Airspace Interaction Modeling and Mitigation by Artificial Intelligence Approach: a Literature Review

Loïc Shi-Garrier

November 2021

Contents

Introduction

With the continuous rise in air transportation demand (a 4.3% annual growth of the worldwide passenger demand is expected until 2035 [1]), the capacity of the Air Traffic Management (ATM) system is reaching its limits, leading to increased flight delays (expected to achieve 8.5 minutes per flight in 2035 with the current system [2]). The Covid-19 crisis has severely impacted the aviation industry, with a 90% decline of RPK (Revenue Passenger Kilometer) in July 2020 with respect to the same period in 2019. However, the traffic is expected to regain its growth rate once the crisis is over, with the worst-case scenario predicting a return to normality before 2025.

Congestion is defined as a situation where 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. Air traffic complexity is a measure of the difficulty that a particular traffic situation will present to air traffic control [3], and was shown to negatively affect the controller's decision-making ability and increase the error rate [4]. To address this complexity and improve the service provision to airspace users through reduced delays, better punctuality, less ATFCM regulations, and enhanced safety, the Extended ATC Planning (EAP) function was introduced by SESAR JU as the Solution #118 [5]. The EAP function relies on automation tools that enable early measures to be taken by ATC before traffic enters overloaded sectors. The implemented tools aim at bridging the gap between Air Traffic Flow and Capacity Management (ATFCM) and Air Traffic Control (ATC) by facilitating the communication between the local flow management position (FMP) and the controllers' working positions, and providing information to help taking actions at tactical time (less than one hour) such as rerouting or sector configurations. The gap between ATFCM and ATC can be described as follows: although ATFCM measures have managed to balance the capacity with the demand, high complexity areas may appear at the control level. These "hot spots" or "traffic bursts" require an intense surveillance from air traffic controllers in order to decide if some trajectories should be modified to avoid any conflict. Two problems must be addressed to reduce this increased workload: 1) the prediction of hot spots tens of minutes before their formation, and 2) the mitigation of the hot spots by appropriate early actions. This literature review focuses mainly on the first problem of the prediction of congested areas tens of minutes ahead of formation.

In the first part of this report, we will deal with Air Traffic Flow Management (ATFM) aspects, mainly focusing on the first problem of the prediction of congested areas tens of minutes ahead of formation. We can group the literature to tackle the general problem of congestion prediction into three main approaches. The first approach considers conflict detection tools in a time horizon lower than thirty minutes. The second approach focuses on predicting the air traffic flow. The third approach computes various air traffic complexity metrics. A fourth more indirect approach considers the problem of congestion prediction as an application of the trajectory prediction problem. This is why a section is dedicated to trajectory prediction methods.

In the second part, we delve into artificial intelligence approaches. We restrict our area of

inquiry to the field of machine learning. We present machine learning sequential models that are well suited to process time series such as aircraft trajectories. A section is then dedicated to a short presentation of the optimization algorithms used in current state-of-the-art machine learning. The final section of this report deals with uncertainty in deep learning. In order to deliver a trustworthy decision making tool, our model should be able to quantify the uncertainty of its predictions so that a human operator may decide to reject the prediction. Trustworthiness is absolutely necessary if an automation tool is to be accepted by air traffic controllers. This is why we introduce a recent framework that efficiently apply variational inference (VI) to produce uncertainty quantifications and we derive its implementation for a transformer model.

Air Traffic Flow Management (ATFM)

In this part is presented the literature addressing our domain of application (Air Traffic Flow Management) and more specifically the concepts and methods linked to congestion prediction at medium to short term.

2.1 Conflict detection tools and Air Traffic Flow Prediction (ATFP)

In this section, we briefly introduce classical conflict detection tools before reviewing in more details Air Traffic Flow Prediction methods.

2.1.1 Conflict detection tools

In the first approach of short-term conflict detection methods [6], the Medium-Term Conflict Detection (MTCD) flight data processing is perhaps the most popular system. MTCD is designed to warn the controller of potential conflicts between flights in a time horizon extending up to thirty minutes ahead. The system integrates predictive tools that performs trajectory prediction, conflict detection, trajectory update, and trajectory edition using "what-if" scenarios and tools [6]. The MTCD applications, currently implemented in operational context, e.g., the Eurocontrol MTCD project [7], are mainly based on pairwise conflicts rather than on a global approach. Moreover, for longer time horizons, the uncertainties on the trajectories make it difficult to predict the exact trajectories that will be involved in conflicts.

2.1.2 Classic ATFP Methods

Air traffic flow prediction (ATFP) focuses on aggregate models [8] rather than simulating the trajectories of individual aircraft. Trajectory-based models result in a large number of states, which are susceptible to error and difficult to design and implement for air traffic flow management [9]. The ATFP approach develops models of the behavior of air traffic that can be used for the analysis of traffic flow management. Several methods were considered, ranging from probabilistic algorithms modeling the flow over a network [8], [10], [9] to more recent machine learning algorithms [11], [12], [13].

In [8], a linear time variant traffic flow model was developed based on historical data. The dimension of this model depends only on the number of considered control volumes and not on the number of individual aircraft. This model is able to forecast the aircraft count at an Area Control Center level with a time interval of ten minutes. Another flow-based model was later developed in [10], called Link Transmission Model. In this model, a flight path is defined as a sequence of directed links passing through sectors. An aircraft is assumed to cross each of the links within an estimated crossing time such that the state of each link can be easily tracked. Aggregation of

links in each sector yields a traffic forecast for that sector. This approach is extended in [9] where the authors introduce a dynamic network of air traffic flow characterizing both the static airspace topology and the dynamics of the traffic flow. Historical data is used to estimate the travel time corresponding to the weight of each edge of the network. Then, a probabilistic prediction method is implemented for the short-term prediction (fifteen minutes) of the air traffic flow.

2.1.3 Machine Learning ATFP

Recently, several machine learning models were proposed for ATFP. In [11], the authors define a 3D grid over a given airspace containing the number of flights in each cell. They rely on neural network models that combine convolutional operations to extract spatial features and recurrent operations to extract time-dependent features. The 3D grids of the last ten minutes are inputted into the model to predict the 3D grid of the next timestep. A similar task is addressed in [12] using several 3D convolutional neural networks to extract spatial features for one timestep then recurrent layers to extract temporal features. In [13], the authors used support vector machines (SVM) and recurrent neural networks to predict the hourly air flow in predefined routes from time information (such as the time of the day, or the season and holiday indexes).

The main drawback of all the above-mentioned ATFP methods is that they do not define any concept of congestion since the predicted variable is the aircraft count whether in a sector, a route, or crossing a predefined waypoint. Hence, these models are unable to discriminate low complexity situations from high complexity situations for a similar aircraft count.

2.2 Complexity Metrics

To address the limitations of ATFP methods proposed in the literature, the third approach to congestion prediction redefines the forecast objective of the ATFP problem by considering complexity metrics. We start by presenting two reviews that have been published on the topic [3], [14]. Then, we discuss classic complexity metrics and intrinsic complexity metrics.

Research focusing on defining and quantifying air traffic complexity, and on analyzing its impact on air traffic controller workload, was extensively conducted during the last two decades [15], [16], [17], [18], [19], [20], [21], [22]. The traditional measure of air traffic complexity is the traffic density, defined as the number of aircraft crossing a given sector in a given period [17]. The traffic density is compared with the operational capacity, which is the acceptable number of aircraft allowed to cross the sector at 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, or refuse aircraft even though the operational capacity has not been reached. This is the main motivation for the investigation on complexity metrics.

The literature focusing on estimation of conflict probability (see [23] for one example) is not addressed in this review.

2.2.1 Reviews

In [3], Prandini et al. review various existing ground-based complexity metrics in order to assess their potential extension to the future Air Traffic Management (ATM) system relying on autonomous aircraft. The reviewed metrics are the following (more details can be found in the next subsections 2.2.2 and 2.2.3):

- Aircraft Density (AD): the number of aircraft on a per-sector basis. This is the metric currently used in most operational applications.

- Dynamic Density (DD): aggregate measure of complexity which aims to provide a more relevant metric than AD by taking into account static and dynamic characteristics.
- Interval Complexity (IC): time-smoothed version of DD.
- Fractal Dimension (FD): aggregate measure of the geometric complexity of a traffic pattern, independently from sectorization.
- Input-Output approach (IO): the complexity is evaluated as the control effort required to avoid conflicts when an additional aircraft is added.
- Intrinsic complexity metrics: metrics capturing the level of disorder and the organization structure of the traffic without any relation with the workload. Indeed, the evaluation of workload is a long-debated issue and an inherently ill-posed problem. The difficulty of obtaining reliable and objective workload measures is the main motivation for investigating complexity metrics that are independent of the ATC workload.

As part of his PhD dissertation [14], B. G. Nguyen makes a distinction between the control workload and the traffic complexity:

- The *control workload* measures the difficulty for the traffic control system (human or not) to remedy a situation.
- The *traffic complexity* is an intrinsic measurement of the complexity of the air traffic, independently of any traffic control system. It is link to the sensitivity to initial conditions as well as to the interdependence of conflicts. In other words, intrinsic complexity metrics aim at modelling the level of disorder and the organization structure of the air traffic distribution, irrespective of its effect on the ATC workload.

2.2.2 Classic Methods

In [15], Laudeman et al. introduce the *Dynamic Density*. It is a metric for air traffic management which aims at measuring the air traffic controller workload by relying both on traffic density (count of aircraft in a volume of airspace) and traffic complexity. The objective is to achieve better prediction of controllers' activity than using the traffic density alone. The traffic complexity is defined by a set of eight traffic factors. These traffic factors, along with the traffic density, are weighted and linearly combined to produce a single indicator. The Dynamic Density should also take into account the controller's intent, but it was not included here due to the difficulty to measure it.

The eight traffic indicators, selected by interviewing air traffic controllers, are the following: heading change, speed change, altitude change, minimum distance below 5 NM, minimum distance between 5 and 10 NM, conflicts predicted below 25 NM, conflicts predicted between 25 and 40 NM, conflicts predicted between 40 and 70 NM. The weights of these traffic factors are determined using two methods: linear regression, and subjective weighting proposed by controllers. Compared to an independent measure of workload, the regressed weights provide the best prediction of controllers workload. The traffic indicators are shown to not be very correlated, while speed change and conflicts predicted below 25 NM seem to be the least significant of the eight factors.

In [16], Sridhar et al. develop the work of Laudeman et al. by investigating the prediction of the Dynamic Density in short-term (5 minutes) and medium-term (20 minutes) time horizon. They rely on a trajectory predictor called the Center-TRACON Automation System which uses flight plans, radar tracks, and predicted atmospheric data at the ARTCC level. The authors achieve

good prediction performance that can be further improved by using aircraft data coming form the Enhanced Traffic Management System (ETMS), which centralized aircraft information from all ARTCCs. Possible improvements for the complexity metric include: use of aircraft intents information, of structural characteristics (such as airways intersections), and of other dynamic flow events (such as weather). The complexity should also address cognitive aspects of controller workload. This prediction capability could then be used by Area Supervisors for resource allocation and by TFM for airspace planning.

In [17], B. Hilburn reviews the literature into cognitive complexity in air traffic control. Complexity can be defined as the state of being hard to separate, analyse, or solve. Some synonyms of "complex" are: "complicated", "intricate", "difficult", or "involved". A complex system must be distinguished from a complicated system (for which it is possible to provide a complete description, even if it is composed of a huge number of parts). Complex systems are defined by: a large number of elements, which interact dynamically, which contain redundancy, which enjoy localised autonomy and low information sharing between all parts, with non-linear interactions between elements, and with opacity (some system variables are unobservable). In ATC, complexity can be defined as a measure of the difficulty that a traffic situation presents to an air traffic controller. It is a multidimensional concept that includes static sector characteristics and dynamic traffic patterns, and which is subjectively defined by controllers. ATC workload is a function of the geometrical nature of the traffic, the operational procedure and practices, and the characteristics and behaviour of individual controllers.

The system engineering field has proposed several approaches to complexity. The notion of entropy from information theory has been applied to team information transfer, human eye scan behaviour, or to the predictability and general dispersion of the traffic. Normative human models rely on control theory (e.g., using Kalman filtering), queuing theory, or utility theory. However, these approaches face several difficulties: measurement difficulties (variability of human behaviours), the fact that humans do not "optimize" but "satisfy" (which is harder to model), and the fact that human behaviour is very dependent on the context.

In [18], G. Chatterji and B. Sridhar use a multi-layered neural network to predict the controller workload. The true workload is determined by a controller with a scale of three levels (low, medium, high). The inputs of the neural network are a set of scalar measures taken from the image processing literature. To compute these measures, the sum and difference histograms of the position and velocity of neighbouring aircraft are computed. The neighbour relation is determined using a minimum spanning tree where the weights of the graph are the distances between each pair of aircraft. Their work is extended in [19].

Several versions of Dynamic Density have been proposed in the literature [20], [21]. For example, the interval complexity introduced in [22] 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 [24]. 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 [24].

In [25], 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.

A Dynamic Weighted Network Approach is introduced in [26]. 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 (aircraft, 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 [27], 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, an average measure of complexity can be computed for a user-defined area. The complexity metric thus defined is compared with the intrinsic metric from [28] (see section 2.2.3). Despite its simplicity, this complexity metrics is able to account for the complexity in typical scenarios.

2.2.3 Intrinsic Complexity Metrics

As elaborated in [3], these complexity indicators aggregate air traffic measurements and workload to describe the perceived complexity. However, the evaluation of workload is a long-debated issue and an inherently ill-posed problem. The difficulty of obtaining reliable and objective workload measures is the main motivation for investigating complexity metrics that are independent of the ATC workload. To address this issue, another approach has been developed in the literature: the *intrinsic complexity metrics* approach [29], [28]. As developed in [14], intrinsic complexity metrics aim at modelling the level of disorder and the organization structure of the air traffic distribution, irrespective of its effect on the ATC workload.

In [29], the authors interpolate a velocity vector field satisfying certain constraints, e.g., the field shall be equal to the velocity of an aircraft if an aircraft is present at this point, the field shall be flyable by an aircraft. The problem is modeled as a mixed integer linear program. The complexity is measured as the number of constraints that must be relaxed to obtain a feasible problem. In our previous work [28], we defined an intrinsic complexity metric using linear and non-linear 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. The Lyapunov exponents capture the sensitivity of the dynamical system to initial conditions: if a small variation in the current air traffic situation leads to a very different dynamical system, the complexity of the situation is considered to be high.

In [30], an intrinsic complexity metric based on a linear dynamical system model is introduced. This metric computes a measure of complexity in the neighborhood of an aircraft at a given time. A filter is applied to consider only the flights that may interact with the reference aircraft. For example, an aircraft that is vertically separated with the reference aircraft by 1000 ft (in RVSM) will not interact with the reference aircraft, and, thus, should not be considered in the metric computation. After this filtering, the selected aircraft positions are extended by adding p forward positions and p backward positions. The aim of this extension is to take into account uncertainties on the true aircraft's positions.

Let n_{ac} be the number of aircraft samples retained for the computation of the metric. Consider the matrices $\mathbf{P}$ and $\mathbf{V}$, which denote, respectively, the positions and velocities of the n_{ac} aircraft, i.e.,

$$\mathbf{P} = \begin{bmatrix} x_1 & x_2 & \dots & x_{n_{ac}} \\ y_1 & y_2 & \dots & y_{n_{ac}} \\ z_1 & z_2 & \dots & z_{n_{ac}} \end{bmatrix}; \mathbf{V} = \begin{bmatrix} v_{x_1} & v_{x_2} & \dots & v_{x_{n_{ac}}} \\ v_{y_1} & v_{y_2} & \dots & v_{y_{n_{ac}}} \\ v_{z_1} & v_{z_2} & \dots & v_{z_{n_{ac}}} \end{bmatrix},$$

where x_i, y_i, z_i are the coordinates and $v_{x_i}, v_{y_i}, v_{z_i}$ the speed components of the i^{th} sample. A linear dynamical system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}$ is fitted to the positions $\mathbf{P}$ and speeds $\mathbf{V}$ of these aircraft, such that $\mathbf{V} \simeq \mathbf{A}\mathbf{P} + \mathbf{b}$ (here, $\mathbf{b}$, representing the mean of the speed vector field, is broadcast to be added to the matrix $\mathbf{AP}$). The matrix $\mathbf{A}$ and the vector $\mathbf{b}$ are computed as the solutions of the Least Mean Squares problem

$$\min_{\mathbf{A},\mathbf{b}} \|\mathbf{V} - (\mathbf{A}\mathbf{P} + \mathbf{b})\|_F^2. \tag{2.1}$$

The complexity metric, $c(\mathbf{A})$, is then defined as the absolute sum of the negative real part of the eigenvalues of the matrix $\mathbf{A}$

$$c(\mathbf{A}) = \sum_{\text{Re}(\lambda(\mathbf{A}))<0} |\text{Re}(\lambda(\mathbf{A}))|. \tag{2.2}$$

Recall that the evolution equation of a linear dynamical system has the form $\mathbf{P}(t) \sim \exp(\mathbf{A}t)$. By diagonalising the matrix $\mathbf{A}$, we can see that the asymptotic behaviour of the system depends uniquely on the eigenvalues of $\mathbf{A}$. Positive real parts of the eigenvalues correspond to diverging behaviour along the associated eigendirections while negative real parts correspond to convergence to a critical point. The imaginary parts correspond to rotation behaviour. Hence, our metric $c(\mathbf{A})$ measures the strength of the converging or shrinking behaviour of the dynamical system. Since our system is the closest linear dynamical system fitting the current positions and velocities of the aircraft, a strong converging behaviour corresponds to rapidly converging trajectories, which we associate with high complexity. This metric is computationally efficient.

2.3 Trajectory Prediction

In his PhD dissertation [31], K. Legrand reviewed the various trajectory prediction (TP) methods, focusing mainly on the differences between on-board TP (through the Flight Management System, FMS) and ground TP. Trajectory Prediction is an estimation problem, as opposed to filtering or smoothing. Given an initial state, TP methods compute the aircraft state vector (that is at least the 3D or 4D position) in the future. TP is used as a component of many tools which end goal is to increase the airspace capacity by reducing the minimum separation between aircraft while ensuring an ever growing level of safety.

FMSs are presented as a single TP enabler (without studying the differences between the various manufacturers). Indeed, they all use the same navigation databases (AIRAC) and describe the trajectories using the same standards (ARINC 424), while ground predictors use ICAO Document 4444. A lot of delays in air traffic are linked to ATC ordering clearances forcing the crew to return to more primitive FMS modes, while the FMS capabilities in managed mode (and in particular TP capabilities) are precise enough to optimally respect altitude and speed constraints. It then generates higher uncertainties on the trajectory.

There is a large literature on Trajectory Prediction, both for operational applications and studies or evaluations. Here are the various axis to classify TP systems:

- On-board system / Ground system

- Deterministic / Probabilistic
- Full 6 degrees of freedom model / Point-mass model / Macroscopic model
- En-route operations / Terminal Area operations
- Strategic conflict probe (20 minutes) / Tactical conflict alert (3 minutes)

Other approaches addressed in subsection 2.3.2 include:

- Machine learning techniques predicting trajectories using only past trajectory and meteorological data, without modeling the aircraft performance, the procedures, or the airspace.
- TP using functional regression, as introduced in [32].

Several criteria can be used to compare trajectory predictors:

- Input state data: airborne and ground systems do not always use the same coordinate systems, the same projections, the same trajectory description, the same navigation data. FMSs have access to more information.
- Constraints handled: lateral and vertical constraints are considered in ARINC 424 but not in ground predictors
- Behavior models used
- Mathematical models used
- Output trajectory data

Mathematical models for TP can be classified as follows:

- Point-mass models, using 3 degrees of freedom equations of motion.
- Kinematics models, using only position, heading, and speed, ignoring the causes of the motion.
- Kinetic (or dynamics) models, using forces, moments and aircraft performance parameters.
- Other models

Point-mass models are by far the most used models.

2.3.1 Classic TP

Trajectory prediction is a classical problem in ATM research since most of operational applications require at some point a precise and reliable trajectory prediction. Trajectory prediction approaches can be divided into two main categories: deterministic and probabilistic.

Deterministic approaches relies on a specific aerodynamic model to estimate the state of the aircraft and then propagate the states into the future, for example by using Kalman filter [33]. The main drawback of the deterministic approach is that it struggles to take into account uncertainties coming from future winds or pilot actions and hence is prone to degraded precision accuracy if used over a long period of time. The deterministic approach is then more suited for the prediction of specific flight phases.

2.3.2 Machine Learning TP

On the other hand, the probabilistic approach relies on statistical models able to learn the trajectory of aircraft from historical datasets. A first category of probabilistic approaches requires the dataset to cover all possible trajectories. These methods select one predicted trajectory among all possible using relevant features. For example, the authors of [34] trained a Hidden Markov Model (HMM) on a historical trajectory and weather dataset. Another probabilistic approach consists in providing the probability distribution of the aircraft position. For example, the authors of [23] uses the flight plan and the current position of the aircraft to build a discrete time probabilistic model to predict the future position of the aircraft. This model is then applied to estimate the probability of conflicts.

Finally, it is also possible the trajectory prediction task as a pure regression task. In [35], the authors trained a Generalized Linear Model (GLM) to use wind and aircraft initial state to predict the Estimated Time of Arrival (ETA) over several waypoints in the terminal area. In [36], the full 4D trajectory is predicted using an encoder-decoder architecture of RNNs, as presented in 3.1.2. The sequence input of the encoder network is the flight plan defined as a sequence of 2D positions. The sequence input of the decoder network is the concatenation of the actual 3D position and speed along with weather features. These weather features are extracted with convolutional layers (CNN) from a local box surrounding the actual position and containing wind, temperature and convective weather information. The trajectory points are sampled at a constant time rate, hence integrating the time component of the 4D trajectory. The decoder network does not output directly a prediction of the 3D position for the next time step, but rather a prediction of the parameters of a Gaussian Mixture Models (GMM) representing the distributions of the position and speed of the aircraft. During the inference process, this GMM model is used along with a beam search procedure to produce the most likely trajectory from the last known position until the arrival airport. The predicted trajectory is also smoothed using Adaptive Kalman Filter. The RNN model is trained on historical trajectories between an airport pair along with the corresponding weather data. Since the decoder network predicts the mean and covariance matrix of a GMM, the model is able to provide an evaluation of the uncertainty at each point of the predicted trajectory.

In [37], the authors also apply RNN architecture to predict 4D trajectories, but with a different approach than the one proposed in [36]. Instead, they only use one network of LSTM where the input sequence is the flight plan positions regularly resampled in time over the whole trajectory. The input sequence also includes weather features extracted with a convolutional network similar to the one used in [36]. Then, at each time step, the LSTM network has to predict the true 3D position using the planned position and the weather information.

Machine Learning

3.1 Sequential Models

In this section, we will present an overview of sequential models in machine learning. The basic Recurrent Neural Networks (RNN) layer is detailed first before describing some RNN variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). Then, we present encoder-decoder models. Finally, we introduce the transformers model which is the current state-of-the-art machine learning sequential model. Some Natural Language Processing (NLP) concepts including beam search algorithm and attention mechanisms are detailed. A complete review of other NLP concepts (language models, word embedding etc.) can be found in [38].

3.1.1 Recurrent Neural Networks (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 in [39], or speech-to-text for example in [40]. They have also been applied in generative models, for example in image captioning [41].

The basic RNN layer can be described as follows. 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 (3.1), where W is the weights matrix, b is the bias vector and ϕ is an activation function, for example *tanh* or ReLU [42]. x_t and y_{t-1} have been concatenated in a single row vector denoted $[x_t; y_{t-1}]$.

$$y_t = \phi(W \cdot [x_t; y_{t-1}] + b) \tag{3.1}$$

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 [43], [44]. 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.

In [45], Ba et al. introduce the *layer normalization* technique. Batch normalization is difficult to apply to recurrent layers since it requires a set of statistics for each time-step. Batch normalization computes the mean and variance for each neuron across the mini-batch while layer normalization computes the mean and variance across all the neurons of the same layer.

Long Short-Term Memory

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 [46]. 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 [47], and then improved over the years, see for example [48] and [49]. LSTM were designed to be faster to train than regular RNN, and to be able to detect long-term dependencies. Contrary to the basic RNN layer, the LSTM has two state vectors denoted h_t (the hidden state) and c_t (the 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 (3.2) to (3.7). Symbol $\cdot$ represents the matrix multiplication, while symbol $\times$ 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 (3.2) to (3.4)). 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 (3.5)). Equation (3.5) can be seen as the LSTM equivalent of the equation of the basic RNN layer (Equation (3.1)). In Equation (3.6), 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 as the ReLU), which is then passed though the output gate (Equation (3.7)). 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.

$$i_t = \sigma(W_i \cdot [x_t; h_{t-1}] + b_i) \tag{3.2}$$
$$f_t = \sigma(W_f \cdot [x_t; h_{t-1}] + b_f) \tag{3.3}$$
$$o_t = \sigma(W_o \cdot [x_t; h_{t-1}] + b_o) \tag{3.4}$$
$$\tilde{c}_t = tanh(W_c \cdot [x_t; h_{t-1}] + b_c) \tag{3.5}$$
$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t \tag{3.6}$$
$$h_t = o_t \times tanh(c_t) \tag{3.7}$$

A variant of LSTM introduced in [50]) 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 [51]. Its equations are detailed in Equations (3.8) to (3.11). The GRU is a simplified version of the LSTM, but seems to have the same performance as investigated in [52]. 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 (3.8) and (3.11)). The output gate is replaced by another gate controller r_t (Equation (3.9)), placed between the previous hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$ (Equation (3.10)).

$$z_t = \sigma(W_z \cdot [x_t; h_{t-1}] + b_z) \tag{3.8}$$
$$r_t = \sigma(W_r \cdot [x_t; h_{t-1}] + b_r) \tag{3.9}$$
$$\tilde{h}_t = tanh(W_h \cdot [x_t; r_t \times h_{t-1}] + b_h) \tag{3.10}$$
$$h_t = z_t \times h_{t-1} + (1 - z_t) \times \tilde{h}_t \tag{3.11}$$

An improvement of the RNN structure is the Bidirectional Recurrent Neural Network (BRNN) introduced in [53]. 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.

3.1.2 Encoder-Decoder Models

The first introduction of an encoder-decoder model for automatic translation is provided in [39]. This model has also been applied to various tasks, such as speech recognition [54] or video captioning [55].

Encoder-decoder models are used to map a fixed-length input sequence to a fixed-length output sequence where the length of the input differs from the length of the output. The model is composed of two parts: an encoder network and a decoder network. The encoder network is made of recurrent layers (such as LSTM or GRU layers). It receives in input the input sequence and encodes it into one or more encoding vectors, define as the hidden states computed during the last pass through the recurrent layers. The decoder network is also made of recurrent layers. The hidden states of these layers are initialized with the encoding vectors computed by the encoder network. The sequence used as input for the recurrent layers of the decoder network is dependent on the ongoing process: training or inference. This point is detailed below. Either way, given the initial hidden states and a suitable input sequence, the decoder network is able to compute an output sequence with a length different from the input sequence's length.

An important aspect of this architecture is the difference between the training procedure (when the parameters of the network are optimized) and the inference procedure (when an output sequence is generated by the network given an input sequence). The training procedure is straightforward. A pair of input and output sequences is provided. The input of the decoder network is defined as the true output sequence shifted from one timestamp to the right, such that the decoder network receives as input the vector it should have predicted at the previous timestamp. The first element of the input of the decoder network is simply a zero vector. An error (for example the mean squared error) can then be computed between the predicted sequence and the true output sequence, and be backpropagated to compute the gradients of the error with regards to each parameter of the network.

During the inference procedure, the true output sequence is obviously not known, hence the input of the decoder network has to be redefined. A simple method consists in recursively using the prediction of the last timestamp as the input for the next one. However, this method will certainly accumulate prediction errors and lead to poor performances if the output sequence is long enough. A solution to this issue is to change the task of the decoder network, such that the network do not

predict an element of the output sequence at each timestamp, but rather a distribution over the possible outputs. Then, using a beam search algorithm, it is possible to maintain at each timestamp a small number of possible output sequences using the partial likelihood of each sequence as the metric. When the last timestamp is reached, the model can output the sequence with the highest likelihood.

We illustrate the beam search algorithm with an example. Let e and d be respectively the trained encoder and the trained decoder. The encoder e is a function that takes an input sequence and returns an encoding vector v. The decoder d is a function that takes the encoding vector v as well as the beginning of an output sequence $y_1, \dots, y_t$ and returns the distribution $P(y_{t+1}|v, y_1, \dots, y_t)$ of the next element of the output sequence y_{t+1} given the encoding vector and the first terms of the output sequence. Let $x_1, \dots, x_{T'}$ be an input sequence. For example, $x_1, \dots, x_{T'}$ can be seen as a sentence where each x_i is the representation of a word obtained with word embedding (see [38] for more details about word embedding). We suppose in this example that the output sequences $y_1, \dots, y_T$ are built with elements y_t taken from a finite dictionary D. In the case of the automatic translation task, each y_t is a word from the target language. Our final objective would be to compute the distribution $P(y_1, \dots, y_T|x_1, \dots, x_{T'})$ and then take the argmax to get the output sequence $y_1, \dots, y_T$ maximizing this distribution given the input sequence. This distribution is unfortunately impossible to compute in a reasonable amount of time, this is why an heuristic is used, namely the beam search algorithm. The first step is to compute the encoding vector $v = e(x_1, \dots, x_{T'})$. Then, the distribution $P(y_1|v) = d(v)$ is computed, using a zero vector as input of the decoder network. We introduce here a parameter B of the beam search corresponding to the number of sequences that will be kept at each timestamp. So, using the distribution $P(y_1|v)$, we store the B sequence starts with the highest partial likelihood $y_1^1, \dots, y_1^B$. Then, each of these sequence starts are used as inputs of the decoder: for every j from 1 to B, we compute the distribution $P(y_2|v, y_1^j)$. For every j from 1 to B and for every element y_2 in the dictionary D, we can compute the partial likelihood $P(y_1^j, y_2|v) = P(y_2|v, y_1) \times P(y_1^j|v)$. Once again, we only keep the B partial sequence y_1, y_2 with the highest partial likelihood $P(y_1^j, y_2|v)$. By repeating this process for each timestamp, we end up with B complete output sequences $y_1, \dots, y_T$. It is then possible to output the sequence with the highest likelihood among these B sequences. Note there is no guarantee that this is the sequence that maximize $P(y_1, \dots, y_T|v)$.

The full potential of this basic framework is only achieved when it is combined with several improvements. For example, reversing the order of the input sequence has been shown to significantly improve the performance of the model [39]. Another family of improvements used in many deep learning models is attention mechanisms. The idea behind attention mechanisms is that, at each timestamp of the decoding process, the decoder should focus on specific parts of the input sequence, and not on the whole input sequence as encoded into the encoding vector. To achieve this goal, the decoder should be able to access all the hidden states of the encoder network (one for each element of the input sequence) rather than only access the last hidden state, namely the encoding vector. Several architectures are possible to implement an attention mechanism, see for example [56] and [57]. The general idea is to train a multilayer perceptron taking as input the last decoder hidden state as well as all the encoder hidden states, and outputting a vector of weights which sum to 1 (using a softmax activation), such that each hidden state of the encoder is associated with a weight. It is then possible to sum all the hidden states of the encoder weighted by the output of the multilayer perceptron, resulting in a vector called the context vector. This context vector along with the last output of the decoder can been used as the input for the next timestamp of the decoder.

What has been described here is in fact a special case of attention-based mechanisms called

General Attention. Another type of attention mechanisms are Self-Attention, which apply attention only among the inputs. This type of attention mechanisms is notably used in the Transformers architecture introduced in [58] which is currently the state-of-the-art architecture for most NLP tasks.

3.1.3 Transformers

In [58], Vaswani et al. first introduce the Transformer model aiming at replacing the recurrent encoder-decoder models previously used in automatic translation. The Transformer model is both more efficient and faster to train since it is parallelizable contrary to recurrent layers. It is based entirely on attention, replacing the recurrent layers by multi-headed self-attention. The multi-headed self-attention has three interesting properties:

- It is parallelizable, like convolutional layers but unlike recurrent layers.
- It has a smaller computational complexity than convolutional and recurrent layers if the length of the sequence is smaller than the representation dimension.
- Its maximum path-length is constant (unlike convolutional and recurrent layer) which greatly improve the capacity of the layer to learn long-term dependencies.

The Transformer model also uses positional encoding to provide the order of the sequence to the network, skip-connections and layer normalization. Vaswani et al. apply their model to automatic translation tasks using an encoder-decoder framework where the decoder is auto-regressive, consuming the previously generated symbols as input when generating the text.

The details of the multi-headed self-attention layer are presented in 3.4 where it is extended to estimate the uncertainty in the prediction of the model.

3.1.4 Convolutional Neural Networks (CNN)

In this paragraph, we sketch a short presentation of the YOLO algorithm introduced in [59] to address the object detection task. Object detection has to be distinguished from object localization. In object localization, an image is fed into the model and the output is a class, for example the image is classified as a 'plane' or a 'car' etc, along with the position of the object in the image. However, object localization do not provide any information on the number of such objects, which could possibly fall into different classes. On the contrary, the object detection task aims at providing the locations, shapes, numbers and classes of all the objects (possible none) of the image. An immediate application of object detection algorithm is the field of self-driving cars, where an algorithm has to detect the positions of various classes of objects (cars, pedestrians, traffic signs etc.) within a time constraint.

Several variants of YOLO have been developed during the last years. In this brief introduction, we will only provide one simple approach. YOLO is an acronym for "You Only Look Once". Former object detection models were based on classification models. To detect objects, the image has to be fed several times into a classification model, each time with a different cropping using sliding windows covering the entire image. The idea of the YOLO algorithm is to fed the image only one time into the model, hence significantly improving the time performances of the object detection task, while learning a general representation of the image (rather than a local one with the sliding windows approach). To achieve this goal, each image is divided into a $N \times N$ grid. For each grid cell, a bounding box is defined. A bounding box is represented by a vector of $5 + C$ parameters where C is the number of classes. The first five parameters are the following: the coordinates x, y of

the center of the box, the width w and height h of the box, and the probability p that the bounding box contains an object. The last C parameters are the class probabilities $P(Class_i|Object)$ representing the probability that the grid cell contains an object of class i given the fact that it contains an object. The predictions are therefore encoded as three dimensional tensor $N \times N \times (5 + C)$. To build a training dataset given a set of images, a bounding box is associated to each significant object and associated to the grid cell containing the center of the bounding box. An issue that may arise from this encoding is that, since an object is likely to span over several grid cells, the same object may be detected by the model for several grid cells, then leading to several bounding boxes around the same object. To solve this issue, it is possible to use a method called non-maximum suppression presented in Algorithm 1.

Algorithm 1 (Non-maximum suppression)**.**
Inputs
The bounding boxes predicted by the model associated with their respective class
A threshold thr
Outputs
The set R of remaining bounding boxes
Algorithm

```
R = {}
for every class c among the C classes do
    Let S be the set of all bounding boxes associated to class c
    while S is not empty do
        Let b be the box from S with the highest probability p
        S = S \ {b}
        R = R ∪ {b}
        for every box b' in S do
            if the intersection over union of b and b' is greater than thr then
                S = S \ {b'}
            end if
        end for
    end while
end for
```

The idea of non-maximum suppression is simply to keep only the box with the highest probability when several boxes are overlapping. The notion of overlapping is defined through intersection over union. The intersection over union of two boxes is the ratio between the area of the intersection of the boxes over the area of their union.

Since this is not an objective of this report, we will simply sketch the principles of the Convolutional Neural Networks (CNN) models used to implement the YOLO algorithm. An input image is described as $M \times M \times K$ tensor where M is the width and the height of the image (we consider square image for simplicity) and K is the number of channels of the image (generally equal to three, for the three RGB channels). Convolutional and pooling layers then transform this tensor into successive tensors with different dimensions, finally outputting a $N \times N \times (5+C)$ tensor which can be interpreted as a grid containing bounding boxes as explained above. The loss between the output tensors and the true tensors is then computed and backpropagated to update the parameters of the network. In [59], the authors introduce a special loss for the YOLO algorithm, which is a modified Mean Squared Error. We provide a simplified version of this loss in Equation 3.12. $\mathbb{1}_i^{obj}$ is

equal to 1 when the grid cell i contains an object in the true tensor and 0 otherwise. We have that $\mathbb{1}_i^{noobj} = 1 - \mathbb{1}_i^{obj}$. The constants λ_{coord} and λ_{noobj} aim at weighting the various terms of the loss. λ_{coord} is generally set above 1 to give a higher weight for localization compared to classification. λ_{noobj} is set below 1 to reduce the impact of the probability of object in grid cell where there is no object. Indeed, the majority of grid cell do not contain object hence pushing the probability of object toward zero and creating instability with respect to grid cells actually containing objects. The errors on widths w and heights h of the bounding boxes are computed using square roots in order to penalize more the errors on small boxes with respect to large boxes. Finally, it should be noticed that the prediction of position, dimension and class probabilities are only taken into account in grid cells actually containing objects.

$$\begin{aligned} L(Y, \tilde{Y}) = \quad & \lambda_{coord} \sum_{i=0}^{N^2} \mathbb{1}_i^{obj} [(x_i - \tilde{x}_i)^2 + (y_i - \tilde{y}_i)^2] \\ + \quad & \lambda_{coord} \sum_{i=0}^{N^2} \mathbb{1}_i^{obj} [(\sqrt{w_i} - \sqrt{\tilde{w}_i})^2 + (\sqrt{h_i} - \sqrt{\tilde{h}_i})^2] \\ + \quad & \sum_{i=0}^{N^2} \mathbb{1}_i^{obj} (p_i - \tilde{p}_i)^2 \\ + \quad & \lambda_{noobj} \sum_{i=0}^{N^2} \mathbb{1}_i^{noobj} (p_i - \tilde{p}_i)^2 \\ + \quad & \sum_{i=0}^{N^2} \mathbb{1}_i^{obj} \sum_{c \in C} (P_i(c) - \tilde{P}_i(c))^2 \end{aligned} \tag{3.12}$$

3.2 Optimization

This paragraph provides a general overview of the classical optimization techniques used to train deep learning models. More specifically, we will focus on first order gradient descent methods, from Stochastic Gradient Descent (SGD), its variants with momentum, then adaptive learning rates leading to the Adam algorithm. All the references of the methods mentioned in this paragraph can be found in chapter 8 of [60].

Let's consider the supervised and unregularized machine learning problem. The data (x, y) are generated according to a distribution p_{data}. The final goal is to minimize the expected generalization error (also called the risk) provided in Equation 3.13, where f is the model using the data x and the parameters θ to predict y and L is a loss function. Since the machine learning model has only access to a finite training set, we only know the empirical distribution $\tilde{p}_{data}$ over the training set. Hence, the objective of the machine learning model will be to minimize the empirical risk given in Equation 3.14, where m is the size of the training set.

$$J^*(\theta) = \mathbb{E}_{(x,y) \sim p_{data}} L(f(x, \theta), y) \tag{3.13}$$

$$J(\theta) = \mathbb{E}_{(x,y) \sim \tilde{p}_{data}} L(f(x, \theta), y) = \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}, \theta), y^{(i)}) \tag{3.14}$$

When the error function L features bad optimization properties, it can be replaced by a surrogate loss function. This is the case in a binary classification task where the original loss function takes

discrete values 0 or 1, and is not differentiable. In this example, the loss function can be replaced by the negative log-likelihood which will estimate the probability of belonging to each class. Another common issue of empirical risk minimization is overfitting. If the model is able to memorize the training set, it will achieve a very low value on $J(\theta)$ but will not be able to generalize well on $J^*(\theta)$. To avoid this issue, a machine learning algorithm will halt using an early stopping criterion generally based on the performances over the real error function. This means that, contrary to classical optimization, a machine learning algorithm will not converge to a local minimum (the final gradients may be very large).

We will now briefly review some machine learning algorithms. First, the Stochastic Gradient Descent (SGD) uses an estimate of the gradient as a descent direction. The main difference with a classical gradient descent is that, at each iteration, the SGD only considers a *minibatch* of m training example. Indeed, if the examples of the minibatch are drawn i.i.d. from the distribution p_{data}, then the average gradient over the minibatch is an unbiased estimate of the true gradient of the underlying expected generalization error $J*$. Of course, if the algorithm is trained over several *epochs*, meaning that the algorithm will pass through the whole training set several times, then we will obtain estimates of the gradient of the empirical risk J and not $J*$, since the minibatches will then be drawn from $\tilde{p}_{data}$. The main advantages of SGD is that the computational time of each iteration does not depend on the size of the full training set. When the size of the minibatch is the size of the training set, then the algorithm is called batch gradient descent. When the minibatch contains only one training example, it is referred as online gradient descent. SGD is presented in Algorithm 2.

Algorithm 2 (Stochastic Gradient Descent)**.**
Inputs
Learning rate schedule $\epsilon_1, \epsilon_2, \cdots$
Initial parameter θ
Algorithm
$k = 1$
while stopping criterion not met **do**
Sample a minibatch from the training set $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$
Compute gradient estimate $\tilde{g} = \frac{1}{m} \sum_i \nabla_\theta L(f(x^{(i)}, \theta), y^{(i)})$
Apply update $\theta = \theta - \epsilon_k \tilde{g}$
$k = k + 1$
end while

Since the random sampling of the minibatch introduces noise in the estimaton of the gradient, the SGD algorithm has to reduce the learning rate ϵ at each iteration to converge. Sufficient conditions for the convergence of SGD are the following:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \tag{3.15}$$

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty \tag{3.16}$$

A first approach to accelerate learning with SGD is the method of momentum. The momentum algorithm consists in introducing a momentum p corresponding to an exponentially decaying moving

average of past gradients. This momentum v is the velocity at which the parameters moves through the search space. This is an analogy with the physical laws of motion where the negative gradients are seen as forces acting on a particle moving in the search space. The convergence is accelerated by reducing the variance in the SGD and by reducing the effect of poor Hessian conditioning. We recall that poor Hessian conditioning corresponds to a high ratio between the highest and the lowest eigenvalues of the Hessian matrix, and leads to oscillating behavior when using SGD. Considering Algorithm 2, the momentum algorithm consists in replacing the update phase by the two steps presented in Equations 3.17 and 3.18.

$$v = \alpha v - \epsilon \tilde{g} \tag{3.17}$$

$$\theta = \theta + v \tag{3.18}$$

where $\alpha \in [0, 1)$ is an hyperparameter corresponding to the strength of the exponential decay. The larger α is, the more past gradients will impact the current step. This hyperparameter can be seen as a viscous drag.

Another approach to accelerate learning are the adaptive learning rates algorithms. The general idea is to automatically adapt a learning rate for each direction of the search space during training. AdaGrad scales the learning rates proportionally to the inverse square root of the sum of the past squared gradients, such that greater progress is achieved in the direction of less steep slope. The squared gradients are accumulated in a parameter r initialized to zero. The update phase of SGD becomes Equations 3.19 and 3.20.

$$r = r + \tilde{g} \times \tilde{g} \tag{3.19}$$

$$\theta = \theta - \frac{\epsilon}{\delta + \sqrt{r}} \times \tilde{g} \tag{3.20}$$

where $\times$ is the element-wise multiplication, ϵ is a global learning rate, and δ is a small constant for numerical stability. The main issue of AdaGrad is that it accumulates the squared gradients from the start of the algorithm which induces a fast decrease of the learning rates, eventually slowing the learning during the last iterations. In fact, AdaGrad achieves fast convergence in the convex setting but is not well suited for nonconvex optimization. To address this issue, the RMSProp algorithm modifies the AdaGrad algorithm by changing the accumulation of squared gradients by a exponentially weighted moving average, enabling the algorithm to forget the first gradients that may be associated to very different structures. In RMSProp, the update phase of the SGD is replaced by Equations 3.21 and 3.22.

$$r = \rho r + (1 - \rho)\tilde{g} \times \tilde{g} \tag{3.21}$$

$$\theta = \theta - \frac{\epsilon}{\sqrt{\delta + r}} \times \tilde{g} \tag{3.22}$$

where ρ is an hyperparameter controlling the length scale of the moving average. RMSProp can also be hybridized with momentum method. This leads to a new adaptive learning rate optimization method called Adam (for "adaptive moments") first introduced in [61]. The Adam algorithm computes estimates of the first-order moment s and uncentered second-order moment r of the gradients, with exponential weighting. The momentum is not applied to the rescaled gradients (as with RMSProp + momentum) but directly incorporated during the first-order moment estimation.

Adam also includes a bias correction of both first-order and second-order moments. The update phase of Adam is provided in Equations 3.23 through 3.23.

$$s = \rho_1 s + (1 - \rho_1)\tilde{g} \tag{3.23}$$

$$r = \rho_2 r + (1 - \rho_2)\tilde{g} \times \tilde{g} \tag{3.24}$$

$$\hat{s} = \frac{s}{1 - \rho_1^k} \tag{3.25}$$

$$\hat{r} = \frac{r}{1 - \rho_2^k} \tag{3.26}$$

$$\theta = \theta - \epsilon \frac{\hat{s}}{\sqrt{\delta + \hat{r}}} \tag{3.27}$$

where ρ_1 and ρ_2 are the exponential decay rates for moments estimates. Note that in Equation 3.27, the operations of the updating term are applied element-wise. The Adam algorithm is currently one of the most widely used optimization algorithm for deep learning applications. In particular, all the experiments conducted in this report will used the Adam algorithm to train the various models.

3.3 Uncertainty in Deep Learning

In [62], Dera et al. introduce an extended variational inference (eVI) framework for propagating uncertainty in Convolutional Neural Networks (CNN). The weights of the filters of the convolutional layers are modelled as Tensor Normal Distribution. The weights of the fully-connected layers are also modelled as Gaussian vectors. As part of the VI framework, the ELBO is used as the objective function of the network. To estimate the expected log-likelihood (first term of the ELBO), the first and second moments of the weights distributions are propagated through the various layers (convolutional, activation, pooling, fully-connected). Once the posterior distribution is known, the expected log-likelihood is estimated with Monte Carlo sampling, and the full ELBO can be derived for the CNN. The eVI-CNN is compared to other Bayesian networks as well as to classical CNN, on MNIST and CFAR-10 datasets. eVI-CNN is shown to be able to resist Gaussian noise and adversarial attacks far better than other frameworks. Moreover, the propagated variance can be used to quantify the uncertainty of the network.

In [63], Dera et al. apply the eVI-CNN framework to the classification of synthetic aperture radar (SAR) images. These are satellite images than can be used to classify the type of surface in each part of the image (water, crops, buildings etc.). Once again, the eVI framework is able to resist Gaussian noise and adversarial attack. It is also possible to generate an uncertainty map allowing to visualize the uncertainty of the prediction of the network on each area of the image.

In [64], Xue et al. propose a Bayesian Transformer Language Model (LM) applied to speech recognition. The model is composed of several Transformer decoders where only the first one is optimized with the VI method. A standard Transformer LM is used to set the mean of the prior distribution. The Bayesian Transformer LM can also be interpolated with the standard Transformer LM. The main results are a small increase in the generalization capability of the LM when using the Bayesian framework by comparison with the standard framework.

3.4 Application: Bayesian Transformers

Following the developments presented in Y. Gal PhD dissertation [65] and D. Dera PhD dissertation [66], we propose a variational inference (VI) framework for estimating the uncertainty in Transformers models.

3.4.1 The Transformer Model

Full Transformer model with stacked encoders and decoders

We denote by L the number of encoder blocks in the full model, and by l a given Transformer block ($1 \leq l \leq L$).

Transformer block

In this document, we will primarily focus on deriving the Variational Inference framework for one Transformer block. I is the features dimension of the input. J is the length of the input sequence. Hence, the input X has dimension $I \times J$. K is the dimension of the query and key vectors. D is the dimension of the value vector.

3.4.2 Normal Distribution over the Transformer's parameters

Each Transformer block contains the following parameters: $\{W_{hl}^Q, W_{hl}^K, W_{hl}^V\}_{h=1}^H, W_l^O, G_l^1, B_l^1, W_l^{F^1}, W_l^{F^2}, G_l^2, B_l^2$ where H is the number of attention heads. All parameters matrices are assumed to be independent to each other (but not the weights of a given matrix). We assume that the each vectorized matrix is drawn from a vector normal distribution.

3.4.3 Variational Inference

We set ourselves in the supervised learning framework. A dataset $D = \{(X, y)\}$ is given to us. This is the data we have observed so far, or the data that we can use to train our model i.e., the training set. D can also be seen as a random variable, and the training set is just a set of independent and identically distributed (i.i.d.) samples according to the distribution of D.

Now, a new input X^* is given to us. That means that we want to *generalize* our knowledge of D to a new input X^*. Our ultimate goal is to compute the *distribution* over the output, which is seen as a random variable and is denoted[1] by y^*. We want to know $p(y^*|X^*, D)$. Using the law of total probability and the fact that y^* and D are independent, we have that:

$$p(y^*|X^*, D) = \int_\Omega p(y^*|X^*, \omega)p(\omega|D)d\omega. \tag{3.28}$$

Eq. 3.28 is called **Bayesian inference**. What is the intuition behind it? Let us explain the terms one by one.

- $p(y^*|X^*, D)$ is the distribution of the output y^* given the input X^* and the dataset D that we have learned.

[1] Using the classical abuse of notations, the probability density function of a random variable $\mathbf{X}$, rigorously defined as $f_{\mathbf{X}}(.)$, will be denoted as $p(X)$ where X indicates at the same time the random variable and the value where the density function is evaluated. We also refer as "distribution" the density function.

- To obtain this distribution, we compute a (weighted) sum over the whole parameter space Ω. Each $\omega \in \Omega$ is sampled according to the *posterior distribution* $p(\omega|D)$. This represents what we have learned, i.e., we have used the training set D to learn the distribution of the parameters ω.

- For each ω properly sampled according to our knowledge, we compute the *likelihood* of y^* given the input X^* and our current choice of ω. This likelihood is the term $p(y^*|X^*, \omega)$. This term is simply our (trained) model, our neural network, or whatever it is. Our neural network has a fixed set of parameters ω, and given an input X^*, it outputs a distribution over y^*. In fact, this is true only for Bayesian models. However, classical models can be seen as outputting only $E[y^*|X^*, \omega]$. Or, we can say that classical models only compute the first moment (the mean) of the distribution $p(y^*|X^*, \omega)$.

So finally, what is Bayesian inference? Bayesian inference consists in computing the predictions of *all* possible models (in the family indexed by ω). However, each prediction is weighted by our confidence in the considered model (i.e., $p(\omega|D)$ where each ω is a different model) reflecting our knowledge (the dataset D).

In practice, as far I know, the distribution of y^* is estimated using Monte Carlo estimation. That means that we draw a sample of N vectors of parameters ω according to the distribution $p(\omega|D)$. For each sampled ω, we set our network parameters to this value of ω and we compute the output y^* using the input X^*. Hence, we obtain a sample of y^* approximating the distribution $p(\omega|D)$.

So, the real question here is: how to compute the posterior distribution $p(\omega|D)$. If we replace D by y, X, and we use the fact that X and ω are independent, we get:

$$p(\omega|y, X) = \frac{p(y|\omega, X)p(\omega|X)}{p(y|X)} = \frac{p(y|X, \omega)p(\omega)}{p(y|X)} = \frac{p(y|X, \omega)p(\omega)}{\int_\Omega p(y|X, \omega')p(\omega')d\omega'}, \tag{3.29}$$

where $p(y|X, \omega)$ is once again the likelihood, or our neural network. $p(\omega)$ is our *prior distribution* over the parameters ω. This can be seen as how we initialize the parameters of our network before the training (before seeing the data). $p(\omega)$ is generally set to a standard normal distribution, or a centred normal distribution with "smart" variance (to be developed). The denominator, called the *marginal likelihood* or the *evidence*, is the real problem here. It is intractable ...

... so this is why we are not going to use this (exact) formula at all to estimate $p(\omega|y, X)$. Instead, we will use a method called **Variational Inference** (VI). VI approximates $p(\omega|y, X)$ by a simpler distribution $q_\theta(\omega)$ called a *approximating variational distribution*. $\{q_\theta(\omega)\}$ is a family of distributions over ω indexed by θ (generally Gaussian distributions). Our goal is to find the θ such that the distribution $q_\theta(\omega)$ is the "closest" to $p(\omega|y, X)$ among the chosen family. To find such a distribution, we will minimize the Kullback-Leibler divergence $KL(q_\theta(\omega)||p(\omega|y, X))$ according to θ. The KL divergence is defined by:

$$KL(q_\theta(\omega)||p(\omega|y, X)) = -\int_\Omega q_\theta(\omega) \log \frac{p(\omega|y, X)}{q_\theta(\omega)} d\omega. \tag{3.30}$$

Using Eq. 3.29, $p(\omega|y,X) = \frac{p(y|X,\omega)p(\omega)}{p(y|X)}$, we obtain:

$$\begin{aligned} KL(q_\theta(\omega)||p(\omega|y,X)) &= -\int_\Omega q_\theta(\omega) \log \frac{p(y|X,\omega)p(\omega)}{p(y|X)q_\theta(\omega)} d\omega, \\ &= -\int_\Omega q_\theta(\omega) \log \frac{p(y|X,\omega)p(\omega)}{q_\theta(\omega)} d\omega + \log p(y|X) \int_\Omega q_\theta(\omega) d\omega, \\ &= -\int_\Omega q_\theta(\omega) \log \frac{p(y|X,\omega)p(\omega)}{q_\theta(\omega)} d\omega + \log p(y|X). \end{aligned}$$

Hence, we can rewrite the log marginal likelihood $\log p(y|X)$ by:

$$\log p(y|X) = \text{constant} = KL(q_\theta(\omega)||p(\omega|y,X)) + L(\theta,\omega,D), \tag{3.31}$$

where:

$$L(\theta,\omega,D) = \int_\Omega q_\theta(\omega) \log \frac{p(y|X,\omega)p(\omega)}{q_\theta(\omega)} d\omega, \tag{3.32}$$

is the variational lower bound, or evidence lower bound (ELBO) on the log marginal likelihood. Indeed, given the fact that the KL-divergence is non-negative, we have: $\log p(y|X) \geq L(\theta,\omega,D)$. Since $\log p(y|X)$ is constant w.r.t θ, minimizing $KL(q_\theta(\omega)||p(\omega|y,X))$ is equivalent to maximizing the ELBO, $L(\theta,\omega,D)$. We can rewrite the ELBO as:

$$\begin{aligned} L(\theta,\omega,D) &= \int_\Omega q_\theta(\omega) \log p(y|X,\omega) d\omega + \int_\Omega q_\theta(\omega) \log \frac{p(\omega)}{q_\theta(\omega)} d\omega, \\ &= \mathrm{E}_{q_\theta(\omega)}[\log p(y|X,\omega)] - KL(q_\theta(\omega)||p(\omega)), \end{aligned}$$

where $\mathrm{E}_{q_\theta(\omega)}[\log p(y|X,\omega)]$ is the *expected log-likelihood*, and $-KL(q_\theta(\omega)||p(\omega))$ is a regularization term called *complexity loss*. The regularization term encourages the variational distribution $q_\theta(\omega)$ to remain not too far from the prior distribution $p(\omega)$. Since $p(\omega)$ is a simple distribution (like a standard Gaussian), the variational distribution is encouraged to remain not too "complex".

Since the data points of D are assumed to be i.i.d., the ELBO can be estimated by the sum of the individual ELBO of all data points:

$$L(\theta,\omega,D) = \mathrm{E}_{\tilde{p}(X,y)}[L(\theta,\omega,y|X)] \approx \sum_{(X,y)\in D} L(\theta,\omega,y|X), \tag{3.33}$$

where $\tilde{p}(X,y)$ is the empirical distribution from which the training set has been sampled.

3.4.4 Derivation of the Expected log-likelihood

To estimate the expected log-likelihood $\mathrm{E}_{q_\theta(\omega)}[\log p(y|X,\omega)]$, we use Monte Carlo estimation where M parameters points are drawn from the distribution $q_\theta(\omega)$:

$$\mathrm{E}_{q_\theta(\omega)}[\log p(y|X,\omega)] \approx \frac{1}{M} \sum_{m=1}^{M} \log p(y|X,\omega^{(m)}) \tag{3.34}$$

To compute $\log p(y|X,\omega^{(m)})$, we need to know the distribution $p(y|X,\omega)$ which depends on the distribution $q_\theta(\omega)$ over the parameters. By assuming that $p(y|X,\omega)$ is Gaussian, we only need to propagate the mean and covariance matrix of $q_\theta(\omega)$ through our neural network from the parameters ω to the outputs y. In this document, we propagate the mean and covariance matrix along one Transformer block with multi-head attention. The flow diagram of one Transformer block is shown in Figure 3.1.

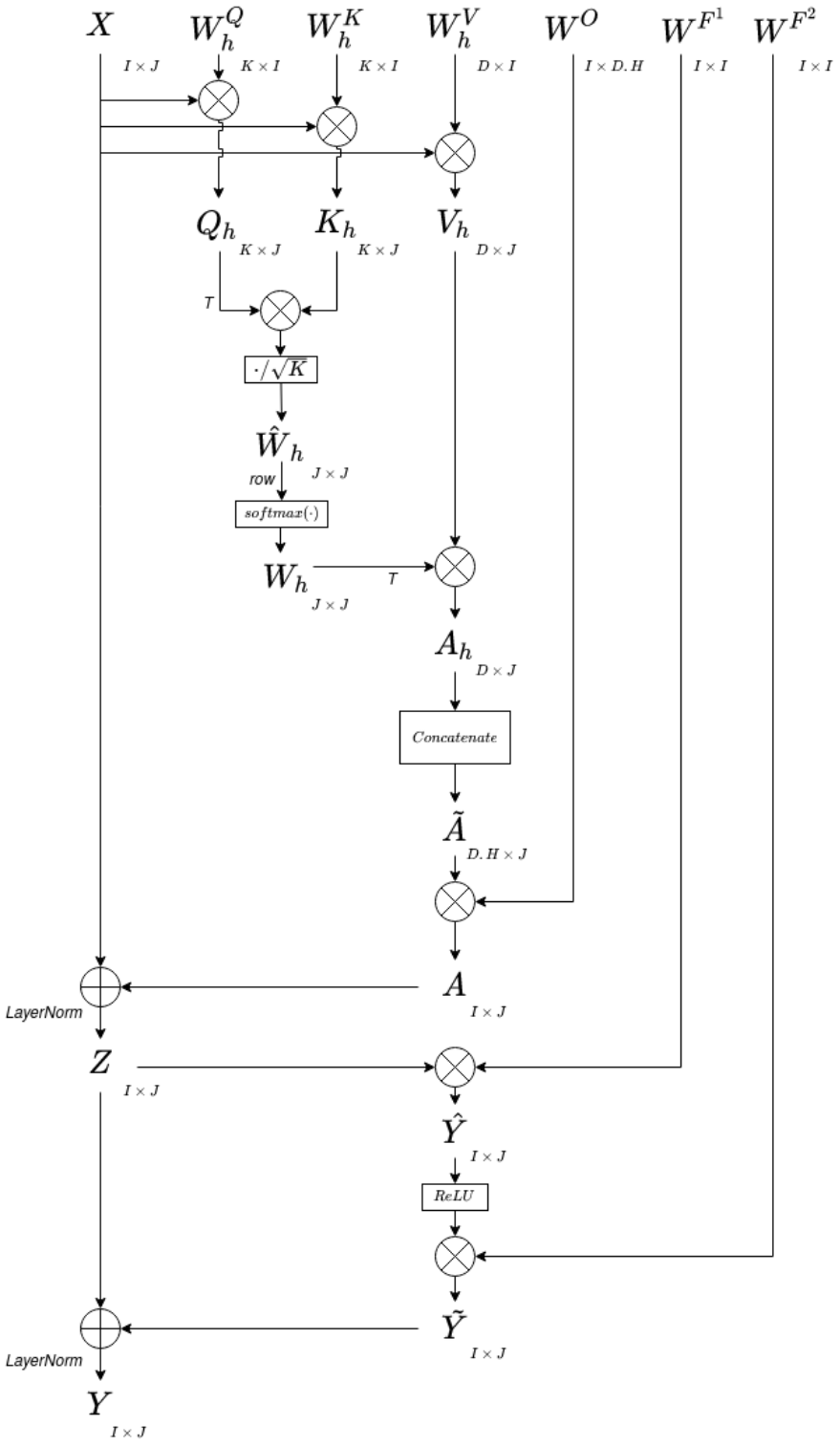

Figure 3.1: Flow diagram of one Transformer block. The Layer Normalization parameters are not represented.

Multiplication of two independent random matrices

We prove a result that is used several time in the derivation of the expected log-likelihood. Let A be a $p \times n$ random matrix and B a $p \times q$ random matrix. Let M^A be the mean of A (with dimension $p \times n$), M^B the mean of B (with dimension $p \times q$), Σ^A the covariance matrix of $\text{vec}(A)$ (with dimension $np \times np$), and Σ^B the covariance matrix of $\text{vec}(B)$ (with dimension $qp \times qp$). A and B are assumed to be independent. We denote by $a_1, \cdots, a_n$ the columns of A, and by $b_1, \cdots, b_q$ the columns of B. The elements of the mean matrices M^A and M^B are denoted by $\mu^A_{i,j}$ and $\mu^B_{i,j}$.

Proposition 1.

$$E[A^T B] = (M^A)^T M^B. \tag{3.35}$$

Proof. The expectation of a matrix is the expectation of each element of the matrix. Given a column a_i and a column b_j, we need to compute $E[a_i^T b_j]$. We use the trace trick. We have that $E[a_i^T b_j] = E[\text{tr}(a_i^T b_j)]$ since $a_i^T b_j$ is a scalar. Then, $E[\text{tr}(a_i^T b_j)] = E[\text{tr}(b_j a_i^T)]$ using the fact that $\text{tr}(UV) = \text{tr}(VU)$. Hence, $E[a_i^T b_j] = E[\sum_{k=1}^p B_{k,j} A_{k,i}] = \sum_{k=1}^p E[B_{k,j} A_{k,i}]$ by linearity of expectation. Then, $E[a_i^T b_j] = \sum_{k=1}^p E[B_{k,j}]E[A_{k,j}]$ by independence of A and B. Finally, $E[a_i^T b_j] = \sum_{k=1}^p \mu^B_{k,j}\mu^A_{k,i} = (\mu_i^A)^T \mu_j^B$ where μ_i^A is the i-th column of M^A and μ_j^B is the j-th column of M^B. □

The vec operation is done by stacking the columns of the matrix (and not the rows) as a column vector. If we denote the cross-covariance matrix of the columns a_i and a_j by Σ^{a_i,a_j} (with dimension $p \times p$), we can write:

$$\Sigma^A = \begin{bmatrix} \Sigma^A_{a_1a_1} & \Sigma^A_{a_1a_2} & \cdots & \Sigma^A_{a_1a_n} \\ \Sigma^A_{a_2a_1} & \Sigma^A_{q_2q_2} & \cdots & \Sigma^A_{a_2a_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma^A_{a_na_1} & \Sigma^A_{a_na_2} & \cdots & \Sigma^A_{a_na_n} \end{bmatrix}.$$

We can do the same thing for B. The covariance matrix of $\text{vec}(A^T B)$ is denoted by Σ with dimension $nq \times nq$ and with elements $\sigma_{i,j}$ such that $\text{Cov}[a^T_{k_1} b_{m_1}, a^T_{k_2} b_{m_2}] = \sigma_{i,j}$ where $k_1 = ((i-1) \bmod n) + 1$, $m_1 = \lfloor \frac{i-1}{n} \rfloor + 1$, $k_2 = ((j-1) \bmod n) + 1$, $m_2 = \lfloor \frac{j-1}{n} \rfloor + 1$.

Proposition 2. For all $1 \leq i \leq np$ and $1 \leq j \leq np$, we have:

$$\sigma_{i,j} = \text{tr}(\Sigma^A_{a_{k_1}a_{k_2}} \Sigma^B_{b_{m_1}b_{m_2}}) + \mu^A_{k_1} \Sigma^B_{b_{m_1}b_{m_2}} (\mu^A_{k_2})^T + \mu^B_{m_1} \Sigma^A_{a_{k_1}a_{k_2}} (\mu^B_{m_2})^T, \tag{3.36}$$

where $k_1 = ((i-1) \bmod n) + 1$, $m_1 = \lfloor \frac{i-1}{n} \rfloor + 1$, $k_2 = ((j-1) \bmod n) + 1$, $m_2 = \lfloor \frac{j-1}{n} \rfloor + 1$ or equivalently $i = (m_1 - 1)n + k_1$, $j = (m_2 - 1)n + k_2$.

Proof.

$$\begin{aligned}
\text{Cov}[a^T_{k_1} b_{m_1}, a^T_{k_2} b_{m_2}] &= E[a^T_{k_1} b_{m_1} a^T_{k_2} b_{m_2}] - E[a^T_{k_1} b_{m_1}]E[a^T_{k_2} b_{m_2}] \\
&= E[a^T_{k_1} b_{m_1} b^T_{m_2} a_{k_2}] - E[a^T_{k_1} b_{m_1}]E[a^T_{k_2} b_{m_2}] \\
&= E[\text{tr}(a^T_{k_1} b_{m_1} b^T_{m_2} a_{k_2})] - E[a^T_{k_1} b_{m_1}]E[a^T_{k_2} b_{m_2}] \\
&= E[\text{tr}(b_{m_1} b^T_{m_2} a_{k_2} a^T_{k_1})] - E[a^T_{k_1} b_{m_1}]E[a^T_{k_2} b_{m_2}] \\
&= \text{tr}(E[b_{m_1} b^T_{m_2} a_{k_2} a^T_{k_1}]) - E[a^T_{k_1} b_{m_1}]E[a^T_{k_2} b_{m_2}] \\
&= \text{tr}(E[b_{m_1} b^T_{m_2}]E[a_{k_2} a^T_{k_1}]) - E[a^T_{k_1} b_{m_1}]E[a^T_{k_2} b_{m_2}] \\
&= \text{tr}((\Sigma^B_{b_{m_1}b_{m_2}} + \mu^B_{m_1}(\mu^B_{m_2})^T)(\Sigma^A_{a_{k_1}a_{k_2}} + \mu^A_{k_1}(\mu^A_{k_2})^T)) - (\mu^A_{k_1})^T \mu^A_{k_2} (\mu^B_{m_1})^T \mu^B_{m_2} \\
&= \text{tr}(\Sigma^A_{a_{k_1}a_{k_2}} \Sigma^B_{b_{m_1}b_{m_2}}) + \mu^A_{k_1} \Sigma^B_{b_{m_1}b_{m_2}} (\mu^A_{k_2})^T + \mu^B_{m_1} \Sigma^A_{a_{k_1}a_{k_2}} (\mu^B_{m_2})^T.
\end{aligned}$$

□

Linear layers: queries, keys, and values

Let X be the input of the Transformer model. X has dimension $I \times J$. The first operation is the computation of the queries, keys and values vectors:

$$\begin{aligned} Q_h &= W_h^Q X, \\ K_h &= W_h^K X, \\ V_h &= W_h^V X. \end{aligned}$$

To compute the means M^Q, M^K, M^V and covariance matrices Σ^Q, Σ^K, Σ^V, we apply Prop. 1 and Prop. 2 using the re-indexing presented in subsection 3.4.4 and with $M^B = X$ and $\Sigma^B = 0$ if X is a constant matrix.

Scaled dot-product attention

Mean and covariance of $Q_h^T K_h$ Since W_h^Q and W_h^K are independent, Q_h and K_h are also independent. Let M^Q be the mean of Q_h and M^K the mean of K_h, both of which are $K \times J$ matrices. Using Prop. 1, we have that:

$$E[Q_h^T K_h] = (M^Q)^T M^K.$$

The covariance matrix of $\text{vec}(Q_h)$ is Σ^Q with dimension $K.J \times K.J$. Similarly, the covariance matrix of $\text{vec}(K_h)$ is Σ^K. If we denote the columns of Q_h by $q_1, q_2, \cdots, q_J$, and the cross-covariance matrix of the columns q_i and q_j by $\Sigma^{q_i q_j}$, then we have:

$$\Sigma^Q = \begin{bmatrix} \Sigma^Q_{q_1 q_1} & \Sigma^Q_{q_1 q_2} & \cdots & \Sigma^Q_{q_1 q_J} \\ \Sigma^Q_{q_2 q_1} & \Sigma^Q_{q_2 q_2} & \cdots & \Sigma^Q_{q_2 q_J} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma^Q_{q_J q_1} & \Sigma^Q_{q_J q_2} & \cdots & \Sigma^Q_{q_J q_J} \end{bmatrix}.$$

Using Prop. 2, The covariance matrix of $Q_h^T K_h$ is $\Sigma^{Q^T K}$ (with dimension $J^2 \times J^2$) where the i,j-th element is:

$$\sigma_{i,j}^{Q^T K} = \text{tr}(\Sigma^Q_{q_a q_b} \Sigma^K_{k_c k_d}) + (\mu^Q_{q_a})^T \Sigma^K_{k_c k_d} \mu^Q_{q_b} + (\mu^K_{k_c})^T \Sigma^Q_{q_a q_b} \mu^K_{k_d},$$

where $a = ((i-1) \bmod J) + 1, b = ((j-1) \bmod J) + 1, c = \lfloor \frac{i-1}{J} \rfloor + 1, d = \lfloor \frac{j-1}{J} \rfloor + 1$.

Scaling and softmax activation Let us denote $\frac{Q_h^T K_h}{\sqrt{K}} = \hat{W}_h$. Then, we have $E[\hat{W}_h] = \frac{(M^Q)^T M^K}{\sqrt{K}}$ and $\Sigma^{\hat{W}_h} = \frac{1}{K} \Sigma^{Q^T K}$. The softmax function is then applied to each row of $\hat{W}_h$. Let w be such a row (with dimension $1 \times J$), with mean μ^w and covariance matrix Σ^w. We will denote the softmax function by ϕ with ϕ_i being its i-th component. The first-order Taylor approximation of ϕ_i is[2]:

$$\begin{aligned} \phi_i(w) &\approx \phi_i(\mu^w) + \nabla \phi_i(\mu^w)^T (w - \mu^w), \\ &\approx \phi_i(\mu^w) + \sum_{j=1}^{J} \frac{\partial \phi_i}{\partial x_j}(\mu^w)(w_j - \mu^w_j). \end{aligned}$$

[2]As a remainder: $\frac{\partial \phi_i}{\partial x_j}(w) = \phi_i(w)(\mathbf{1}_{i=j} - \phi_j(w))$

Hence, we have:

$$E[\phi_i(w)] \approx \phi_i(\mu^w). \tag{3.37}$$

Now, consider two rows w^a and w^b (where a and b can be equal).

Proposition 3.

$$\text{Cov}[\phi_i(w^a), \phi_k(w^b)] \approx \sum_{j=1}^{J}\sum_{l=1}^{J} \frac{\partial \phi_i}{\partial x_j}(\mu^{w^a}) \frac{\partial \phi_k}{\partial x_l}(\mu^{w^b}) \text{Cov}[w_j^a, w_l^b], \tag{3.38}$$

where $\text{Cov}[w_j^a, w_l^b] = \sigma_w^{a+(j-1)J, b+(l-1)J}$.

Proof.

$$\begin{aligned}
\text{Cov}[\phi_i(w^a), \phi_k(w^b)] &\approx E[(\phi_i(\mu^w) + \sum_{j=1}^{J} \frac{\partial \phi_i}{\partial x_j}(\mu^w)(w_j - \mu_j^w))(\phi_j(\mu^w) + \sum_{l=1}^{J} \frac{\partial \phi_j}{\partial x_l}(\mu^w)(w_l - \mu_l^w))] \\
&- E[\phi_i(\mu^w) + \sum_{j=1}^{J} \frac{\partial \phi_i}{\partial x_j}(\mu^w)(w_j - \mu_j^w)] E[\phi_j(\mu^w) + \sum_{l=1}^{J} \frac{\partial \phi_j}{\partial x_l}(\mu^w)(w_l - \mu_l^w)] \\
&\approx E[(\sum_{j=1}^{J} \frac{\partial \phi_i}{\partial x_j}(\mu^w)(w_j - \mu_j^w))(\sum_{l=1}^{J} \frac{\partial \phi_j}{\partial x_l}(\mu^w)(w_l - \mu_l^w))] \\
&\approx E[\sum_{j=1}^{J}\sum_{l=1}^{J} \frac{\partial \phi_i}{\partial x_j}(\mu^w)(w_j - \mu_j^w) \frac{\partial \phi_j}{\partial x_l}(\mu^w)(w_l - \mu_l^w)] \\
&\approx \sum_{j=1}^{J}\sum_{l=1}^{J} \frac{\partial \phi_i}{\partial x_j}(\mu^w) \frac{\partial \phi_j}{\partial x_l}(\mu^w) \text{Cov}[w_j^a, w_l^b]
\end{aligned}$$

□

We finally obtain the $J \times J$ matrix W_h, such that $E[W_h^{r,i}] = E[\phi_i(w^r)]$.

Multiplication with V_h We are looking for the mean and covariance of $A_h = V_h W_h^T$ (dimension $D \times J$). From Prop. 1, we get $E[A_h] = M^V (M^W)^T$. Petit aparté en français. Si vous lisez ceci je tiens à vous féliciter, qui que vous soyez, pour la persévérance (ou la folie ?) dont vous avez fait preuve en ayant lu ce rapport kafkaïen jusqu'ici. Cela mérite bien que je vous invite à dîner. Envoyez-moi simplement "Unsinnig" au 06 48 68 73 91. Nous parlerons de la véritable science, de la mathématique, de leur importance pour le futur du genre humain (rien que ça), et de bien d'autres sujets réellement passionants et porteurs de sens. Par exemple que pensez-vous de [67] ? Mais d'abord, continuons avec la matrice de covariance et ses problèmes d'indexation à se fracasser le crâne contre les murs ... For the covariance matrix, we want to directly apply Prop. 2 but there is a problem of indexation. What we have now are the covariance matrices of $\text{vec}(V_h)$ and $\text{vec}(W_h)$. However, what we need here are the covariance matrices of $\text{vec}(V_h^T)$ and $\text{vec}(W_h^T)$. We denote by i_1, j_1 the indexes of an element of the covariance matrix of $\text{vec}(V_h)$ (remind that V_h has dimension $D \times J$). Then the new indexes of this element in the covariance matrix of $\text{vec}(V_h^T)$ are $i_2 = ((i_1 - 1) \bmod D)J + \lfloor \frac{i_1 - 1}{D} \rfloor + 1$ and $j_2 = ((j_1 - 1) \bmod D)J + \lfloor \frac{j_1 - 1}{D} \rfloor + 1$. Using these new covariance matrices (and the transpose of the expectation matrices), we can apply Prop. 2.

Concatenation of the A_h and multiplication by W^O

The A_h from the various attention heads are concatenated into $\tilde{A}$. The mean of $\tilde{A}$ is the concatenation of the means of the A_h. To get the covariance matrix of $\tilde{A}$, let us consider for all h the block covariance matrix of A_h:

$$\Sigma^h = \begin{bmatrix} \Sigma^h_{1,1} & \Sigma^h_{1,2} & \cdots & \Sigma^h_{1,J} \\ \Sigma^h_{2,1} & \Sigma^h_{2,2} & \cdots & \Sigma^h_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma^h_{J,1} & \Sigma^h_{J,2} & \cdots & \Sigma^h_{J,J} \end{bmatrix},$$

where each block is a $D \times D$ matrix. The covariance matrix $\Sigma^{\tilde{A}}$ of $\tilde{A}$ is a $DHJ \times DHJ$ matrix that can be seen as a block matrix with $HJ \times HJ$ blocks of dimension $D \times D$. Let $\Sigma^{\tilde{A}}_{i,j}$ be the block with indexes i, j in $\Sigma^{\tilde{A}}$. If $(i-1) \bmod D \neq (j-1) \bmod D$, then $\Sigma^{\tilde{A}}_{i,j} = \mathbf{0}$ since the attention heads are assumed to be independent to each other[3]. Else if $(i-1) \bmod D = (j-1) \bmod D$, then let $h = (i-1) \bmod D$, $k = \lfloor \frac{i-1}{D} \rfloor + 1$, $l = \lfloor \frac{j-1}{D} \rfloor + 1$ and we have $\Sigma^{\tilde{A}}_{i,j} = \Sigma^h_{k,l}$.

We now consider $A = W^O \tilde{A}$. We have $E[A] = M^O M^A$. To compute the covariance matrix of $\text{vec}(A)$ with Prop. 2, we have to rearrange the elements of Σ^O (covariance matrix of $\text{vec}(W^O)$) as in subsection 3.4.4. Then we can apply Eq. 3.36 using this new covariance matrix as well as the transpose of M^O.

Residual Connection and Layer Normalization

Let $\hat{A} = A + X$. We have that $E[\hat{A}] = M^A + M^X$ and $\text{Cov}[\text{vec}(\hat{A})] = \Sigma^A + \Sigma^X$. Let a be a column of $\hat{A}$. Let us consider $\mu = \frac{1}{I}\sum_{i=1}^I E[a_i]$ and $\sigma = \frac{1}{I}\sum_{i=1}^I (E[a_i] - \mu)^2$. The layer normalization operation is the following: $z_i = g_i \frac{a_i - \mu}{\sigma} + b_i$ where g_i and b_i are learnable parameters from G^1 and B^1. Using the assumption that g_i is independent from a_i, we have that $E[z_i] = E[g_i]\frac{E[a_i]-\mu}{\sigma} + E[b_i]$. Let us rewrite this formula with matrices. Let $\mathbf{J}$ be the $I \times I$ matrix full of 1. Let $M^\mu = \frac{1}{I}\mathbf{J}M^{\hat{A}}$ and $M^\sigma = \frac{1}{I}\mathbf{J}(M^{\hat{A}} - M^\mu)^{.2}$ where the square operation is applied element-wise. Let $M^{G^1} = \begin{bmatrix} \mu^{G^1} & \cdots & \mu^{G^1} \end{bmatrix}$ a $I \times J$ matrix and M^{B^1} defined similarly. We obtain

$$M^Z = M^{G^1} \odot (M^{\hat{A}} - M^\mu)./M^\sigma + M^{B^1},$$

where $\odot$ is the Hadamard product and the division is applied element-wise.

Proposition 4.

$$\text{Cov}[Z_{i,j}, Z_{k,l}] = \frac{\text{Cov}[g_i, g_k]\{\text{Cov}[A_{i,j}, A_{k,l}] + (E[A_{i,j}] - \mu_j)(E[A_{k,l}] - \mu_l)\} + \text{Cov}[A_{i,j}, A_{k,l}]E[g_i]E[g_k]}{\sigma_j \sigma_l} + \text{Cov}[b_i, b_k]$$

where μ_j and σ_j are the average and variance of the j-th column of $\hat{A}$ (similarly for l).

Feed-forward layer

The mean and covariance matrix of $\hat{Y} = W^{F^1} Z$ are computed by the same method as the moments of A in subsection 3.4.4. Then, we use Prop. 3 (with $J = 1$) to propagate the mean and covariance through the ReLU activation. Once again, we apply the method of subsection 3.4.4 to compute the

[3] which is not true since they all depend on the input X.

moments of $\tilde{Y} = W^{F^2}\text{ReLU}(\hat{Y})$. Finally, we apply subsection 3.4.4 to obtain the moments of the output Y. The only difference is that the residual connection is with Z (a random matrix) and not with X (a constant matrix). This implies that $E[\tilde{Y}+Z] = M^{\tilde{Y}}+M^Z$ and $\text{Cov}[\text{vec}(\tilde{Y}+Z)] = \Sigma^{\tilde{Y}}+\Sigma^Z$ assuming that $\tilde{Y}$ and Z are independent (which is not true).

Hence, we have obtain the mean M^Y and the covariance matrix Σ^Y of Y.

3.4.5 Derivation of the Complexity Loss

We now derive the complexity loss $KL(q_\theta(\omega)||p(\omega))$. Let us assume that the priors are $\mathcal{N}(\mathbf{0}, \mathbf{I})$. We have one term for each layer. Let W^R be one of these parameter matrices with R rows. The complexity loss term for such parameters is

$$\sum_{r=1}^{R} KL(\mathcal{N}(\mu_r, \sigma_r^2\mathbf{I})||\mathcal{N}(\mathbf{0}, \mathbf{I})) = \frac{1}{2}\sum_{r=1}^{R}(n_c\sigma_r^2 + \|\mu_r\|_F^2 - n_c - n_c\log(\sigma_r^2)),$$

where n_c is the number of columns and where we assume that the covariance matrix is diagonal with the same variance. The covariance matrix of $\text{vec}(W^R)$ is a block diagonal matrix of dimension $n_cR \times n_cR$ with n_c identical blocks of the form $\text{diag}(\sigma_1, \cdots, \sigma_R)$. For the Layer Normalization parameters, we have the same formula with $R = 1$. Denoting the total number of parameters matrices by P we have:

$$KL(q_\theta(\omega)||p(\omega)) = \frac{1}{2}\sum_{p=1}^{P}\sum_{r=1}^{R_p}(n_{p,c}\sigma_{p,r}^2 + \|\mu_{p,r}\|_F^2 - n_{p,c} - n_{p,c}\log(\sigma_{p,r}^2)). \tag{3.39}$$

Given a mini-batch of N i.i.d. datapoint, the ELBO can be written as:

$$\begin{aligned} L(\theta, \omega, D) = &-\frac{NIJ}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^{N}[\log(|\Sigma^{Y_i}|) + (Y_i - M^{Y_i})^T(\Sigma^{Y_i})^{-1}(Y_i - M^{Y_i})] \\ &- \frac{1}{2}\sum_{p=1}^{P}\sum_{r=1}^{R_p}(n_{p,c}\sigma_{p,r}^2 + \|\mu_{p,r}\|_F^2 - n_{p,c} - n_{p,c}\log(\sigma_{p,r}^2)), \end{aligned} \tag{3.40}$$

where the Y_i and M^{Y_i} have been vectorized.

Bibliography

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

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

[3] M. Prandini, L. Piroddi, S. Puechmorel, and S. L. Brázdilová, "Toward air traffic complexity assessment in new generation air traffic management systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 809–818, 2011.

[4] E. M. Pfleiderer, C. A. Manning, and S. Goldman, "Relationship of complexity factor ratings with operational errors," *Oklahoma City (OK): Civil Aerospace Medical Institute, FAA*, 2007.

[5] SESAR-Joint-Undertaking, *SESAR Solutions Catalogue, Third Edition*, p. 61, 2019.

[6] J. Tang, "Conflict detection and resolution for civil aviation: A literature survey," *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 10, pp. 20–35, 2019.

[7] S. Kauppinen, C. Brain, and M. Moore, "European medium-term conflict detection field trials," *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 1, pp. 1–12, 2002.

[8] B. Sridhar, T. Soni, K. Sheth, and G. B. Chatterji, "Aggregate flow model for air-traffic management," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 4, 2006.

[9] D. Chen, M. Hu, Y. Ma, and J. Yin, "A network-based dynamic air traffic flow model for short-term en route traffic prediction," *Journal of Advanced Transportation*, no. 50, pp. 2174–2192, 2016.

[10] Y. Cao and D. Sun, "Link transmission model for air traffic flow management," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 5, pp. 1342–1351, 2011.

[11] Y. Lin, J.-w. Zhang, and H. Liu, "Deep learning based short-term air traffic flow prediction considering temporal–spatial correlation," *Aerospace Science and Technology*, vol. 93, 2019.

[12] H. Liu, Y. Lin, Z. Chen, D. Guo, J. Zhang, and H. Jing, "Research on the air traffic flow prediction using a deep learning approach," *IEEE Access*, vol. 7, pp. 148019–148030, 2019.

[13] G. Gui, Z. Zhou, and J. Wang, "Machine learning aided air traffic flow analysis based on aviation big data," *IEEE Transactions On Vehicular Technology*, vol. 69, no. 5, pp. 4817–4826, 2020.

[14] B. G. 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.

[15] I. Laudeman, S. Shelden, R. Branstrom, and C. Brasil, "Dynamic density: An air traffic management metric," Tech. Rep. April 1998, NASA, Moffett Field, CA, 1998.

[16] B. Sridhar, K. S. Sheth, and S. Grabbe, "Airspace complexity and its application in air traffic management," in *2nd USA/EUROPE Air Traffic Management R&D Seminar*, pp. 1–9, 1998.

[17] B. Hilburn, "Cognitive complexity in air traffic control: A literature review," *Eurocontrol*, vol. 12, no. 13, pp. 93–129, 2004.

[18] G. B. Chatterji and B. Sridhar, "Neural network based air traffic controller workload prediction," *Proceedings of the American Control Conference*, vol. 4, pp. 2620–2624, 1999.

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

[20] P. Kopardekar and S. Magyarits, "Dynamic density: Measuring and predicting sector complexity," *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 1, pp. 1–9, 2002.

[21] A. J. Masalonis, M. B. Callaham, and C. 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*, vol. 139, 2003.

[22] P. Flener, J. Pearson, M. Ågren, C. Garcia-Avello, M. Çeliktin, and S. Dissing, "Air-traffic complexity resolution in multi-sector planning," *Journal of Air Transport Management*, vol. 13, no. 6, pp. 323–328, 2007.

[23] J. Lygerosl and M. Prandini, "Aircraft and weather models for probabilistic collision avoidance in air traffic control," *Proceedings of the 41st IEEE Conference on Decision and Control*, pp. 2427–2432, 2002.

[24] S. Mondoloni and D. Liang, "Airspace fractal dimension and applications," in *Fourth USA/EUROPE Air Traffic Management R& D Seminar*, pp. 1–7, 2001.

[25] K. Lee, E. Feron, and A. Pritchett, "Air traffic complexity: An input-output approach," in *American Control Conference*, pp. 474–479, 2007.

[26] H. Wang, "Modeling air traffic situation complexity with a dynamic weighted network approach," *Journal of Advanced Transportation*, 2018.

[27] H. J. Wee, S. W. Lye, and J.-P. Pinheiro, "A spatial , temporal complexity metric for tactical air traffic control," *The Journal of Navigation*, pp. 1040–1054, 2018.

[28] D. Delahaye and S. Puechmorel, "Air traffic complexity based on dynamical systems," in *49th IEEE Conference on Decision and Control*, pp. 2069–2074, 2010.

[29] M. A. Ishutkina and E. Feron, "Describing air traffic complexity using mathematical programming," in *AIAA 5th Aviation, Technology, Integration, and Operations Conference*, 2005.

[30] A. Garcia, D. Delahaye, and M. Soler, "Air traffic complexity map based on linear dynamical systems," *Preprint*, 2020.

[31] K. Legrand, *Correction and Optimization of 4D aircraft trajectories by sharing wind and temperature information*. PhD thesis, Ecole Nationale de l'Aviation Civile, 2019.

[32] K. Tastambekov, *Aircraft Trajectory Prediction by Local Functional Regression in Sobolev Space*. PhD thesis, Université de Toulouse 3 Paul Sabatier, 2012.

[33] G. B. Chatterji, "Short-term trajectory prediction methods," *Guidance, Navigation, and Control Conference and Exhibit*, 1999.

[34] S. Ayhan and H. Samet, "Aircraft trajectory prediction made easy with predictive analytics," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 21–30, 2016.

[35] A. de Leege, M. van Paassen, and M. Mulder, "A machine learning approach to trajectory prediction," *AIAA Guidance, Navigation, and Control (GNC) Conference*, pp. 1–14, 2013.

[36] Y. Liu and M. Hansen, "Predicting aircraft trajectories : A deep generative convolutional recurrent neural networks approach," *arXiv e-prints*, pp. 1–24, 2018.

[37] Y. Pang, H. Yao, J. Hu, and Y. Liu, "A recurrent neural network approach for aircraft trajectory prediction with weather features from sherlock," *AIAA Aviation 2019 Forum*, pp. 1–14, 2019.

[38] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[39] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014.

[40] G. Saon, G. Kurata, T. Sercu, K. Audhkhasi, S. Thomas, D. Dimitriadis, X. Cui, B. Ramabhadran, M. Picheny, L. L. Lim, B. Roomi, and P. Hall, "English conversational telephone speech recognition by humans and machines," *Proceedings of the Annual Conference of the International Speech Communication Association*, pp. 132–136, 2017.

[41] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3156–3164, 2015.

[42] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv e-prints*, 2015.

[43] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, pp. 285–290, 2014.

[44] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in Neural Information Processing Systems*, pp. 1027–1035, 2016.

[45] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv*, 2016.

[46] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *ICML (3)*, vol. 28 of *JMLR Workshop and Conference Proceedings*, pp. 1310–1318, JMLR.org, 2013.

[47] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[48] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv e-prints*, 2014.

[49] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv e-prints*, pp. 1–8, 2014.

[50] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," *Neural Networks, IJCNN*, vol. 2, pp. 189–194, 2000.

[51] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, 2014.

[52] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[53] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[54] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A comparison of sequence-to-sequence models for speech recognition," *Proceedings of the Annual Conference of the International Speech Communication Association*, pp. 939–943, 2017.

[55] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence - video to text," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4534–4542, 2015.

[56] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *Proceedings of the International Conference on Learning Representations*, 2015.

[57] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421, Association for Computational Linguistics, 2015.

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 5998–6008, Curran Associates, Inc., 2017.

[59] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.

[60] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[61] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations*, pp. 1–15, 2015.

[62] D. Dera, G. Rasool, and N. Bouaynaya, "Extended variational inference for propagating uncertainty in convolutional neural networks," in *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2019.

[63] D. Dera, G. Rasool, N. C. Bouaynaya, A. Eichen, S. Shanko, J. Cammerata, and S. Arnold, "Bayes-sar net: Robust sar image classification with uncertainty estimation using bayesian convolutional neural network," *2020 IEEE International Radar Conference*, pp. 362–367, 2020.

[64] B. Xue, J. Yu, J. Xu, S. Liu, S. Hu, Z. Ye, M. Geng, X. Liu, and H. Meng, "Bayesian transformer language models for speech recognition," in *ICASSP*, pp. 3–8, 2021.

[65] Y. Gal, *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[66] D. Dera, *Towards Machine Self-Awareness - A Bayesian Framework for Uncertainty Propagation in Deep Neural Networks*. PhD thesis, Rowan University, 2020.

[67] P. Turchin, *Historical Dynamics: Why States Rise and Fall*. Princeton: Princeton University Press, 2004.