Predicting congested areas at tactical time with recurrent neural networks

Loïc Shi-Garrier

1 Introduction

With the continuous rise in air transportation demand (a 4.3% annual growth of the worldwide passenger demand is expected until 2035 [14]), the capacity of the Air Traffic Management system reaches its limits, leading to increased flight delays (expected to achieve 8.5 minutes per flight in 2035 with the current system [35]). From the air traffic controllers perspective, potential converging proximity are strongly demanding, because they have to constantly monitor numerous trajectories to decide if they should act on them or not.

In a time horizon lower than 20 minutes, existing Medium Term Conflict Detection (MTCD) tools provide efficient conflicts detection. The objective of this work is to apply machine learning methods to provide congestion predictions in an extended time horizon, between 40 to 60 minutes. For this time horizon, the uncertainties on the trajectories make it difficult to predict the exact trajectories that will be involved in a conflict. This is why we will focus on predicting congested areas rather than individual trajectories involved in conflicts. Congestion appears when a set of trajectories strongly interact in a given area and time interval, leading to potential conflicts and hence requiring high monitoring from the controllers. The aim of predicting congested areas is to provide a decision making tool for the Flight Management Position (FMP), providing information to help taking actions at tactical time (less than one hour) such as rerouting or sector configurations. The machine learning advances over the last decades have already been successfully applied in various problems of Air Traffic Management, for example to predict estimated take-off times [4], or to detect controllers' actions [25]. In this application, the objective is to make predictions by using sequences of aircraft states (trajectories). Recurrent Neural Networks (RNN) are well suited to deal with sequences of data, and this is why they have been chosen to build a first prediction model for congested areas. In order to quantify the notion of congestion and to design a supervised learning task, a measure of the air traffic complexity is required. This will be achieve by using a complexity metric already developed in the literature.

In section 2, we present the related works on complexity metrics and introduce the basic concepts of Recurrent Neural Networks. Section 3 describes the dataset and the proposed model. Section 4 presents some preliminary results, including fine-tuning of the model, and a first insight on its predicting performances. Finally, section 5 provide several options to design a more efficient model and to evaluate it, as well as new applications that could benefit from it. These options will be further investigated during the internship.

2 Literature review

In this part, we will first conduct a literature review presenting a sample of the state-of-the-art complexity metrics for Air Traffic Management. Then, a brief introduction to recurrent neural networks is provided.

2.1 Complexity Metrics

In this section, we will briefly review the main approaches to complexity metrics developed in the literature. Extensive reviews on this topic can be found in [28]. The literature focusing on conflict detection, trajectory prediction, and estimation of conflict probability is not addressed in this review.

The traditional measure of the air traffic complexity is the traffic density, defined as the number of aircraft traversing a given sector in a given period [12]. The traffic density is compared with the operational capacity, which is the acceptable number of aircraft allowed to traverse the sector in the same time period.

This crude metric does not take into account the traffic structure and the geometry of the airspace. Hence, a controller may continue to accept traffic beyond the operational capacity, as well as refusing aircraft even though the operational capacity has not been reached.

The Dynamic Density (introduced in [19]) aims at producing an aggregate measure of complexity by combining complexity factors, including static air traffic characteristics (such as airway crossings) and dynamic air traffic characteristics (such as the number of aircraft, the closing rates etc.). A list of these complexity factors can be found in [12]. The complexity factors can be combined either linearly (for example [33]) or through a neural network ([1], [2]). Several versions of Dynamic Density have been proposed in the literature ([18], [22]). For example, the interval complexity (introduced in [6]) is a variant of Dynamic Density where the chosen complexity factors are averaged over a time window.

Another approach is the Fractal Dimension (introduced in [23]). Fractal Dimension is an aggregate metric for measuring the geometrical complexity of a traffic pattern which evaluates the number of degrees of freedom used in a given airspace. The fractal dimension is a ratio providing a statistical index of complexity comparing how detail in a pattern changes with the scale at which it is measured. Applied to air route analysis, it consists in computing the fractal dimension of the geometrical figure composed of existing air routes. A relation between fractal dimension and conflict rate (number of conflicts per hour for a given aircraft) is also shown in [23].

In [21], an Input-Output approach to traffic complexity is exposed. In this approach, air traffic complexity is defined in terms of the control effort needed to avoid the occurrence of conflicts when an additional aircraft enters the traffic. The input-output system is defined as follow. The air traffic within the considered region of the airspace is the system to be controlled. The feedback controller is an automatic conflict solver. The input to the closed-loop system is a fictitious additional aircraft entering the traffic. The output is computed using the deviations of the aircraft already present in the traffic due to the new aircraft (issued by the automatic conflict solver). This amount of deviations needed to solve all the conflicts provides a measure of the air traffic complexity. This metric is dependent on the conflict solver and on the measure of the control effort.

Another approach of complexity metrics is the intrinsic complexity metrics approach (for example in [15]). In this case, a difference is made between the control workload and the traffic complexity [24]. The control workload is a measurement of the difficulty for the traffic control system (human or not) of treating a situation. This is linked to the cognitive process of traffic situation management. The traffic complexity, on the other hand, is an intrinsic measurement of the complexity of a traffic situation, independently to any traffic control system. Intrinsic complexity metrics ignore the control workload and aim at modelling the traffic complexity only using the geometry of trajectories. In [5], an intrinsic complexity metrics is described using linear and nonlinear dynamical system models. The air traffic situation is modeled by an evolution equation (the aircraft trajectories being integral lines of the dynamical system). The complexity is measured by the Lyapunov exponents of the dynamical system. Intuitively, The Lyapunov exponents capture the sensitivity of the dynamical system to initial conditions. In this study, we will use a similar approach, based on linear dynamical system computed in the neighborhood of a trajectory point and taking into account the uncertainty on the position of the considered aircraft. With this metric, a linear dynamical system is fitted to the aircraft speed vectors V and position vectors X, such that $V \simeq A.\cdot X + b$. The eigenvalues of A are used to identify different organizational structures of the aircraft speed vectors, such as translation, rotation, divergence, convergence, or a mix of them (see Figure 1 taken from [5]). This last complexity metric is computationally efficient [8].

A Dynamic Weighted Network Approach is introduced in [38]. Three types of complexity relations are defined: aircraft-aircraft (risk of conflicts), aircraft-waypoints (i.e the proximity with the entry/exit points of a sector) and aircraft-airways (deviation of an aircraft from its route). These complexity relations are combined in a dynamic network, where the nodes are the air traffic units (aircrafts, waypoints and airways) and the edges are the complexity relations between them, weighted by a measure of this complexity. A aggregate complexity metric is then derived from this network.

In [40], the authors define flight conflict shape movements on each point of an aircraft's trajectory based on the aircraft position and speed, as well as its likely angle of deviation and the safety standards of separation. Using this flight conflict shape movement, a complexity value is attributed to each pair of aircraft. Then, a average measure of complexity can be computed for a user-defined area. The complexity metric thus defined is compared with the intrinsic metric from [5]. Despite its simplicity, this complexity metrics is able to

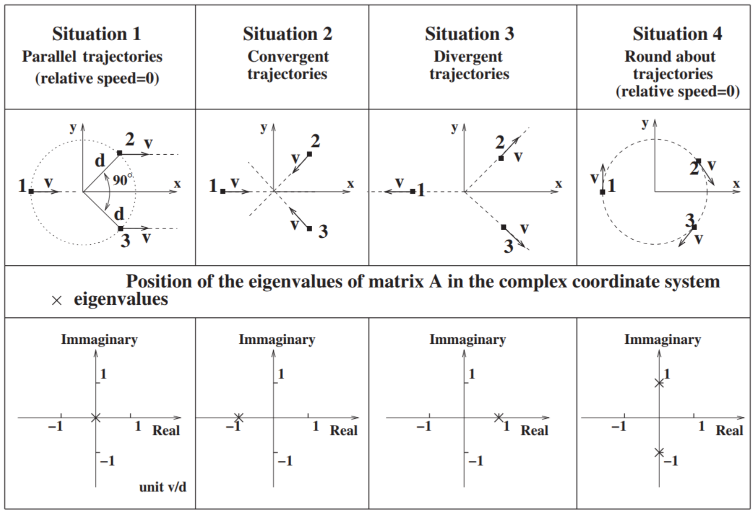
Situation 1
Parallel trajectories (relative speed=0)
Situation 2
Convergent trajectories
Situation 3
Divergent trajectories
Situation 4
Round about trajectories (relative speed=0)
Position of the eigenvalues of matrix A in the complex coordinate system
× eigenvalues
Immaginary
Real
unit v/d

Figure 1: Eigenvalues loci for several typical situations.The small squares are the initial positions of aircraft at a given time (this represents the observation given by a radar for instance with the associated speed vector). Source: [5]

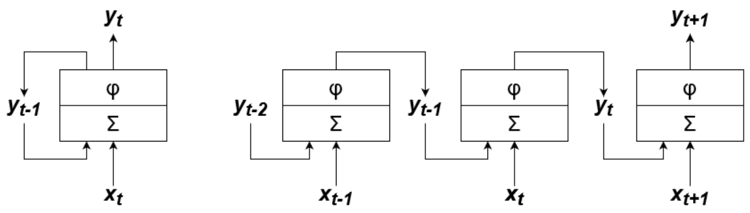

Figure 2: Basic RNN layer (left) and the same layer unrolled through time (right). Σ represents linear mapping and ϕ represents the activation function

account for the complexity in typical scenarios.

2.2 Recurrent Neural Networks

In this section, we will present a brief overview of recurrent neural networks (RNN). The basic RNN layer is detailed first, along with a typology of the tasks that are efficiently achieved by a RNN. Then, we will describe some RNN variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). Natural Language Processing (NLP) concepts such as language models, word embeddings or Beam search algorithm will not be covered (see [41] for a complete review). Attention mechanisms [41] are also outside the scope of this short introduction to RNN.

RNN is a class of nets that is designed to process sequences of data rather than fixed-sized inputs. Hence, they are very effective at analysing and predicting time series, and have been extensively used in Natural Language Processing including automatic translation (for example [34]) or speech-to-text (for example [30]). They have also been applied in generative models, for example in image captioning ([37]).

The basic RNN layer is shown in Figure 2. At each time step t, the recurrent cell receives the input x_t as well as its own output from the previous time step y_{t-1}. The output y_t is computed according to Equation (1), where W is the weights matrix, b is the bias vector and ϕ is an activation function (for example $tanh$ or ReLU, [20]). x_t and y_{t-1} have been concatenated in a single row vector denoted $[x_t; y_{t-1}]$.

$$y_t = \phi(W \cdot [x_t; y_{t-1}] + b) \tag{1}$$

Depending on the task aimed to be solved by the network, various architectures are possible. A simple

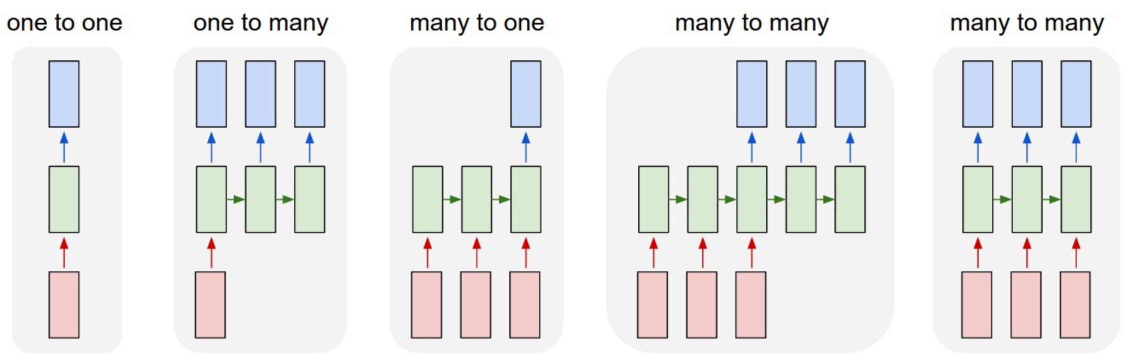

Figure 3: Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state. In the basic RNN layer, the RNN's state is equal to the output vector, but they are generally different in more complex layers (i.e LSTM). Source: [16]

typology of these architectures is provided in Figure 3 taken from [16]. One-to-one networks correspond to classical (non-recurrent) feedforward networks. In one-to-many networks, a single input vector is used to output a sequence, for example in image captioning. Alternatively, in many-to-one networks, a sequence of inputs can be used to get a single output vector, for example in sentiment analysis. The first many-to-many architecture is an *encoder-decoder*, where a sequence of inputs is encoded into a single vector, then decoded into a sequence of outputs. This type of model can be used for automatic translation, where we need to read the full input sequence before beginning the translation process. Finally, the second many-to-many model also transform a sequences of inputs into a sequences of outputs, but a new output is generated right after a new input is provided to the network. An example of application is the video classification, where each frame of a video is labeled.

To train the network, the classical optimization algorithms (generally based on gradient descent) need the gradients of the parameters with respect to the outputs. To compute these gradients, we can simply use the backpropagation method, often referred as *backpropagation through time* (BPTT) when applied to RNN.

To prevent overfitting, a technique often used with RNN is dropout ([27], [7]). When applied to a given layer, dropout consists in randomly remove (set to zero) a fixed proportion of the outputs of the layer. The cancelled outputs changed at each forward pass in the network. The dropout proportion (also referred as the dropout rate) is an hyperparameter.

The basic RNN layer introduced above has difficulty learning long term dependencies when the length of the sequence is very long. This problem has been referred as the *vanishing gradient* problem [26]. To solve it, several architectures of RNN layer with long-term memory have been introduced. We will present two of these new type of layers: Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU).

Long-Short Term Memory were first introduced in [13], and then improved over the years (for example [29] and [42]). LSTM were designed to be faster to train than regular RNN, and to be able to detect long-term dependencies. Figure 4 shows the architecture of the LSTM cell. Contrary to the basic RNN layer, the LSTM has two state vectors denoted h_t (the hidden state) and c_t (he cell state) which can be seen respectively as the short-term state and the long-term state. The LSTM's equations are presented in Equations (2) to (7). $\cdot$ represents the matrix multiplication, while $\times$ represents the element-wise multiplication. σ is the logistic activation (to get outputs between 0 and 1). The current input x_t and the previous hidden state h_{t-1} are used to compute the controllers of three *gates*: the input gate controller i_t, the forget gate controller f_t and the output gate controller o_t (Equations (2) to (4)). These gate controllers are simply computed by a linear mapping followed by a logistic activation. In parallel, a candidate cell state $\tilde{c}_t$ is computed using the current input and the previous hidden state (Equation (5)). Equation (5) can be seen as the LSTM equivalent of the equation of the basic RNN layer (Equation (1)). In Equation (6), the current cell state c_t is then updated by adding the previous cell state c_t (filtered by the forget gate) and the candidate cell state $\tilde{c}_t$ (filtered by the input gate). The hidden state h_t is finally computed by activating the cell state c_t with the *tanh* function (or another activation function such s the ReLU), which is then passed though the output gate (Equation

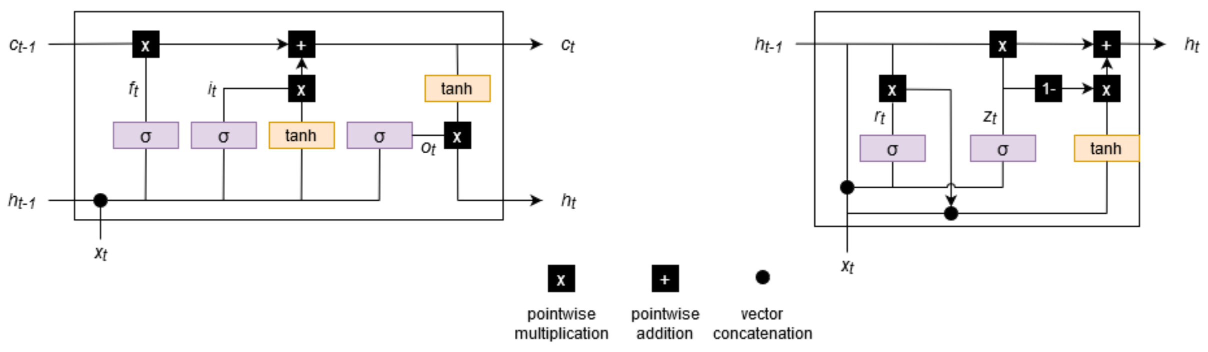

Figure 4: LSTM cell (left) and GRU cell (right)

(7)). With the input gate i_t, the LSTM is able to store information in the cell state c_t, to extract it with the output gate o_t and to discard it with the forget gate f_t.

$$i_t = \sigma(W_i \cdot [x_t; h_{t-1}] + b_i) \tag{2}$$
$$f_t = \sigma(W_f \cdot [x_t; h_{t-1}] + b_f) \tag{3}$$
$$o_t = \sigma(W_o \cdot [x_t; h_{t-1}] + b_o) \tag{4}$$
$$\tilde{c}_t = tanh(W_c \cdot [x_t; h_{t-1}] + b_c) \tag{5}$$
$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t \tag{6}$$
$$h_t = o_t \times tanh(c_t) \tag{7}$$

A variant of LSTM (introduced in [9]) called *peephole connections* uses the cell states c_{t-1} and c_t to compute the gate controllers. Another variant of the LSTM is the Gated Recurrent Unit (GRU) introduced in [3]. The Figure 4 illustrates the GRU cell and its equations are detailed in Equations (8) to (11). The GRU is a simplified version of the LSTM, but seems to have the same performance (see [10]). The GRU has a unique state vector h_t, while the forget and input gates are merged into a single gate controller z_t (Equation (8) and (11)). The output gate is replaced by another gate controller r_t (Equation (9)), placed between the previous hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$ (Equation (10)).

$$z_t = \sigma(W_z \cdot [x_t; h_{t-1}] + b_z) \tag{8}$$
$$r_t = \sigma(W_r \cdot [x_t; h_{t-1}] + b_r) \tag{9}$$
$$\tilde{h}_t = tanh(W_h \cdot [x_t; r_t \times h_{t-1}] + b_h) \tag{10}$$
$$h_t = z_t \times h_{t-1} + (1 - z_t) \times \tilde{h}_t \tag{11}$$

An improvement of the RNN structure is the Bidirectional Recurrent Neural Network (BRNN) introduced in [31]. In the basic RNN layer, the hidden state h_t is computed using only the sequence of inputs $x_0, ..., x_t$. However, the prediction at time step t may depend on elements that are further in the input sequence (for example in automatic translation). BRNN consists in creating two networks, with one of these networks going through the input sequence in a forward pass (x_0 until x_T where T is the length of the input sequence), and the other one going through the input sequence backward (x_T until x_0). Hence, we obtain two sequences of hidden states, and at each time step, the prediction can be computed using these two hidden states. The main drawback of BRNN is that the full input sequence is needed to produce predictions, so online predictions is impossible.

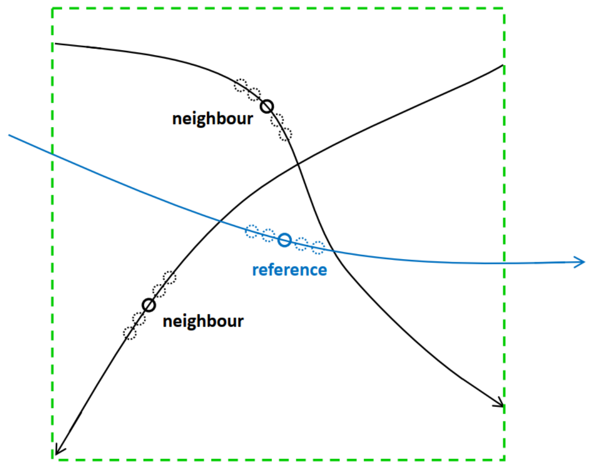

Figure 5: Neighborhood of the reference aircraft. The dotted circles represents the extension of the aircraft positions. Source: [8]

3 Data and Methodology

The objective of the report is to build a machine learning model able to predict congested areas in a time horizon of 40 minutes. This objective will be defined as a supervised learning regression task. To achieve this, a training set has to be built using pairs of training inputs/outputs.

The data used in this study are simulated trajectories over the French airspace. These trajectories are created by inputting flight plans to the simulator without any deconflicting actions. The final dataset is composed of 8011 simulated trajectories spread along 3025 time steps (1 time step is equal to 15 seconds).

Let us first describe the training outputs. Since we want to predict the congested areas, the training outputs have to provide a measure of the congestion in each point of the airspace at a given time. This measure of congestion is achieved with a complexity metric, as the ones described in subsection 2.1. The selected complexity metric is the intrinsic complexity metric based on linear dynamical system ([8]), since it provides a measure of the traffic complexity independent from the perceived workload, and only linked with the geometrical organisation of the trajectories. Other metrics may also be tested in further studies. As described in [8], the metric based on linear dynamical system can be adapted to compute a measure of the complexity in the neighborhood of an aircraft, at a given time. To do so, the neighborhood is defined as a $24,8NM \times 24,8NM$ box horizontally and $\pm 30FL$ vertically, centered on the reference aircraft. A filter is applied to consider only the flight that may interact with the reference aircraft. For example, another aircraft in level flight vertically separated with the reference aircraft by 1000 ft (in RVSM) will not interact with the reference aircraft, and should not be considered in the metric computation. After this filtering, the selected aircraft positions are extended by adding 10 forward positions (separated by one time step) and 10 aft positions (see Figure 5). The aim of this extension is to take into account the uncertainties on the true aircraft's positions. Each of this added points is treated as a new aircraft. Finally, a linear dynamical system $\dot{X} = AX + b$ is fitted to the positions and speeds of these aircraft. The complexity metric C is then defined as the sum, in absolute value, of the negative real part of the eigenvalues of the matrix A as shown in the following expression $C = -\sum_{\mathrm{Re}(\lambda(A))<0} \mathrm{Re}(\lambda(A))$. To build the training outputs, we defined for each time step t a $N \times N$ matrix $y[t]$. This matrix is superimposed over the airspace, such that each element of the matrix covers a small area of the airspace. Then, for a given area, the corresponding element of the matrix takes the maximum value of the complexity metric of the aircraft inside this area at time step t. The complexity value are then scaled by a logarithmic function to obtain a more uniform distribution. The heat map in Figure 6 provides an example of a matrix $y[t]$. The $y[t]$ matrices are then flattened as N^2 vectors to be used in the machine learning model.

Let us know focus on the training inputs. We will defined three different types of input features and compare their performances to predict the future congested areas. The first type of inputs (referred as Type

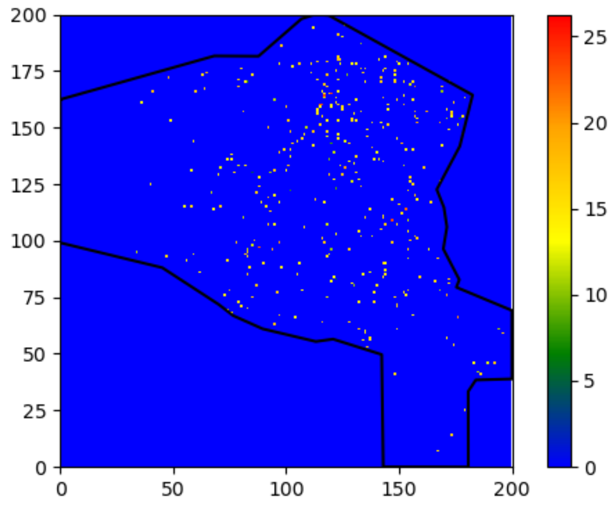

Figure 6: Example of visualisation with $N = 200$. The color scale indicates the complexity metric value. The black lines are the limits of the considered airspace.

I) is simply a sequence of flattened $y[t]$ matrices for t ranging from t_0 and t_0+T-1. Hence, the input vectors are $x[t_0] = y[t_0]$ until $x[t_0+T-1] = y[t_0+T-1]$ while the corresponding training output is $y[t_0+T+T_{pred}-1]$ (where T_{pred} is the number of time steps between the end of the input sequence and the prediction). The second type of inputs (Type II) will use the aircraft states. For a given point in an aircraft trajectory, we define the aircraft state as $state = \begin{bmatrix} longitude & latitude & altitude & ground\_speed & heading \end{bmatrix}$. At a given time t, the input vector $x[t]$ is built by concatenating all the aircraft states currently in the airspace, ordered by increasing longitude (and by increasing latitude if the longitude is equal), such that the positional information is preserved in the structure of the inputs. To get a fixed-sized input vector, the dimension of $x[t]$ is set to the maximum number of simultaneous aircraft states in the dataset (multiplied by the length of an aircraft state). If the number of states is strictly lower, the remaining elements of $x[t]$ are padded with zeros. Finally, Type III input is similar to Type II but with a different aircraft state, referred as an extended state, and defined as $extended\_state = \begin{bmatrix} longitude & latitude & altitude & ground\_speed & heading & C & a_{1,1} & a_{1,2} & \cdots & a_{3,3} \end{bmatrix}$ where C is the complexity metric defined above and the $a_{i,j}$ are the elements of the A matrix of the linear dynamical system previously mentioned.

We can know describe the training set as pairs of (input sequence, output vector) such that $\left(\begin{bmatrix} x[t_0] & \cdots & x[t_0+T-1] \end{bmatrix}, y[t_0+T+T_{pred}-1]\right)$ where x can either be Type I, II or III input features. In the following, the dataset will be randomly divided between training examples (90% of the dataset) and validation examples (10% of the dataset).

Since the input features are defined as sequences, it is natural to introduce a RNN model. The model selected for the experiments is presented in the Figure 7. This is a many-to-one network, as introduced in subsection 2.2. The input features are fed into L successive recurrent layers (which can either be LSTM or GRU). After the last time step, the hidden state of the L-th recurrent layer is fed into a dense layer to output a prediction $y_{pred}[t_0+T+T_{pred}-1]$. The selected error function is the Mean Squared Error between the prediction and the ground-truth $y[t_0+T+T_{pred}-1]$.

4 Results

In this section, we will present some preliminary results. The model is implemented with the Keras library using the TensorFlow 2 backend.

First, some model's hyperparameters are described and tuned. Concerning the architecture of the model, the following hyperparameters are evaluated: length of the input sequence T, class of recurrent layers rl (LSTM or GRU), number of layers L and dimension of the hidden states of each layer. Concerning the optimization algorithm, we use the Adam algorithm [17] with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and learning

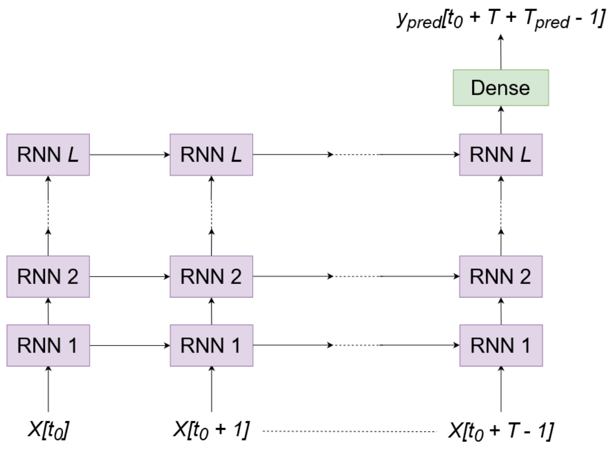

Figure 7: RNN model unrolled through time

rate decay $d = 10^{-2}$. The learning rate lr and batch size bs are evaluated. The impact of the L2 regularization coefficient lbd and the dropout rate dr is also considered. Finally, we evaluate the gradient norm scaling value cn, which is a threshold for the gradient norm aiming at avoiding the exploding gradient problem. To evaluate the impacts of these hyperparameters, we chose a default set of hyperparameters, and then change one hyperparameter's value at a time from the default values. The metric used to measure the model's performance is the average of the validation loss over the 10 last epochs. The default hyperparameters values are the following: $T = 160$, $rl = GRU$, $l = [512, 512, 512, 512]$ (meaning 4 recurrent layers with 512 hidden units in each one of them), $bs = 64$, $lr = 10^{-4}$, $dr = 0.1$, $lbd = 10^{-5}$, $cn = 1$. The model is trained over 200 epochs with a prediction time $T_{pred} = 160$ (corresponding to 40 minutes).

Hyperparameters values	Validation loss ($\times 10^4$)
Default	180
$T = 80$	182
$T = 320$	188
$rl = LSTM$	154
$l = [256, 256, 256, 256]$	168
$l = [1024, 1024, 1024]$	191
$l = [512, 512, 512, 512, 512, 512]$	213
$l = [1024, 256, 256, 1024]$	189
$bs = 32$	158
$bs = 128$	232
$lr = 10^{-5}$	720
$lr = 10^{-3}$	112
$dr = 0$	176
$dr = 0.3$	202
$lbd = 0$	102
$lbd = 10^{-4}$	312
$cn = 10^{-1}$	189
$cn = 10$	198

Table 1: Validation loss for various hyperparameters values

In the Table 1, we present the results of the hyperparameters search. In the left column, only the modified hyperparameter is shown with its new value. The hyperparameters having a significant impact on the model's performance are: the learning rate lr which slows down training when reduced, the L2 regularization which also harms the performances, the batch size bs which slows down training when increased (because the parameters updates are less frequent), and the choice of recurrent layer (LSTM seems to have slightly better performance). For the next experiments, the following values will be used: $lr = 10^{-3}$, $bs = 32$,

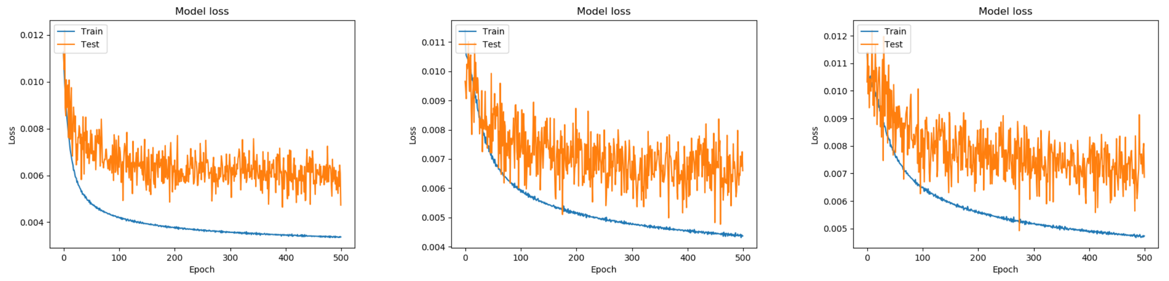

Figure 8: Training and validation (referred as test) losses per epoch using Type I input (left), Type II (center) and Type III (right)

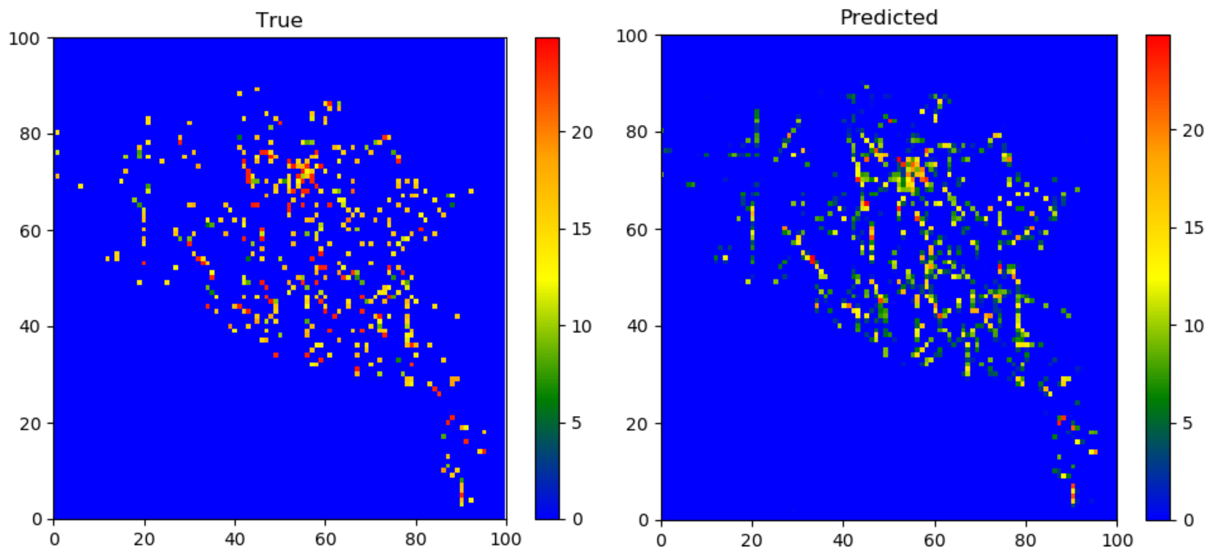

Figure 9: Validation example prediction (right) compared to the true output (left)

$rl = LSTM$. No regularization techniques will be implemented (no L2 regularization and no dropout). The other hyperparameters are kept to their default values.

Three model are now trained on 500 epochs with each input types (I, II and III). Loss curves for each model are on Figure 8. Each model seems to achieve similar performances, with a final validation loss in the order of $7 \cdot 10^{-3}$. Type I input (using directly $y[t]$ matrix as inputs) seems to achieve slightly better performance, which was expected since in this case the inputs and outputs share the same structure. However, Type I input requires the computation of the complexity metric before making predictions, while Type II input needs only the aircraft states. Type III input seems to bring no improvement over Type II input.

Finally, Figure 9 shows the prediction on a randomly chosen validation example using the Type II input, and compares it with the true complexity heat map. The model seems to be able to predict the location of the congested areas, even if the corresponding complexity values seem underestimated. An evaluation of these predictions by air traffic controllers is needed to assess if the model provides useful information for the FMP. This evaluation should also indicate what improvements are desirable with respect to the current information provision by existing tools, to help anticipate traffic bursts and increase the airspace capacity.

5 Conclusion

In this report, we have presented a simple RNN model using a sequence of aircraft states to predict the complexity of the air traffic in all areas of an airspace, in a time horizon of 40 minutes. An hyperparameters search was conducted to fine-tune their values. The hyperparameters search could be improved by using a Bayesian optimization framework [32], which has been extensively used for hyperparameters values optimization. Then, we presented some preliminary results showing the model's predictions with various inputs features. An evaluation by air traffic controllers from Flight Management Position is needed to assess the model's performances and to identify what level of performance is expected for the information provision from an operational perspective.

Several potential improvements have been identified. In particular, methods coming from Natural Language Processing could be applied. 1D Convolutional Neural Network layers (CNN) could be use to improve sequence processing, and then be combined with attention mechanisms through ”transformers” [36]. More work is also needed in feature engineering of the aircraft states, in particular to reduce the dimension of the inputs. This could be achieve with autoencoder [39]. The model should also be modified to take into account the future controllers' actions, since they impact the formation of congested areas.

Once evaluated, the prediction model can be used as an input to a mitigation model, which would be able to suggest actions on trajectories (changing heading, speed, or altitude) to prevent the formation of congested areas. This mitigation model could applied machine learning methods as well as more classical optimization techniques (such as metaheuristics, for example [11]) and conflict resolution algorithms.

Acknowledgments: This internship, taking place at the ENAC laboratory, is made possible thanks to the supervision of Professor Daniel Delahaye. I would also like to thank Professor Nidhal Bouaynaya and Professor Ghulam Rasool from the Rowan University, as well as Ian Levitt, Andrew Cheng and Albert Schwartz from the FAA WJH Technical Center for helping me with their respective expertise in Natural Language Processing and Air Traffic Flow Management.

References

[1] Gano B. Chatterji and Banavar Sridhar. Neural network based air traffic controller workload prediction. *Proceedings of the American Control Conference*, 4:2620–2624, 1999.

[2] Gano B. Chatterji and Banavar Sridhar. Measures for air traffic controller workload prediction. *1st AIAA, Aircraft, Technology Integration, and Operations Forum*, 2001.

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014.

[4] Ramon Dalmau, Franck Ballerini, Herbert Naessens, Seddik Belkoura, and Sebastian Wangnick. Improving the predictability of take-off times with machine learning a case study for the maastricht upper area control centre area of responsibility. *SESAR Innovation Days*, 2019.

[5] Daniel Delahaye and Stéphane Puechmorel. Air traffic complexity based on dynamical systems. In *49th IEEE Conference on Decision and Control*, pages 2069–2074, 2010.

[6] Pierre Flener, Justin Pearson, Magnus Ågren, Carlos Garcia-Avello, Mete Çeliktin, and Søren Dissing. Air-traffic complexity resolution in multi-sector planning. *Journal of Air Transport Management*, 13(6):323–328, 2007.

[7] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1027–1035, 2016.

[8] Adrian Garcia, Daniel Delahaye, and Manuel Soler. Air traffic complexity map based on linear dynamical systems. *Preprint*, 2020.

[9] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. *Neural Networks, IJCNN*, 2:189–194, 2000.

[10] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[11] Eulalia Hernández-Romero, Alfonso Valenzuela, Damián Rivas, and Daniel Delahaye. Metaheuristic approach to probabilistic aircraft conflict detection and resolution considering ensemble prediction systems. *SESAR Innovation Days*, 2019.

[12] Brian Hilburn. Cognitive Complexity in Air Traffic Control: A Literature Review. *Eurocontrol*, 12(13):93–129, 2004.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] ICAO. *Long-Term Traffic Forecasts, Passenger and Cargo*, 2018.

[15] Mariya A Ishutkina and Eric Feron. Describing Air Traffic Complexity Using Mathematical Programming. In *AIAA 5th Aviation, Technology, Integration, and Operations Conference*, 2005.

[16] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*, 2015. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

[17] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[18] Parimal Kopardekar and Sherri Magyarits. Dynamic density: Measuring and predicting sector complexity. *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, 1:1–9, 2002.

[19] I V Laudeman, S G Shelden, R Branstrom, and C L Brasil. Dynamic density: An air traffic management metric. Technical Report April 1998, NASA, Moffett Field, CA, 1998.

[20] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *arXiv e-prints*, 2015.

[21] Keumjin Lee, Eric Feron, and Amy Pritchett. Air traffic complexity: An input-output approach. In *American Control Conference*, pages 474–479, 2007.

[22] Anthony J Masalonis, Michael B Callaham, and Craig R Wanke. Dynamic density and complexity metrics for realtime traffic flow management: Quantitative analysis of complexity indicators. *5th USA/Europe Air Traffic Management R & D Seminar, Budapest, Hungary*, 139, 2003.

[23] Stephane Mondoloni and Diana Liang. Airspace fractal dimension and applications. In *Fourth USA/EUROPE Air Traffic Management R& D Seminar*, pages 1–7, 2001.

[24] Bang Giang Nguyen. *Classification in functional spaces using the BV norm with applications to ophthalmologic images and air traffic complexity.* PhD thesis, Université de Toulouse 3 Paul Sabatier, 2014.

[25] Xavier Olive, Jeremy Grignard, Thomas Dubot, and Julie Saint-lot. Detecting controllers' actions in past mode s data by autoencoder-based anomaly detection. *SESAR Innovation Days 2018*, 2018.

[26] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org, 2013.

[27] Vu Pham, Theodore Bluche, Christopher Kermorvant, and Jerome Louradour. Dropout Improves Recurrent Neural Networks for Handwriting Recognition. In *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, volume 2014-Decem, pages 285–290, 2014.

[28] Maria Prandini, Luigi Piroddi, Stéphane Puechmorel, and Silvie Luisa Brázdilová. Toward Air Traffic Complexity Assessment in New Generation Air Traffic Management Systems. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):809–818, 2011.

[29] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *arXiv e-prints*, 2014.

[30] George Saon, Gakuto Kurata, Tom Sercu, Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis, Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn Li Lim, Bergul Roomi, and Phil Hall. English conversational telephone speech recognition by humans and machines. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2017-August:132–136, 2017.

[31] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.

[33] Banavar Sridhar, Kapil S Sheth, and Shon Grabbe. Airspace Complexity and its Application in Air Traffic Management. In *2nd USA/EUROPE Air Traffic Management R&D Seminar*, pages 1–9, 1998.

[34] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

[35] SESAR Joint Undertaking. *Airspace Architecture Study*, 2019.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[37] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 3156–3164, 2015.

[38] Hongyong Wang. Modeling Air Traffic Situation Complexity with a Dynamic Weighted Network Approach. *Journal of Advanced Transportation*, 2018, 2018.

[39] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2015.

[40] Hong Jie Wee, Sun Woh Lye, and Jean-philippe Pinheiro. A Spatial , Temporal Complexity Metric for Tactical Air Traffic Control. *The Journal of Navigation*, pages 1040–1054, 2018.

[41] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing [Review Article]. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018.

[42] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *arXiv e-prints*, pages 1–8, 2014.