiple actors to build a common, robust machine learning model without sharing data, thus allowing to address critical issues such as data privacy, data security, data access rights and access to heterogeneous data.

Today, federated learning (abv. FL) finds applications in many industries ([2]), but it is particularly exploited by tech giants, who cannot afford to centralize all the data generated by their users. For example, Google uses FL to train speech prediction models on mobile phones, laptops…



(1): Illustration of the usage of FL by Google on its mobile phones

In detail, here is how FL takes place:

- A central server and clients are organized in a “star” pattern (all the clients are linked to the central server but not to each other). In our case, a client could be an aircraft, a ground station…
- The central server chooses a statistical model to be trained by the clients, and sends the instruction to all client nodes to train a specific model (e.g. neural network, linear regression…). Most of the time, not all available clients are chosen for the training of the model. Only a fraction of them is chosen, either randomly and/or according to certain criteria
- The model is chosen according to a certain need, but can also take into account the amount of data available, an information that can be provided by clients without significant cost (*i.e.* broadband usage)
- The chosen clients train the model locally, with their own datasets

- The clients send their trained model to the central server which averages them out and sends back the resulting model. Averaging can be done by attributing a certain weight to each client.
- The new model is further trained by the clients and so on...
- This cycle repeats until a stopping criterion is met (accuracy threshold, maximum number of cycles…).

We can see how this process can be preferable to more traditional techniques in many ways:
- This framework provides much more data privacy and security guarantees, as the local data does not leave the client.
- It also takes advantage of the computing power of the clients, reducing the load on the central server (the training dataset is locally smaller, while all the data is used globally).
- In the context of personal devices, it can improve user experience, as it allows the "customization" of the model, while also taking into account data from different clients.

However, there are also downsides to Federated Learning:
- There is a risk of a bottleneck effect on the central server, as all clients must wait for the averaging and reception of the new model to proceed.
- Non-independent and identically distributed (i.i.d.) datasets can be an advantage but also a drawback, as they may distort the final model.

Moreover, the common challenges associated with FL are the following:
- Expensive communications (sending model weights tens or hundreds of times may require more bandwidth usage than just sending the training data once)
- Systems heterogeneity (clients of a different nature can possess varying computing power)
- Statistical heterogeneity (non-i.i.d. datasets are a risk. Moreover, systems of different nature may store different features from its sensors)
- Privacy in some cases (information about the training data can be inferred from weights for some specific models)

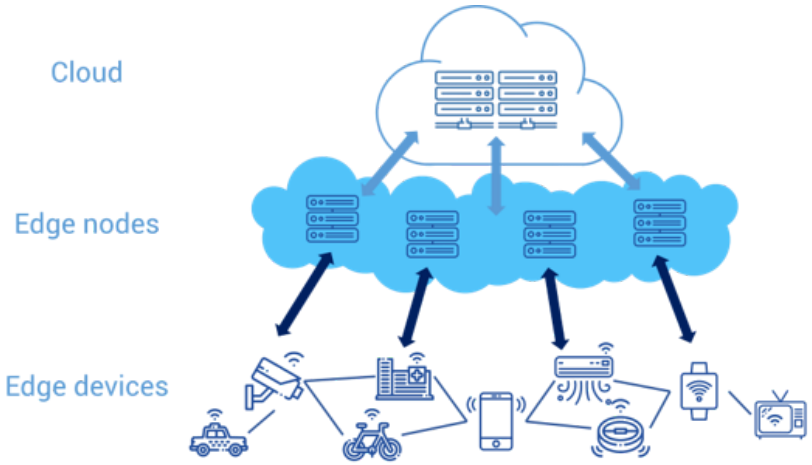
The literature proposes many improvements to federated learning to tackle these challenges: ([2], [3]) sum up the state of Federated Learning and highlight improvement opportunities for further research.

In detail, these improvements try to reduce the amounts of data transfer:
- Discretization / sparsification of weights updates
- Lossy weight compression
- Low-rank weights matrix

Edge Computing

Edge computing is a framework that brings computation capacity and data warehouses closer to where it is needed to reduce response times and bandwidth usage. In practice it consists of installing servers between the clients and the cloud server. The function of these intermediary servers is to process and reduce the data before sending it to the cloud, as well as acting as a database closer to the users, allowing shorter query response times (because there is less data to query and the message transmission phase is shorter).

Edge Computing is particularly used in the context of the Internet of Things, where sensors generate too much data for it to be processed by or even sent to a remote global server.



(2): Illustration of a system using Edge Computing

The Edge Computing framework is widely employed today, but does not seem to fit our problem, because aircrafts are very mobile. As a result, it is difficult to imagine edge nodes "close" to our aircrafts. However:

- Stratospheric Platforms technology offers 5G access to remote places thanks to UAVs. Each drone can fly for 1 week uninterrupted and covers a zone of 140km in radius. These drones can be used as edge nodes to gather and retransmit Mode-S data to ground stations and other aircrafts.
- SITA / ARINC ground stations can also act as edge nodes to reduce the data stored in airports / national databases.

Research on the Air Traffic Management System

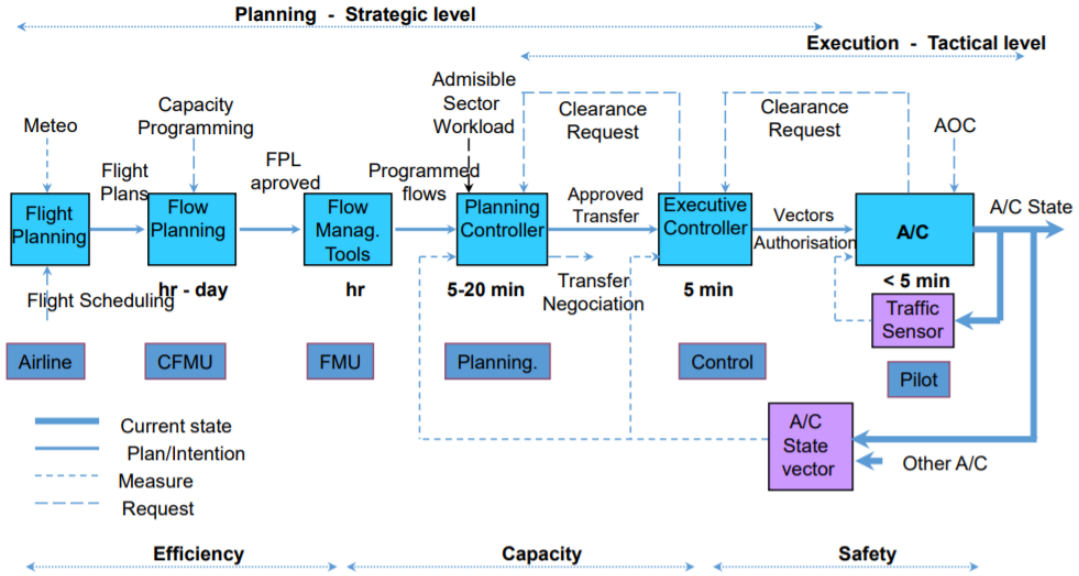
In this section we summarize the information relevant to the project regarding the organization of the civil aviation ecosystem.

Main Stakeholders

- ICAO (International Civil Aviation Organization)
  The ICAO is a specialized and funding agency of the United Nations. It changes the principles and techniques of international air navigation and fosters the planning and development of international air transport to ensure safe and orderly growth. The ICAO Council adopts standards and recommended practices concerning air navigation, its infrastructure, flight inspection, prevention of unlawful interference, and facilitation of border-crossing procedures for international civil aviation.

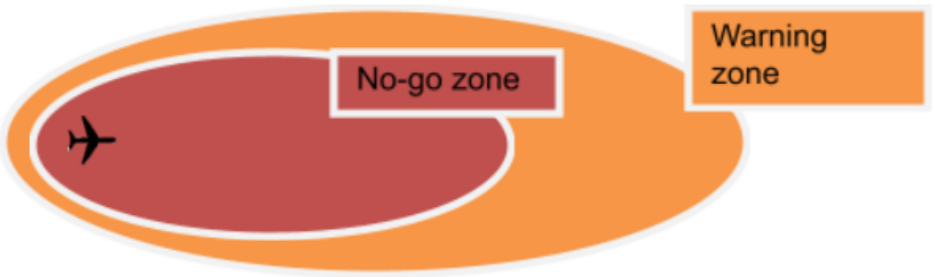
- Eurocontrol
  The European Organisation for the Safety of Air Navigation is an international organisation working to achieve safe and seamless air traffic management across Europe. Its tasks include coordination and planning of air traffic control for all of Europe.

- Operational level
  There are many more organizations that collaborate to allow aircrafts to fly safely and efficiently.



(3): Flowchart illustrating flight planning, flow management and safety protocols for one flight. Source: Provided by D. Panagiotakopoulos
The main concern long before the flight is efficiency (seeking the fastest/most economical path). As the departure approaches, the stakeholders make sure that airport and airspace capacity is respected. As soon as the engines start, safety becomes the main concern until the engines are turned off again.

Years prior, airlines communicate the flights they want to schedule to the Airspace Management (ASM), which provides relevant feedback. Days before the operation, the airline companies have to provide the expected flight plans, which is mainly determined to reduce fuel consumption and flight time. Hours before the operation, the Air Traffic Flow and Capacity Management have to coordinate the aircrafts to ensure the absence of conflict on the airport's tarmac. Once the aircrafts have begun flying, the main focus becomes safety, with the monitoring of their relative positions. There are in-flight "safety zones" delimited by an aircraft, as illustrated below:



(4): Aircrafts entering another aircraft's warning zone are prompted to modify their trajectory as soon as possible, to avoid entering the no-go zone. These zones typically measure hundreds of meters vertically and a few kilometres horizontally.

Types of communications

- ADS-B & ADS-C
  Automatic Dependent Surveillance Broadcast (ADS-B) is a surveillance technology in which aircrafts determine their position thanks to satellites or sensors. The obtained

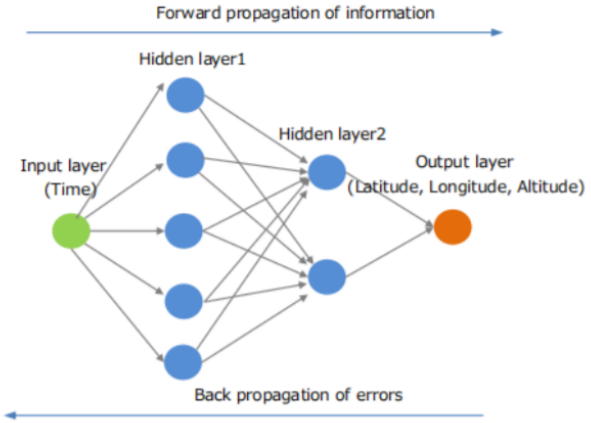
position is periodically broadcast to ground stations and nearby aircrafts. The broadcast takes place without prompting from the ground, and grants nearby aircrafts situational awareness.
Automatic Dependance Surveillance Contract (ADS-C) is a similar protocol, but it takes place between an aircraft and a specific ground station. The communication period is usually higher than ADS-B's. It is used by aircrafts travelling through regions where ground stations are scarce and where communication is more complicated.
Both are performed by the aircraft thanks to a Mode-S transponder.

- ASTERIX protocols
  ASTERIX (All Purpose Structured Eurocontrol Surveillance Information Exchange) is a standard for the transmission of air traffic information. Operated by Eurocontrol, ASTERIX is a set of universal surveillance data formats which include target reports from surveillance sensors such as radars as well as processed information such as aircraft tracks and various system status messages.

- Voice communications
  At the beginning of aviation, voice comms were the main communication channel to ground stations. Although normalized data transfers through data links have become more widespread, the ATC system still relies heavily on voice communications despite the many failures that can happen through this channel.

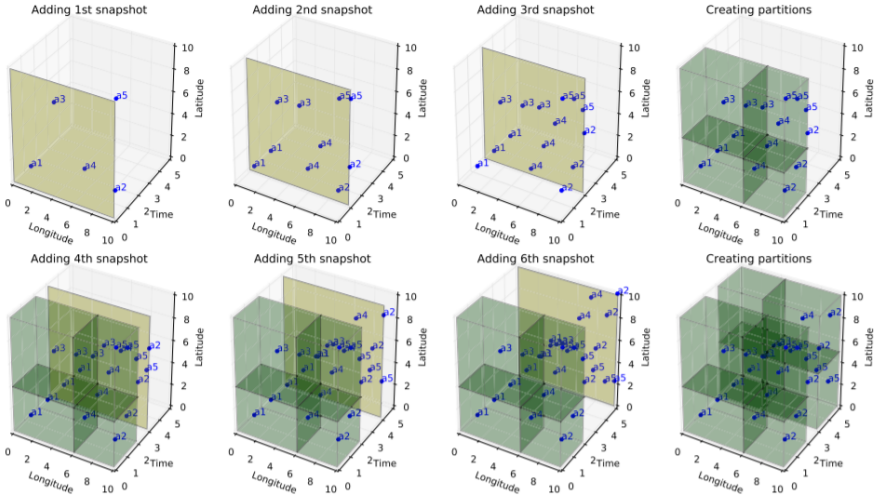
The literature proposes many improvements to the ATC system:

- In ([4]), the authors propose to use various methods to tackle the problem of errors in retrieving an aircraft's position via Mode-S communication, the most common types of error being missing, incoherent or duplicated data points. These methods include using simple dense neural networks to predict an aircraft's position based on the sole timestamp, to fill the gaps of missing data. They also use clustering (DBSCAN) to detect and remove outliers.



(5): The neural network used to predict aircraft's positions. Taken from the original paper.

- ([5]) attempts to improve the indexing and compression of the ATC databases, suffering from increasing volumes of data. The main idea is to partition an ADS-B database into a collection of spatio-temporal blocks, which allows to selectively load relevant blocks during query answering. They also develop an efficient column-based compression technique, which significantly reduces the amount of storage per block by exploiting similarities in the input data.



(6): Visualization of ADS-BI index creation. Dots represent aircrafts, yellow planes visualize snapshots and green boxes model spatio-temporal aggregations. Taken from the original paper.

4D trajectory prediction

The papers mentioned before are all paths that could be explored to come up with a data-wise reduction of the ATC system. However, the one I will be presenting in this section is the one we choose to proceed with. The improvements proposed in the other papers are further improvements that could be added on top of the solution we are trying to create.

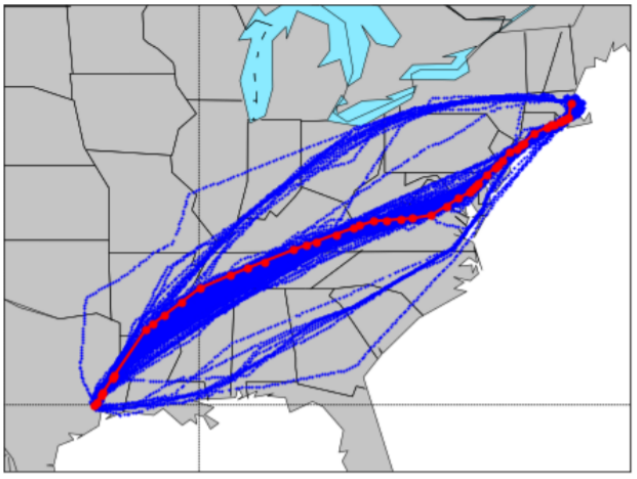
State of the art

A first document ([6]) from 1999 by EUROCONTROL linking to several papers, we can find information about the PHARE project on aircraft trajectory prediction. They explain that their goal was to consider safety constraints that may arise during the flight, while staying as close as possible to the optimal trajectory. However, the document does not provide the technical details of the algorithms used to make these predictions.

Another document ([7]) provides more up-to-date and detailed information about the concept and application of the trajectory prediction, but the specifics of the algorithm still aren't available.

Recent research

Predicting Aircraft Trajectories: A Deep Generative Convolutional Recurrent Neural Networks Approach **(Liu et al, 2018)** ([1]) is a paper using deep neural networks to predict an aircraft's trajectory.
Aircrafts all have a flight plan before departure. However, they almost never follow their flight plan, as can be seen on the illustration below.



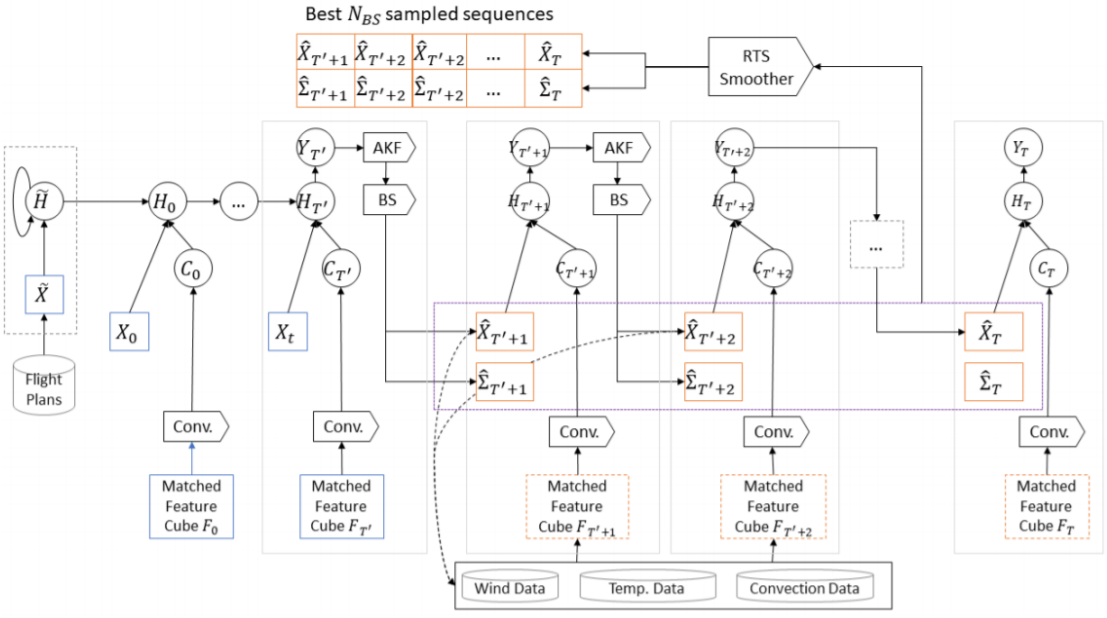
(7): The red path is the flight plan before departure, the blue paths are all the trajectories taken by the aircrafts with this flight plan. From the original paper.

One of the reasons for this is weather conditions, to which pilots and airlines are trying to react in order to improve fuel consumption or avoid turbulence. This paper attempts to take this into account to predict an aircraft's trajectory from the flight plan and meteorological data, which is of three kinds: Temperature map (cold air is denser, which allows engines to generate more power), wind vector map (tail wind is more fuel-efficient) and weather convective cells (vertical air movement should be avoided as much as possible).

Here are the technical details of the proposed neural network:

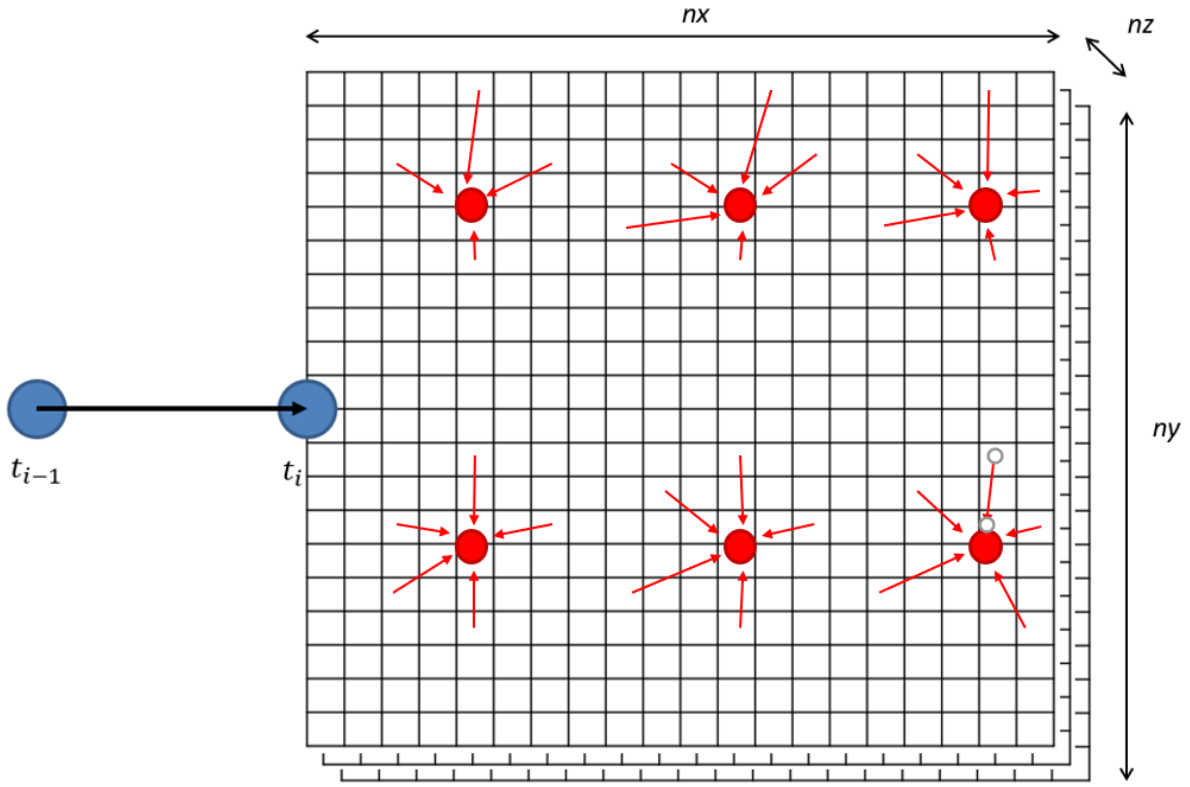
- We assume that the state of the aircraft (position & speed) at each timestamp follows Gaussian mixtures whose parameters are learned by the decoder network
- A convolutional neural network is used to embed the weather data
- A « Feature cubes grids » system is created to match the aircraft's position with the relevant portions of the weather dataset (illustrated below)
- LSTM cells use flight plans and the first 20 data points to make the predictions
- An adaptive Kalman Filter smooths the output

Here is an overview of the inference process by this network:



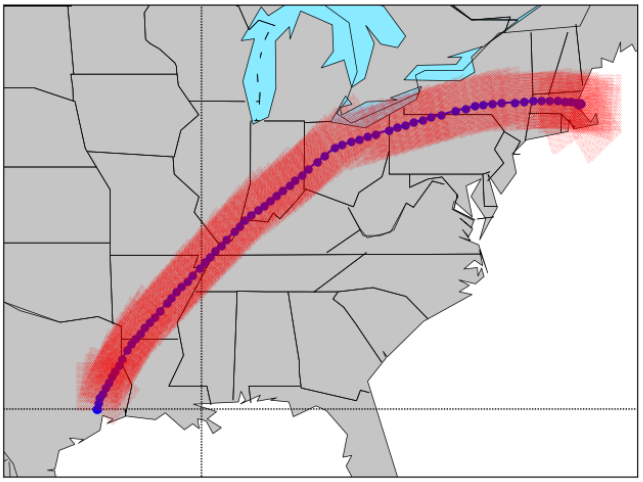
(8): Inference framework

Below is an explanation of the concept of “Feature cube matching”.



(9): Illustration of a feature cubes grid at one timestamp.

The 3D grid (*nx* by *ny* by *nz* cubes) is oriented according to the aircraft’s second last position (blue dots). We only take into account the weather in front of the aircraft. Each cube of the grid takes the value of the closest point for which we have weather data (red dots). The (four-dimensional) matching is done using a tree method to reduce computational complexity. The whole grid is embedded using a CNN for the prediction of the position at the next timestamp. One grid is created for each timestamp.



(10): Feature cubes grid path example. Source: Original paper

II – Coding

Lexicon

A lexicon for the terms used in this section:

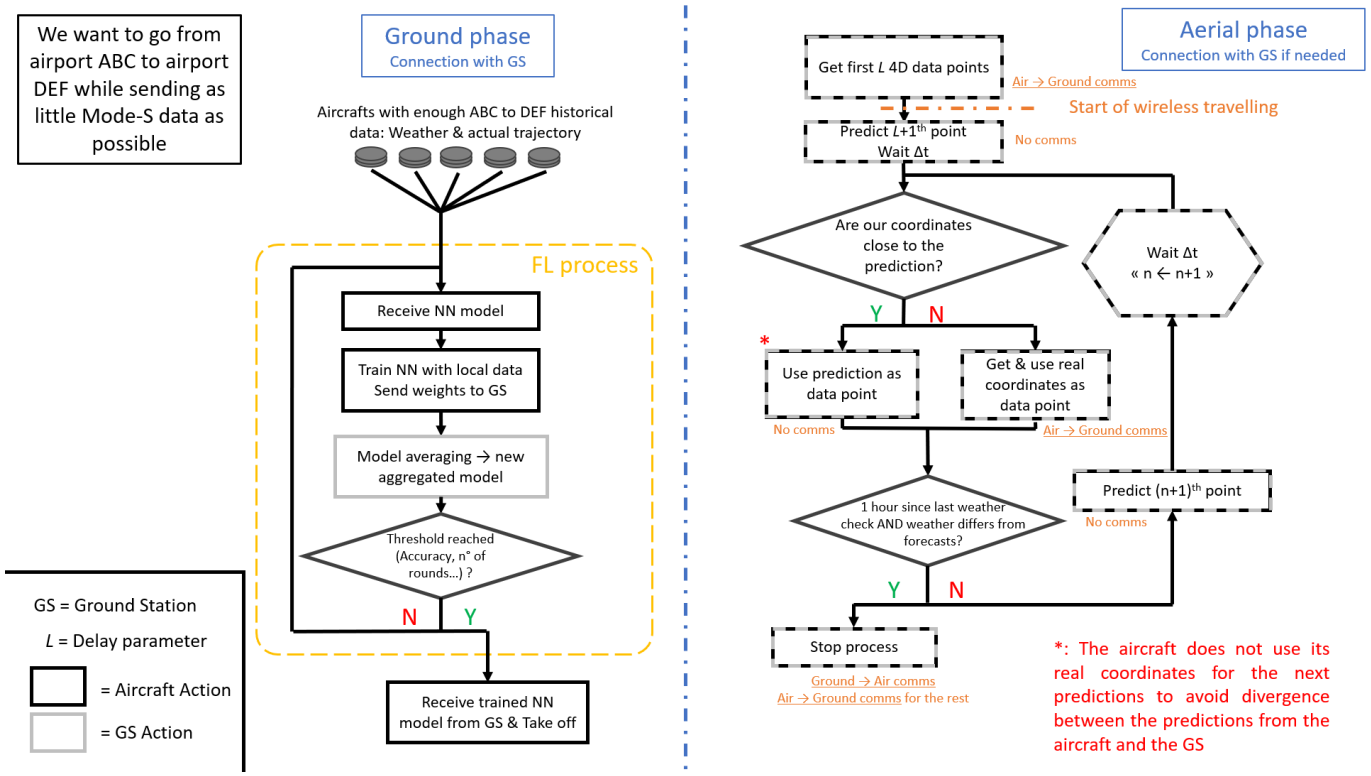
Batch: Set of observations sent forth and back through the network at once for the training
Batch size: Number of observations in each batch. The last batch contains the remaining observations if the batch size doesn't divide the total number of observations.
Epoch: Cycle of sending all batches through the network for training
[Federated Learning] Client: Aircraft that trains the network with its own data
[Federated Learning] Central server: Ground station (ex: ANSP) which collects and aggregates the networks of all clients at each cycle
[Federated Learning] Cycle: The process of training the model locally by the clients, sending the model to the central server, averaging the models, and sending it back to the clients.

Problematic to solve

We choose to abandon Edge Computing for this project, and we try federated learning to see if it improves the amounts of data transferred during the training:

- We create a model (neural network or other) that will be able to predict 4D trajectories based on weather data and flight plans.
- Both the aircraft and the Ground Station (GS) hold a copy of the network and of weather data (and forecasts) to make predictions.
- After take-off, if the flight "goes as planned", no air-ground Mode-S communication is needed. By this we mean that the aircraft is close enough to the model's prediction according to a certain threshold. In this case, if the ground station receives no ADS-B message, it means that the aircraft's position can be approximated to the model's prediction, which is then used for the next predictions. This continues as long as there is no contact between the aircraft and the ground station.
- Federated Learning is used for the model training phase.

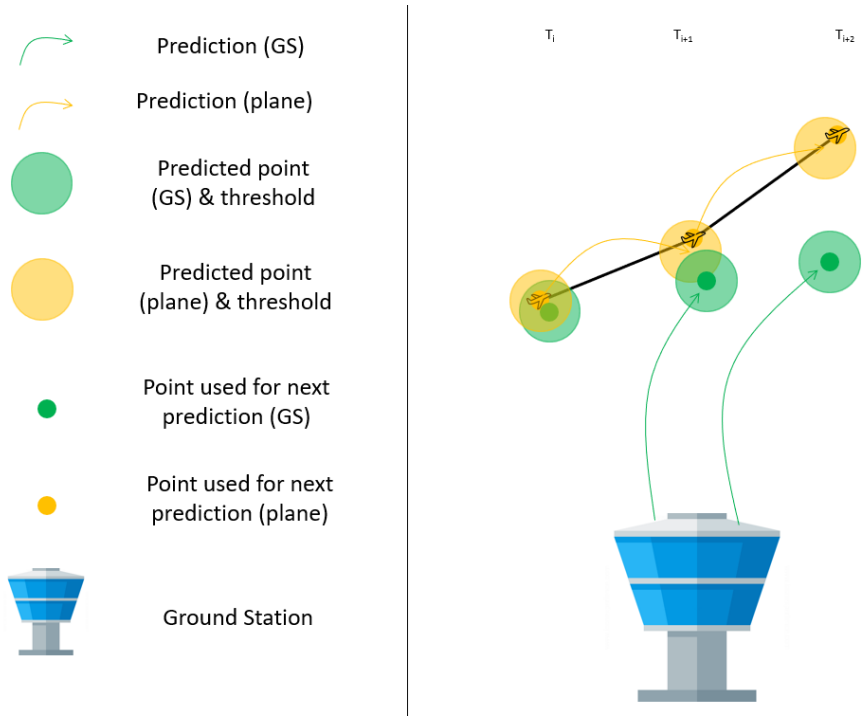
Here is the flowchart of the system, summarizing the whole process.



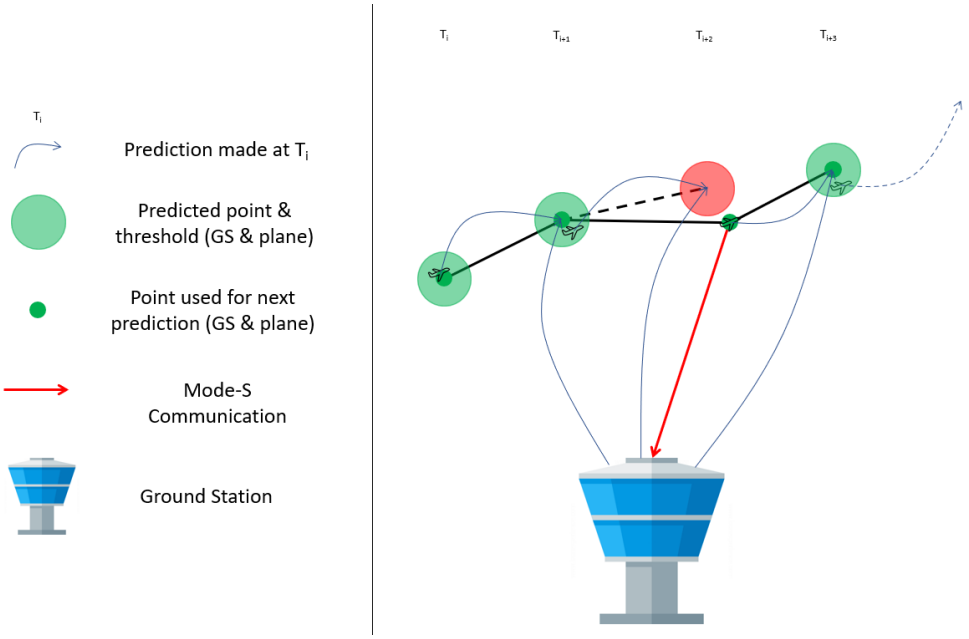
(11): System Flowchart

Since we want to reduce overall broadband usage, we have to make sure that every step of this flowchart is “data efficient”. The main concern is the “weather update” part, as weather datasets can be heavy, especially in our case. To solve this problem, we use weather forecasts for as long as possible, but if the ground station notices that every hour, the actual weather differs too much from the forecasts made at the departure, we abort communication-less travel, and resume classical mode of Mode-S communication for the rest of the flight. There is no real reason for choosing to check the weather every hour, it is rather a parameter to experiment with: try checking every 30 minutes, every 2 hours… and see how it affects the performance/cost trade-off (cost in terms of weather data storage volume required)

Below is an illustration of the *statement in red at the bottom of the flowchart:



(12): If the aircraft uses its actual coordinates for the next predictions, it can stay within its own threshold range, which does not trigger communication
→ Divergence and security hazard



(13): If the aircraft uses the model's prediction for the next predictions, the ground station and the aircraft are "on the same page", so their threshold ranges are the same, allowing no divergence.

Language & Framework

After trying to adapt the original paper's code as a starting point for this project, tests have been carried out on homemade code. The paper's code can be found on GitHub (link) and is

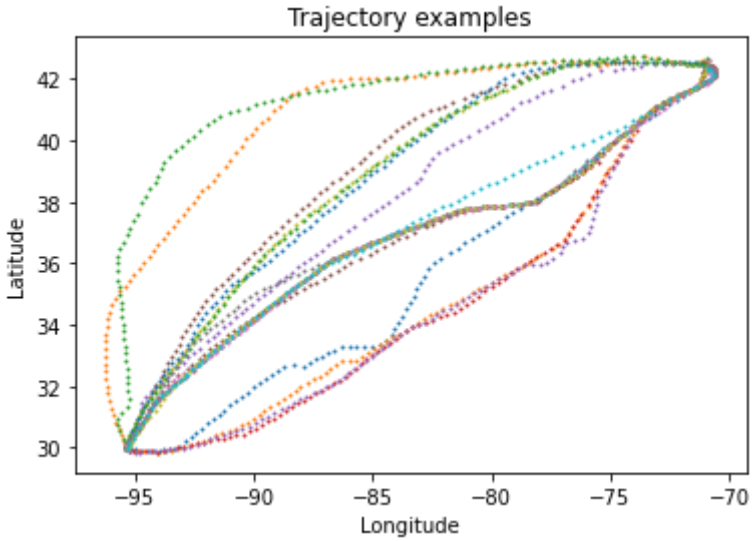
exclusively in Python. It uses Tensorflow to build neural networks and pandas for CSV manipulation.

Alternatives to Neural Networks

On NATS' request, we try to find alternatives to recurrent neural networks for time series predictions, as these are very heavy and hard to explain despite their high potential.
The most common alternative to neural networks for these tasks are linear models (Moving Average, Auto Regression and their many variants such as ARIMA), or nonlinear ones.
The documentation for a Python library implementing all of these methods can be found here: statsmodels.

Trajectory Data

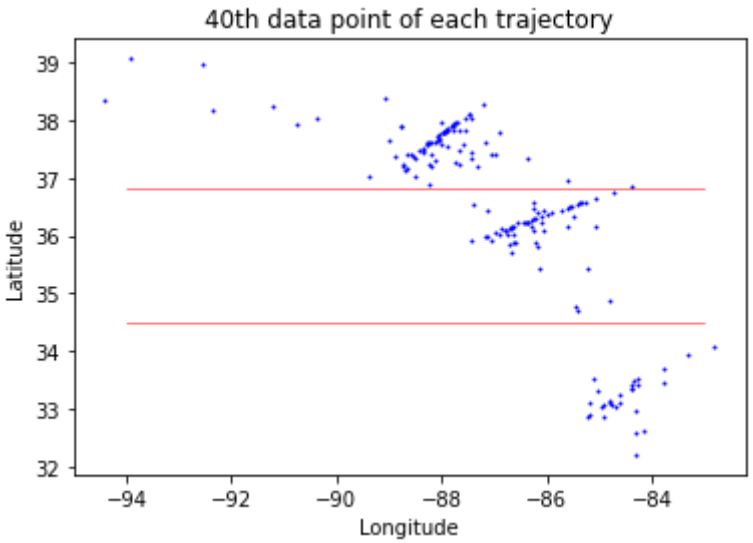
We use a set of 169 real trajectories from Houston to Boston airports. This data is available on the Github page for Liu's paper. We only keep this specific route in our training data to make it easier for the model to yield better predictions.
Each trajectory contains 80 to 120 data points (abr. DP) under the following format: [Longitude, Latitude, Altitude, Speed]. Since the NN takes fixed-sized input, we split each trajectory in segments of length *L*, *L* being the "delay" parameter.
Example: For a trajectory of 100 DPs, with L = 20, we can generate sequences made up of the DPs 1 to 20 (the observation is the DP n°21), DPs 2 to 21 (observation: DP n°22) and so on.



(14) 15 examples of trajectory data points available

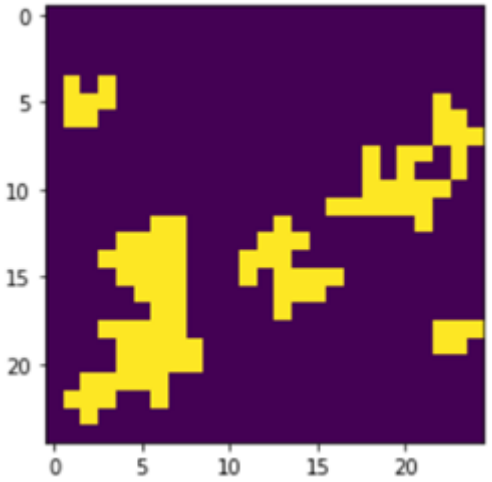
Weather Data

For the training, it would have been ideal to use actual historical weather datasets for the Boston - Houston area which correspond to the conditions the aircrafts had to adapt to during their flight. However, since weather data for a specific timestamps is difficult to find, we randomly generate weather data and expect the neural network to overfit on it. We then split trajectories into several categories and assign a weather map to each category. The criterion to decide which category each trajectory belong to is rather arbitrary: If we plot every trajectory's 40th DP, we notice that there are 3 main routes that aircrafts take:

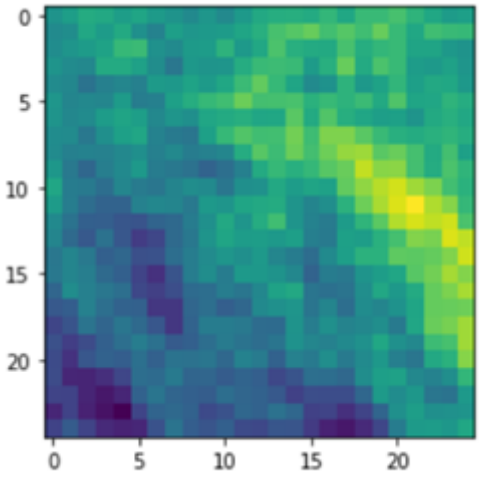


(15)

As a result, we decide to separate categories on the basis of their 40th data point's latitude coordinate, as shown by the red lines.



(16): Air convective cells (binary map)
Generated in such a way that groups are formed

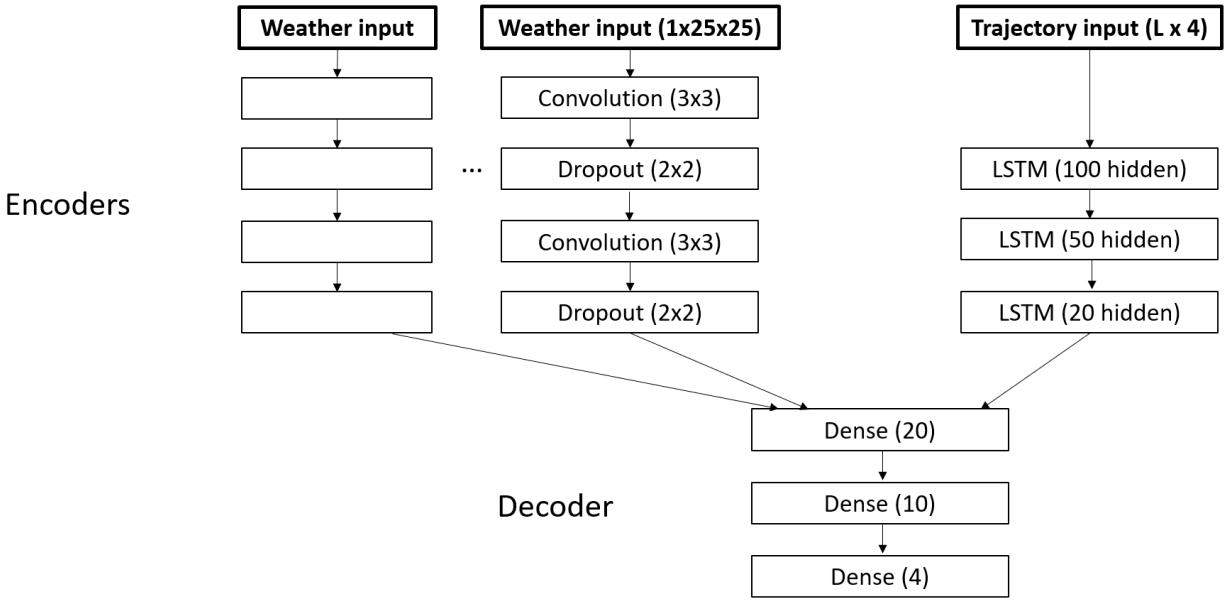


(17): Temperature map / Wind vector coordinates
Generated "continuously"

For the tests, weather maps are sized 25x25 each: Large enough to fit into the NN's encoder, but small enough to reduce computing time.

Neural Network Architecture

We use 2D convolutional layers followed by Dropout layers for weather features. For the following tests, only one weather encoder is used to reduce computation time and because of the fact that we are counting on overfitting to obtain predicting power from the weather input.
We use recurrent layers for the embedding of trajectories (LSTM layers).
The decoding is done using Dense layers.



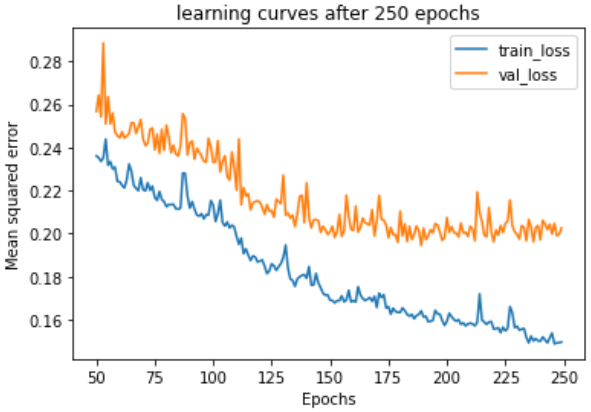
(18): Neural Network architecture

Testing & Results

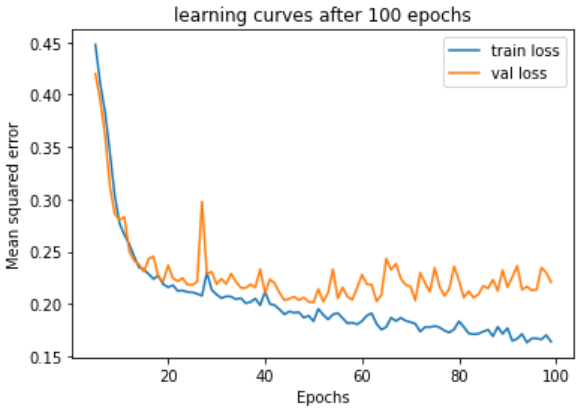
In this section, we implement the network and train it with the data generated as explained previously. Our goal is obviously to obtain a model as accurate as possible, because if the model's prediction is within the safety threshold, the aircraft will not need to communicate its position to the ground station, which knows in which range the aircraft is situated thanks to the same model, running in parallel on the ground. As a result, this will reduce broadband usage.

L = 20

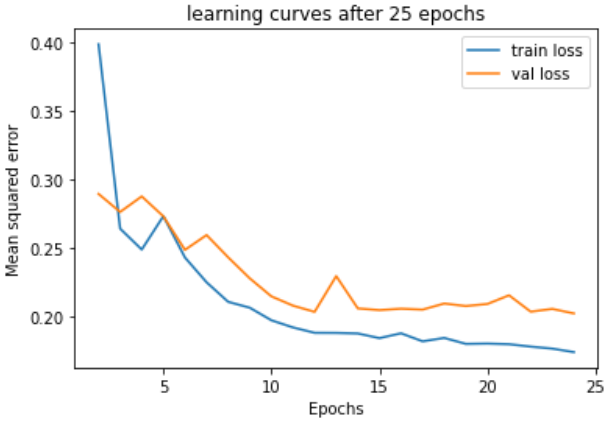
We first try L = 20, as recommended by Liu's paper ([1]). We train many networks while modifying the batch size. The loss function is the mean squared error. The plots may be truncated at the first epochs to make the other part of the curves more readable.



(19): 250 epochs, 1000 batch size
→ 850s training time

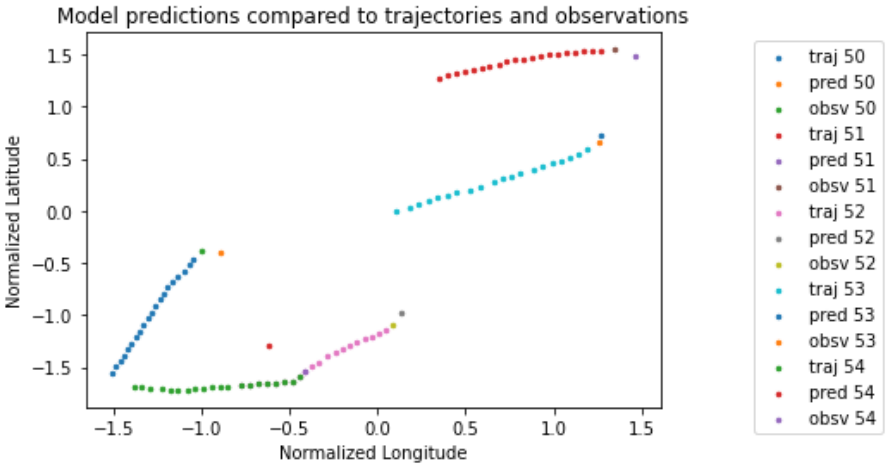


(20): 100 epochs, 500 batch size
→ 300s training time



(21): 25 epochs, 50 batch size
→ 150s training time

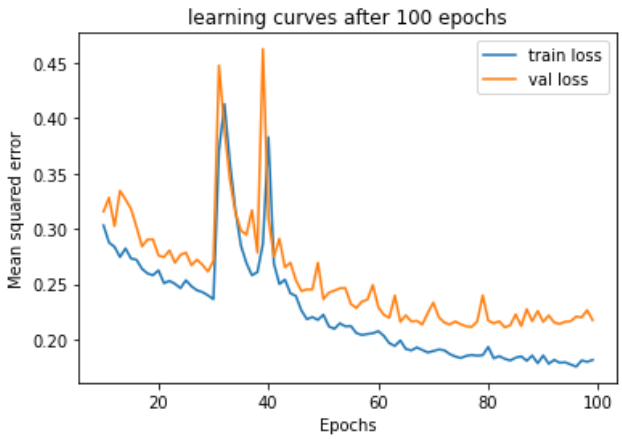
We are essentially getting the same results regardless of the batch size, in terms of optimal performance. The validation loss follows the training loss for the first epochs but eventually goes back up, while the train loss keeps decreasing. The validation loss does not go below the 0.20 mark, while the training loss decreases below 0.16. This is a clear sign of overfitting. In consequence, we stop the training, as the model is incapable of generalization: It cannot yield good predictions on unseen data.
However, we notice that lower batch sizes lead to quicker convergence to model optimality, while the risk for the loss to explode at some point is higher.



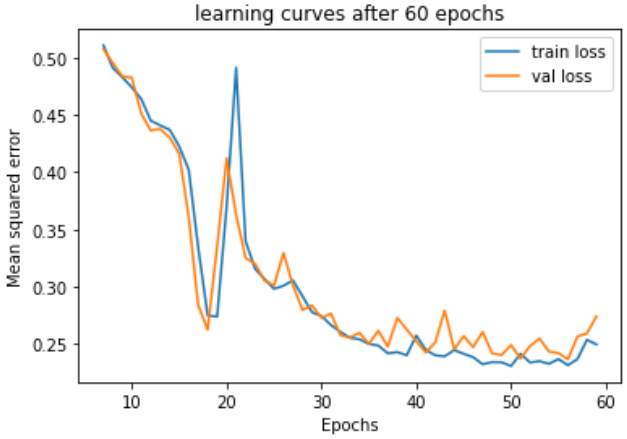
(22)

L = 30

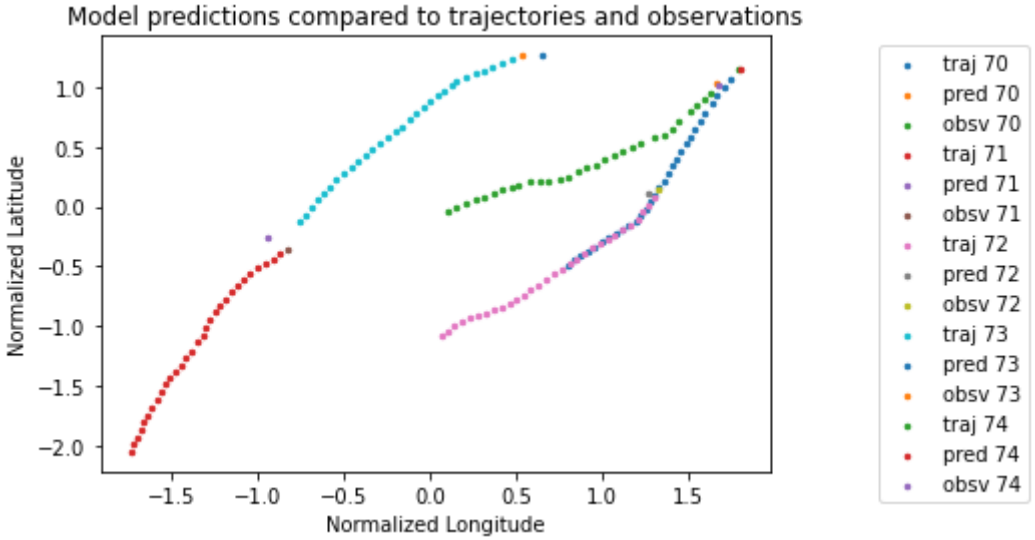
We can think that increasing the number of DPs used for the prediction will improve the prediction power.



(23): 100 epochs, 500 batch size → 370s training time

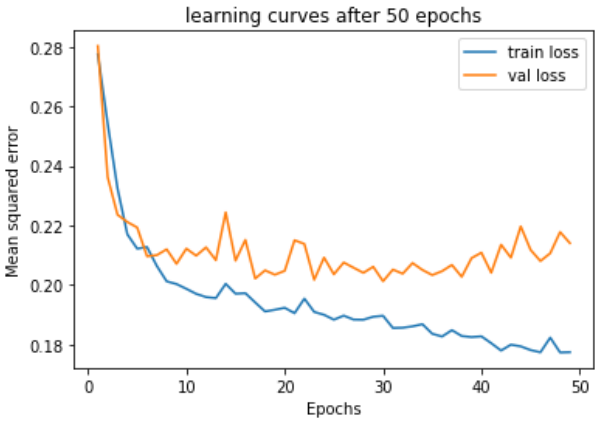


(24): 25 epochs, 50 batch size → 150s training time



(25)

With the L parameter still equal to 30, we try to increase the model complexity in order to get better results: We add more hidden states to the dense and LSTM layers. We train for 50 epochs, with a batch size of 125. After 620s of training time, we obtain the following learning curve:



(26)

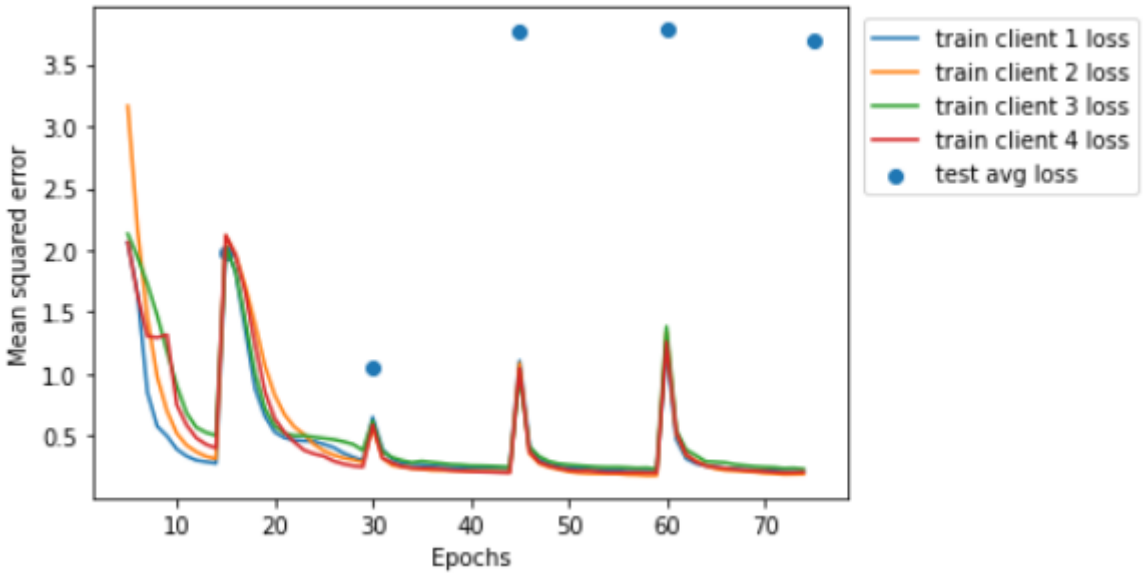
We see that results aren't any better, and that we still observe overfitting if we get past a certain number of epochs.

Federated Learning

In this section we are implementing the training of our neural network using federated learning. Because we don't have real aircrafts with their own data, we need to create them artificially and spread data among them. We choose to divide the data we have evenly among all aircrafts. For more realistic simulations, it would be preferable to spread data unevenly. This way we could analyse and resolve one of the issues associated with FL, as explained in the literature review.

Our stopping criterion is to reach a certain number of cycles. An accuracy criterion would be preferable to know more accurately the predicting power of our model, but here our model's predictions aren't good enough to do so.

For the following results, we use the following parameters:
FL cycles: 5
N° of aircrafts: 4
Epochs per cycle: 15
Batch size: 256
Delay parameter: 30



(27): Learning curves for each client and test error of the averaged model at the end of each cycle

We notice that the client models adapt well to the central server's updates during the whole training. The test loss of the central model decreases well at first, but then remains stuck at high values, which clearly shows overfitting.

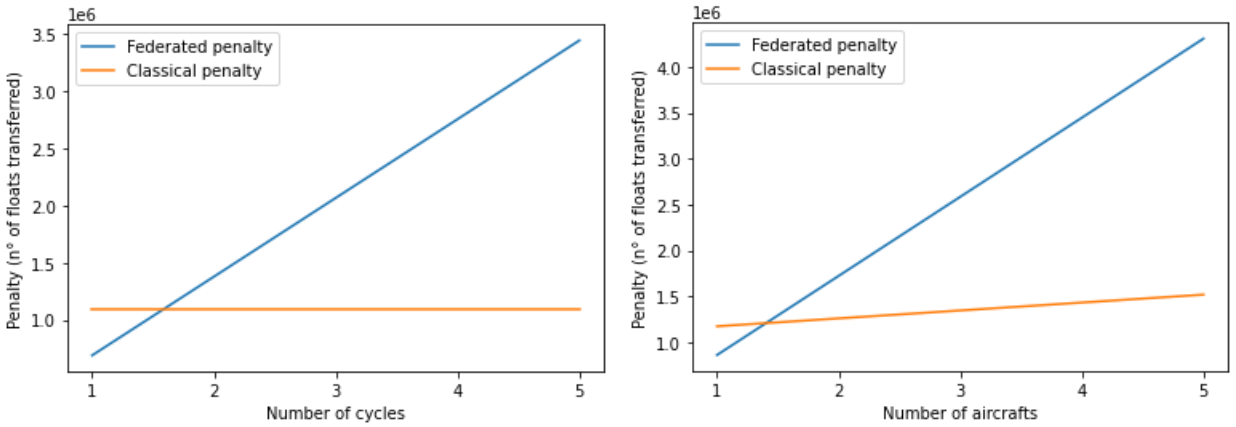
Data transfer reduction analysis

Training phase (FL *vs* no FL):
We introduce penalty functions to compare the data transfers induced by federated learning and "classical" learning (all aircrafts send their datasets to the central server). The penalty functions correspond to the number of floating-point numbers that would be sent between the aircrafts and the central server over the whole training process. They are calculated with the following formulas:

$$penalty_{FL} = n_{cycles} * 2 * n_{aircrafts} * (model\ size)$$

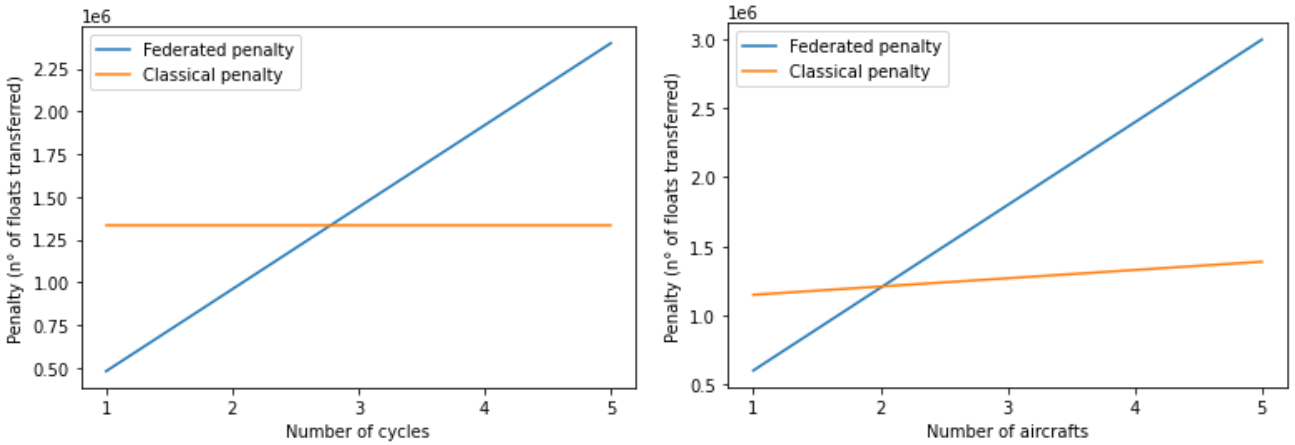
$$penalty_{classical} = n_{aircrafts} * (model\ size) + \sum_{i=1}^{n_{aicrafts}} (dataset\ size)_i$$

- FL penalty: Every cycle, the model is sent at the end of the training from each aircraft to the central server, and then sent back (the aggregated version), hence the "2" factor.
- Classical penalty: Each aircraft sends their dataset once, to the central server (sum term on the right). Once trained, the model is sent to every aircraft.



(28): Both penalties as a function of the number of cycles and aircrafts
y values in millions

We can see that the FL framework becomes less preferable above one training cycle.
We try to simplify the model in order to reduce the number of weights in our model. The LSTM layers' weights make up more than 90% of the model size. We remove the second LSTM layer and observe the change in performance as well as in the penalty plotted above:
We obtain slightly worse predictions (10% higher loss at best performance), but the model weight is almost divided in half. We get the following plots:



(29): Same plots after removing the second LSTM layer

We see how the FL framework can become preferable as the model reduces in size.

Still, with the initial parameters and models, the FL costs about 3 times more than classical training.
This can be improved by reducing the number of model parameters while keeping good predictions, but for this we would need good predictions first.

Flight phase

Let's first say that the results obtained during these tests will always be optimistic when compared to a real implementation, because of the fact that we assume that the weather conditions remain the same over the whole flight, so the contactless flight cannot stop because of inaccurate weather forecasts (bottom right of the flowchart).
We introduce a function allowing to determine whether the prediction is close enough to the actual coordinates:

```
def dist(lon1,lon2,lat1,lat2,alt1,alt2): # Function to calculate the distance in km based on coordinates and altitudes
    a = 2*pi/360 # Conversion from degree to radian

    #Trigonometry formula for the distance in km regardless of altitude
    d_flat = 6371 * np.arccos((np.sin(a*lat1) * np.sin(a*lat2)) + np.cos(a*lat1) * np.cos(a*lat2) * np.cos(a*lon2 - a*lon1))

    d_h = alt1-alt2 # Vertical distance (in km)
    return(np.sqrt(d_flat * d_flat + d_h * d_h)) # Pythagoras theorem for the final distance
```

We then simulate the flight conditions: For a given trajectory, we use consecutive sequences of data points for the predictions, and we determine if the spatial distance is within a certain range.

We find that predictions are too inaccurate to provide useful value yet.
We find that, with a threshold of 200 km (way too much for safety), the data reduction remains below 10%
→ The data link must be used almost systematically.

Another remark on practical implementation

When considering a practical implementation of our system, we also need to take into account hardware constraints: The computers installed on commercial flights are much weaker than common personal computers, and their storage capacity (for datasets or models) is very limited as well. Such board computers may be able to run the proposed neural networks (provided the necessary storage space and software is available), but probably not train it.

However, UAVs, which are the original motivation for this project, are often equipped with stronger computers, such as Raspberry Pi's and equivalents, which have proven to be able to train and run some neural networks easily ([8], [9])

III – Timeline

Organization

The project started officially on 08/04/21 with a kick-off meeting with NATS and Cranfield University stakeholders. We then met about every week with Dr. Panagiotakopoulos and every two weeks with Pr. Guo to show them my advancement. The most important for this first part was to make sure that the project was well understood and to find a problem to solve. After a catch-up meeting with NATS on 24/05/21, we used their feedback and interrogations to prepare the next meeting on 01/07/21 ; in the meantime, I familiarized myself with the code proposed by the main paper's author.
After noticing that understanding and adapting the existing code would take too much time, I started writing the code from scratch. This includes:

- Extracting and processing the trajectory data
- Generating the weather data
- Building, training and testing the neural network
- Simulating a federated learning framework
- Simulating the flight phase
- Calculating penalty functions to compare data reduction (FL vs no FL; flight phase)
- Creating various plots for visualizing outputs

Conclusion

Summary

After reviewing many papers to find which angle we were going to attack the broad problem of data handling in the Air Traffic System, we chose to go for four-dimensional trajectory prediction. For this, we designed a complete framework allowing ground stations and aircrafts to coordinate to train and use an artificial neural network to reduce the amount of Mode-S data transferred during the flight. The application didn't yield convincing results (poor prediction capacity ; federated learning framework less efficient than traditional techniques). Nonetheless, it gave us a chance to implement techniques covered during Cranfield University's AAI MSc, as well as other concepts such as federated learning.

Advice for further research

Since we ran a bit low on time towards the end of this project, here are some ways in which further research could be conducted on the proposed system:

- Try to make the model more complex to have it yield better predictions, then try to reduce it by reducing existing layers' sizes or removing useless layers
- Replace mock weather data by real data if available ; add more trajectories of the same path if available
- Try to determine how much we can reduce training data to reduce the penalty for classical training
- See if boosting (averaging of the predictions of many weak models) leads to better predictions
- Add to the 4D trajectory prediction method other improvements exposed in the literature review section.

References

[1]: Predicting Aircraft Trajectories: A Deep Generative Convolutional Recurrent Neural Networks Approach, Liu et al, 2018
[2]: Federated Learning: Challenges, methods, and future directions, Li et al, 2019
[3]: Federated Learning: Strategies For Improving Communication Efficiency, Konecny et al, 2016
[4]: Efficient and Effective 4D Trajectory Data Cleansing, Tan et al, 2020
[5]: ADS-BI: Compressed Indexing of ADS-B Data, Wandelt et al, 2018
[6]: Trajectory Prediction, EUROCONTROL, 1999, link
[7]: EUROCONTROL Specification for Trajectory Prediction, EUROCONTROL, 2017, link
[8]: Article reporting the use of Raspberry Pi to train and run a depth-estimation ANN, 2021, link
[9]: Article exposing the performance of various Raspberry Pi's on running ANNs, link