RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Université de Toulouse

Doctorat de l'Université de Toulouse

préparé à l'Université Toulouse III - Paul Sabatier

Contributions à la Robustesse de l'Apprentissage Machine avec Applications à la Prédiction de Trafic

---

Thèse présentée et soutenue, le 14 octobre 2024 par

Loïc SHI-GARRIER

École doctorale
AA - Aéronautique, Astronautique

Spécialité
Mathématiques et Applications

Unité de recherche
ENAC-LAB - Laboratoire de Recherche ENAC

Thèse dirigée par
Daniel DELAHAYE et Nidhal BOUAYNAYA

Composition du jury
M. Pierre MARÉCHAL, Président, Université Toulouse III - Paul Sabatier
M. Roman SHTERENBERG, Rapporteur, University of Alabama at Birmingham
Mme Lyudmila MIHAYLOVA, Rapporteure, University of Sheffield
M. Sameer ALAM, Examinateur, Nanyang Technological University
M. Daniel DELAHAYE, Directeur de thèse, ENAC
Mme Nidhal BOUAYNAYA, Co-directrice de thèse, Rowan University

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

ENAC

Résumé

L'émergence de l'apprentissage profond est porteur de nombreuses opportunités pour l'aviation civile. Les méthodes d'apprentissage peuvent automatiser des tâches pénibles (e.g., transcription de la parole vers le texte pour les rapports d'incidents) ou introduire de nouvelles fonctions visant à améliorer la sécurité, l'efficacité, ou l'impact environnemental du transport aérien (e.g., réduction de la complexité du trafic aérien). Cependant, les caractéristiques propres aux modèles d'apprentissage machine (fondés sur les données, stochastiques, manque de transparence) limitent leur introduction dans les applications où la sécurité est un enjeu central. Au-delà de la performance en termes de précision, ces applications exigent des modèles fiables, capables de fournir des garanties en termes de robustesse et d'explicabilité.

Cette thèse explore divers aspects de la robustesse des réseaux de neurones contre les attaques par exemples contradictoires en s'appuyant sur des concepts empruntés à la géométrie de l'information. Dans une première partie, nous introduisons un terme de régularisation visant à améliorer la robustesse des réseaux de neurones contre les attaques L_2. Ce terme pousse le modèle à être localement isométrique par rapport à la métrique euclidienne (en entrée) et à la métrique d'information de Fisher (en sortie). Cette méthode est évaluée sur les jeux de données MNIST et CIFAR-10. Dans la seconde partie, on introduit une méthode pour étudier la robustesse des réseaux de neurones récurrents basée sur les équations différentielles stochastiques. Plusieurs expériences sont ensuite menées sur des données synthétiques et sur MNIST afin d'étudier l'interaction entre la vulnérabilité aux exemples contradictoires et les foliations induites par le noyau de la métrique d'information de Fisher dans l'espace d'entrée. Dans la troisième partie, nous appliquons les réseaux de neurones récurrents à la prédiction de la congestion dans l'espace aérien. Finalement, nous développons une méthode fondée sur l'inférence variationnelle pour la quantification de l'incertitude dans les modèles de type Transformer. Cette méthode est évaluée sur des données de trafic routier.

Abstract

The advent of deep learning brings many opportunities to the civil aviation community. Data-driven methods can automatize tedious tasks (e.g., speech-to-text for incident reporting) or introduce new functions to improve the safety, efficiency, or environmental impact of the air transportation system (e.g., reduction of the complexity of air traffic). However, the unique features of machine learning models (data-driven, stochastic, lack of transparency) prevent their introduction into safety-critical applications. Beyond accuracy, safety-critical applications require the model to be trustworthy by providing guarantees on robustness and explainability.

This thesis explores several aspects of neural networks robustness against adversarial noise using concepts borrowed from information geometry. In the first part, a regularization term is introduced to improve the robustness of neural networks against L_2 attacks. This term pushes the model to be locally isometric with respect to the Euclidean metric (in input) and the Fisher information metric (in output). The method is evaluated on MNIST and CIFAR-10 datasets. In the second part, a method is introduced to study the robustness of recurrent neural networks using stochastic differential equations. Then, several experiments are performed on synthetic data and on MNIST in order to study the interaction between adversarial vulnerability and the foliations induced by the kernel fisher information metric in the input space. In the third part, recurrent neural networks are applied for the prediction of airspace congestion. Finally, a method is derived for uncertainty quantification for Transformer models based on variational inference. This method is evaluated on road traffic data.

Acknowledgments

First of all, I would like to thank the Direction Générale de l'Aviation Civile for trusting me and allowing me to pursue this PhD at the Ecole Nationale de l'Aviation Civile, as well as the Rowan University Department of Electrical and Computer Engineering for hosting me during six months.

I would like to express my gratitude for the following people who have helped me undertake this journey:

My advisors, Daniel Delahaye and Nidhal Bouaynaya, for their guidance and support.

The reviewers of my manuscript, Roman Shterenberg and Lyudmila Mihaylova, as well as the other members of my committee, Pierre Maréchal and Sameer Alam.

All the scholars who helped me at one point or another. In particular: Giuseppina Carannante, Nicolas Couellan, Dimah Dera, Gregory Ditzler, Stéphane Puechmorel, and Laurent Lapasset for giving me access to his computation platform at ENAC.

All the students with whom I enjoyed fruitful discussions. In particular: Christopher Angelini, Taha Bouhsine, Hakim Cherfi, El-Mehdi Djelloul, Hikmat Khan, Kyle Naddeo, Eliot Tron, and my dear friend Nicolas Gault with whom I hope to continue our highly important research program for many more years.

Contents

Résumé **3**

Abstract **4**

Acknowledgments **5**

Contents **6**

List of Figures **8**

List of Tables **11**

Notations **12**

1 Introduction: Trustworthy and Certifiable AI for Civil Aviation **13**
1.1 The Promises of Machine Learning . . . 13
1.2 Trustworthy AI in Aviation . . . 15
1.2.1 Challenges for AI Certification . . . 15
1.2.2 The Trustworthiness Trade-off . . . 15
1.3 What is Robustness? . . . 16
1.4 Thesis Organization and Review of Contributions . . . 17

2 Machine Learning Robustness **19**
2.1 Adversarial Machine Learning . . . 19
2.1.1 Adversarial Attacks . . . 20
2.1.2 Adversarial Defenses . . . 22
2.1.3 Information Geometry and Adversarial Machine Learning . . . 25
2.1.4 Proposed Explanations . . . 27
2.1.5 Challenges for Model Stability . . . 31
2.2 Verification of Model Stability . . . 33
2.2.1 Randomized Smoothing . . . 34
2.2.2 Lipschitz Neural Networks . . . 35
2.3 Uncertainty Quantification . . . 37
2.3.1 Bayesian Inference . . . 37
2.3.2 Variational Inference . . . 38
2.3.3 Variational Inference for Uncertainty Quantification . . . 39
2.3.4 Stochastic Differential Equation Network (SDE-Net) . . . 40
2.4 Conclusion . . . 41

3 Partial Isometry Regularization **43**
3.1 Definitions and Robustness Condition . . . 43
3.1.1 Geometrical Definitions . . . 43
3.1.2 Robustness Condition . . . 46
3.2 Derivation of the Regularization Method . . . 47
3.2.1 The Partial Isometry Condition . . . 47
3.2.2 Coordinate Change . . . 49
3.2.3 The Fisher–Rao Distance . . . 50
3.3 Experiments . . . 50

3.3.1 Experiments on MNIST Dataset . . . 51
3.3.2 Experiments on CIFAR-10 Dataset . . . 52
3.4 Discussion . . . 53
3.5 Conclusion . . . 54
3.6 Proofs . . . 55

4 Robustness and Geometry **59**
4.1 The Image and Kernel Foliations . . . 59
4.2 Robust Time Series Prediction . . . 60
4.2.1 A Model for Recurrent Neural Networks . . . 61
4.2.2 Stochastic Differential Equations . . . 64
4.2.3 Visualizing the Kernel Foliation . . . 66
4.3 Foliations . . . 70
4.3.1 Adversarial Attacks and Leaves . . . 71
4.3.2 Trajectories on Leaves . . . 74
4.3.3 Robustified Dataset . . . 78
4.4 Conclusion . . . 82

5 Traffic Prediction and Uncertainty Quantification **83**
5.1 Context . . . 83
5.2 Air Traffic Flow Prediction (ATFP) . . . 84
5.2.1 Conflict detection tools . . . 84
5.2.2 Classic ATFP Methods . . . 84
5.2.3 Machine Learning ATFP . . . 85
5.3 Complexity Metrics . . . 85
5.3.1 Reviews . . . 86
5.3.2 Classic Methods . . . 86
5.3.3 Intrinsic Complexity Metrics . . . 88
5.4 Recurrent Model . . . 89
5.4.1 Review of Sequential Architectures . . . 90
5.4.2 Encoder-Decoder Long Short-Term Memory Network . . . 94
5.4.3 Methodology and experiments . . . 96
5.4.4 Results and Discussion . . . 99
5.5 Transformer Model . . . 100
5.5.1 The Transformer framework . . . 100
5.5.2 Moment propagation for Transformers . . . 102
5.5.3 Experiments . . . 106
5.6 Conclusion . . . 108

6 Conclusion and Perspectives **111**
6.1 General Conclusion . . . 111
6.2 Future Research Directions . . . 112
6.2.1 Rethinking Robustness . . . 112
6.2.2 Bayesian Priors, Simplicity, and Robustness . . . 113

Bibliography **117**

List of Figures

2.1 The geodesic between two normal distributions is a hyperbola (in blue). Its length is the Fisher-Rao distance between the two distributions. 25

3.1 ϵ-robustness at $\boldsymbol{x}$ is enforced if and only if $f(\overline{b}(\boldsymbol{x}, \epsilon)) \subseteq \mathcal{A}_{\boldsymbol{x}}$. 45

3.2 ϵ-robustness at $\boldsymbol{x}$ is enforced if $\overline{b}(\boldsymbol{x}, \epsilon) \subseteq \tilde{b}(\boldsymbol{x}, \delta)$. 46

3.3 Accuracy of the baseline (dashed, blue), regularized (solid, green), and adversarially trained (dotted, red) models for various attack perturbations on the MNIST dataset. The perturbations are obtained with PGD using L_∞ norm. 52

4.1 The objective is to learn the dynamic of a system living on a manifold S, given a measure function F. However, a random noise $w(t)$ is added to the measures. This noise can be a Gaussian noise or an adversarial attack. Thus, $x(t) = F(\boldsymbol{s}(t)) + w(t)$. The prediction of the model is $\hat{y}(t) = f_\tau(x(t), \boldsymbol{h}(t-\tau))$ where τ is a time step. When $\tau \longrightarrow 0$, the difference equation on $\boldsymbol{h}$ can be converted into a differential equation. A loss function is used to train the network by computing the error $e(t) = \mathcal{L}(\hat{y}(t), F(\boldsymbol{s}(t+\tau))$. 62

4.2 Simplified model. It is assumed that $m = 1$ hence, $\boldsymbol{h}(t) = x(t-1)$. Moreover, one has $\boldsymbol{h}(t) = f_\tau(w(t), \boldsymbol{h}(t-\tau)) = (a\tau + 1)\boldsymbol{h}(t-\tau) + b\tau + c\tau w(t)$. A constant 1 is added in input to include the bias in the matrix of f_τ. 63

4.3 An example of a curve $\boldsymbol{\theta}(t) = (\mu(t), \sigma^2(t))$ solution of equation 4.2 in the Poincaré half-plane $\mathcal{H}_1$ (in red). The velocity tensor field $\dot{\boldsymbol{\theta}}(t)$ is represented with blue arrows. The covariant derivative $D_t\dot{\boldsymbol{\theta}}(t)$ is represented with green arrows. The parameters are $a = -1, b = 1, c = 1$ and $\mu_0 = -1$. The curve is plotted between $t = 0.01$ and $t = 2$. Some geodesics of $\mathcal{H}_1$ are plotted for better visualization (in gray). The Y-axis corresponds to the variance $v = \sigma^2$. 66

4.4 Dataset consisting of trajectories of a simple dynamical system (time t on the X-axis, $y(t)$ on the Y-axis). 67

4.5 True trajectories (blue), predicted trajectories (green), and standard deviations (red). 69

4.6 Embedding of the state space in the input space. The red parts correspond to the training data while the blue parts were not seen by the model during training. 70

4.7 Kernel leaves (in green) of points belonging to the embedded state space, viewed from two different angles. The magenta arrows are basis of the kernel at each of these points. 71

4.8 FGSM attack 72

4.9 Original example, anti-adversarial example, projection on the data leaf, and FGSM example. An anti-adversarial example is characterized as follows: the model is unsure about how to classify the example despite being clearly recognizable by humans. 73

4.10 Entropy and predicted probabilities of each class with respect to the iterations of Algorithm 1 [1]. Note that the x-axis is the iteration not the distance. After the algorithm converges at the $\sim$ 100th iteration, the entropy and probabilities are constant because the distance between $\boldsymbol{x}_0$ and the current iterate does not increase anymore. 74

4.11 A plane of the input space where the Y-axis is an adversarial direction, while the X-axis is a random direction orthogonal to the adversarial direction. 74

4.12 Entropy and probabilities along paths whose velocities are eigenvectors of the local data matrix. The associated eigenvalues are sorted with the smallest one in the top left corner and the highest one in the bottom right corner (the increase is along row first). 75

4.13 Distance along paths whose velocities are eigenvectors and examples of images along these paths. 76

4.14 Examples of images from the robust MNIST training set. The true label is indicated above each image. 78

4.15 Histograms of distances using 20 images from the test set. 80

4.16 Histograms of distances using 20 images from the training set. 81

4.17 Loss function and horizontal path between two original images of the training set using f_{std} and f_{adv} with 5000 iterations. The first image is the origin, the last image is the destination, the five central images are sampled regularly along the horizontal path. Above each image, the iteration and the predicted class are indicated. 81

4.18 Histogram of distances between images of the same class using f_{std} and 100 images from the training set. 82

5.1 Basic RNN layer (left) and the same layer unrolled through time (right). Σ represents linear mapping and ϕ represents the activation function. 90

5.2 A Long Short-Term Memory (LSTM) layer. The current input $\boldsymbol{x}_t$ is combined with the previous hidden state $\boldsymbol{h}_{t-1}$ to compute the three gates used to update the cell state $\boldsymbol{c}_{t-1}$ and compute a new hidden state $\boldsymbol{h}_t$. 90

5.3 Encoder-decoder LSTM model for complexity prediction. The encoder network (left) uses a sequence of aircraft states $\boldsymbol{x}_{t_0}, \cdots, \boldsymbol{x}_{t_0+t_{in}-1}$ to compute an *encoding sequence*. The decoder network (right) uses the encoding sequence and the complexity matrix outputted at the last timestep $\hat{\boldsymbol{H}}_{t-1}$ to compute a prediction. 95

5.4 Depiction of the entire framework. The dataset consists of trajectories where $\boldsymbol{x}_t^i$ is the state vector of the ith trajectory at time t. The complexity values are computed as described in Section 5.3.3. The dataset, consisting of the aircraft states and the complexity matrices, is divided into training and validation sets. The encoder-decoder LSTM model is trained (validated) with the training (validation) sets. 96

5.5 A heatmap of the complexity matrices of one sequence of trajectories from the training set. The longitude is represented horizontally and the latitude vertically. True values after 40 minutes (left) and predicted values (right). 97

5.6 A heatmap of the complexity matrices of one sequence of trajectories from the validation set. The longitude is represented horizontally and the latitude vertically. True values after 40 minutes (left) and predicted values (right) 98

5.7 Distribution of non-zero absolute errors between prediction and ground truth over the validation set. 99

5.8 Prediction of the model on one example of the test set of the simulated trajectories dataset. From left to right: ground truth for each cell of the airspace, predicted classes, predictive variance, absolute error between ground truth and predicted classes. 106

5.9 Histograms of the predictive variances for the correctly (left) and incorrectly (right) classified examples over the test set of MNIST. 107

5.10 Left: accuracy of the deterministic and Bayesian models on the test set of PeMS-SF as a function of the signal-to-noise ratio. Right: predictive variance of the Bayesian model on the test set of PeMS-SF as a function of the signal-to-noise ratio. 108

List of Tables

3.1 Clean and robust accuracy on MNIST against AA, averaged over 10 runs. The number in parentheses is the attack strength. 52

3.2 Clean and robust accuracy on CIFAR-10 against PGD. The number in parentheses is the attack strength. 53

4.1 Model evaluation over the training set and four test datasets. The relative error measures the accuracy of the model, while the likelihood measures the quality of the uncertainty quantification of the model. Both metrics are reported in %. . . 68

4.2 Predicted class and probability for the original example (Figure 4.9a), FGSM example (Figure 4.9d), and projection of the FGSM example on the data leaf (Figure 4.9c). 72

4.3 Distances and angles between various examples. 73

4.4 Standard and adversarial accuracy for the three models. 79

Notations

Unless specified otherwise, the following notations throughout the thesis.

$\mathbb{R}, \mathbb{R}^*, \mathbb{R}_+$	Sets of reals, nonzero reals, and non-negative reals respectively
$\mathbb{N}, \mathbb{N}^*$	Sets of non-negative integers, and positive integers respectively
$f(\cdot)$	Model
$\mathcal{L}$	Loss functions
$\boldsymbol{x}$	Model input
$\hat{\boldsymbol{y}}$	Model output
$\boldsymbol{y}$	Target of the model (i.e., ground truth)
$\boldsymbol{\theta}$	Model parameters, or parameters of a probability distribution
$\boldsymbol{w}$	Weights (parameters used in linear or affine layers)
$\boldsymbol{b}$	Bias (parameters used in linear or affine layers)
n	Dataset size
c	Number of classes
d	Dimension of the input $\boldsymbol{x}$
m	Sequence length
$\mathbb{P}[\cdot]$	Probability measures
$p(\cdot), q(\cdot)$	Probability distributions or densities
$\mathbb{E}[\cdot], \text{var}[\cdot]$	Expectation operator, variance operator
$\boldsymbol{\mu}, \sigma, \boldsymbol{\Sigma}$	Mean, variance, covariance matrix
$\text{KL}[\cdot\|\|\cdot]$	Kullback-Leibler divergence
$\text{ELBO}(\cdot)$	Evidence lower bound
$\text{rg}(\boldsymbol{M}), \text{rk}(\boldsymbol{M}), \|\boldsymbol{M}\|, \lambda(\boldsymbol{M})$	Range, rank, determinant, spectrum of a matrix $\boldsymbol{M}$
$\|\cdot\|_p$	L_p norms. The L_2 norm is sometimes denoted by $\|\cdot\|$
$\|\cdot\|_F$	Frobenius norm
$\odot$	Element-wise product (i.e., Hadamard product)
∇	Gradient operator
δ_{ij}	Kronecker delta
$\mathbb{1}_A$	Indicator function of the set A
$\text{Re}(\cdot)$	Real part
$s(\cdot)$	Softmax function
$\text{ReLU}(\cdot)$	Rectified linear unit i.e., $\max\{\cdot, 0\}$

Chapter 1

Introduction: Trustworthy and Certifiable AI for Civil Aviation

1.1 The Promises of Machine Learning

Since 2012, the field of machine learning has been flourishing thanks to the advent of “deep learning". Relying on increased computation power, machine learning models are now able to solve previously intractable problems for which writing explicit sets of instructions is impossible. For example, one can mention image recognition [2], object detection [3], or automatic translation [4] and other natural language processing tasks [5]. The remarkable performances of machine learning stem from its ability to extract complex correlations in huge datasets, and to exploit them in real time.

A *machine learning model* refers to any data-driven algorithm, i.e., an algorithm whose behavior is mainly determined by training data, as opposed to explicit instructions. Such model can be used to extract knowledge from data, which is termed *unsupervised learning*. These models can also be used for prediction (either as classification or regression) in the *supervised learning* setting. In this case, the model should be able to *generalize* by doing prediction on data that were not seen during training. Finally, machine learning models are used in *reinforcement learning*, where their goal is to interact with an environment and assign values to states (or action-state pairs) in order to maximize an expected reward. This thesis focuses on deep learning neural networks for offline supervised classification or regression. “Offline" means that, after training, the network is fixed and cannot learn from new data seen during inference. A *neural network* is a differentiable composition of linear maps and non-linear functions called “activation functions". Such models are usually trained by minimizing a loss function using a gradient descent algorithm.

In this introductory chapter, the promises of machine learning for the civil aviation community are briefly discussed as well as the challenges arising for the certification of machine learning systems. Special attention is given to the concept of *Trustworthy AI*, and more specifically to *robustness against adversarial attacks* which is one of the main topic developed in this thesis.

The performances of machine learning pave the way of countless new use cases in virtually all domains related to civil aviation. To give an idea of the broad range of applications, several domains are listed below with a few examples for each:

- **Aircraft design and operations**

- The Visual Landing System developed at the FAA [6] uses object detection to provide a landing guidance for the general aviation with a similar interface as the ILS.
- Voice recognition and suggestion of radio frequencies [7] §F.2.2.
- In the long term, machine learning could be used for single-pilot operations with a virtual co-pilot (Pilot AI teaming) [8].

- **Aicraft production and maintenance**
  - Controlling corrosion by usage-driven inspections [7] §F.4.1.
  - Damage detection in images [7] §F.4.2.
- **Air Traffic Management / Air Navigation Services**
  - AI-based augmented 4D trajectory prediction [7] §F.3.1.
  - Detection of abnormal approaches with GAN [9].
  - Air traffic structuration with reinforcement learning [10].
  - Speech-to-text for air traffic control [11].
- **Aerodromes**
  - AI-based screening for airport security systems [12].
  - Detection of degradation on runway's pavement [13].
- **Urban Air Mobility**
  - ACAS Xu [14].

Among the examples listed above, the only use cases that are actually implemented are not safety-critical. Implementing machine learning-based tools for safety-critical use cases is still an ongoing research topic. As discussed in the next section, machine learning models depart significantly from conventional software, and thus require the design of new development processes in order to demonstrate a sufficient level of safety. Safety-critical communities, such as the civil aviation community, are reluctant to adopt new software methods [15]. Since the introduction of new methods requires designing new processes, the benefits must widely exceed the costs. For example, Object-Oriented Programming (OOP) has been a major paradigm of software development for decades. However, OOP was only introduced in the aviation standards in 2011 (with the DO-332, supplement to DO-178C).

Furthermore, the impressive successes and promises of machine learning are the result of an "accuracy rush". In particular, very few guarantees exist to ensure the robustness of these data-driven models. Accuracy is not enough to exploit the full potential of these technologies. Safety-critical applications such as civil aviation, where a formal certification process may be mandatory for deployment, require other properties, among them explainability as well as robustness to stochastic environmental perturbations and deliberate attacks from adversaries. Even for non safety-critical applications, a model must be trustworthy in order to be accepted by the final user. However, it has been shown that the confidence level of classification models are not calibrated [16]. To be trustworthy, a model should inform the user of potential errors, which requires the ability to quantify the uncertainty of its own predictions. These properties are subsumed under the concept of Trustworthy Machine Learning.

1.2 Trustworthy AI in Aviation

1.2.1 Challenges for AI Certification

The goal of certification is to ensure that the system will safely perform its intended function under all foreseeable operating and environmental conditions. The certification practices and standards rest on decades of experience that enabled the design of a requirement-based methodology based on development assurance [17] and the V-cycle. In this methodology, high-level functions are refined into requirements allocated to subsystems and items. Each line of code must be traced back to the requirements. While conventional algorithms can automate some tasks, machine learning is able to introduce *autonomous* tools, i.e., tools that do not rely on a pre-defined set of behaviors which could be completely verified. In the long run, machine learning could deliver the autonomous aircraft as well as the autonomous air traffic control [18].

However, the current certification practices are inadequate for machine learning tools. If the current standards can easily be extended to machine learning (e.g., CS 25.1309 for the avionic domain), it will be necessary to develop new *means of conformity* well suited to the specificity of each application ([17] §E.8). For the last few years, several working groups have initiated a preliminary work on this matter (see chapter 3 of [19]). In particular, at the European level, the EASA has launched an “Artificial Intelligence Roadmap" [17] in 2019 and has published its first recommandations [7] in 2021 (restricted to offline supervised learning of level 1 “assistance to human"), with a new safety evaluation methodology tailored for machine learning labeled “learning assurance".

Machine learning models have properties that make current certification practices unsuitable:

- Machine learning models are data-driven, their behavior is dictated by the data, not by the code;
- Machine learning models are black-box, the learned parameters cannot be easily translated into instructions and explanations understandable by humans;
- When they are trained with variants of stochastic gradient descent, machine learning models are non-deterministic.

These properties bring new challenges for certification [15] in terms of:

- *Specifiability:* correct and complete capture of item requirements. Does the data correctly represent the Operational Design Domain (ODD)? Is the model really implementing the intended function? Specifiability requires the model to be *robust* to perturbations;
- *Traceability:* relationship between item requirements and code (i.e., learned parameters). Traceability requires the model to be *explainable* and *transparent* in order to be confident that the model implements the intended function correctly and safely;
- *Innocuity:* does the model introduce unintended behavior?

1.2.2 The Trustworthiness Trade-off

The various aspects required to address the challenges described in the previous paragraph can be gathered together under the label of *trustworthy machine learning*. Trustworthiness can be defined as the combination of two processes: a certification process and an explanation process

[20]. The certification process consists in demonstrating to the relevant certification authority that the model satisfies the high-level requirements and can perform safely its intended function. This process happens before the deployment of the model. On the other hand, the explanation process happens after the deployment. Two types of explanations are required: for the users, and for the investigators after an incident. More precisely, trustworthiness is concerned with the following aspects [21], including ethical considerations:

- Robustness and Generalization;
- Explainability, Transparency, and Reproductibility;
- Ethical requirements: Fairness and Privacy.

In the literature, the various requirements of trustworthiness are generally addressed separately. However, these requirements are not independent and can strongly interact. For example, robustness and explainability can be incompatible with high accuracy and generalization. Transparency and privacy can also be antagonistic, while fairness and explainability have been shown to be sometimes incompatible. Combining approaches that address each requirement does not guarantee that the final model will be trustworthy. Trustworthiness should be considered from an holistic perspective where all requirements are jointly optimized [21].

1.3 What is Robustness?

In [7], robustness is defined as the ability of a system to maintain its level of performance under all foreseeable conditions. More precisely, three types of robustness can be distinguished:

- Robustness in adverse conditions, i.e., outside the Operational Design Domain (ODD). Adverse conditions include edge cases, out-of-distribution samples, and distribution shift;
- Model stability, i.e., inside the ODD. Model stability deals with stability to small perturbations;
- Stability with respect to parameters and hyperparameters. This is linked with embeddability issues, since the values of parameters implemented in the deployed model may be slightly different from the parameters of the model trained during the development.

This thesis primarily focuses on model stability for neural networks. Robustness against out-of-distribution samples is also discussed in the context of uncertainty quantification.

In the machine learning literature, model stability is often related to robustness against adversarial attacks. Adversarial attacks are generally defined as imperceptible perturbations that can fool a model. However, seeing model stability as a cybersecurity issue in the face of deliberate attacks is misleading for several reasons [22]. First, in a real-world context, adversarial attacks are not the main vulnerabilities of machine learning models from a cybersecurity point-of-view. This is due to the fact that real adversaries have little knowledge about the model (that is often a subsystem of a larger system) and no direct access to it. Moreover, they have limited resources to conduct attacks. Adversaries can cause far more damage with lower cost by exploiting domain knowledge and social engineering techniques, as well as targeting other parts of the overall system. Second, small perturbations can be caused by many phenomenons beyond deliberate attacks: noise, sensor fault, human error etc.

As discussed in paragraph 2.1.5, the so-called adversarial vulnerability of neural networks is better described as the worst-case accuracy of the model. Adversarial examples are a specific manifestation of poor generalization. They highlight an inconsistency between the model and human oracles. This is the main reason why adversarial robustness is central to machine learning trustworthiness.

1.4 Thesis Organization and Review of Contributions

This thesis focuses on the robustness of machine learning models against evasion attacks in the context of civil aviation.

Chapter 2 reviews and discusses the literature about adversarial attacks and defenses, with an emphasis on applications of information geometry to adversarial machine learning. The chapter also covers the literature about verification methods for adversarial robustness, and concludes with a review of various uncertainty quantification methods for neural networks.

In chapter 3, an adversarial defense method based on notions borrowed from information geometry is presented. The method relies on a regularization term encouraging the model to be partially isometric, which improve the robustness of neural network classifiers against L_2 attacks. This work has been published in the Entropy Special Issue “Information Geometry for Data-Analysis" [23].

Chapter 4 explores several preliminary studies on the connections between information geometry and neural networks robustness. The chapter begins with a model of recurrent neural networks seen as a stochastic differential equation. Following this, several experiments related to the data leaf hypothesis proposed in [1] are presented.

Chapter 5 explores applications for traffic prediction by introducing two machine learning models. It begins with a review of the literature on air traffic flow prediction and complexity metrics, then describes the first model: a Long Short-Term Memory (LSTM) network for the prediction of air traffic congestion. This work was presented at the Fourteenth USA/Europe Air Traffic Management Research and Development Seminar [24]. This model has several limitations:

- It focuses on accuracy performance and is not robust against noise or adversarial attacks.
- It relies on recurrent networks that are prone to vanishing gradients and cannot be parallelized.

To address these limitations, a Transformer model applied to traffic data from the PeMS system is developed. This model is derived in two versions: a deterministic version and a Bayesian version that is able to quantify its uncertainty, in particular against noise or adversarial attacks. Note that this work has not been completed yet, and only preliminary results are provided.

Chapter 6 summarizes the main takeaways of the thesis and outlines future research directions emerging from this work.

Chapter 2

Machine Learning Robustness

In this chapter, three different aspects of machine learning robustness are reviewed. The first section addresses adversarial attacks and defenses with a focus on applications of information geometry to adversarial machine learning, which remains an underexplored topic. Several explanations proposed in the literature to account for the phenomenon of adversarial attacks in neural networks are also reviewed. It is argued that a satisfactory explanation has yet to be established. Finally, various challenges and reformulations of the adversarial problem are presented.

The second section reviews the literature about verification methods for adversarial robustness. Note that the expressions “certification" or “certified robustness" are often used in the literature in place of “verification". The word “certification" used in the literature has a different meaning as in “AI certification" as discussed in the introduction. “AI certification" refers to the entire process of demonstrating a sufficient level of safety to a relevant authority, while “certified robustness" consists in providing guarantees about the robustness of a model against specific attacks. In order to avoid any confusion, this thesis uses the expressions “verification" and “verifiability" for the specific goal of formally proving that a model is robust, and keep the word “certification" for the overall process described in the introduction. Verifiability could be part of the certification process of an AI, but is also relevant for improving the AI trustworthiness even for machine learning models that are not intended to be formally certified. This chapter focuses on recent methods tailored for neural networks as opposed to the more mature field of formal verification.

The last section discusses uncertainty quantification for neural networks, with a focus on the moment propagation method. These methods can be used to detect noise, adversarial examples, as well as out-of-distribution examples. They also improve the explainability and acceptability of machine learning models.

2.1 Adversarial Machine Learning

The expression “adversarial attacks" refers to any strategies aiming at deliberately altering the expected behavior of a model and/or extracting information from a model. Such attacks can occur at training time by corrupting the training data (either the input, the label, or both) and are thus referred as *poisoning attacks*. When they occur at inference time (i.e., after training), adversarial attacks are called *evasion attacks*. Instead of altering the behavior of the model, an attack can also aim at accessing information about the model itself, or about the data that were used to train it, which pose privacy issues. In the rest of this work, only evasion attacks are

considered. Since there is no risk of confusion, the expressions “adversarial attack" and “evasion attack" will be used interchangeably.

Evasion attacks are imperceptible malicious input perturbations that cause a “well-trained” model to completely alter its output, causing it to confidently arrive at deleterious conclusions. For deep learning models[1], they were first discovered in computer vision by Szegedy *et al.* [26] in 2013. More severely, it was found that adversarial examples are transferable [27, 28, 29], i.e., adversarial examples generated from one model can fool another different model with a high probability [30], provided that they have been trained on the same task. Adversarial examples are also ubiquitous: given a state-of-the-art model trained in a standard manner, every point of the training set has a adversarial example close to it. Given such evidence, one can justly assume that all deep learning models are vulnerable, and therefore the deployment of these models in the real-world is hinged on the understanding and diminishing of the adversarial threat [31]. Realizing the full potential of machine and deep learning requires establishing robustness guarantees for these systems, and notably combating the adversarial threat. Otherwise, adversarial machine learning can be a disaster in safety critical applications.

Evasion attacks are often classified along several criteria:

- *White-box, gray-box, physical attacks.* What is the threat model? Can the attacker access the model's parameters or the model's inputs? Black-box attacks are also concerned with transferability issues.
- *Targeted, untargeted, confidence reduction.* What is the objective of the attacker? Does the attacker want the adversarial example to be classified as a specific target class, or simply to be misclassified? Does the attacker want to completely fool the model, or simply to make it unconfident about its prediction? Attacks can also be universal in the sense that they will work on any input example.
- *Constrained, unconstrained.* Should the perturbation be restricted in some way? If yes, the attacker must choose a *dissimilarity measure*, generally a L_p norm (including the L_0 “norm"), or a semantic perturbations, or an optimal transport metric (Wasserstein metric).
- *Data type* and *tasks.* The adversarial machine learning community generally focus on images, but attacks have also be designed for text, time series, graphs etc. Attacks generally target classification tasks, but also natural language processing [32], or reinforcement learning [33, 34, 35] etc.

Studying adversarial attacks and defenses provides insights to understand the causes of adversarial vulnerability, and helps to design strategies to quantify this vulnerability, mitigate it, assess its safety/security impact, and provide guarantees. In other words, adversarial attacks and defenses can guide the efforts to address the robustness and verifiability challenges, which are core aspects of trustworthy machine learning.

2.1.1 Adversarial Attacks

In this paragraph, several classical adversarial attacks are presented, with a focus on image classification in a white-box setting. For a given input $\boldsymbol{x}$, the loss between the predicted logits $f(\boldsymbol{x})$ and the ground truth $\boldsymbol{y}$ is denoted by $\mathcal{L}(f(\boldsymbol{x}), \boldsymbol{y})$. The logit associated to the class i is denoted by $f_i(\boldsymbol{x})$.

[1] Adversarial attacks for simpler models were already studied earlier [25].

One-step attacks The simplest one-step gradient attack is the Fast Gradient Sign Method (FGSM), introduced by Goodfellow *et al.* [36] in 2015. FGSM is a L_∞ attack. For a given budget $\epsilon > 0$, the perturbed example $\boldsymbol{x}_{adv}$ is obtained from the clean example $\boldsymbol{x}$ with:

$$\boldsymbol{x}_{adv} = \boldsymbol{x} + \epsilon \times \text{sign}\left(\nabla_{\boldsymbol{x}} \mathcal{L}(f(\boldsymbol{x}), \boldsymbol{y})\right),$$

where $\nabla_{\boldsymbol{x}}$ stands for the gradient operator with respect to $\boldsymbol{x}$. Several improvements of FGSM have been proposed, for example by developing a targeted version of FGSM [37], or by using momentum to improve the quality of the attack in a black-box setting [38].

In 2016, Papernot *et al.* [39] introduced the Jacobian saliency maps attack (JSMA). It is an L_0 attack where the attacker will modify a small number of pixels in the input image that maximally impacts the predicted class.

The one-step spectral attack introduced by Zhao *et al.* [40] relies on information geometry. More precisely, it uses the eigenvector of the Fisher information matrix associated with the largest eigenvalue as the attack direction.

Iterative attacks The first iterative attack, called the basic iterative method, was introduced by Kurakin *et al.* [37]. It is a simple iteration of the FGSM attack. The most well-known iterative attack is the Projected Gradient Descent (PGD), introduced by Madry *et al.* [41] in 2018. It can be used for both L_2 and L_∞ budgets by projecting the perturbed example into the corresponding L_p ball after each iteration.

In 2016, Moosavi-Dezfooli *et al.* [42] proposed DeepFool which is an unrestricted attack, since the only parameter is the number of iterations. The idea behind DeepFool is to start with a linear classifier for binary classification. In this setting, there is a closed-form formula to obtain the distance of any point to the decision boundary, which is an hyperplane in the linear case. For a nonlinear model, the gradient is computed to obtain a linear approximation of the model around the current iterate and use the same formula as in the linear case to make a step towards the decision boundary. For multiclass classification, a chosen number of classes with the highest probabilities are selected (except the predicted class). Then, the algorithm makes a step towards the closest class, according to the linear approximation.

It has been shown that one-step attacks are more transferable than iterative attacks [43].

Optimization attacks The very first adversarial attack proposed by Szegedy *et al.* [26] in 2013 was an optimization attack called L-BFGS, based on the famous quasi-Newton BFGS optimization algorithm. However, its computation cost is too high for practical purposes.

A well-known optimization attack was introduced by Carlini and Wagner (C&W) [44] and was able to break several defenses including defensive distillation (presented in section 2.1.2.1). C&W is a targeted attack with L_p budget. Let t be the target class. The perturbation $\boldsymbol{\epsilon}$ applied to the clean example $\boldsymbol{x}$ is obtained by minimizing the following expression:

$$\min_{\boldsymbol{\epsilon}} \|\boldsymbol{\epsilon}\|_p + \kappa \times \left(\max_{i \neq t} f_i(\boldsymbol{x} + \boldsymbol{\epsilon}) - f_t(\boldsymbol{x} + \boldsymbol{\epsilon})\right),$$

where $\kappa > 0$ is a hyperparameter, and f_i is the i-th component of f.

Semantic attacks Semantic attacks (also referred to as *parametric attacks*) do not work on the pixel space of images and do not restrict the perturbation to be in a L_p ball. Instead, the perturbation is visible but does not change the semantic meaning of the image, e.g., adversarial

rotations, translations, deformations [45], or using a generative model [46]. Xiao *et al.* [47] have proposed a semantic “spatial" attack where pixel intensity is moved along flows that do not change the semantic content of the image.

2.1.2 Adversarial Defenses

In this paragraph, some classical empirical defenses against adversarial attacks are presented, focused on image classification. The defenses are organized according to the following taxonomy:

- Gradient masking;
- Robust optimization;
- Adversarial detection.

2.1.2.1 Gradient Masking

The gradient masking strategy consists in making gradient-based attacks difficult or impossible by limiting the information accessible with backpropagation. The main weakness of this strategy is that it can only “confound” the adversaries: it cannot eliminate the existence of adversarial examples [48].

Defensive distillation Defensive distillation is a variation of model distillation, where a new (generally smaller) model is trained to mimic another already trained model, adding some regularization terms to the loss in order to encourage the two models to be similar.

The idea of defensive distillation [49] is to train a first model with one-hot encoding. This model will be accurate but not robust. Then, a second model is trained by learning to predict the predictions of the first model (called the “teacher model"). Since the first model will not predict one-hot vectors, there is a kind of smoothing, or randomization, that helps the second model to be more robust. Let $f_T(\boldsymbol{x})$ be the logits of the teacher model. A *soft label* $\tilde{\boldsymbol{y}}$ is computed as:

$$\tilde{\boldsymbol{y}} = s\left(\frac{f_T(\boldsymbol{x})}{\tau}\right),$$

where s is the softmax function, and τ is a hyperparameter called the *temperature*. If $f(\boldsymbol{x})$ are the logits of the distilled model, the loss function is:

$$\mathcal{L}(f(\boldsymbol{x}), \tilde{\boldsymbol{y}}) = -\sum_{i=1}^{c} \tilde{\boldsymbol{y}}_i \log(s_i(f(\boldsymbol{x}))),$$

where c is the number of classes, and s_i is the i-th component of s.

In [50], the authors build a dataset containing only robust features using a robust model. It can be seen as a “data distillation" instead of a model distillation.

Breaking gradient descent In [51], the authors propose a defense where the model is trained on adversarial examples crafted on another model. They show that such a model exhibits a gradient masking effect: the loss landscape is very curved and the gradient does not point toward the direction of a local maxima.

Obfuscated gradients Obfuscated gradients correspond to various techniques: shattered gradients, stochastic gradients, exploding & vanishing gradients [48]. In the shattered gradients strategy, non-differentiable operations can be added such that the model is no longer differentiable with respect to the inputs, or the computation of the gradient can be numerical unstable. Stochastic gradients correspond to several methods: training several models and using one of them randomly, dropping randomly some neurons, or transforming the input randomly. Finally, exploding & vanishing gradients consists in doing several neural networks evaluations. More precisely, the method uses a generative model to project the attack to the "data manifold" then feed the projected point into the model. The concatenation of the generative model and the original model creates a very deep network such that the gradients with respect to the input tend to vanish/explode, and hence are hardly usable to craft adversarial attacks. Exploding & vanishing gradients are also linked to denoising methods, where the adversarial noise is suppressed from the input before being fed to the model. Denoising methods include the Deep Contractive Networks [52] and the High-Level Representation Guided Denoiser [53].

2.1.2.2 Robust Optimization

The expression "robust optimization" as used here is an umbrella term that gathers various defense strategies where the model is robustified thanks to an adapted learning mechanism. Robust optimization methods are divided into regularization methods where the training objective is modified in order to increase robustness, and adversarial training where adversarial examples are directly used during training.

Two strategies are possible to robustify a model with optimization methods:

- Minimize the average adversarial loss (i.e., the maximal training loss in a neighborhood of each training point);
- Maximize the average minimal perturbation distance.

Regularization methods In [54], Cisse *et al.* introduced the Parseval networks, which consist in penalizing the Lipschitz constant of each layer. The authors argue that adversarial sensitivity can be controlled by the Lipschitz constant of the network. Another work [55] provided an upper bound for the generalization error using the Lipschitz constants of the model and of the loss function. This suggests that the Lipschitz constants also control the generalization, which means that accuracy and robustness may not be antagonistic.

The Lipschitz constant Λ_p of the model (with respect to the L_p norm) is less than or equal to the product of the Lipschitz constants of each layer. Hence, if the Lipschitz constants of the layers are greater than 1, Λ_p can grow exponentially with the depth of the network. The Lipschitz constant of the k-th layer is basically the L_p matrix norm of the weights $\boldsymbol{W}^{(k)}$. Recall that $\|\boldsymbol{W}^{(k)}\|_2$ is the *spectral norm* which is the largest singular value of $\boldsymbol{W}^{(k)}$, and $\|\boldsymbol{W}^{(k)}\|_\infty$ is the maximum 1-norm of the rows. When going through an aggregation layer, the Lipschitz constants are simply summed. For activation function, it is enough to have a Lipschitz constant less than or equal to 1, which is the case for ReLU.

To control Λ_p, Parseval networks ensure that each layer has a Lipschitz constant less than or equal to 1 (to avoid the exponential grow). Then, it uses an usual regularization scheme (like weight decay) to control the overall Lipschitz constant of the network. Two modifications are used to ensure that the Lipschitz constant of each layer is smaller than 1:

1. Orthogonality of weight matrices. To control Λ_2 for linear and convolutional layer, it suffices to ensure that the largest singular value is 1. This can be achieve by ensuring that the rows of the weight matrices are orthogonal (or "Parseval tight frame" when the matrix is not square) because the singular values of an orthogonal matrix are all equal to 1. To control Λ_∞, it is possible to rescale each row to have a 1-norm smaller than 1.
2. Convex combination in aggregation layers. The coefficients of the combination are learnable. Since the combination is convex, Λ_p will remain smaller than 1.

In order to stay on the manifold of orthogonal matrices (i.e., the *Stiefel manifold*), the authors propose to do one gradient step of the layer-wise regularizer:

$$R_\beta(\boldsymbol{W}^{(k)}) = \frac{\beta}{2}\|\boldsymbol{W}^{(k)}(\boldsymbol{W}^{(k)})^\top - \boldsymbol{I}\|_2^2, \tag{2.1}$$

where β is a hyperparameter, and $\boldsymbol{I}$ is the identity matrix.

Another method called *label smoothing* can be used in classification tasks. Let c be the number of classes and $\alpha \in [0, 1)$ a hyperparameter. Label smoothing consists in replacing the true distribution $\boldsymbol{q}$ (where $\boldsymbol{q}_i = 1$ if i is the true label and $\boldsymbol{q}_j = 0$ for $j \neq i$) by a "smoothed" distribution $\boldsymbol{q}^{LS}$ such that

$$\boldsymbol{q}_i^{LS} = (1 - \alpha)\boldsymbol{q}_i + \frac{\alpha}{c}.$$

It is a linear combination of the true distribution with the uniform distribution. In [56], Müller *et al.* show that label smoothing reduces the over-confidence of the model, but impairs knowledge distillation. The largest eigenvalue suppression method introduced by Shen *et al.* [57] is equivalent to label smoothing.

Adversarial training Adversarial training is the most famous adversarial defense, and the only defense that yields good empirical robustness. This method was first described by Goodfellow *et al.* [36] and developed by Madry *et al.* [41]. Let n be the number of training examples. The idea is to minimize

$$\frac{1}{n}\sum_{i=1}^{n} \max_{\boldsymbol{\epsilon} \in \Delta(\boldsymbol{x}_i)} \mathcal{L}(f(\boldsymbol{x}_i + \boldsymbol{\epsilon}), \boldsymbol{y}_i),$$

instead of

$$\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(\boldsymbol{x}_i), \boldsymbol{y}_i),$$

that is used for normal training. The set $\Delta(\boldsymbol{x}_i)$ contains all the allowed adversarial perturbations for $\boldsymbol{x}_i$.

Many variants and improvements of the adversarial training method have been proposed for the last few years [58]. For example, adversarial training can be improved by using another loss function. This is the approach followed by TRADES [59] and MART [60].

2.1.2.3 Adversarial detection

Instead of robustifying the model, another strategy consists in detecting the adversarial examples before they are fed into the machine learning model.

Some detection methods rely on an auxiliary model to classify adversarial and clean examples [61]. Other methods use statistical properties of the inputs to detect adversarial examples, using PCA [62] or other statistical tests to see if two examples are from the same distribution or not [63].

2.1.3 Information Geometry and Adversarial Machine Learning

This paragraph focuses on applications of information geometry to adversarial machine learning, which is an underexplored application for information geometry.

2.1.3.1 Motivations

Information geometry deals with *statistical manifolds*. In its most simple form, a statistical manifold is a parameterized family of distributions $\mathcal{S} = \{f(\boldsymbol{u}, \boldsymbol{\theta})\}$. Here, f is a probability density function over the sampling variable $\boldsymbol{u}$, and parameterized by $\boldsymbol{\theta}$. Moreover, this family of distributions is assumed to be *regular* in the sense that there is a one-to-one mapping $f(\boldsymbol{u}, \boldsymbol{\theta}) \mapsto \boldsymbol{\theta}$ to an open set of an Euclidean space $\mathbb{R}^d$ (this is why the family is *parameterized*), and this mapping has full rank (in the sense that the d functions $\{\partial f(\boldsymbol{u}, \boldsymbol{\theta})/\partial \boldsymbol{\theta}^i\}_{1 \leq i \leq d}$ are linearly independent). Information geometry studies the properties of this family of distributions that are invariant under coordinate change $\boldsymbol{\theta} \mapsto \boldsymbol{\theta}'$, i.e., properties that are *intrinsic* to this family of distributions. It is important to note here that information geometry is not directly related to the branch of machine learning called "manifold learning" where the goal is to learn a latent manifold from valued data.

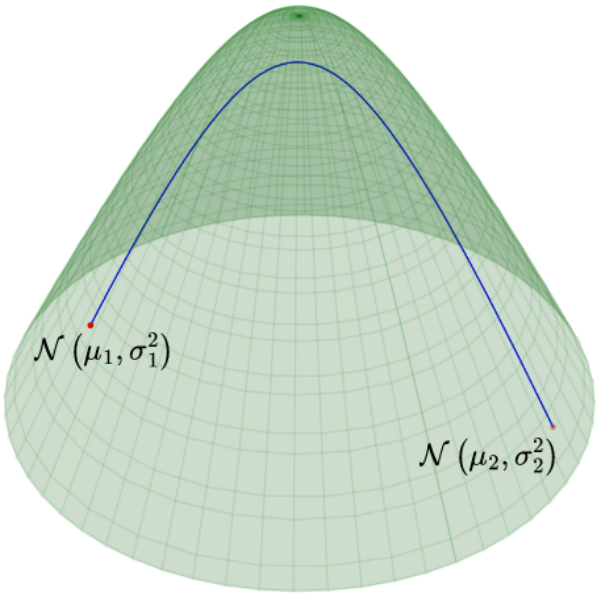


Figure 2.1: The geodesic between two normal distributions is a hyperbola (in blue). Its length is the Fisher-Rao distance between the two distributions.

A natural Riemannian metric for any statistical manifold (i.e., a metric that reflects its statistical properties) is the *Fisher information metric* (FIM) [64, 65]. The distance induced by the FIM is called the Fisher-Rao distance. The invariance properties of the FIM (explained below) makes the Fisher-Rao distance the "true" information distance between distributions. Figure 2.1 provides an example: the statistical manifold of the univariate Normal distributions $\mathcal{N}(\mu, \sigma^2)$ - endowed with the FIM - can be immersed into the Euclidean space $\mathbb{R}^3$ as a hyperboloid sheet. In simple examples, such as this one, a close-formed formula can be derived for the Fisher-Rao distance [66, 67]. Unfortunately, this is not possible in general.

The FIM has two remarkable properties that motivate its choice as the natural metric on a statistical manifold. First, it is the "infinitesimal distance" of the *relative entropy* (Theorem 4.4.5 in [68]). More precisely, if KL is the relative entropy (also known as the Kullback–Leibler divergence) and if $\boldsymbol{d}_{FR}$ is the Fisher–Rao distance, then given two distributions $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, it follows that (see Theorem 4.4.5 in [68]):

$$\mathrm{KL}\left[\boldsymbol{\theta}_1 || \boldsymbol{\theta}_2\right] = \frac{1}{2} \boldsymbol{d}_{FR}^2\left(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\right) + o\left(\boldsymbol{d}_{FR}^2\left(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\right)\right).$$

The same result can be restated infinitesimally using the FIM $\boldsymbol{g}$, as follows:

$$\mathrm{KL}\left[\boldsymbol{\theta} || \boldsymbol{\theta} + d\boldsymbol{\theta}\right] = \frac{1}{2} \boldsymbol{g}_{\boldsymbol{\theta}}\left(d\boldsymbol{\theta}, d\boldsymbol{\theta}\right) + o\left(\boldsymbol{g}_{\boldsymbol{\theta}}\left(d\boldsymbol{\theta}, d\boldsymbol{\theta}\right)\right), \tag{2.2}$$

where $d\boldsymbol{\theta}$ is seen as a vector of the tangent space $T_{\boldsymbol{\theta}}\mathcal{S}$.

The other remarkable property of the FIM is the Chentsov's theorem [69] which claims that the FIM is the *unique* Riemannian metric on $\mathcal{S}$ that is invariant under sufficient statistics (up to a multiplicative constant). Informally, the FIM is the only Riemannian metric that is statistically meaningful. In [70], Amari and Nagaoka state a more general result. Along with the FIM, they introduce a family of affine connections parameterized by a real parameter α, called the α-connections. Theorem 2.6 in [70] states that an affine connection is invariant under sufficient statistics if and only if it is an α-connection for some $\alpha \in \mathbb{R}$. In other words, the α-connections are the only affine connections that have a statistical meaning.

While equation 2.2 gives the second-order approximation of the relative entropy, an α-connection can be seen as the third-order term in the Taylor approximation of some divergence [70]. More precisely, a given α-connection can be canonically associated with a unique divergence (while the second-order term is always given by the FIM). If $\alpha = \pm 1$, the canonical divergences are the relative entropy and its dual (obtained by switching the arguments in $\mathrm{KL}[\boldsymbol{\theta}_2||\boldsymbol{\theta}_1]$). More generally, for $\alpha \neq 0$, the canonical divergence is not symmetric. The only canonical divergence that is symmetric is obtained for $\alpha = 0$, and is precisely the square of the Fisher–Rao distance. Thus, the Fisher–Rao distance is the only statistically meaningful distance.

More discussion about information geometry can be found in section 3.1.1.

2.1.3.2 Application to Adversarial Robustness

In deep learning models, a statistical manifold can be constructed by either representing the weights of the model as parameters of a family of distributions or by viewing the inputs of the model as parameters. When representing the weights as parameters, the FIM has been applied to optimize neural networks based on the natural gradient descent [71, 72, 73, 74], as well as meta-learning [75, 76].

The development of information geometric tools for adversarial robustness is a more recent and still an under-explored research direction. Here, the inputs of the model are considered to be the parameters of the statistical manifold. In these works, the FIM has been adopted to construct new adversarial attacks, such as the one-step spectral attack [40] and its two-step extension using the Riemannian curvature of the FIM [77]. Adversarial defenses against FIM attacks include an extension of adversarial training [58], regularization [78, 57], and detection of adversarial examples [79]. A variant of the FIM, called the "local data matrix" can be used to show that the training dataset lies on a geometric structure of a foliation, the *data leaf* [1]. The FIM can also help investigate and explain the behavior of neural networks by taking inspiration from the biophysical mechanisms of operation in the brain [80].

Another interesting line of work empirically shows that adversarial samples are features that help the standard generalization of deep learning models by disentangling non-robust and robust features [50]. A follow-up work, based on the FIM metric, points out that achieving decent performance and being adversarially robust are not two completely contradicting objectives [81]. It is still possible that there exists a perfect balance point where both objectives could be satisfied simultaneously.

Shen *et al.* [57] developed an approach to suppress the FIM eigenvalues, which was shown to be equivalent to the well-known label smoothing method [56]. Label smoothing was experimentally proven to prevent the network from becoming over-confident [56]. Martin *et al.* [79] use the weights instead of the inputs as parameters to detect adversarial attacks by measuring the average magnitude of the derivative of the log-likelihood evaluated at the learned parameter.

Picot *et al.* [58] introduce a regularizer called FIRE to optimize the trade-off between accuracy and robustness. This regularization is an improvement of the adversarial training framework [36, 41].

While Zhang *et al.*, [59] use the KL divergence to regularize the cross-entropy loss, Picot *et al.* [58] use the Fisher-Rao distance i.e., the Riemannian distance induced by the FIM. Shi *et al.* [81] study the generalization of deep neural networks by disentangling an information-theoretic non-robust component, responsible for adversarial vulnerability, and a robust component. As already argued by Ilyas *et al.* [50], Shi *et al.* claim that deep neural networks rely on both robust and non-robust features to achieve high test accuracy. However, non-robust features can easily be perturbed to craft adversarial examples. Thus, there is a trade-off between robustness and accuracy since preventing the models to rely on non-robust features will increase robustness and decrease accuracy.

Grementieri and Fioresi [1] also study the FIM of neural networks with respect to inputs. Denoting the pullback of the FIM by $\widetilde{\boldsymbol{G}}$, the author prove that the distribution (in the geometrical sense) defined on the input space by $\boldsymbol{x} \mapsto (\ker \widetilde{\boldsymbol{G}}_{\boldsymbol{x}})^{\perp}$ is integrable for ReLU networks. Moreover, the authors claim that one of the integral submanifold, which they call the *data leaf*, contains all the training and testing data. The data leaf has dimension strictly less than the number of classes, thus it is a low-dimensional representation of the data as seen by the model. The data leaf will be discussed in more detail in chapter 4.

The FIM has thus been harnessed to develop attacks and defenses, but a precise robustness analysis is yet to be proposed. In particular, no provable guarantees were derived using information geometric tools. More importantly, one may describe geometry as the discipline studying *classification properties* (i.e., the study of invariants) and *local-global properties*, e.g., the Gauss-Bonnet theorem relating a local property (the curvature) with a global property (the Euler characteristic) [82]. The study of adversarial robustness, which is non-local by definition and contrary to accuracy, should benefit greatly from a geometrical vision. However, except for a few modest endeavors [58, 1, 77], the current literature on adversarial robustness is mainly concerned with the FIM and its spectrum (which are very local objects) without unfolding the full arsenal developed in information geometry or differential geometry, such as, for example, the dual structure on statistical manifolds [83]. To ensure the applicability of such machinery, one caveat is the question of whether it will be computationally tractable and scalable. Geometry may also provide impossibility results regarding robustness guarantees with limited computational power. Indeed, Bubeck *et al.* [84] argued that adversarial vulnerability of classifiers in high dimension is "likely not due to information theoretic limitations, but rather it could be due to computational constraints". They provided evidence to support the hypothesis that "identifying a robust classifier from limited training data is information theoretically possible but computationally impossible". Interestingly, their evidence weakens the notion that identifying a robust classifier requires huge amount of training data.

2.1.4 Proposed Explanations

Despite the huge research effort[2], there is no consensus to explain the existence of adversarial examples. Thus, it is not known if a robust model can be built, in theory as well as in practice. Another topic of contention is the existence (or nonexistence) of an "accuracy-robustness trade-off", with some works arguing in favor of its existence [85, 50], and other in favor of its nonexistence [86]. This paragraph provides a non-exhaustive list of explanations for adversarial

[2]More than 8,000 papers related to adversarial robustness have been published on arXiv during the last decade. See `https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html`.

examples proposed in the literature. These explanations are classified in three categories: explanations relating adversarial vulnerability with high-dimension datasets, explanations relating adversarial vulnerability with overfitting, and finally, the “non-robust features" hypothesis.

2.1.4.1 High-dimensionality

Concentration of measure in high-dimensions This explanation was developed by Gilmer *et al.* [87] (2018), Mahloujifar *et al.*, [88] (2018), Fawzi *et al.* [89] (2018), and Shafahi *et al.* [90] (2019).

The high dimensionality of the input space can present fundamental barriers on the robustness of classifiers. Theoretical and experimental proofs have shown that, *for certain data distributions*, any decision boundary is close to a large fraction of inputs and hence no classifier can be robust against small perturbations. This theory attempts to explain why, despite countless propositions for adversarial defenses, there are still new attacks that manage to break those defenses.

This theory certainly explains partially the ubiquity of adversarial attacks. It provides bounds on the robustness that can be achieved by any classifier of certain datasets and it is not incompatible with other explanations. However, this theory has several drawbacks:

- It implies that adversarial attacks are far less frequent for small dimensions. However, a model in two dimensions can also overfit the training set and be very brittle in the face of adversarial attacks.
- There exist models that are empirically robust to a wide range of attacks, even in high dimensions.
- Biological systems such as the human visual system process high dimensional data, but are not prone to the adversarial vulnerability exhibited by machine learning models.

In [87], the authors use a synthetic dataset for a binary classification task (two concentric spheres). First, they show experimentally that a model with a very low error rate still has adversarial attacks *on the data manifold* (i.e., on the same sphere as the original point). Then, they provide an upper bound for the adversarial robustness which depends on the error rate and on the dimension of the problem. This is a consequence of the concentration of measure in high dimensions. In high dimension, almost all the points of the sphere are close to the equator *for any equator*[3]. Let E be the error set, i.e., the set of points that are misclassified. Even if E has small measure and is well concentrated at one area of the sphere (and not dispersed across the sphere), then E will be close to every equator. In particular, if there is an equator such that all of E is on one hemisphere, then the opposite pole (which is the farthest point to E) will be at a distance $\approx \sqrt{2}r$ from E where r is the radius of the sphere[4]. It must be emphasized that this result is highly dependent on the geometry of the synthetic dataset. There are no similar bounds for specific real-world datasets like MNIST or CIFAR-10 (at least in the literature reviewed for this thesis), and even less for arbitrary datasets. Thus, the consequences of this result remain limited.

[3]Let $\boldsymbol{x} = (x_1, \ldots, x_d)$ be a random point uniformly sampled over the unit sphere, whose coordinates are expressed in some orthonormal basis. An equator is the set of points on the sphere such that $x_i = 0$ for some i. Each x_i has the same distribution and $\sum x_i^2 = 1$. Since d is large, each x_i is very likely to be small. Hence, the point $\boldsymbol{x}$ is very likely to be close to all equators.

[4]The maximum distance between two points of the sphere is $2r$.

Linear explanation This was the first explanation to achieve a consensus before being challenged. It was proposed by Goodfellow *et al.* [36] in 2015.

In [26], it was supposed that adversarial examples are due to extreme nonlinearity of deep neural networks, combined with insufficient regularization (overfitting). The idea was to imagine that adversarial examples are a dense set with measure zero, such that all examples from the training and test sets are *not* adversarial.

For Goofellow *et al.* [36], these hypothesis are unnecessary. Linear behavior in high-dimensional spaces is sufficient to cause adversarial examples. Using this assumption, they propose the FGSM attack which proves to be very effective. They show that adversarial training provides additional regularization and reduces the model's vulnerability, while generic regularization methods (dropout, pretraining, model averaging) do not reduce the vulnerability to adversarial examples. They claim that using very nonlinear models increases the robustness, but this claim has been challenged (see [50]).

For Goodfellow *et al.*, the trade-off is between *models that are easy to train due to their linearity and models that use nonlinear effects to resist adversarial perturbation.* For them, the solution to adversarial attacks lies in the efficiency of the optimization methods. They claim that linear models cannot escape adversarial examples (the authors of [91] claim that this is not true), so only nonlinear models (with hidden layers) can be trained to resist adversarial attacks.

The reasoning is as follows. Consider a linear model $\boldsymbol{w}^\top \boldsymbol{x}$ for some weight vector $\boldsymbol{w}$. Let $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\eta}$ be an adversarial example. One has $\boldsymbol{w}^\top \tilde{\boldsymbol{x}} = \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{w}^\top \boldsymbol{\eta}$. Assume that a bound is imposed on the max norm of the perturbation: $\|\boldsymbol{\eta}\|_\infty < \epsilon$. Assume that the average magnitude of the weight $\boldsymbol{w}$ is $\mu = \sum_i \boldsymbol{w}_i / d$. The increase caused by $\boldsymbol{w}^\top \boldsymbol{\eta}$ can be maximized by choosing $\boldsymbol{\eta} = \epsilon \times \text{sign}(\boldsymbol{w})$. In this case, one has $\boldsymbol{w}^\top \boldsymbol{\eta} = \sum_i \epsilon |\boldsymbol{w}_i| = \epsilon \mu d$. For a fixed-size perturbation ϵ, the increase in the activation grows linearly with d. To conclude, the problem is due to:

- The model being linear. More precisely, a linear model is forced to attend exclusively to the signal that *aligns most closely with its weights*, even if other signals have greater amplitude (but are not aligned with the weights).
- The dimension of the input being high.

Adversarial examples generalize across different models because different models learn similar functions when trained on the same task. They also claim that the *direction of perturbation is more important than the specific point in space*, which is a counter-argument for adversarial examples begin "dense". This also explains why adversarial perturbations generalize across different clean examples. They also show that ensemble methods are not resistant to adversarial examples.

According to [91], the linearity argument is unconvincing because, even if $\boldsymbol{w}^\top \boldsymbol{\eta}$ grows linearly with d, this is also the case for $\boldsymbol{w}^\top \boldsymbol{x}$, such that *their ratio stays constant.*

Piecewise linear decision boundaries This explanation was proposed by Shamir *et al.* [92] in 2019.

They study $\mathbb{R}^d$ with the L_0 (Hamming) metric and explain why a deep neural networks trained to distinguish between c classes can be fooled with adversarial examples of Hamming distance $\sim c$.

They work on targeted attack (which is harder than untargeted attack). According to the authors, all the classes defined by neural networks in the input space $\mathbb{R}^d$ are intertwined in a

fractal-like way so that *any point in any class is simultaneously close to all the boundaries with all the other classes.* The problem with this work (as mentioned by [50]) is that they are using the L_0 “norm" (number of elements that changed) and their arguments cannot extend to other norms. They show that a small number of directions ($\sim d$) can be chosen such that if one move along these directions *as far as desired*, then any other class can be reached. This is particularly true if the input dimension is high. In particular, this type of attack does not respect the input range (for example, image pixels may be bounded between 0 and 255).

2.1.4.2 Overfitting

Insufficient data According to Schmidt *et al.* [93], adversarial examples arise due to insufficient information about the true data distribution. Robust models may achieve robustness by reducing the effective/useful amount of information in the training data (discarding non-robust features).

Boundary tilting This explanation was proposed by Tanay and Griffin [91] in 2016.

They argue that the decision boundaries extend beyond the data manifold and can be lying close to it under certain circumstances. Regularization is the solution to adversarial examples by preventing finite-sample overfitting. This explanation is in contradiction with the distillation hypothesis which suggests that adversarial examples are made of features inherent to the data distribution, hence lying on the data manifold.

The mathematical analysis of [91] focuses on linear classifiers, like SVM or logistic regression. The authors use in-depth geometrical analysis but using very classical tools of Euclidean geometry. They do not provide explanations about how their framework extends to deep neural networks.

2.1.4.3 Non-robust features

This explanation was introduced by Ilyas *et al.* [50] and Tsipras *et al.* [85] in 2019. It can be seen as a form of dataset distillation since their goal is to show that robustness can be achieved *by modifying only the training set and not the training process.* It shows that adversarial vulnerability is mainly a property of the dataset.

This theory provides an explanation of *adversarial transferability* which is a natural consequence of the existence of non-robust features. Other works (see [50]) have studied adversarial transferability from a theoretical perspective, using *simple models* or *unbounded perturbations.* In particular, Tramer *et al.* [94] (2017) showed empirically the existence of *adversarial subspaces* i.e., linear subspaces containing only adversarial perturbations. Moreover, they showed that there is significant overlap in the adversarial subspaces between different models, thus explaining transferability. The directions of these subspaces might correspond to non-robust features.

The non-robust features theory also explains *universal adversarial perturbations* i.e., perturbations that work on different inputs. These perturbations correspond to non-robust features that appear in different images/input points.

In [50], other works are mentioned to study the manipulation of dataset features and its impact to robustness. It seems that removing information from the training set improve the robustness, which is once again coherent with the non-robust features theory.

The authors of [50] argue that there is a trade-off between accuracy and robustness. If non-robust

features are not used, the model will lose accuracy but gain robustness. However, discarding non-robust features means that the model is more interpretable.

An answer to [50] by Nakkiran[5] shows that it is possible to construct an attack which is non-transferable by design. Hence, this type of attack is really a “bug" of the model under attack and not a “feature" of the dataset. Nakkiran speculates that Ilyas *et al.* were unable to detect the “bug-type" attack because they use the PGD method to robustify their dataset. However, the PGD method is intrinsically prone to create attacks “along directions of the data" that are transferable, and hence that are of the “feature-type". In other words, the adversarial examples that one obtains depend on the attack that one uses.

2.1.5 Challenges for Model Stability

Beyond the security threat, adversarial attacks are the symptom of the dramatic lack of robustness of deep learning systems and neural networks. The first proposed attacks were defeated by defenses that have been themselves defeated by other stronger attacks in a never-ending cat-and-mouse chase [44, 95, 96, 97, 98, 99]. Despite the attempts to explain adversarial vulnerability described in the previous section, a fundamental understanding of how neural networks learn is still missing. For example, it has been shown experimentally that, even for simple tasks, neural networks trained with stochastic gradient descent (SGD) cannot converge to an intuitive decision boundary, leading to high adversarial vulnerability while achieving very high accuracy [87].

Neural networks do not understand learning tasks The main result of [87] was already discussed in the previous section. This paragraph emphasizes several important points from [87] that are relevant with respect to adversarial robustness:

- There may be an upper bound on the adversarial robustness (given an error rate and the dimension of the problem), but this bound seems to be high for real-world datasets and thus is not the main reason behind adversarial vulnerability of current deep learning models. For example, this bound is not reached on MNIST, even with adversarial training.
- Accuracy and robustness may *not* be antagonistic since the upper bound on the robustness is proportional to the accuracy of the model (more precisely to the generalization capability of the model).
- Models trained with cross entropy and SGD seem to achieve low error rate *without really understanding the task*. The models learn to detect statistical correlation by “adding wrong numbers that add to good predictions". Even a model initialized with zero error rate and zero adversarial vulnerability (but nonzero cross entropy) will converge to a model with non zero (but very small) error rate and adversarial vulnerability. This is because the cross entropy is not perfectly matched with accuracy, and the model will reduce the average cross entropy *while dramatically increasing the worst-case cross entropy for a very small number of points*. It seems to indicate that something is wrong about the way models are trained.
- Models are even able to achieve very low error rate *while using only a fraction of the entire input dimensions*, while it is impossible to achieve zero error rate without using all dimensions. Hence, understanding a (simple) task and reaching zero error rate is *very* different from achieve a very low (but nonzero) error rate by looking at statistical correlations.

[5] `https://staging.distill.pub/2019/advex-bugs-discussion/response-5/`

- There is nothing special about adversarial examples in the sense that, for a very accurate model, any incorrectly classified point is close to a correctly classified point. In other words, every incorrectly classified point *is* an adversarial example, which is in contradiction with the adversarial detection strategy.

Choice of the dissimilarity measure Most adversarial attacks and defenses focus on pixel-based perturbations measured with L_p norms.

In [100], the authors introduces a new typology of attacks. They call *sensitivity attacks* every attack where a small perturbation changes the prediction of the model while the true label has not changed. All attacks mentioned in section 2.1.1 are sensitivity attacks. The author introduces *invariance attacks*, defined as small perturbations that change the true label, while the prediction of the model does not change. The authors claim that there is a trade-off between sensitivity robustness and invariance robustness. When addressing sensitivity attacks, the true label is assumed to not change in a small neighborhood of each training and test points. This is in contradiction with invariance attacks, and it is certainly not true as illustrated in [100] where a small change in L_2 norm leads to a semantically different image. This is coherent with the idea that pixel-based coordinates and L_p norm are bad measures of "dissimilarity": examples that are closed in terms of L_p norm can belong to different classes. Another example comes from [101] where the authors craft adversarial examples for image recognition using physical raindrops. The perturbations are large (i.e., the L_p norm between the clean and perturbed images is large) and clearly visible, but the perturbed image looks natural even thought it can fool the network. As discussed in section 2.1.1, semantic attacks such as in [47] can be indistinguishable from a human perspective while fooling a model and having a large L_p norm.

This discussion suggests that the notion of adversarial examples is not well-defined. The expression "adversarial attack" conveys the idea of an ill-intentioned, imperceptible perturbation that can fool a model. However, adversarial attacks have limited relevance in terms of cybersecurity [22]. Real-world adversaries have little knowledge about the actual machine learning model embedded into a larger system, and no direct access to it (which is generally required to craft efficient adversarial examples, even with black-box attacks). Moreover, real adversaries have economic constraints. Even if they could damage a system with adversarial attacks, the cost of crafting successful adversarial examples is often prohibitively high. In fact, real adversaries rely on domain knowledge (i.e., dependent on the target system) as well as on social engineering to achieve their goals.

With this in mind, one may ask what do adversarial attacks really mean? This thesis argues that the expression "adversarial attack" is misleading, because there is no need of an adversary. Consider a classification task and let $\boldsymbol{h}$ be a human oracle, f a machine learning model, and $\boldsymbol{x}$ an input such that $f(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x})$, meaning that the model and the human oracle agree on $\boldsymbol{x}$. Let $\eta(\boldsymbol{x})$ be a "neighborhood" of $\boldsymbol{x}$ defined with some suitable dissimilarity measure. An adversarial example is an input $\boldsymbol{x}'$ satisfying:

$$(\boldsymbol{x}' \in \eta(\boldsymbol{x})) \wedge (f(\boldsymbol{x}') \neq \boldsymbol{h}(\boldsymbol{x}))\,.$$

In other words, "adversarial attacks" highlight an inconsistency of the decision boundaries between machine learning models and human oracles. Machine learning models and humans do not understand the task in the same way.

What are the implications for safety? Since, according to [87], adversarial examples exist in subsets with measure almost equal to zero, it begs the question of how much extensive testing can really build trust against model instability. Empirically probing the model with classical attacks cannot guarantee that there do not exist adversarial examples outside the scope of

the chosen attack methods. More generally, how can an AI assurance process and rigorous data management build trust that a model will not exhibit unintended behavior caused by adversarial examples?

Another implication is that robustness and explainability are not independent from each other. A model that is not robust because it does not understand the task that it is learning, cannot be explained. This is another manifestation of the fact that trustworthiness cannot be tackled from one perspective at a time. Finally, robustness is a consequence of the training process, which limits the effectiveness of post-hoc methods that try to improve the robustness of models that are already trained.

2.2 Verification of Model Stability

Following the discussion of the last paragraph, this section moves away from the question of improving the stability of a model to the question of providing guarantees that a given model is robust against a specific set of perturbations (i.e., any L_2 perturbations with norm less than a given ϵ). A model that is provably robust against a given set of perturbations is said to be *verified* for this set of perturbations.

For the overall certification process of model stability for machine learning models, the machine learning community can draw inspiration from the certification of traditional software [20], including processes such as:

- *Reviews*, of both the development process and the properties of the software items.
- *Testing*, using methodologies such as coverage criteria, test case generation, mutation testing, simulation, scenario etc. For relevant evaluation of machine learning robustness, the literature uses specialized benchmark relying on adaptive attacks, such as AutoAttack [102].
- *Verification*, also called *formal methods*. A model can be verifiable by-design, or verified post-hoc.

Formal methods is a mature field [103] and will not be reviewed it in this thesis. This paragraph will briefly highlight several formal methods that have been adapted to neural networks. Verification methods can be split into *exact* and *conservative* methods. Given an input point $\boldsymbol{x}$, exact methods provide the exact set of points around $\boldsymbol{x}$ with the same class as $\boldsymbol{x}$, i.e., they are both sound and complete. Exact formal methods can rely on Satisfiability Modulo Theories [14] or mixed integer linear programming [104]. However, since exact verification is a NP-hard problem, exact methods are unable to scale beyond medium-sized models (a few hundreds neurons), and thus are completely inapplicable to modern deep learning models [105].

Conservative methods are sound but incomplete in the sense that they may exclude points that are robust, or refuse to take a decision. Conservative methods can be either deterministic or probabilistic. Deterministic formal methods can bound the global Lipschitz constant of the model, e.g., by imposing an orthogonality constraint on the weight matrix of each layer [106, 107], or by relying on convex relaxation [108, 109, 110]. Once again, these methods are unable to scale to very large models.

This section will focus on recent conservative methods, both deterministic and probabilistic, that were developed explicitly for neural networks, and that can scale to large models and large datasets. Two families of verification methods for neural networks are reviewed: randomized smoothing and Lipschitz neural networks.

2.2.1 Randomized Smoothing

Randomized smoothing is a probabilistic verification method that is applied post-hoc. The main advantage of randomized smoothing is that it is model-agnostic: it can work on any classifier, even if it does not come from a neural network.

Randomized smoothing for verification was introduced in [111] and [112]. Randomized smoothing consists in using a *base classifier* to create a *smoothed classifier*, which is certifiably robust (with high probability) to adversarial perturbations under some given norm. The certified radius depends on the accuracy of the original classifier under some chosen noise. Let $\mathcal{Y}$ be a set of c classes, and let $f : \mathbb{R}^d \to \mathcal{Y}$ be any base classifier. For an input $\boldsymbol{x} \in \mathbb{R}^d$, the smoothed classifier's prediction $g(\boldsymbol{x})$ is defined to be the class which f is most likely to predict when $\boldsymbol{x}$ is perturbed by a given random noise:

$$g(\boldsymbol{x}) = \arg\max_{c \in \mathcal{Y}} \mathbb{P}[f(\boldsymbol{x} + \boldsymbol{\epsilon}) = c],$$

where $\boldsymbol{\epsilon}$ is a random variable following a chosen smoothing distribution. Using a generalization of the Neyman-Pearson lemma, Cohen *et al.* [113] provide a tight certified radius for randomized smoothing with Gaussian noise against L_2 adversary. If there is a class $c_A \in \mathcal{Y}$ and two probabilities $\underline{p_A}, \overline{p_B}$ such that:

$$\mathbb{P}[f(\boldsymbol{x} + \boldsymbol{\epsilon}) = c_A] \geq \underline{p_A} \geq \overline{p_B} \geq \max_{c \neq c_A} \mathbb{P}[f(\boldsymbol{x} + \boldsymbol{\epsilon}) = c], \tag{2.3}$$

then one has $g(\boldsymbol{x} + \boldsymbol{\eta}) = c_A$ for all $\|\boldsymbol{\eta}\|_2 \leq R$ where:

$$R = \frac{\sigma}{2}\left(\Phi^{-1}(\underline{p_A}) - \Phi^{-1}(\overline{p_B})\right), \tag{2.4}$$

with Φ^{-1} the quantile function of the standard normal distribution. This bound requires that $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2)$ where the hyperparameter $\sigma > 0$ controls the robustness-accuracy trade-off. Indeed, the radius is larger when $\underline{p_A}$ is close to 1 and $\overline{p_B}$ is close to 0. The higher is σ, the less likely this will be true, thus there is a trade-off. This trade-off depends on the ability of the model f to be robust to Gaussian noise. Hence, this result shows a direct link between robustness to Gaussian noise and verified robustness to adversarial attacks. Since it is not possible to exactly evaluate the prediction of g at $\boldsymbol{x}$, Monte Carlo algorithms are used.

Moreover, Cohen *et al.* [113] showed that this bound is tight, i.e., if $\underline{p_A} + \overline{p_B} \leq 1$, then for any perturbation $\boldsymbol{\delta}$ such that $\|\boldsymbol{\delta}\|_2 \geq R$, there exists a classifier f verifying equation 2.3 for which $g(\boldsymbol{x} + \boldsymbol{\delta}) \neq c_A$. Hence, the set of perturbations to which the smoothed classifier is provably robust is precisely a L_2 ball. Thus, smoothing with other distributions (other than Gaussian) lead to verified guarantees for other L_p norm, or even other type of attacks. For example, Hao *et al.* [114] have developed a randomized smoothing verification against semantic attacks.

Salman *et al.* [115] proved the same radius with arguments on the Lipschitz constant of the smoothed classifier. Teng *et al.* [116] derive a similar certified radius against L_1 perturbations using Laplace noise instead of Gaussian noise. This suggests that there is a link between the chosen L_p perturbation and the best smoothing distribution to obtain the largest radii.

The first attempt to develop a full theory of randomized smoothing is provided by Yang *et al.* [117]. Relying on measure theoretic considerations, they generalized the proof of Cohen *et al.* [113] and provided the certified radii for several pairs (smoothing distribution, L_p norm). Using the theory of Wulff crystals, they highlighted some properties that a smoothing distribution should exhibit to maximize the certified radius for L_1, L_2 and L_∞ adversaries. Finally, they show a no-go theorem concerning the robustness-accuracy trade-off: randomized smoothing can achieve nontrivial certified accuracy for any L_p adversary ($p \in [1, +\infty]$) only if the budget

$\eta \leq O\left(\min\left\{1, d^{1/p-1/2}\right\}\right)$, where d is the dimension of the input space. Recent works have shown that this result can be bypassed by retrieving more information about the base classifier [118, 119].

However, the results of Yang *et al.* apply only for isotropic adversaries (e.g., L_p adversaries). Moreover, the parameters of the smoothing distribution are fixed for the certification of the entire input space. Recent works suggest to implement *data-dependent* [120] and *anisotropic* [121, 122] randomized smoothing, one of them relying on information geometry [123]. Nevertheless, a full theory of randomized smoothing maximizing the certified radii for a general class of adversaries, and quantifying precisely the accuracy-robustness trade-off, is still missing.

In [124], the authors prove several theoretical results about input-dependent randomized smoothing (IDRS). IDRS consists in using a variance $\sigma(\boldsymbol{x})$ that depends on the input $\boldsymbol{x}$, instead of a constant variance σ. The authors do not investigate how to choose the function $\sigma(\boldsymbol{x})$: they assume that $\sigma(\boldsymbol{x})$ is given and investigate how to derive tight certified radii, both in theory and in practice.

Moreover, they show that IDRS suffers from curse of dimensionality in the sense that the certified radius for a design $\sigma(\boldsymbol{x})$ will shrink when the dimension increases, except if the function $\sigma(\boldsymbol{x})$ is r-semi-elastic with very small r, i.e., the function $\sigma(\boldsymbol{x})$ is almost constant and thus, any benefit of using an input-dependent scheme is lost.

In [120], the authors investigate IDRS from an empirical perspective. They use backpropagation and gradient descent to directly optimize $\sigma(\boldsymbol{x})$ to obtain large certified radius. In order to achieve verification, they use an heuristic method which is not proved to be tight. Nonetheless, the authors claim that their method achieves larger radii than Cohen *et al.*[113] with constant σ.

Randomized smoothing has several drawbacks. First, it is a probabilistic Monte-Carlo method. Thus, the certified radius is only guaranteed up to a small probability of error. Moreover, the MC sampling may suffer from the curse of dimensionality. Second, randomized smoothing is a post-hoc method. Thus, it does not provide any insight about how to design a robust base model that "understand the task". These drawbacks are partially addressed by Lipschitz neural networks.

2.2.2 Lipschitz Neural Networks

The Lipschitz constant of a function quantifies of much its output can change when the input is changed with a perturbation of norm 1. Since model stability can be seen as a high sensitivity of the output to a small input change, it is natural to try to evaluate or constraint the Lipschitz constant of a neural network to measure and improve the robustness against adversarial attacks. Constraining the Lipschitz constant can be done with regularization [54]. However, it is not possible to show that a regularized network has a small enough local Lipschitz constant at every point. Instead, this paragraph focuses on networks that are Lipschitz by-design and whose Lipschitz constant is controlled everywhere such that it is possible to verify the adversarial robustness of the network.

In [107], the authors propose to prove the robustness against L_2 perturbations in a classification setting by upper-bounding the global Lipschitz constant of the network. The upper-bound is simply the product of the Lipschitz constants of each layer. In order to obtain a provable robustness, they add a new class, denoted $\perp$, which corresponds to a non-robust prediction. Assume that, at a given point $\boldsymbol{x}$, the network outputs the largest logit $f_1(\boldsymbol{x})$ to class 1. Let K_1 be the global Lipschitz constant of f_1, and K_i be the global Lipschitz constants of the other

logits f_i. Then, the logit of the class $\perp$ is set to be:

$$f_\perp(\boldsymbol{x}) = \max_i f_i(\boldsymbol{x}) + \epsilon(K_1 + K_i),$$

where $\epsilon > 0$ is a chosen certified radius. With this setting, when the model predicts a class that is not $\perp$, then it is guaranteed that there is no L_2 perturbation less than ϵ that can change the prediction of the network.

In [125], the authors introduce a new method to build Lipschitz networks with global Lipschitz constant equal to 1 (referred to as 1-Lipschitz networks). Since 1-Lipschitz functions are closed under composition, to build a 1-Lipschitz network, it suffices to compose 1-Lipschitz affine transformations and activations. Since they are interested with L_2 perturbations, the matrix norm used in this paragraph is the spectral norm and the vector norm is the Euclidean norm[6]. Both norms are denoted $\|\|_2$. If a norm-constrained network f uses 1-Lipschitz *element-wise* monotonic activation functions, and if one wants $\|\nabla f(\boldsymbol{x})\|_2 = 1$ almost everywhere, then f *must be linear*. Thus, there is a tension between preserving gradient norm (i.e., decreasing the Lipschitz constant) on one hand, and nonlinear processing on the other hand.

If a norm-constrained network f verifies $\|\nabla f(\boldsymbol{x})\|_2 = 1$ almost everywhere, then each weight matrix $\boldsymbol{W}$ can be replaced with a matrix $\tilde{\boldsymbol{W}}$ whose singular values all equal 1 (without changing the computed function). Thus, the authors of [125] are looking for Lipschitz networks, with *orthonormal weight matrices*, and *activations that preserve the gradient norm during backpropagation* (and which are expressive enough). Such networks are called *Gradient Norm Preserving* (GNP) networks. Classical activations (such as ReLU, tanh, maxout) are all 1-Lipschitz. However, they are not gradient norm prerserving. This is why the authors introduce a new activation called *GroupSort*. GroupSort separates the pre-activations into groups, then sorts each group, and outputs the combined group sorted vector. GroupSort is 1-Lipschitz and gradient norm preserving (its Jacobian is a permutation matrix which preserves any L_p norm).

In order to enforce L_2 orthonormality (i.e., singular values equal to 1) for the affine layers, the authors use Bjorck algorithm. Given a matrix, this algorithm finds the closest orthonormal matrix:

$$\boldsymbol{A}_{k+1} = \frac{1}{2}\boldsymbol{A}_k\left(3\boldsymbol{I} - \boldsymbol{A}_k^\top \boldsymbol{A}_k)\right).$$

Since the algorithm is fully differentiable, it is possible to optimize the weights of the matrices directly on the space of orthonormal matrices (called the Stiefel manifold).

The provable robustness is obtained as follows. Let K be the Lipschitz constant of some network. Let $\boldsymbol{x}$ be an input with class t and let $f(\boldsymbol{x})$ be the logits associated to $\boldsymbol{x}$. The *margin* is:

$$\mathcal{M}(\boldsymbol{x}) = \max\left\{0, f_t(\boldsymbol{x}) - \max_{i \neq t} f_i(\boldsymbol{x})\right\}.$$

If $\mathcal{M}(\boldsymbol{x}) > K\epsilon/2$, then the network is robust at $\boldsymbol{x}$ to all perturbations $\boldsymbol{\eta}$ such that $\|\boldsymbol{\eta}\|_2 < \epsilon$. The networks are trained with L_2-norm constrained weights using *multi-class hinge loss*:

$$\mathcal{L}(f(\boldsymbol{x}), t) = \sum_{i \neq t} \max\left\{0, \kappa - (f_t(\boldsymbol{x}) - f_i(\boldsymbol{x}))\right\}.$$

The hyper-parameter κ controls the margin enforcement. It depends on the Lipschitz constant K and the desired perturbation tolerance ϵ as $\kappa > K\epsilon/2$.

The main result of [125] is that L_∞-norm constrained networks with GroupSort activations can approximate any 1-Lipschitz function (with range in $\mathbb{R}$) in L_p distance. However, there is no

[6] However, the method can be extended to L_∞ perturbations with small modifications.

such universal approximation result for L_2-norm constrained networks, even if it seems true empirically.

In [126], the authors provide an in-depth analysis of the properties of 1-Lipschitz networks with GroupSort activation, including expressiveness, accuracy-robustness trade-off, and generalization. Every neural network[7] g is K-Lipschitz, thus $f = (1/K)g$ is in 1-Lipschitz. The network f has the same accuracy and also the *same robustness to adversarial attacks* as g. However, computing K is NP-hard. 1-Lipschitz networks have limited expressiveness for regression tasks, however *this is not true for classification.* This is because Lipschitz constraint is not a constraint on the shape of the boundary (which can be arbitrarily complex) but on the slope of the landscape of f. The main thesis of [126] is that *1-Lipschitz networks are theoretically better grounded than unrestricted neural networks for classification.*

Providing certified robustness for 1-Lipschitz networks does not increase runtime contrary to methods based on bounding boxes or abstract interpretation (formal methods), as well as randomized smoothing (whose Monte Carlo sampling can be costly). Moreover, 1-Lipschitz networks do not require adversarial training to achieve robustness. For any f, the robustness radius ϵ of binary classifier $\text{sign} \circ f$ at example $\boldsymbol{x}$ verifies $\epsilon \geq |f(\boldsymbol{x})|$. However, enforcing the orthonormal constraint during training means that the training time is larger.

If during training of an unrestricted neural network, the empirical loss over the training set converges to 0, then *the Lipschitz constant of the network diverges to* ∞. There is at least one weight matrix whose operator norm converges to ∞. Moreover, the predicted probabilities (after logistic activation) converge to either 0 or 1 (saturated probabilities). The saturation of the predicted probabilities means that they do not contain any useful information on the true confidence of the classifier. These issues are avoided when using 1-Lipschitz networks. One drawback of 1-Lipschitz networks is that they are only robust against L_p perturbations. The robustness of 1-Lipschitz networks against semantic attacks has not been studied yet.

2.3 Uncertainty Quantification

In order to deliver a trustworthy decision making tool, a 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 the end users, such as air traffic controllers. This section presents methods to quantify the uncertainty of machine learning models relying on approximation of Bayesian inference. First, the terminology of Bayesian inference is recalled and a method for approximating probability distributions called *variational inference* is presented. Then, the uncertainty quantification methods themselves are reviewed. Finally, another uncertainty quantification method based on stochastic differential equations is presented.

2.3.1 Bayesian Inference

Let $\boldsymbol{D} = (\boldsymbol{x}_1, \boldsymbol{y}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{y}_n)$ be a dataset. It is seen as a random vector whose subvectors $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ are *independent and identically distributed* according to an unknown joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$. One can write $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})$.
Given a random variable $\boldsymbol{x}^*$ sampled from $p(\boldsymbol{x})$ and independent from all $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, the goal is to compute the *conditional distribution* $p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{D})$.
Consider a *family of models* $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$. The parameter $\boldsymbol{\theta}$ is seen as a random

[7] Using affine layers and Lipschitz acitvations but no recurrent mechanisms.

variable. The conditional distribution is obtained by marginalizing the parameter $\boldsymbol{\theta}$:

$$p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{D}) = \int p(\boldsymbol{y}, \boldsymbol{\theta}|\boldsymbol{x}^*, \boldsymbol{D}) d\boldsymbol{\theta}.$$

The integrand can be expanded:

$$p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{D}) = \int p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{\theta}, \boldsymbol{D}) p(\boldsymbol{\theta}|\boldsymbol{x}^*, \boldsymbol{D}) d\boldsymbol{\theta}.$$

Now, two assumptions have to be made:

- The parameter $\boldsymbol{\theta}$ is independent with $\boldsymbol{x}^*$. In fact, $\boldsymbol{\theta}$ and $\boldsymbol{x}_i$ are assumed to be independent for all i. In other words, before training, the parameter $\boldsymbol{\theta}$ is independent from the data collection process.
- The chain $\boldsymbol{D} \rightarrow \boldsymbol{\theta} \rightarrow \boldsymbol{y}|\boldsymbol{x}^*$ is Markov. After training, the parameter $\boldsymbol{\theta}$ contains all the information about the dataset $\boldsymbol{D}$ that are necessary to infer $p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{D})$.

Thus, the *Bayesian Model Averaging* (BMA) is derived:

$$p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{D}) = \int p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{D}) d\boldsymbol{\theta}.$$

The term $p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{\theta})$ corresponds to a model from the family of model. The prediction of this model is weighted by $p(\boldsymbol{\theta}|\boldsymbol{D})$. *Bayesian inference* is used to compute $p(\boldsymbol{\theta}|\boldsymbol{D})$ as :

$$p(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{p(\boldsymbol{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta})}{\int p(\boldsymbol{D}|\boldsymbol{\theta}') p(\boldsymbol{\theta}') d\boldsymbol{\theta}'}.$$

The various terms are:

- The *prior* $p(\boldsymbol{\theta})$. It is a choice made by the modeler. The rationales behind this choice will be discussed later.
- The *likelihood*:
$$p(\boldsymbol{D}|\boldsymbol{\theta}) = \prod_{i=1}^{n} p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta}) p(\boldsymbol{x}_i|\boldsymbol{\theta}) = \prod_{i=1}^{n} p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta}) p(\boldsymbol{x}_i).$$
Here, the assumption that $\boldsymbol{x}_i$ and $\boldsymbol{\theta}$ are independent has been used. Thus, $p(\boldsymbol{x}_i|\boldsymbol{\theta}) = p(\boldsymbol{x}_i)$ does not depend on $\boldsymbol{\theta}$. Maximizing $\prod_{i=1}^{n} p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ is equivalent to maximizing $p(\boldsymbol{D}|\boldsymbol{\theta})$.
The negative log-likelihood $-\sum_{i=1}^{n} \log p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta})$ is often used as the loss function in non-Bayesian machine learning (along with regularization terms).
- The *evidence* (or *model evidence*, or *marginal likelihood*) $p(\boldsymbol{D}) = \int p(\boldsymbol{D}|\boldsymbol{\theta}') p(\boldsymbol{\theta}') d\boldsymbol{\theta}'$. It is considered to be intractable except for very simple models. Its value is used for *model selection*: better models should yield higher evidence.

2.3.2 Variational Inference

There are two methods to approximate $p(\boldsymbol{\theta}|\boldsymbol{D})$ when the evidence $p(\boldsymbol{D})$ is intractable:

- Markov Chain Monte Carlo (MCMC).

- Variational Inference (VI).

MCMC is more precise (and has asymptotic guarantees) but is more computationally heavy. In the remainder of this document, only Variational Inference will be considered.

Let $q(\boldsymbol{\theta}|\boldsymbol{\xi})$ be a *variational family* parameterized by $\boldsymbol{\xi}$. Variational Inference consists in using optimization to find a parameter $\boldsymbol{\xi}^*$ such that $q(\boldsymbol{\theta}|\boldsymbol{\xi}^*)$ is "close" to $p(\boldsymbol{\theta}|\boldsymbol{D})$. Then, the variational family $q(\boldsymbol{\theta}|\boldsymbol{\xi}^*)$ will be used instead of $p(\boldsymbol{\theta}|\boldsymbol{D})$ to compute $p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{D})$. The "closeness" is measured with the KL divergence:

$$\begin{aligned}\mathrm{KL}\left[q(\boldsymbol{\theta}|\boldsymbol{\xi})||p(\boldsymbol{\theta}|\boldsymbol{D})\right] &= \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log\frac{q(\boldsymbol{\theta}|\boldsymbol{\xi})}{p(\boldsymbol{\theta}|\boldsymbol{D})}\right], \\ &= \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log q(\boldsymbol{\theta}|\boldsymbol{\xi})\right] - \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log p(\boldsymbol{\theta}|\boldsymbol{D})\right], \\ &= H\left(q(\boldsymbol{\theta}|\boldsymbol{\xi}), p(\boldsymbol{\theta}|\boldsymbol{D})\right) - H\left(q(\boldsymbol{\theta}|\boldsymbol{\xi})\right),\end{aligned}$$

where $H(q)$ is the entropy, and $H(q,p)$ is the cross-entropy.

Since $p(\boldsymbol{\theta}|\boldsymbol{D})$ is unknown, this KL divergence cannot be computed. Thus, another quantity is computed such that its minimum is achieved at the same $\boldsymbol{\xi}^*$ than the minimum of the KL divergence. Using the law of contional probability:

$$\begin{aligned}\mathrm{KL}\left[q(\boldsymbol{\theta}|\boldsymbol{\xi})||p(\boldsymbol{\theta}|\boldsymbol{D})\right] &= \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log q(\boldsymbol{\theta}|\boldsymbol{\xi})\right] - \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log p(\boldsymbol{D},\boldsymbol{\theta})\right] + \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log p(\boldsymbol{D})\right], \\ &= \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log q(\boldsymbol{\theta}|\boldsymbol{\xi})\right] - \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log p(\boldsymbol{D},\boldsymbol{\theta})\right] + \log p(\boldsymbol{D}).\end{aligned}$$

Here, the fact that $\log p(\boldsymbol{D})$ is a constant with respect to $\boldsymbol{\theta}|\boldsymbol{\xi}$ has been used.
Define the *Evidence Lower Bound* or *negative variational free energy*:

$$\begin{aligned}\mathrm{ELBO}(\boldsymbol{\xi}) &= \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log p(\boldsymbol{D},\boldsymbol{\theta})\right] - \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log q(\boldsymbol{\theta}|\boldsymbol{\xi})\right], \\ &= -U\left(q(\boldsymbol{\theta}|\boldsymbol{\xi}), p(\boldsymbol{D},\boldsymbol{\theta})\right) + H\left(q(\boldsymbol{\theta}|\boldsymbol{\xi})\right),\end{aligned}$$

where $U(q(\boldsymbol{\theta}|\boldsymbol{\xi}), p(\boldsymbol{D},\boldsymbol{\theta})) = -\mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log p(\boldsymbol{D},\boldsymbol{\theta})\right]$ is the energy. This relationship is similar to the free energy A in thermodynamics: $A = U - TS$, where U is the energy, S the entropy, and T the temperature (a constant).
The ELBO can also be expressed as:

$$\begin{aligned}\mathrm{ELBO}(\boldsymbol{\xi}) &= \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log p(\boldsymbol{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\xi})\right], \\ &= \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[\log p(\boldsymbol{D}|\boldsymbol{\theta})\right] - \mathrm{KL}\left[q(\boldsymbol{\theta}|\boldsymbol{\xi})||p(\boldsymbol{\theta})\right],\end{aligned}$$

which corresponds to the expected log-likelihood plus a regularization term.
The following holds:

$$\mathrm{KL}\left[q(\boldsymbol{\theta}|\boldsymbol{\xi})||p(\boldsymbol{\theta}|\boldsymbol{D})\right] = \log p(\boldsymbol{D}) - \mathrm{ELBO}(\boldsymbol{\xi})$$

Minimizing the KL divergence wrt $\boldsymbol{\xi}$ is equivalent to maximizing the ELBO. The positivity of the KL divergence (i.e., Gibbs' inequality) implies that $\log p(\boldsymbol{D}) \geq \mathrm{ELBO}(\boldsymbol{\xi})$, hence the name "evidence lower bound". Equality is achieved if and only if $q(\boldsymbol{\theta}|\boldsymbol{\xi}) = p(\boldsymbol{\theta}|\boldsymbol{D})$.

2.3.3 Variational Inference for Uncertainty Quantification

The variational inference framework has been adapted to quantify the uncertainty of deep learning models. In [127], Blundell *et al.* introduce the *Bayes-by-Backprop* framework, which uses variational inference to train a fully-connected network where the parameters are modeled as random variables with Gaussian distributions. In [128], Gal and Ghahramani introduce Dropout

CNN. They use variational inference to train a Convolutional Neural Networks (CNN) whose parameters are random variables sampled from a Bernoulli distribution. Both [127] and [128] rely on Monte Carlo techniques to sample from the posterior distribution. The uncertainty is estimated by computing the variance of the samples drawn from the posterior.

In [129], Dera *et al.* introduce the moment propagation framework for uncertainty quantification in CNN. In section 5.5.2, the moment propagation framework is fully derived for Transformers. Thus, the method is only briefly described here. In the moment propagation framework, the parameters of the CNN are modeled as Gaussian distributions. As part of the variational inference framework, the ELBO is used as the objective function of the network. To estimate the expected negative log-likelihood (first term of the ELBO), the first and second moments of the parameters distributions are propagated through the various layers (convolutional, activation, pooling, fully-connected). The model is compared to other Bayesian networks as well as to classical CNN, on MNIST and CIFAR-10 datasets. The Bayesian 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. Contrary to previous methods, the moment propagation framework do not require to perform Monte Carlo sampling to evaluate the variational posterior. Indeed, the variational posterior is approximated by a Gaussian distribution whose mean and variance are obtained by propagation through the model.

In [130], Dera *et al.* apply the moment propagation 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 moment propagation 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 [131], Xue *et al.* propose a Bayesian Transformer[8] Language Model (LM) applied to speech recognition. The model consists of several Transformer decoders where only the first one is optimized with the variational inference 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.

2.3.4 Stochastic Differential Equation Network (SDE-Net)

Uncertainty is sometimes split into two components: aleatoric uncertainty and epistemic uncertainty. Aleatoric uncertainty corresponds to the natural randomness of the task (due to class overlap, data noise etc.). It cannot be reduced with more data. Epistemic uncertainty, on the other hand, is caused by the ignorance of the model due to lack of data. It is high in regions where the data is sparse.

SDE-Net was introduced by Kong *et al.* [132]. It brings the following benefits:

- Separate aleatoric and epistemic uncertainty;
- No need to specify model prior distributions and infer posterior distributions (as with Bayesian methods);
- Applicable to classification and regression tasks.

[8] The Transformer model is described in detail in section 5.5.1.

Neural networks can be viewed as state transformations of a dynamic system. The idea is to add a Brownian motion term to quantify epistemic uncertainty. SDE-Net consists of:

- A *drift net* that parameterizes an ODE to fit the predictive function;
- A *diffusion net* that encourages high diffusion for data outside the training distribution.

SDE-Net is the discretization of the following stochastic differential equation:

$$d\boldsymbol{x}_t = f(\boldsymbol{x}_t, t)dt + g(\boldsymbol{x}_t, t)d\boldsymbol{w}_t,$$

where $g(\boldsymbol{x}_t, t)$ is the variance of the Brownian motion and represents the epistemic uncertainty. The aleatoric uncertainty is captured by the drift net by representing the model output as a probability distribution (categorical for classification, Gaussian for regression).

The objective function is the following:

$$\min_{\boldsymbol{\theta}_f} \mathbb{E}_{\boldsymbol{x}_0 \sim p_{train}}[\mathcal{L}(\boldsymbol{x}_T)] + \min_{\boldsymbol{\theta}_g} \mathbb{E}_{\boldsymbol{x}_0 \sim p_{train}}[g(\boldsymbol{x}_0; \boldsymbol{\theta}_g)] + \max_{\boldsymbol{\theta}_g} \mathbb{E}_{\tilde{\boldsymbol{x}}_0 \sim p_{OOD}}[g(\tilde{\boldsymbol{x}}_0; \boldsymbol{\theta}_g)],$$
$$s.t.\ d\boldsymbol{x}_t = f(\boldsymbol{x}_t, t; \boldsymbol{\theta}_f)dt + g(\boldsymbol{x}_0; \boldsymbol{\theta}_g)d\boldsymbol{w}_t,$$

where $\mathcal{L}(.)$ is the loss function dependent on the task, T is the terminal time of the stochastic process (the last "hidden layer"), p_{train} is the distribution for training data, p_{OOD} is the distribution for out-of-distribution data. In order to reduce the complexity: the parameters are shared by each layer; the variance g only depends on $\boldsymbol{x}_0$ instead of $\boldsymbol{x}_t$; and to avoid the explosion of solution, the variance g should be bounded by a hyper-parameter σ_{max}.

To quantify the uncertainty, multiple random realizations of $\boldsymbol{x}_T$ are obtained. Then, the aleatoric uncertainty is given by the expected predictive entropy $\mathbb{E}_{\boldsymbol{x}_T|x_0,\boldsymbol{\theta}_{f,g}}[\mathcal{H}[p(\boldsymbol{y}|\boldsymbol{x}_T)]]$ for classification, and the expected predictive variance $\mathbb{E}_{\boldsymbol{x}_T|\boldsymbol{x}_0,\boldsymbol{\theta}_{f,g}}[\sigma(\boldsymbol{x}_T)]$ for regression. The epistemic uncertainty is given by the variance of the final solution $\text{var}[\boldsymbol{x}_T]$. This is similar to ensembling methods but it requires the training of only one model.

For training, there is no closed form solution for $\boldsymbol{x}_T$. Instead, the authors use the Euler-Maruyama scheme with fixed step size:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + f(\boldsymbol{x}_k, t; \boldsymbol{\theta}_f)\Delta_t + g(\boldsymbol{x}_0; \boldsymbol{\theta}_g)\sqrt{\Delta t}Z_k, \tag{2.5}$$

where $Z_k \sim \mathcal{N}(0, 1)$ and $\Delta t = T/N$ with N the number of iterations.

SDE-Net is tested on the following tasks: out-of-distribution detection, misclassification detection, adversarial sample detection, and active learning. It has been shown to have strong performance compared to other uncertainty quantification techniques.

2.4 Conclusion

In this chapter, several aspects and challenges surrounding the concept of robust machine learning have been reviewed. The most well-known evasion attacks were presented, as well as and some defenses against them, including less known approaches relying on information geometry. Then, taking a step back, various interpretations of adversarial vulnerability were explored as well as some of their limitations. Verification methods can partially address some of these limitations, and some of these methods developed recently for machine learning models have been reviewed: randomized smoothing and Lipschitz neural networks. Finally, some methods for uncertainty quantification were presented, which is another important aspect of machine learning

robustness. In particular, methods based on variational inference have been described and will be applied in chapter 5.

The next chapter builds on ideas introduced in this chapter to develop a robustification method against L_2 attacks. More particularly, the next chapter relies on ideas coming from information geometry as discussed in section 2.1.3.

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

ENAC

Chapter 3

Partial Isometry Regularization

This chapter offers an information geometric perspective on adversarial robustness in machine learning models. It is shown that robustness can be achieved by encouraging the model to be isometric in the orthogonal space of the kernel of the pullback Fisher information metric (FIM). A regularization defense method for adversarial robustness is subsequently formulated. While the focus of this chapter is on L_2 white-box attacks within multi-class classification tasks, the method's applicability extends to more general settings, including unrestricted attacks and black-box attacks across various supervised learning tasks. The regularized model is evaluated on MNIST and CIFAR-10 datasets against projected gradient descent (PGD) L_∞ attacks and AutoAttack [102] with L_∞ and L_2 norms. Comparisons with the unregularized model, defensive distillation [49], Jacobian regularization [133], and Fisher information regularization [57] show significant improvement in robustness. Moreover, the regularized model is able to ensure robustness against larger perturbations compared to adversarial training.

Section 3.1 introduces definitions related to geometry. Then, a sufficient condition is derived for adversarial robustness at a given sample point. Section 3.2 presents a method for approximating the robustness condition, which involves promoting model isometry in the orthogonal complement of the kernel of the pullback of the FIM. In section 3.3, several experiments are presented to evaluate the proposed method. Section 3.4 discusses the results in the context of related work on adversarial defense. Finally, section 3.5 concludes the chapter and outlines potential extensions of this research. Section 3.6 provides the proof of the results stated in the previous sections.

This work has been published in the Entropy Special Issue "Information Geometry for Data-Analysis" [23].

3.1 Definitions and Robustness Condition

Let $d, c \in \mathbb{N}^*$ such that $d \geq c > 1$. In the learning framework, d will be the dimension of the input space, while c will be the number of classes. In this chapter, the components of a vector $\boldsymbol{v}$ is denoted by $\boldsymbol{v}^i \in \mathbb{R}$ with a superscript.

3.1.1 Geometrical Definitions

Consider a multi-class classification task. Let $\mathcal{X} \subseteq \mathbb{R}^d$ be the input domain, and let $\mathcal{Y} = \{1, \ldots, c\} \subset \mathbb{N}$ be the set of labels for the classification task. For example, the input domain

of MNIST is $\mathcal{X} = [0,1]^d$ (with $d = 784$), and $c = 10$. The input domain $\mathcal{X}$ is assumed to be a d-dimensional embedded smooth connected submanifold of $\mathbb{R}^d$.

Definition 3.1.1 (Probability simplex)**.** Define the probability simplex of dimension $c-1$ by

$$\Delta^{c-1} = \left\{ \boldsymbol{\theta} \in \mathbb{R}^{c-1} : \forall k \in \{1, \dots, c-1\}, \boldsymbol{\theta}^k > 0 \text{ and } \sum_{i=1}^{c-1} \boldsymbol{\theta}^i < 1 \right\}. \tag{3.1}$$

Δ^{c-1} is a smooth submanifold of $\mathbb{R}^c$ of dimension $c-1$. The vector $\boldsymbol{\theta} = (\boldsymbol{\theta}^1, \dots, \boldsymbol{\theta}^{c-1})$ can be seen as a coordinate system from Δ^{c-1} to $\mathbb{R}^{c-1}$. Then, one can define $\boldsymbol{\theta}^c = 1 - \sum_{i=1}^{c-1} \boldsymbol{\theta}^i$.

A machine learning model (e.g., a neural network) is often seen as assigning a label $\hat{\boldsymbol{y}} \in \mathcal{Y}$ to a given input $\boldsymbol{x} \in \mathcal{X}$. Instead, in this chapter, a model is seen as assigning the parameters of a random variable $\boldsymbol{y}$ to a given input $\boldsymbol{x} \in \mathcal{X}$. The random variable $\boldsymbol{y}$ has a probability density function $p(\boldsymbol{y}|\boldsymbol{\theta})$ belonging to the family of c-dimensional categorical distributions $\mathcal{S} = \{p(\boldsymbol{y}|\boldsymbol{\theta}) : \boldsymbol{\theta} \in \Delta^{c-1}\}$.

$\mathcal{S}$ can be endowed with a differentiable structure by using $p(\boldsymbol{y}|\boldsymbol{\theta}) \in \mathcal{S} \mapsto (\boldsymbol{\theta}^1, \dots, \boldsymbol{\theta}^{c-1}) \in \mathbb{R}^{c-1}$ as a global coordinate system. Hence, $\mathcal{S}$ becomes a smooth manifold of dimension $c-1$ (more details on this construction can be found in [134], Chapter 2). The density $p(\boldsymbol{y}|\boldsymbol{\theta})$ can be identified with $(\boldsymbol{\theta}^1, \dots, \boldsymbol{\theta}^{c-1})$.

A machine learning model is seen as a smooth map $f : \mathcal{X} \to \Delta^{c-1}$ that assigns to an input $\boldsymbol{x} \in \mathcal{X}$, the parameters $\boldsymbol{\theta} = f(\boldsymbol{x}) \in \Delta^{c-1}$ of a c-dimensional categorical distribution $p(\boldsymbol{y}|\boldsymbol{\theta}) \in \mathcal{S}$. In practice, a neural network produces a vector of logits $l(\boldsymbol{x})$. Then, these logits are transformed into the parameters $\boldsymbol{\theta}$ with the softmax function: $\boldsymbol{\theta} = s(l(\boldsymbol{x}))$.

In order to study the sensitivity of the predicted $f(\boldsymbol{x}) \in \Delta^{c-1}$ with respect to the input $\boldsymbol{x} \in \mathcal{X}$, a distance has to be defined both in $\mathcal{X}$ and in Δ^{c-1}. In order to measure distances on smooth manifolds, each manifold has to be equipped with a Riemannian metric.

First, consider Δ^{c-1}. As described above, the set Δ^{c-1} is seen as the family of categorical distributions. A natural Riemannian metric for Δ^{c-1} is the Fisher information metric (FIM).

Definition 3.1.2 (Fisher information metric)**.** For each $\boldsymbol{\theta} \in \Delta^{c-1}$, the Fisher information metric (FIM) $\boldsymbol{g}$ defines a symmetric positive-definite bilinear form $\boldsymbol{g}_{\boldsymbol{\theta}}$ over the tangent space $T_{\boldsymbol{\theta}}\Delta^{c-1}$. In the standard coordinates of $\mathbb{R}^c$, for all $\boldsymbol{\theta} \in \Delta^{c-1}$ and all tangent vectors $\boldsymbol{v}, \boldsymbol{w} \in T_{\boldsymbol{\theta}}\Delta^{c-1}$, the following holds:

$$\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{v}, \boldsymbol{w}) = \boldsymbol{v}^\top \boldsymbol{G}_{\boldsymbol{\theta}} \boldsymbol{w}, \tag{3.2}$$

where $\boldsymbol{G}_{\boldsymbol{\theta}}$ is the Fisher information matrix for parameter $\boldsymbol{\theta} \in \Delta^{c-1}$, defined by

$$\boldsymbol{G}_{\boldsymbol{\theta},ij} = \frac{\delta_{ij}}{\boldsymbol{\theta}^i} + \frac{1}{\boldsymbol{\theta}^c}. \tag{3.3}$$

For any $\boldsymbol{\theta} \in \Delta^{c-1}$, the matrix $\boldsymbol{G}_{\boldsymbol{\theta}}$ is symmetric positive-definite and non-singular (see Proposition 1.6.2 in [68]). The FIM induces a distance on Δ^m, called the Fisher–Rao distance, denoted by $\boldsymbol{d}_{FR}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ for any $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \Delta^{c-1}$.

As already discussed section 2.1.3.1, the choice of the FIM as a metric for Δ^{c-1} is motivated by two properties:

- The FIM is the "infinitesimal distance" of the relative entropy, which is the loss function used to train a multi-class classification model.
- Chentsov's theorem [69] which claims that the FIM is the unique Riemannian metric on Δ^{c-1} that is invariant under sufficient statistics (up to a multiplicative constant).

Now, consider $\mathcal{X}$. Since the topic of interest is adversarial robustness, the desired metric should formalize the idea that two close data points are "indistinguishable" from a human perspective (or any other relevant perspective). A natural choice is the Euclidean metric induced from $\mathbb{R}^d$ on $\mathcal{X}$.

Definition 3.1.3 (Euclidean metric)**.** Consider the Euclidean space $\mathbb{R}^d$ endowed with the Euclidean metric $\overline{\boldsymbol{g}}$. It is defined in the standard coordinates of $\mathbb{R}^d$ for all $\boldsymbol{x} \in \mathbb{R}^d$ and for all tangent vectors $\boldsymbol{v}, \boldsymbol{w} \in T_{\boldsymbol{x}}\mathbb{R}^d$ by

$$\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v}, \boldsymbol{w}) = \boldsymbol{v}^\top \boldsymbol{w}, \tag{3.4}$$

thus, its matrix is the identity matrix of dimension d, denoted by $\boldsymbol{I}_d$. The Euclidean metric induces a distance on $\mathbb{R}^d$ that is denoted with the L_2-norm: $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ for any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$.

From now on, the following are fixed:

- A smooth map $f : (\mathcal{X}, \overline{\boldsymbol{g}}) \to (\Delta^{c-1}, \boldsymbol{g})$. The i-th component of f in the standard coordinates of $\mathbb{R}^c$ is denoted by f^i;
- A point $\boldsymbol{x} \in \mathcal{X}$;
- A positive real number $\epsilon > 0$.

Define the Euclidean open ball centered at $\boldsymbol{x}$ with radius ϵ by

$$\overline{b}(\boldsymbol{x}, \epsilon) = \left\{ \boldsymbol{z} \in \mathbb{R}^d : \|\boldsymbol{z} - \boldsymbol{x}\| < \epsilon \right\}. \tag{3.5}$$

Definition 3.1.4. Define the set (Figure 3.1):

$$\mathcal{A}_{\boldsymbol{x}} = \left\{ \boldsymbol{\theta} \in \Delta^{c-1} : \arg\max_i \boldsymbol{\theta}^i = \arg\max_i f^i(\boldsymbol{x}) \right\}. \tag{3.6}$$

For simplicity, assume that $f(\boldsymbol{x})$ is not on the "boundary" of $\mathcal{A}_{\boldsymbol{x}}$, such that $\arg\max_i f^i(\boldsymbol{x})$ is well-defined.

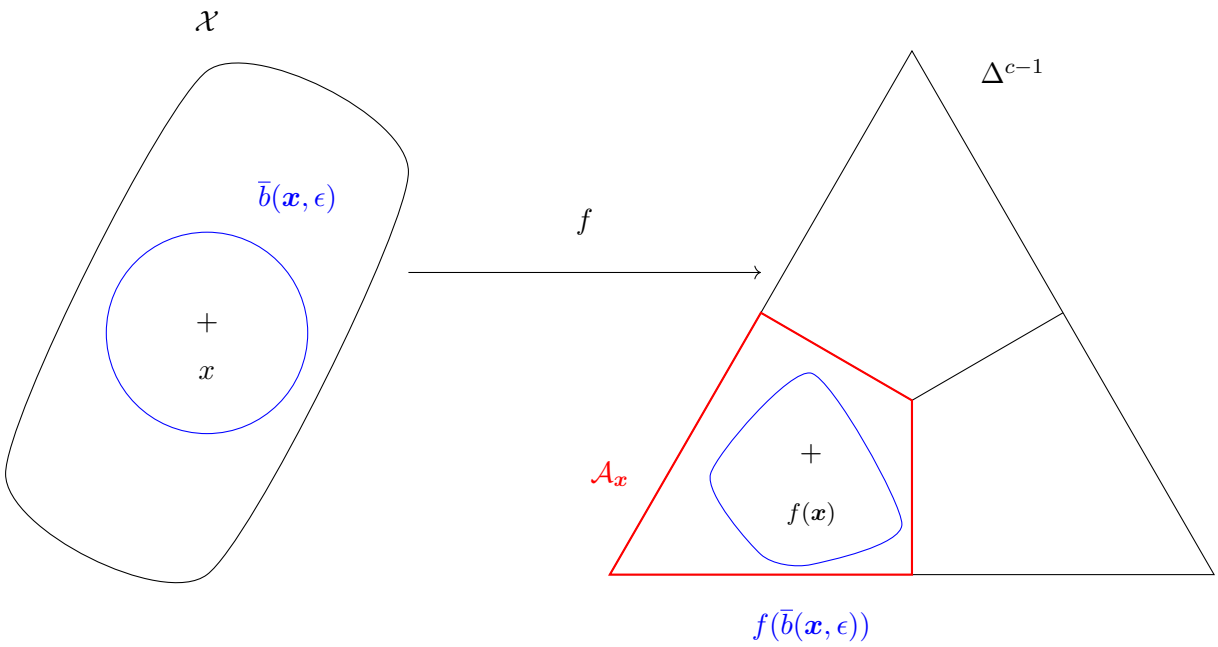


Figure 3.1: ϵ-robustness at $\boldsymbol{x}$ is enforced if and only if $f(\overline{b}(\boldsymbol{x}, \epsilon)) \subseteq \mathcal{A}_{\boldsymbol{x}}$.

The set $\mathcal{A}_{\boldsymbol{x}}$ is the subset of distributions of Δ^{c-1} that have the same class as $f(\boldsymbol{x})$.

Definition 3.1.5 (Geodesic ball of the FIM)**.** Let $\delta > 0$ be the Fisher–Rao distance between $f(\boldsymbol{x})$ and $\Delta^{c-1} \setminus \mathcal{A}_{\boldsymbol{x}}$ (Figure 3.2), i.e., the Fisher–Rao distance between $f(\boldsymbol{x})$ and the closest distribution of Δ^{c-1} with a different class.

Define the geodesic ball centered at $f(\boldsymbol{x}) \in \Delta^{c-1}$ with radius δ by

$$b(f(\boldsymbol{x}), \delta) = \left\{ \boldsymbol{\theta} \in \Delta^{c-1} : \boldsymbol{d}_{FR}(f(\boldsymbol{x}), \boldsymbol{\theta}) \leq \delta \right\}. \tag{3.7}$$

Section 3.2.3 proposes an efficient approximation of δ.

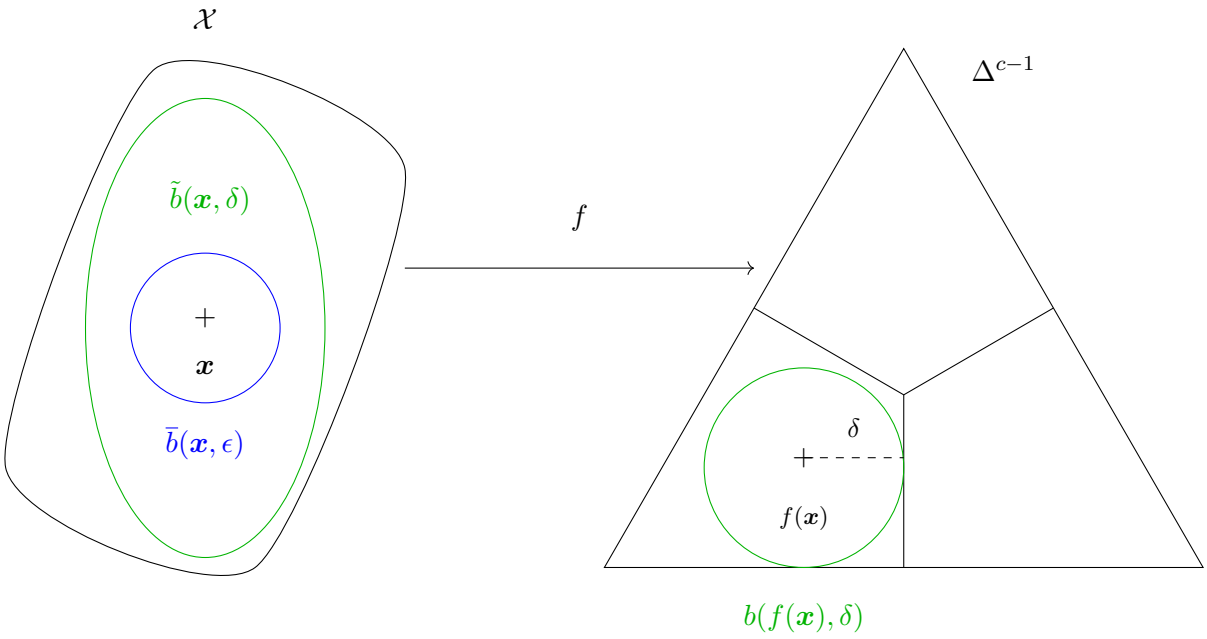


Figure 3.2: ϵ-robustness at $\boldsymbol{x}$ is enforced if $\bar{b}(\boldsymbol{x}, \epsilon) \subseteq \tilde{b}(\boldsymbol{x}, \delta)$.

Definition 3.1.6 (Pullback metric)**.** On $\mathcal{X}$, define the pullback metric $\tilde{\boldsymbol{g}}$ of $\boldsymbol{g}$ by f. In the standard coordinates of $\mathbb{R}^d$, $\tilde{\boldsymbol{g}}$ is defined for all tangent vectors $\boldsymbol{v}, \boldsymbol{w} \in T_{\boldsymbol{x}}\mathcal{X}$ by

$$\tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v}, \boldsymbol{w}) = \boldsymbol{v}^\top \boldsymbol{J}_{\boldsymbol{x}}^\top \boldsymbol{G}_{f(\boldsymbol{x})} \boldsymbol{J}_{\boldsymbol{x}} \boldsymbol{w}, \tag{3.8}$$

where $\boldsymbol{J}_{\boldsymbol{x}}$ is the Jacobian matrix of f at $\boldsymbol{x}$ (in the standard coordinates of $\mathbb{R}^d$ and $\mathbb{R}^c$). Define the matrix of $\tilde{\boldsymbol{g}}_{\boldsymbol{x}}$ in the standard coordinates of $\mathbb{R}^d$ by

$$\widetilde{\boldsymbol{G}}_{\boldsymbol{x}} = \boldsymbol{J}_{\boldsymbol{x}}^\top \boldsymbol{G}_{f(\boldsymbol{x})} \boldsymbol{J}_{\boldsymbol{x}}. \tag{3.9}$$

Definition 3.1.7 (Geodesic ball of the pullback metric)**.** Let $\tilde{\boldsymbol{d}}_{FR}$ be the distance induced by the pullback metric $\tilde{\boldsymbol{g}}$ on $\mathbb{R}^d$. The geodesic ball centered at $\boldsymbol{x}$ with radius δ is defined by

$$\tilde{b}(\boldsymbol{x}, \delta) = \left\{ \boldsymbol{z} \in \mathbb{R}^d : \tilde{\boldsymbol{d}}_{FR}(\boldsymbol{x}, \boldsymbol{z}) \leq \delta \right\}. \tag{3.10}$$

Note that the radius δ is the Fisher–Rao distance between $f(\boldsymbol{x})$ and $\Delta^{c-1} \setminus \mathcal{A}_{\boldsymbol{x}}$ as defined in definition 3.1.5.

3.1.2 Robustness Condition

Definition 3.1.8 (Robustness)**.** The model f is said to be ϵ-robust at $\boldsymbol{x}$ if

$$\forall \boldsymbol{z} \in \mathbb{R}^d, \|\boldsymbol{z} - \boldsymbol{x}\| < \epsilon \Rightarrow f(\boldsymbol{z}) \in \mathcal{A}_{\boldsymbol{x}}. \tag{3.11}$$

Equivalently, one can write (Figure 3.1):

$$f(\overline{b}(\boldsymbol{x}, \epsilon)) \subseteq \mathcal{A}_{\boldsymbol{x}}. \tag{3.12}$$

Proposition 3.1.9 (Sufficient condition for robustness)**.** *If $\overline{b}(\boldsymbol{x}, \epsilon) \subseteq \tilde{b}(\boldsymbol{x}, \delta)$, then f is ϵ-robust at $\boldsymbol{x}$ (Figure 3.2).*

The goal is to start from Proposition 3.1.9 and make several assumptions in order to derive a condition that can be efficiently implemented.

Working with geodesic balls $\overline{b}(\boldsymbol{x}, \epsilon)$ and $\tilde{b}(\boldsymbol{x}, \delta)$ is intractable, so a first assumption consists in using an "infinitesimal" condition by restating Proposition 3.1.9 in the tangent space $T_{\boldsymbol{x}}\mathcal{X}$ instead of working directly on $\mathcal{X}$. In $T_{\boldsymbol{x}}\mathcal{X}$, define the Euclidean ball of radius ϵ by

$$\overline{\mathcal{B}}_{\boldsymbol{x}}(0, \epsilon) = \left\{ \boldsymbol{v} \in T_{\boldsymbol{x}}\mathcal{X} : \overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v}, \boldsymbol{v}) = \boldsymbol{v}^\top \boldsymbol{v} \leq \epsilon^2 \right\}. \tag{3.13}$$

Similarly, in $T_{\boldsymbol{x}}\mathcal{X}$, define the $\tilde{\boldsymbol{g}}_{\boldsymbol{x}}$-ball of radius δ by

$$\widetilde{\mathcal{B}}_{\boldsymbol{x}}(0, \delta) = \left\{ \boldsymbol{v} \in T_{\boldsymbol{x}}\mathcal{X} : \tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v}, \boldsymbol{v}) = \boldsymbol{v}^\top \widetilde{\boldsymbol{G}}_{\boldsymbol{x}} \boldsymbol{v} \leq \delta^2 \right\}. \tag{3.14}$$

Assumption 3.1.10. Proposition 3.1.9 is replaced by

$$\overline{\mathcal{B}}_{\boldsymbol{x}}(0, \epsilon) \subseteq \widetilde{\mathcal{B}}_{\boldsymbol{x}}(0, \delta). \tag{3.15}$$

Proposition 3.1.11. *Equation 3.15 is equivalent to*

$$\forall \boldsymbol{v} \in T_{\boldsymbol{x}}\mathcal{X}, \quad \tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v}, \boldsymbol{v}) \leq \frac{\delta^2}{\epsilon^2} \overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v}, \boldsymbol{v}). \tag{3.16}$$

Since $c - 1 < d$, the Jacobian matrix $\boldsymbol{J}_{\boldsymbol{x}}$ has a rank smaller or equal to $c - 1$. Thus, since $\boldsymbol{G}_{f(\boldsymbol{x})}$ has full rank, $\widetilde{\boldsymbol{G}}_{\boldsymbol{x}} = \boldsymbol{J}_{\boldsymbol{x}}^\top \boldsymbol{G}_{f(\boldsymbol{x})} \boldsymbol{J}_{\boldsymbol{x}}$ has a rank of at most $c - 1$ (when $\boldsymbol{J}_{\boldsymbol{x}}$ has a rank of $c - 1$).
Assumption 3.1.12. The Jacobian matrix $\boldsymbol{J}_{\boldsymbol{x}}$ has a full rank equal to $c - 1$.

Using Assumptions 3.1.10 and 3.1.12, the constant rank theorem ensures that for small enough δ, f is ϵ-robust at $\boldsymbol{x}$. However, contrary to Proposition 3.1.9, Assumption 3.1.10 does not offer any guarantee on the ϵ-robustness at $\boldsymbol{x}$ for arbitrary δ.

3.2 Derivation of the Regularization Method

In this section, a condition for robustness is derived (Proposition 3.2.5), which can be implemented as a regularization method. Then, two useful results are provided for the practical implementation of this method: an explicit formula for the decomposition of the FIM as $\boldsymbol{G} = \boldsymbol{P}^\top \boldsymbol{P}$ (section 3.2.2), and an easy-to-compute upper-bound of δ, i.e., the Fisher–Rao distance between $f(x)$ and $\Delta^{c-1} \setminus \mathcal{A}_{\boldsymbol{x}}$ (section 3.2.3).

3.2.1 The Partial Isometry Condition

In order to simplify the notations:

- $\boldsymbol{J}_{\boldsymbol{x}}$ is replaced with $\boldsymbol{J}$, which is a full-rank $(c-1) \times d$ real matrix;
- $\boldsymbol{G}_{f(\boldsymbol{x})}$ is replaced with $\boldsymbol{G}$, which is an $(c-1) \times (c-1)$ symmetric positive definite real matrix;

- $\widetilde{\boldsymbol{G}}_{\boldsymbol{x}}$ is replaced with $\widetilde{\boldsymbol{G}}$, which is a $d \times d$ symmetric positive-semidefinite real matrix.

Define $D = \left(\ker(\widetilde{\boldsymbol{G}})\right)^{\perp}$. The two following facts will be used.

Fact 3.2.1.

$$D = \mathrm{rg}(\boldsymbol{J}^{\top}) = (\ker(\boldsymbol{J}))^{\perp} = \left(\ker(\boldsymbol{J}^{\top}\boldsymbol{G}\boldsymbol{J})\right)^{\perp} \tag{3.17}$$

Fact 3.2.2. $\boldsymbol{J}^{\top}\boldsymbol{G}\boldsymbol{J}$ is symmetric positive semidefinite. Thus, by the spectral theorem, the eigenvectors associated with its nonzero eigenvalues are all in $D = \mathrm{rg}(\boldsymbol{J}^{\top})$.

In particular, since $\mathrm{rk}(\boldsymbol{J}) = c - 1$, there exists an orthonormal basis of $T_{\boldsymbol{x}}\mathcal{X}$, denoted by $\mathcal{B} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_m, \boldsymbol{e}_{m+1}, \ldots, \boldsymbol{e}_d)$, such that each $\boldsymbol{e}_i$ is an eigenvector of $\boldsymbol{J}^{\top}\boldsymbol{G}\boldsymbol{J}$, and such that $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_m)$ is a basis of $D = \mathrm{rg}(\boldsymbol{J}^{\top})$ and $(\boldsymbol{e}_{m+1}, \ldots, \boldsymbol{e}_d)$ is a basis of $\ker(\boldsymbol{J})$.

The set $D = \mathrm{rg}(\boldsymbol{J}^{\top})$ is an $(c-1)$-dimensional subspace of $T_{\boldsymbol{x}}\mathcal{X}$. $\tilde{\boldsymbol{g}}_{\boldsymbol{x}}$ does not define an inner product on $T_{\boldsymbol{x}}\mathcal{X}$ because $\widetilde{\boldsymbol{G}}$ has a nontrivial kernel of dimension $d - c + 1$. In particular, the set $\widetilde{\mathcal{B}}_{\boldsymbol{x}}(0, \delta)$ is not bounded, i.e., it is a cylinder rather than a ball. However, when restricted to D, $\tilde{\boldsymbol{g}}_{\boldsymbol{x}}|_D$ defines an inner product. The restriction of $\widetilde{\mathcal{B}}_{\boldsymbol{x}}(0, \delta)$ to D is defined as:

$$\widetilde{\mathcal{B}}_D(0, \delta) = \left\{\boldsymbol{v} \in D : \boldsymbol{v}^{\top}\widetilde{\boldsymbol{G}}\boldsymbol{v} \leq \delta\right\}, \tag{3.18}$$

and similarly, the restriction of $\overline{\mathcal{B}}_{\boldsymbol{x}}(0, \epsilon)$ to D is defined as:

$$\overline{\mathcal{B}}_D(0, \epsilon) = \left\{\boldsymbol{v} \in D : \boldsymbol{v}^{\top}\boldsymbol{v} \leq \epsilon^2\right\}. \tag{3.19}$$

Assume that f is such that equation 3.15 holds (i.e., $\overline{\mathcal{B}}_{\boldsymbol{x}}(0, \epsilon) \subseteq \widetilde{\mathcal{B}}_{\boldsymbol{x}}(0, \delta)$). Moreover, the limit case defined as follows is assumed to hold: for any perturbation size, there is a smaller perturbation of f such that equation 3.15 does not hold anymore. This limit case is equivalent to having $\overline{\mathcal{B}}_D(0, \epsilon) = \widetilde{\mathcal{B}}_D(0, \delta)$. In this case, $\widetilde{\mathcal{B}}_{\boldsymbol{x}}(0, \delta)$ is the smallest possible $\tilde{\boldsymbol{g}}_{\boldsymbol{x}}$-ball (for the inclusion) such that equation 3.15 holds. Experimental observations indicate that enforcing this stronger criteria yields a larger robustifying effect. Therefore, the following assumption is made:

Assumption 3.2.3. Equation 3.15 is replaced with

$$\overline{\mathcal{B}}_D(0, \epsilon) = \widetilde{\mathcal{B}}_D(0, \delta). \tag{3.20}$$

Proposition 3.2.4. *Equation 3.20 is equivalent to*

$$\forall \boldsymbol{v} \in D, \quad \tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v}, \boldsymbol{v}) = \frac{\delta^2}{\epsilon^2}\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v}, \boldsymbol{v}). \tag{3.21}$$

Equation 3.21 can be rewritten in matrix form:

$$\forall \boldsymbol{v} \in D, \quad \boldsymbol{v}^{\top}\widetilde{\boldsymbol{G}}\boldsymbol{v} = \frac{\delta^2}{\epsilon^2}\boldsymbol{v}^{\top}\boldsymbol{v}. \tag{3.22}$$

Section 3.2.2 exploits the properties of the FIM to derive a closed-form expression for a matrix $\boldsymbol{P} \in \mathrm{GL}_{c-1}(\mathbb{R})$, such that $\boldsymbol{G} = \boldsymbol{P}^{\top}\boldsymbol{P}$. For now, it is assumed that such a $\boldsymbol{P}$ is easily available and the goal is to identify a condition on $\boldsymbol{P}$ and $\boldsymbol{J}$, which is equivalent to equation 3.22.

Proposition 3.2.5. *The following statements are equivalent:*

$$(i) \qquad \forall \boldsymbol{u} \in D, \quad \boldsymbol{u}^{\top}\boldsymbol{J}^{\top}\boldsymbol{G}\boldsymbol{J}\boldsymbol{u} = \frac{\delta^2}{\epsilon^2}\boldsymbol{u}^{\top}\boldsymbol{u},$$

$$(ii) \qquad \boldsymbol{P}\boldsymbol{J}\boldsymbol{J}^{\top}\boldsymbol{P}^{\top} = \frac{\delta^2}{\epsilon^2}\boldsymbol{I}_{c-1},$$

where $\boldsymbol{I}_{c-1}$ is the identity matrix of dimension $(c-1) \times (c-1)$.

Proposition 3.2.5 constrains the matrix $\boldsymbol{PJ}$ to be a semi-orthogonal matrix (multiplied by a homothety matrix). A smooth map f between Riemannian manifolds $(\mathcal{X}, \overline{\boldsymbol{g}})$ and $(\Delta^{c-1}, \boldsymbol{g})$ is said to be (locally) isometric if the pullback metric (denoted $f^*\boldsymbol{g}$) coincides with $\overline{\boldsymbol{g}}$, i.e., $f^*\boldsymbol{g} = \overline{\boldsymbol{g}}$. Such a map f locally preserves distances. In this case, $f^*\boldsymbol{g} = \tilde{\boldsymbol{g}}$ is not a metric (since its kernel is non-trivial); thus, f cannot be an isometry. However, equation 3.21 ensures that f locally preserves distances along directions spanned by D. Hence, f becomes a partial isometry, at least in the neighborhood of the training points.

Under the Assumptions 3.1.10, 3.1.12 and 3.2.3, equation (ii) in Proposition 3.2.5 implies robustness as defined in definition 3.1.8. In other words, equation (ii) is a sufficient condition for robustness. However, there is no reason for a neural network to satisfy equation (ii). This is why the following regularization term is defined:

$$\alpha\left(\boldsymbol{x}, \epsilon, f\right) = \frac{1}{(c-1)^2} \left\| \boldsymbol{PJJ}^\top \boldsymbol{P}^\top - \frac{\delta^2}{\epsilon^2} \boldsymbol{I}_{c-1} \right\|, \tag{3.23}$$

where $\|\!|\cdot|\!\|$ is any matrix norm, such as the Frobenius norm or the spectral norm. The Frobenius norm is used in the experiments of section 3.3. Computing $\alpha(\boldsymbol{x}, \epsilon, f)$ requires only the computation of the Jacobian matrix $\boldsymbol{J}$, which can be efficiently achieved with backpropagation. Finally, the loss function is:

$$\mathcal{L}_{iso}\left(\boldsymbol{y}, \boldsymbol{x}, \epsilon, f\right) = \mathcal{L}_{CE}\left(\boldsymbol{y}, f(\boldsymbol{x})\right) + \lambda\, \alpha\left(\boldsymbol{x}, \epsilon, f\right), \tag{3.24}$$

where $\mathcal{L}_{CE}$ is the cross-entropy loss, and $\lambda > 0$ is a hyperparameter controlling the strength of the regularization with respect to the cross-entropy loss. The regularization term $\alpha(\boldsymbol{x}, \epsilon, f)$ is minimized during training, such that the model is pushed to satisfy the sufficient condition of robustness.

3.2.2 Coordinate Change

This subsection shows how to compute the matrix $\boldsymbol{P}$ that was introduced in Proposition 3.2.5. To this end, Δ^{c-1} is isometrically embedded into the Euclidean space $\mathbb{R}^c$ using the following inclusion map:

$$\begin{aligned} \mu : \Delta^{c-1} &\longrightarrow \mathbb{R}^c \\ \left(\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{c-1}\right) &\longmapsto 2\left(\sqrt{\boldsymbol{\theta}^1}, \ldots, \sqrt{\boldsymbol{\theta}^{c-1}}, \sqrt{1 - \sum_{i=1}^{c-1} \boldsymbol{\theta}^i}\right) \end{aligned}$$

It is easy to see that μ is an embedding. If $\mathcal{S}^{c-1}(2)$ is the sphere of radius 2 centered at the origin in $\mathbb{R}^c$, then $\mu\left(\Delta^{c-1}\right)$ is the subset of $\mathcal{S}^{c-1}(2)$, where all coordinates are strictly positive (using the standard coordinates of $\mathbb{R}^c$).

Proposition 3.2.6. *Let $\boldsymbol{g}$ be the Fisher information metric on Δ^{c-1} (definition 3.1.2), and $\overline{\boldsymbol{g}}$ be the Euclidean metric on $\mathbb{R}^c$. Then, μ is an isometric embedding of $(\Delta^{c-1}, \boldsymbol{g})$ into $(\mathbb{R}^c, \overline{\boldsymbol{g}})$.*

Now, the stereographic projection is used to embed Δ^{c-1} into $\mathbb{R}^{c-1}$:

$$\begin{aligned} \tau : \mu(\Delta^{c-1}) &\longrightarrow \mathbb{R}^{c-1} \\ (\mu^1, \ldots, \mu^{c-1}, \mu^c) &\longmapsto 2\left(\frac{\mu^1}{2-\mu^c}, \ldots, \frac{\mu^{c-1}}{2-\mu^c}\right), \end{aligned}$$

with $\mu^c = 2\sqrt{1 - \sum_{i=1}^{c-1} \boldsymbol{\theta}^i}$.

Proposition 3.2.7. *In the coordinates τ, the FIM is:*

$$\boldsymbol{G}_{\tau,ij} = \frac{4}{\left(1 + \|\tau/2\|^2\right)^2}\delta_{ij}. \tag{3.25}$$

Let $\widetilde{\boldsymbol{J}}$ be the Jacobian matrix of $\tau \circ \mu : \Delta^{c-1} \to \mathbb{R}^{c-1}$ at $f(\boldsymbol{x})$. Then, it follows that:

$$\boldsymbol{G} = \widetilde{\boldsymbol{J}}^\top \boldsymbol{G}_\tau \widetilde{\boldsymbol{J}} = \frac{4}{\left(1 + \|\tau/2\|^2\right)^2}\widetilde{\boldsymbol{J}}^\top \widetilde{\boldsymbol{J}}. \tag{3.26}$$

Thus, one can choose:

$$\boldsymbol{P} = \frac{2}{1 + \|\tau/2\|^2}\widetilde{\boldsymbol{J}}. \tag{3.27}$$

Write $f(\boldsymbol{x}) = \boldsymbol{\theta} = (\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{c-1})$ and $\boldsymbol{\theta}^c = 1 - \sum_{i=1}^{c-1} \boldsymbol{\theta}^i$. For simplicity, write $\tau^i(\boldsymbol{\theta}) = \tau^i(\mu(\boldsymbol{\theta})) = 2\sqrt{\boldsymbol{\theta}^i}/(1 - \sqrt{\boldsymbol{\theta}^c})$ for $i = 1, \ldots, c-1$. More explicitly, one has:

Proposition 3.2.8. *For $i, j = 1, \ldots, c-1$:*

$$\boldsymbol{P}_{ij} = \frac{\delta_{ij}}{\sqrt{\boldsymbol{\theta}^i}} - \frac{\tau^i(\boldsymbol{\theta})}{2\sqrt{\boldsymbol{\theta}^c}}. \tag{3.28}$$

3.2.3 The Fisher–Rao Distance

In this subsection, a simple upper-bound for δ (i.e., the Fisher–Rao distance between $f(\boldsymbol{x})$ and $\Delta^{c-1} \setminus \mathcal{A}_{\boldsymbol{x}}$) is derived. Proposition 3.2.6 has shown that the probability simplex Δ^{c-1} endowed with the FIM can be isometrically embedded into the $(c-1)$-sphere of radius 2. Therefore, the angle γ between two distributions of coordinates $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ in Δ^{c-1} with $\mu_1 = \mu(\boldsymbol{\theta}_1)$ and $\mu_2 = \mu(\boldsymbol{\theta}_2)$ is such that:

$$\cos(\gamma) = \frac{1}{4}\sum_{i=1}^{c} \mu_1^i \mu_2^i = \sum_{i=1}^{c} \sqrt{\boldsymbol{\theta}_1^i \boldsymbol{\theta}_2^i}. \tag{3.29}$$

The Riemannian distance between these two points is the arc length on the sphere:

$$\boldsymbol{d}_{FR}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = 2\arccos \sum_{i=1}^{c} \sqrt{\boldsymbol{\theta}_1^i \boldsymbol{\theta}_2^i}. \tag{3.30}$$

In the regularization term defined in equation 3.23, δ is replaced with the following upper bound:

$$\delta = \boldsymbol{d}_{FR}\left(f(\boldsymbol{x}), \Delta^{c-1} \setminus \mathcal{A}_{\boldsymbol{x}}\right) \leq \boldsymbol{d}_{FR}\left(f(\boldsymbol{x}), O\right), \tag{3.31}$$

where $O = \frac{1}{c}(1, \ldots, 1)$ is the center of the simplex Δ^{c-1}. Thus,

$$\delta \leq 2\arccos \sum_{i=1}^{c} \sqrt{\frac{f^i(\boldsymbol{x})}{c}}. \tag{3.32}$$

3.3 Experiments

The regularization method introduced in section 3.2 is evaluated on MNIST and CIFAR-10 datasets. The method uses the loss function introduced in equation 3.24.

3.3.1 Experiments on MNIST Dataset

3.3.1.1 Experimental Setup

For the MNIST dataset, a LeNet model is implemented with two convolutional layers of 32 and 64 channels, respectively, followed by one hidden layer with 128 neurons[1]. Three models are trained: one regularized model, one baseline unregularized model, and one model trained with adversarial training. All three models are trained with the Adam optimizer ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) for 30 epochs, with a batch size of 64, and a learning rate of 10^{-3}. For the regularization term, a budget of $\epsilon = 5.6$ is used, which is chosen to contain the L_∞ ball of radius 0.2. The adversarial training is performed with 10 iterations of PGD with a budget $\epsilon_{adv} = 0.2$ using L_∞ norm. It was found that $\lambda = 10^{-6}$ yields the best performance in terms of robustness–accuracy trade-off; this value is small because there was not attempt to normalize the regularization term.

The models are trained on the 60,000 images of MNIST's training set and then tested on 10,000 images of the test set. The baseline model achieves an accuracy of 98.9% (9893/10,000), the regularized model achieves an accuracy of 94.0% (9,403/10,000), and the adversarially trained model achieves an accuracy of 98.8% (9,883/10,000). Although the current implementation of the regularized model is almost six times slower to train than the baseline model, it may be possible to accelerate the training using, for example, the technique proposed by Shafahi *et al.* [135], or using another method to approximate the spectral norm of $\widetilde{\boldsymbol{J}}$. Even without relying on these acceleration techniques, the regularized model is still faster to train than the adversarially trained model.

3.3.1.2 Robustness to Adversarial Attacks

To measure the adversarial robustness of the models, the PGD attack is used with the L_∞ norm, 40 iterations, and a step size of 0.01. The L_∞ norm yields the hardest possible attack for the method, and corresponds more to the human notion of "indistinguishable images" than the L_2 norm. The attacks are performed on the test set, and only on images that are correctly classified by each model. The results are reported in Figure 3.3. The regularized model has a slightly lower accuracy than the baseline model for small perturbations, but the baseline model suffers a drop in accuracy above the attack level $\epsilon = 0.1$. Adversarial training achieves high accuracy for small- to medium-sized perturbations but the accuracy decreases sharply above $\epsilon = 0.3$. The regularized model remains robust even for large perturbations. The baseline model reaches 50% accuracy at $\epsilon = 0.2$ and the adversarially trained model at $\epsilon = 0.325$, while the regularized model reaches 50% accuracy at $\epsilon = 0.4$.

[1]The code is available here: `https://github.com/lshigarrier/geometric_robustness.git`.

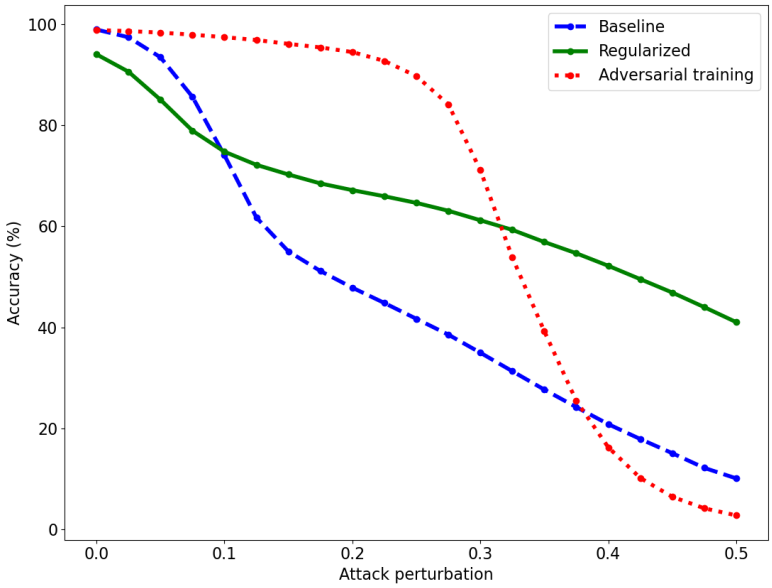


Figure 3.3: Accuracy of the baseline (dashed, blue), regularized (solid, green), and adversarially trained (dotted, red) models for various attack perturbations on the MNIST dataset. The perturbations are obtained with PGD using L_∞ norm.

Table 3.1 provides more results against AutoAttack (AA) [102], which was designed to offer a more reliable evaluation of adversarial robustness. For a fair comparison, and in addition to a baseline model (BASE), the partial isometry defense (ISO) is compared with several other computationally efficient defenses: distillation (DIST) [49], Jacobian regularization (JAC) [133], which also relies on the Jacobian matrix of the network, and Fisher information regularization (FIR) [57], which also leverages information geometry. An adversarially trained (AT) model using PGD is also considered. ISO is the best defense that does not rely on adversarial training. In future work, ISO may be combined with AT to further boost performance. Note that ISO and JAC are more robust against L_2 attacks since they were designed to defend the model against such attacks. On the other hand, AT is more robust against L_∞ attacks, because the adversarial training was performed with the L_∞ norm.

Table 3.1: Clean and robust accuracy on MNIST against AA, averaged over 10 runs. The number in parentheses is the attack strength.

Defense	**BASE**	**ISO**	**DIST**	**JAC**	**FIR**	**AT**
Clean	99.01	96.51	98.81	98.95	98.84	98.98
AA-L_2 (1.5)	35.70	43.38	35.35	38.74	1.68	73.34
AA-L_∞ (0.15)	10.38	22.15	9.63	13.30	0.03	95.43

3.3.2 Experiments on CIFAR-10 Dataset

A DenseNet121 model fine-tuned on CIFAR-10 using pre-trained weights for ImageNet[2] is used. As for the MNIST experiments, the partial isometry defense is compared with distillation (DIST), Jacobian regularization (JAC), and Fisher information regularization (FIR). Here,

[2]The code is available here: `https://github.com/lshigarrier/iso_defense.git`.

adversarial training (AT) relies on the fast gradient sign method (FGSM) attack [136]. All defenses are compared against PGD for various attack strengths. The results are presented in Table 3.2. The defenses are evaluated in a "gray-box" setting where the adversary can access the architecture and the data but not the weights. More precisely, the adversarial examples are crafted from the test set of CIFAR-10 using another unregularized DenseNet121 model. AT is the more robust method, but ISO achieves a robust accuracy 30% higher than the next best analogous method (FIR).

One of the objectives of this method is to provide alternatives to adversarial training (AT). Apart from high computational costs, AT suffers from several limitations: it only robustifies against the chosen attack at the chosen budget and it does not offer a robustness guarantee. For example, under Gaussian noise, AT accuracy decreases faster than baseline accuracy (i.e., no defense). Achieving high robustness accuracy against specific attacks on a specific benchmark is insufficient and misleading to measure the true robustness of the evaluated model. The method developed in this chapter offers a new point of view that can be extended to certified defense methods in future works.

Table 3.2: Clean and robust accuracy on CIFAR-10 against PGD. The number in parentheses is the attack strength.

Defense	**BASE**	**ISO**	**DIST**	**JAC**	**FIR**	**AT**
Clean	92.93	76.86	84.96	86.17	89.98	80.78
PGD (4/255)	2.49	40.17	7.54	8.56	9.74	68.82
PGD (8/255)	0.47	39.68	3.35	3.66	4.05	66.61

3.4 Discussion

In 2019, Zhao *et al.* [40] proposed to use the Fisher information metric in the setting of adversarial attacks. They used the eigenvector associated with the largest eigenvalue of the pullback of the FIM as an attack direction. Following their work, Shen *et al.* [57] suggested a defense mechanism by suppressing the largest eigenvalue of the FIM. They upper-bounded the largest eigenvalue by the trace of the FIM. As in the method developed in this chapter, they added a regularization term to encourage the model to have smaller eigenvalues. Moreover, they showed that their approach is equivalent to label smoothing [56]. In the framework developed in this chapter, their method consists of expanding the geodesic ball $\tilde{b}(\boldsymbol{x}, \delta)$ as much as possible. However, their approach does not guarantee that the constraint imposed on the model will not harm the accuracy more than necessary. In the framework of this chapter, matrix $\boldsymbol{PJ}$ (compared with δ/ϵ) informs the model on the precise restriction that must be imposed to achieve adversarial robustness in the L_2 ball of radius ϵ.

Cisse *et al.* [54] introduced another adversarial defense called Parseval networks (see section 2.1.2.2 for more details). To achieve adversarial robustness, the authors aim to control the Lipschitz constant of each layer of the model to be close to unity. This is achieved by constraining the weight matrix of each layer to be a Parseval tight frame, which is another name for semi-orthogonal matrix. Since the Jacobian matrix of the entire model with respect to the input is almost the product of the weight matrices, the Parseval network defense is similar to the defense proposed in this chapter, albeit with completely different rationales. This suggests that geometric reasoning could successfully supplement the line of work on Lipschitz constants of neural networks, such as in [126].

Following another line of work, Hoffman *et al.* [133] advanced a Jacobian regularization to

improve adversarial robustness. Their regularization consists of using the Frobenius norm of the input–output Jacobian matrix. To avoid computing the true Frobenius norm, they relied on random projections, which are shown to be both efficient and accurate. This method is similar to the method of Shen *et al.* [57] in the sense that it will also increase the radius of the geodesic ball. However, the Jacobian regularization does not take into account the geometry of the output space (i.e., the Fisher information metric) and assumes that the probability simplex Δ^{c-1} is Euclidean.

Although this study focuses on L_2 norm robustness, it must be pointed out that there are other "distinguishability" measures that can be used to study adversarial robustness, including all other L_p norms. In particular, the L_∞ norm is often considered to be the most natural choice when working with images. However, the L_∞ norm is not induced by any inner product and, hence, there is no Riemannian metric that induces the L_∞ norm. However, given an L_∞ budget ϵ_∞, a L_2 budget $\epsilon_2 = \sqrt{d}\epsilon_\infty$ can be chosen such that any attack in the ϵ_∞ budget will also respect the ϵ_2 budget. When working on images, other dissimilarity measures are rotations, deformations, and color changes of the original image. Contrary to the L_2 or L_∞ norms, these measures do not rely on a pixel-based coordinate system. However, it is possible to define unrestricted attacks based on these spatial dissimilarities, for example, in [47] (see section 2.1.1).

In this chapter, the partial isometry regularization has been derived for a classification task. The method can be extended to regression tasks by considering the family of multivariate normal distributions as the output space. Some experiments are conducted on the manifold of univariate normal distributions in section 4.2. On the probability simplex Δ^{c-1}, the FIM is a metric with constant positive curvature, while it has constant negative curvature on the manifold of multivariate normal distributions [137].

Finally, the precise quantification of the robustness condition presented in equation 3.11 and Proposition 3.2.5 paves the way for the development of a certified defense [113] in this framework. By strongly enforcing Proposition 3.2.5 on a chosen proportion of the training set, it may be possible to maximize the accuracy under the constraint of a chosen robustness level, which offers another solution to the robustness–accuracy trade-off [59, 85]. Certifiable defenses are a necessary step for the deployment of deep learning models in critical domains and missions, such as civil aviation, security, defense, and healthcare, where a certification may be required to ensure a sufficient level of trustworthiness.

3.5 Conclusion

In this chapter, an information geometric approach to the problem of adversarial robustness in machine learning models has been introduced. The proposed defense consists of enforcing a partial isometry between the input space endowed with the Euclidean metric and the probability simplex endowed with the Fisher information metric. A regularization term was subsequently derived to achieve robustness during training. The proposed strategy is tested on the MNIST and CIFAR-10 datasets, and shows a considerable increase in robustness without harming the accuracy. Future works could evaluate the method on other benchmarks and real-world datasets. Several attack methods could also be considered in addition to PGD and AutoAttack. Although this chapter focuses on L_2 norm robustness, future work could consider other “distinguishability” measures.

This chapter extends a recent, promising but understudied framework for adversarial robustness based on information geometric tools. The FIM has already been harnessed to develop attacks [40] and defenses [58, 57] but a precise robustness analysis is yet to be proposed. This chapter is

a step toward the development of such an analysis, which might yield certified guarantees relying on these geometric tools. This chapter has demonstrated the usefulness of such an approach by developing a preliminary robustification method. Model robustification is a hard, unsolved yet vital problem to ensure the trustworthiness of deep learning tools in safety-critical applications. The framework proposed here could be extended and applied to existing certification strategies, such as Lipschitz-based [107] or randomized smoothing [113], where statistical models naturally appear (see section 2.2).

In the next chapter, several other research directions are explored about the relationship between machine learning robustness and information geometry. While the method described in this chapter has been fully derived and evaluated, only preliminary results are presented for the various frameworks introduced in the next chapter, along with suggestions for future developments.

3.6 Proofs

Proof of Proposition 3.1.11. (3.16) $\Rightarrow$ (3.15). Assume (3.16). Let $\boldsymbol{v} \in \overline{\mathcal{B}}_{\boldsymbol{x}}(0,\epsilon)$. Thus, $\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v}) \leq \epsilon^2$. One has

$$\tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v}) \leq \frac{\delta^2}{\epsilon^2}\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v}) \leq \frac{\delta^2}{\epsilon^2}\epsilon^2 = \delta^2. \tag{3.33}$$

Thus, $\boldsymbol{v} \in \widetilde{\mathcal{B}}_{\boldsymbol{x}}(0,\delta)$.

(3.15) $\Rightarrow$ (3.16). Assume (3.15). Let $\boldsymbol{v} \in T_{\boldsymbol{x}}\mathcal{X}$, $\boldsymbol{v} \neq 0$. Define $\boldsymbol{w} = \epsilon\,\boldsymbol{v}/\sqrt{\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v})}$. Then, $\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{w},\boldsymbol{w}) = \epsilon^2$. Thus, $\boldsymbol{w} \in \overline{\mathcal{B}}_{\boldsymbol{x}}(0,\epsilon)$. Hence, $\boldsymbol{w} \in \widetilde{\mathcal{B}}_{\boldsymbol{x}}(0,\delta)$. Thus, $\tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{w},\boldsymbol{w}) < \delta^2$. Finally, one has

$$\tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{w},\boldsymbol{w}) = \frac{\epsilon^2}{\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v})}\tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v}) < \delta^2. \tag{3.34}$$

Equation (3.16) is obtained by multiplying by $\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v})/\epsilon^2$. □

Proof of Fact 3.2.1. Only the third equality is proved (the second equality is a well-known fact of linear algebra).

Let $\boldsymbol{u} \in \ker \boldsymbol{J}$. Then, $\boldsymbol{J}^\top\boldsymbol{G}\boldsymbol{J}\boldsymbol{u} = 0$; thus, $\boldsymbol{u} \in \ker(\boldsymbol{J}^\top\boldsymbol{G}\boldsymbol{J})$. Hence, $\Big(\ker(\boldsymbol{J}^\top\boldsymbol{G}\boldsymbol{J})\Big)^\perp \subseteq (\ker(\boldsymbol{J}))^\perp$.

Let $\boldsymbol{v} \in \ker \boldsymbol{J}^\top\boldsymbol{G}\boldsymbol{J}$. Since $\boldsymbol{G}$ is symmetric positive-definite, the function $\boldsymbol{w} \mapsto N(\boldsymbol{w}) = \sqrt{\boldsymbol{w}^\top\boldsymbol{G}\boldsymbol{w}}$ is a norm. It follows that $0 = \boldsymbol{v}^\top\boldsymbol{J}^\top\boldsymbol{G}\boldsymbol{J}\boldsymbol{v} = N(\boldsymbol{J}\boldsymbol{v})^2$. The positive-definiteness of the norm N implies $\boldsymbol{J}\boldsymbol{v} = 0$. Thus, $\boldsymbol{v} \in \ker \boldsymbol{J}$. Hence, $(\ker(\boldsymbol{J}))^\perp \subseteq \Big(\ker(\boldsymbol{J}^\top\boldsymbol{G}\boldsymbol{J})\Big)^\perp$. □

Proof of Proposition 3.2.4. The implication (3.21) $\Rightarrow$ (3.20) is immediate (by double inclusion).

Now, assume (3.20) holds. Let $\boldsymbol{v} \in D$. Define $\boldsymbol{w}_1 = \epsilon\,\boldsymbol{v}/\sqrt{\overline{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v})}$ and $\boldsymbol{w}_2 = \epsilon\,\boldsymbol{v}/\sqrt{\tilde{\boldsymbol{g}}_{\boldsymbol{x}}(\boldsymbol{v},\boldsymbol{v})}$. Then, with a similar argument as in the proof of Proposition 3.1.11, Equation (3.21) can be obtained. Note that $\boldsymbol{w}_2$ is well-defined because $\boldsymbol{v} \notin \ker(\boldsymbol{J})$. □

Proof of Proposition 3.2.5. First, the polar decomposition is introduced.

Let $\boldsymbol{A}$ be a $(c-1) \times d$ matrix.

Define the absolute value of $\boldsymbol{A}$ by $|\boldsymbol{A}| = \Big(\boldsymbol{A}^\top\boldsymbol{A}\Big)^{\frac{1}{2}}$. Note that the square root of $\boldsymbol{A}^\top\boldsymbol{A}$ is well-defined because it is a positive-semidefinite matrix

Define the linear map $u : \mathrm{rg}\,(|\boldsymbol{A}|) \to \mathrm{rg}(\boldsymbol{A})$ by $u(|\boldsymbol{A}|\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ for any $\boldsymbol{x} \in \mathbb{R}^d$.

Using the fact that $|\boldsymbol{A}|$ is symmetric, it follows that $\|\boldsymbol{A}\boldsymbol{x}\|^2 = \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} = (\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{x} = (|\boldsymbol{A}|^2\boldsymbol{x})^\top \boldsymbol{x} = \boldsymbol{x}^\top |\boldsymbol{A}|^\top |\boldsymbol{A}|\boldsymbol{x} = \||\boldsymbol{A}|\boldsymbol{x}\|^2$; thus, u is an isometry (note that u can be arbitrarily extended to the entire $\mathbb{R}^d$, e.g., by setting $\ker(u) = \ker(|\boldsymbol{A}|)$.).

Let $\boldsymbol{U}$ be the matrix associated to u in the canonical basis.

The main result will now be demonstrated.

Let $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{J}$. Using the polar decomposition, one has

$$\boldsymbol{P}\boldsymbol{J} = \boldsymbol{U}|\boldsymbol{P}\boldsymbol{J}|, \tag{3.35}$$

where $\boldsymbol{U}$ is an isometry from $\mathrm{rg}(|\boldsymbol{P}\boldsymbol{J}|) = (\ker|\boldsymbol{P}\boldsymbol{J}|)^\perp = (\ker(\boldsymbol{P}\boldsymbol{J}))^\perp = (\ker(\boldsymbol{J}))^\perp = D$ to $\mathrm{rg}(\boldsymbol{P}\boldsymbol{J}) = \mathbb{R}^{c-1}$ (using the assumption that $\mathrm{rk}(\boldsymbol{J}) = c-1$). Transposing this relation, it follows that

$$\boldsymbol{J}^\top \boldsymbol{P}^\top = |\boldsymbol{P}\boldsymbol{J}|\boldsymbol{U}^\top. \tag{3.36}$$

Hence, by multiplying both relations, one has

$$\boldsymbol{P}\boldsymbol{J}\boldsymbol{J}^\top \boldsymbol{P}^\top = \boldsymbol{U}|\boldsymbol{P}\boldsymbol{J}|^2\boldsymbol{U}^\top = \boldsymbol{U}\boldsymbol{J}^\top \boldsymbol{P}^\top \boldsymbol{P}\boldsymbol{J}\boldsymbol{U}^\top \tag{3.37}$$

Assume that (ii) holds, i.e., $\boldsymbol{P}\boldsymbol{J}\boldsymbol{J}^\top \boldsymbol{P} = \boldsymbol{I}_{c-1}$. Then,

$$\boldsymbol{J}^\top \boldsymbol{G}\boldsymbol{J} = \boldsymbol{J}^\top \boldsymbol{P}^\top \boldsymbol{P}\boldsymbol{J} = \boldsymbol{U}^\top \boldsymbol{P}\boldsymbol{J}\boldsymbol{J}^\top \boldsymbol{P}^\top \boldsymbol{U} = \boldsymbol{U}^\top \boldsymbol{U}. \tag{3.38}$$

Since $\boldsymbol{U}$ is an isometry from D to $\mathbb{R}^{c-1}$, then $\boldsymbol{U}^\top \boldsymbol{U}$ is the projection onto D, denoted by Π_D. Therefore, one has $\boldsymbol{J}^\top \boldsymbol{G}\boldsymbol{J} = \Pi_D$, which is (i).

Now, assume that (i) holds, i.e., $\boldsymbol{J}^\top \boldsymbol{P}^\top \boldsymbol{P}\boldsymbol{J} = \Pi_D$, where Π_D is the projection onto D. It follows that

$$\boldsymbol{P}\boldsymbol{J}\boldsymbol{J}^\top \boldsymbol{P}^\top = \boldsymbol{U}\boldsymbol{J}^\top \boldsymbol{P}^\top \boldsymbol{P}\boldsymbol{J}\boldsymbol{U}^\top = \boldsymbol{U}\Pi_D\boldsymbol{U}^\top. \tag{3.39}$$

Since $\mathrm{rg}(\boldsymbol{U}^\top) = D$; then, $\Pi_D\boldsymbol{U}^\top = \boldsymbol{U}^\top$. Since $\boldsymbol{U}$ is an isometry from D to $\mathbb{R}^{c-1}$; then, $\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}_{c-1}$. Thus, $\boldsymbol{P}\boldsymbol{J}\boldsymbol{J}^\top \boldsymbol{P}^\top = \boldsymbol{I}_{c-1}$ which is (ii). □

Proof of Proposition 3.2.6. The objective is to show that $\mu^*\overline{\boldsymbol{g}} = \boldsymbol{g}$. Using the coordinates $\boldsymbol{\theta}$ on Δ^{c-1} (definition 3.1.1) and the standard coordinates on $\mathbb{R}^c$, and writing $f(\boldsymbol{x}) = \boldsymbol{\theta}_0 = (\boldsymbol{\theta}_0^1, \ldots, \boldsymbol{\theta}_0^{c-1})$ one has

$$\begin{aligned}
\boldsymbol{G}_{ij} &= \boldsymbol{G}_{\boldsymbol{\theta}_0, ij}, \\
&= \sum_{\alpha=1}^{c}\sum_{\beta=1}^{c} \frac{\partial \mu^\alpha(\boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}^i}\frac{\partial \mu^\beta(\boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}^j}\delta_{\alpha\beta}, \\
&= \sum_{\alpha=1}^{c} \frac{\partial \mu^\alpha(\boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}^i}\frac{\partial \mu^\alpha(\boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}^j}.
\end{aligned}$$

For $i = 1, \ldots, c-1$ and $\alpha = 1, \ldots, c-1$ one has:

$$\frac{\partial \mu^\alpha(\boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}^i} = \frac{\delta_{i\alpha}}{\sqrt{\boldsymbol{\theta}_0^i}}, \tag{3.40}$$

and for $\alpha = c$:

$$\frac{\partial \mu^c(\boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}^i} = -\frac{1}{\sqrt{\boldsymbol{\theta}_0^c}}, \tag{3.41}$$

with $\boldsymbol{\theta}_0^c = \sqrt{1 - \sum_{i=1}^{c-1} \boldsymbol{\theta}_0^i}$. Thus,

$$\boldsymbol{G}_{\boldsymbol{\theta}, ij} = \frac{\delta_{ij}}{\boldsymbol{\theta}_0^i} + \frac{1}{\boldsymbol{\theta}_0^c}, \tag{3.42}$$

which is the FIM, as defined in definition 3.1.2. □

Proof of Proposition 3.2.7. For $i = 1, \dots, c-1$, the inverse transformation of $\tau(\mu)$ is

$$\mu^i(\tau) = \frac{2\tau^i}{1 + \|\tau/2\|^2}, \tag{3.43}$$

and

$$\mu^c(\tau) = 2\frac{\|\tau/2\|^2 - 1}{\|\tau/2\|^2 + 1}. \tag{3.44}$$

The proofs of Equations (3.43) and (3.44) are provided below.

Moreover, according to Proposition 3.2.6, the FIM in the coordinates $(\mu^1, \dots, \mu^{c-1})$ is the metric induced on $\mu(\Delta^{c-1})$ by the identity matrix (i.e., the Euclidean metric) of $\mathbb{R}^c$. Hence, it follows that:

$$\begin{aligned}\boldsymbol{G}_{\tau,ij} &= \sum_{\alpha=1}^{c}\sum_{\beta=1}^{c} \frac{\partial\mu^\alpha(\tau)}{\partial\tau^i}\frac{\partial\mu^\beta(\tau)}{\partial\tau^j}\delta_{\alpha\beta}, \\ &= \sum_{\alpha=1}^{c} \frac{\partial\mu^\alpha(\tau)}{\partial\tau^i}\frac{\partial\mu^\alpha(\tau)}{\partial\tau^j}.\end{aligned}$$

For $i = 1, \dots, c-1$ and $\alpha = 1, \dots, c-1$, one has

$$\frac{\partial\mu^\alpha(\tau)}{\partial\tau^i} = \frac{2}{1 + \|\tau/2\|^2}\left(\delta_{i\alpha} - \frac{\tau^\alpha\tau^i}{2\left(1 + \|\tau/2\|^2\right)}\right), \tag{3.45}$$

and for $\alpha = c$:

$$\frac{\partial\mu^c(\tau)}{\partial\tau^i} = \frac{2\tau^i}{\left(1 + \|\tau/2\|^2\right)^2}, \tag{3.46}$$

Thus,

$$\begin{aligned}\boldsymbol{G}_{\tau,ij} =& \frac{4}{\left(1 + \|\tau/2\|^2\right)^2}\left(\sum_{\alpha=1}^{c-1}\left\{\delta_{i\alpha}\delta_{j\alpha} - \frac{\delta_{i\alpha}\tau^j\tau^\alpha}{2\left(1 + \|\tau/2\|^2\right)} - \frac{\delta_{j\alpha}\tau^i\tau^\alpha}{2\left(1 + \|\tau/2\|^2\right)} + \frac{\tau^i\tau^j\left(\tau^\alpha\right)^2}{4\left(1 + \|\tau/2\|^2\right)^2}\right\} + \frac{\tau^i\tau^j}{\left(1 + \|\tau/2\|^2\right)^2}\right), \\ =& \frac{4}{\left(1 + \|\tau/2\|^2\right)^2}\left(\delta_{ij} - \frac{\tau^i\tau^j}{1 + \|\tau/2\|^2} + \frac{\tau^i\tau^j\|\tau/2\|^2}{\left(1 + \|\tau/2\|^2\right)^2} + \frac{\tau^i\tau^j}{\left(1 + \|\tau/2\|^2\right)^2}\right), \\ =& \frac{4}{\left(1 + \|\tau/2\|^2\right)^2}\left(\delta_{ij} - \frac{\tau^i\tau^j}{1 + \|\tau/2\|^2} + \frac{\tau^i\tau^j}{1 + \|\tau/2\|^2}\right), \\ =& \frac{4}{\left(1 + \|\tau/2\|^2\right)^2}\delta_{ij}.\end{aligned}$$

□

Proof of Equations (3.43) and (3.44). One has $\tau^i(\mu) = \lambda\mu^i$ with $\lambda = 2/\left(2 - \mu^c\right)$, where μ^c can be expressed as a function of τ as will be shown now. First, the following holds

$$\|\tau\|^2 = \sum_{i=1}^{c-1}(\tau^i)^2 = \lambda^2\|\mu\|^2. \tag{3.47}$$

Since μ belongs to the sphere of radius 2, one has $\|\mu\|^2 + (\mu^c)^2 = 4$. Thus,

$$\|\tau\|^2 = \lambda^2\left(4 - (\mu^c)^2\right) = 4\frac{4 - (\mu^c)^2}{(2 - \mu^c)^2} = 4\frac{2 + \mu^c}{2 - \mu^c}. \tag{3.48}$$

Isolating μ^c, one has

$$\mu^c(\tau) = \frac{2\|\tau\|^2 - 8}{\|\tau\|^2 + 4} = 2\frac{\|\tau/2\|^2 - 1}{\|\tau/2\|^2 + 1}. \tag{3.49}$$

Now, μ^c can be replaced with the expression of λ. One obtains $\lambda = \left(1 + \|\tau/2\|^2\right)/2$; thus,

$$\mu^i(\tau) = \frac{\tau^i}{\lambda} = \frac{2\tau^i}{1 + \|\tau/2\|^2} \tag{3.50}$$

□

Proof of Proposition 3.2.8. One has

$$\tau^i(\boldsymbol{\theta}) = 2\sqrt{\boldsymbol{\theta}^i} \Big/ \left(1 - \sqrt{\boldsymbol{\theta}^c}\right) \tag{3.51}$$

Thus,

$$\left\|\frac{\tau(\boldsymbol{\theta})}{2}\right\|^2 = \sum_{i=1}^{c-1} \frac{\tau^i(\boldsymbol{\theta})^2}{4} = \frac{\sum_{i=1}^{c-1} \boldsymbol{\theta}^i}{\left(1 - \sqrt{\boldsymbol{\theta}^c}\right)^2} = \frac{1 - \boldsymbol{\theta}^c}{\left(1 - \sqrt{\boldsymbol{\theta}^c}\right)^2} = \frac{1 + \sqrt{\boldsymbol{\theta}^c}}{1 - \sqrt{\boldsymbol{\theta}^c}}.$$

Hence, for any $i = 1, \ldots, c-1$:

$$\frac{2}{1 + \|\tau(\boldsymbol{\theta})/2\|^2} = 1 - \sqrt{\boldsymbol{\theta}^c} = \frac{2\sqrt{\boldsymbol{\theta}^i}}{\tau^i(\boldsymbol{\theta})}. \tag{3.52}$$

Now, $\widetilde{\boldsymbol{J}}$ is computed. Let i and j in $\{1, \ldots, c-1\}$:

$$\frac{\partial \tau^i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^j} = \frac{\delta_{ij}}{\sqrt{\boldsymbol{\theta}^i}\left(1 - \sqrt{\boldsymbol{\theta}^c}\right)} - \frac{\sqrt{\boldsymbol{\theta}^i}}{\sqrt{\boldsymbol{\theta}^c}\left(1 - \sqrt{\boldsymbol{\theta}^c}\right)^2}, \tag{3.53}$$

$$= \frac{\tau^i(\boldsymbol{\theta})}{2}\left(\frac{\delta_{ij}}{\boldsymbol{\theta}^i} - \frac{\tau^i(\boldsymbol{\theta})}{2\sqrt{\boldsymbol{\theta}^i\boldsymbol{\theta}^c}}\right). \tag{3.54}$$

Replacing Equations (3.52) and (3.54) with Equation (3.27) yields the result. □

Chapter 4

Robustness and Geometry

In this chapter, the relationships between information geometry and neural networks robustness are investigated. First, a model of recurrent neural networks seen as a stochastic differential equation is introduced. Then, several experiments around the data leaf hypothesis proposed in [1] are presented.

4.1 The Image and Kernel Foliations

In this section, following [1], the image and kernel foliations that will be used in the next sections of this chapter are presented. Information geometry concepts have already been introduced in sections 2.1.3 and 3.1.1.

Consider a machine learning model $f : (\boldsymbol{x}, \boldsymbol{w}) \mapsto \boldsymbol{\theta}$, where $\boldsymbol{x} \in \mathbb{R}^d$ is the input and $\boldsymbol{w}$ are the learnable parameters of the model. The output $\boldsymbol{\theta}$ is the parameter of a statistical manifold $\mathcal{S} = \{p(\boldsymbol{y}|\boldsymbol{\theta})\}$. The prediction of the model is sampled from $p(\boldsymbol{y}|f(\boldsymbol{x}, \boldsymbol{w}))$. In the remaining of this chapter, the model is assumed to be already trained, and only the behavior of the model at inference is studied. Therefore, the learnable parameters $\boldsymbol{w}$ are fixed and can be discarded to ease the notations. From now on, the model f takes only the input $\boldsymbol{x}$ and returns the parameters $\boldsymbol{\theta} = f(\boldsymbol{x})$ of the predictive distribution. The manifold $\mathcal{S}$ is endowed with the Fisher information metric (FIM) $\boldsymbol{g}$ as defined in section 3.1.1. Consider the pullback of the FIM $f^*\boldsymbol{g}$. For a given $\boldsymbol{x} \in \mathbb{R}^d$, the pullback of the FIM by the model f is represented by a matrix denoted by $\tilde{\boldsymbol{G}}_{\boldsymbol{x}}$ in the standard coordinates of $\mathbb{R}^d$. Following [1], this matrix is called the *local data matrix*. The pullback operator can be seen as "retropropagating" the geometry from the output space $\mathcal{S}$ to the input space $\mathbb{R}^d$. The local data matrix can be written explicitly as:

$$\tilde{\boldsymbol{G}}_{\boldsymbol{x}} = \mathbb{E}_{\boldsymbol{y}|f(\boldsymbol{x})}\left[\nabla_{\boldsymbol{x}} \log p(\boldsymbol{y}|f(\boldsymbol{x})) \left(\nabla_{\boldsymbol{x}} \log p(\boldsymbol{y}|f(\boldsymbol{x}))\right)^\top\right]. \tag{4.1}$$

It can also be formulated as a function of the Jacobian matrix $\boldsymbol{J}_{\boldsymbol{x}}$ of f and of the FIM $\boldsymbol{G}$:

$$\tilde{\boldsymbol{G}}_{\boldsymbol{x}} = \boldsymbol{J}_{\boldsymbol{x}}^\top \boldsymbol{G}_{f(\boldsymbol{x})} \boldsymbol{J}_{\boldsymbol{x}}.$$

According to equation 2.2, if one moves along a small enough perturbation in one of the directions of $\ker \tilde{\boldsymbol{G}}_{\boldsymbol{x}}$ in the input space, then the predictive distribution $p(\boldsymbol{y}|\boldsymbol{\theta})$ is constant. For these directions to exist, a sufficient condition is that the dimension of the input space d is larger than the dimension of the output space $c - 1$, which is typically verified in supervised learning. According to [1], the directions of $\ker \tilde{\boldsymbol{G}}_{\boldsymbol{x}}$ are the *noise directions*, i.e., directions in which the inputs get more and more noisy while the network still predict the same output with the same

confidence. On the other hand, moving along directions in $\left(\ker \tilde{\boldsymbol{G}}_{\boldsymbol{x}}\right)^{\perp}$ changes the predictive distribution $p(y|\boldsymbol{\theta})$. In this case, the model will predict different outputs with high confidence. These are the privileged directions for adversarial attacks.

Assume that $\tilde{\boldsymbol{G}}_{\boldsymbol{x}}$ has constant rank with respect to $\boldsymbol{x}$. This can be verified experimentally. A *distribution* D can be defined by

$$\boldsymbol{x} \in \mathbb{R}^d \mapsto D_{\boldsymbol{x}} = \ker \tilde{\boldsymbol{G}}_{\boldsymbol{x}} \subset \mathbb{R}^d$$

This object was already introduced in section 3.2.1, but more explanations are provided now. A distribution consists in associating to each point $\boldsymbol{x}$ of a manifold $\mathcal{M}$ a subspace of the tangent space $T_{\boldsymbol{x}}\mathcal{M}$ in a smooth way. Moreover, the corresponding subspaces are required to have constant dimension. Here, it is assumed that, for all $\boldsymbol{x}$, one has $\dim\left(\ker \tilde{\boldsymbol{G}}_{\boldsymbol{x}}\right) = r$ for some fixed constant $r > 0$. For each point $\boldsymbol{x} \in \mathbb{R}^d$, the objective is to find a submanifold $N_{\boldsymbol{x}} \subset \mathbb{R}^d$ such that $\boldsymbol{x} \in N_{\boldsymbol{x}}$ and for any $\boldsymbol{x}' \in N_{\boldsymbol{x}}$ the equality $T_{\boldsymbol{x}'}N_{\boldsymbol{x}} = D_{\boldsymbol{x}'}$ holds. If such $N_{\boldsymbol{x}}$ uniquely exists for all $\boldsymbol{x} \in \mathbb{R}^d$, then the distribution D is said to be *integrable*. The submanifold $N_{\boldsymbol{x}}$ is called the *leaf* of $\boldsymbol{x}$ and the set of all leaves for all $\boldsymbol{x} \in \mathbb{R}^d$ is called a *foliation*. The entire manifold $\mathbb{R}^d$ is thus the disjoint union of all the leaves of the foliation. Since the distribution D is defined by the kernel of the local data matrix, it will be called the *kernel distribution*. Its associated foliation is the *kernel foliation*, and its leaves are the *kernel leaves*. Moving along a kernel leaf means moving along a kernel direction at each point. Hence, the output distribution does not change along a kernel leaf.

A necessary and sufficient condition for a distribution to be integrable is given by Frobenius Theorem. First, the notion of *involutive* distribution is defined.

Definition 4.1.1 (Involutive property)**.** Let D be a distribution and let X and Y be smooth vector fields belonging to D (i.e., for all $\boldsymbol{x}$, $X_{\boldsymbol{x}} \in D_{\boldsymbol{x}}$ and $Y_{\boldsymbol{x}} \in D_{\boldsymbol{x}}$). Define the Lie bracket of vector fields by $[X, Y]_{\boldsymbol{x}} = X_{\boldsymbol{x}}Y - Y_{\boldsymbol{x}}X$.
Then, D is said to be involutive if $[X, Y] \in D$.

Then, Frobenius Theorem can be stated.

Theorem 4.1.2 (Frobenius)**.** *A distribution D is integrable if and only if it is involutive.*

There is no reason for an arbitrary neural network to induce an involutive kernel distribution. In [1], the authors show that for a classifier using ReLU activation for the hidden layers and softmax for the last layer, the kernel distribution is involutive (assuming the local data matrix has constant rank). This result comes from the fact that such a network is piecewise linear. However, the authors also claim that it can be shown experimentally that the kernel distribution induced by the local Fisher matrix (obtained by differentiating with respect to the weight $\boldsymbol{w}$ instead of the data point $\boldsymbol{x}$) is not involutive. Then, the authors focus on the *transverse foliation* of the kernel foliation which will be called the *data foliation* defined by $\boldsymbol{x} \mapsto \left(\ker \tilde{\boldsymbol{G}}_{\boldsymbol{x}}\right)^{\perp}$. They show experimentally that all the training dataset lie on a unique leaf of the image foliation. This claim will be examined in section 4.3 by conducting several experiments. The advantage of the image leaves compared to the kernel leaves is that it is possible to use $\tilde{\boldsymbol{G}}_{\boldsymbol{x}}$ to define a Riemannian metric on each image leaf, since by definition, the local data matrix is non-degenerate on $\left(\ker \tilde{\boldsymbol{G}}_{\boldsymbol{x}}\right)^{\perp}$.

4.2 Robust Time Series Prediction

In this section, a framework is described for the prediction of time series with sequential models, and more particularly with Recurrent Neural Networks (RNN). RNN are applied in chapter 5 for

the prediction of airspace congestion. The reader can refer to section 5.4.1 for a more empirical presentation of RNN.

Let S be an open set of $\mathbb{R}^d$ called the *state space*. Let $\phi : S \to S$ be a diffeomorphism of S. The map ϕ is seen as a dynamical system such that $\boldsymbol{s}_{t+1} = \phi(\boldsymbol{s}_t)$. Let $F : S \to \mathbb{R}$ be a smooth function called the *measurement function*. Let $m \geq 2d + 1$. Consider the map $\Psi : \boldsymbol{s} \in S \mapsto (F(\boldsymbol{s}), F(\phi(\boldsymbol{s})), \dots, F(\phi^{m-1}(\boldsymbol{s}))) \in \mathbb{R}^m$. Then, the following result is true [138]:
Theorem 4.2.1 (Takens)**.** *For generic ϕ and F, Ψ is an embedding.*
In other words, there exist a subset $\tilde{S} \subset \mathbb{R}^m$ and a dynamical system $\tilde{\phi} : \tilde{S} \to \tilde{S}$ which is conjugated with ϕ, i.e., $\phi = \Psi^{-1} \circ \tilde{\phi} \circ \Psi$ for some diffeomorphism $\Psi : S \to \tilde{S}$.

The input space $\tilde{S} \subset \mathbb{R}^m$ can be seen as a set of time series of length m. Given a time series $\boldsymbol{x} \in \tilde{S} \subset \mathbb{R}^m$, the model f predicts a time series $\boldsymbol{y} \in \mathbb{R}^c$ of length c. The simplest example is to use $\boldsymbol{x} = (F(\boldsymbol{s}), F(\phi(\boldsymbol{s})), \dots, F(\phi^{m-1}(\boldsymbol{s})))$ to predict the next measurement $\boldsymbol{y} = F(\phi^m(\boldsymbol{s}))$ (here $c = 1$) or a sequence of next measurements $\boldsymbol{y} = (F(\phi^m(\boldsymbol{s})), F(\phi^{m+1}(\boldsymbol{s})), \dots, F(\phi^{m+c-1}(\boldsymbol{s})))$ for arbitrary c. In both examples, the aim of the model is to learn the conjugate dynamic $\tilde{\phi}$. In the general case, it is assumed that there is a smooth function $g : \mathbb{R}^m \to \mathbb{R}$ such that $\boldsymbol{y}^i = g(\tilde{\phi}^i(\boldsymbol{x}))$ for $1 \leq i \leq c$. In the examples above, g was the projection to the last component. A model whose objective is to learn this mapping, i.e., $f(\boldsymbol{x}) \approx (g(\tilde{\phi}(\boldsymbol{x})), \dots, g(\tilde{\phi}^c(\boldsymbol{x})))$, is called a *sequential model*.

The goal is to study the kernel foliation of a sequential model f in order to guarantee the robustness of such models to noise, adversarial attacks, and out-of-distribution samples. To apply this framework, $\boldsymbol{y} \in \mathbb{R}^c$ is seen as a random variable whose distribution is parameterized by the model output $f(\boldsymbol{x}) \in \mathbb{R}^q$. Hence, f does not infer $\boldsymbol{y}$ directly, but only the parameter of the distribution of $\boldsymbol{y}$. In chapter 3, the model output was the parameter of a categorical distribution. In this section, the distribution of $\boldsymbol{y}$ is assumed to belong to the family $\mathcal{H}_c$ of c-dimensional normal distributions, hence $q = c + \frac{c(c+1)}{2} = \frac{c(c+3)}{2}$. A sufficient condition for the existence of the kernel foliation is $m > q$, i.e., $c < (\sqrt{9 + 8m} - 3)/2$ and this condition is assumed to be met in the following. Once equipped with the Fisher information metric $\boldsymbol{G}$, $\mathcal{H}_c$ becomes a q-dimensional Riemannian manifold.

4.2.1 A Model for Recurrent Neural Networks

A recurrent neural network is a function $f(x, \boldsymbol{h})$ where $x \in \mathbb{R}$ is the input of the network and $\boldsymbol{h} \in \mathbb{R}^m$ is the state of the network. The network's prediction also depends on the learnable parameters, but in what follows, these parameters are assumed to be fixed. In other words, the goal is to study an already trained network. To simplify the analysis, it is also assumed that the input x is univariate.

At iteration k, the network produces an output $\hat{y}_k = f(x_k, \boldsymbol{h}_{k-1}) \in \mathbb{R}$. Define the recurrence relation $\boldsymbol{h}_k = g\left(\hat{y}_{k-m}, \hat{y}_{k-(m-1)} \dots, \hat{y}_{k-2}, \hat{y}_{k-1}\right)$ for some function g. In the following, the analysis is limited to the special case where g is such that: $\boldsymbol{h}_k = \left(\hat{y}_{k-(m-1)} \dots, \hat{y}_{k-1}, \hat{y}_k\right)$, which is simply the input sequence shifted by one timestep.

Let X be a dynamical system living on a smooth manifold S of dimension d that is assumed to be unobservable. Let F be a measurement function. If X is in some state $\boldsymbol{s} \in S$, then the associated measurement is $y = F(\boldsymbol{s})$. For example, when studying the dynamics of an aircraft, the state space is $S \subseteq \mathbb{R}^{12}$ since the state space of the complete dynamical model of an aircraft has six degrees of freedom (three positions and three angles) along with their time derivatives. In this example, $y = F(\boldsymbol{s})$ could be a measure of the altitude. In practice, the inputs are noisy and the input is defined as $x = y + w$ with some noise w. In the absence of noise, Takens'

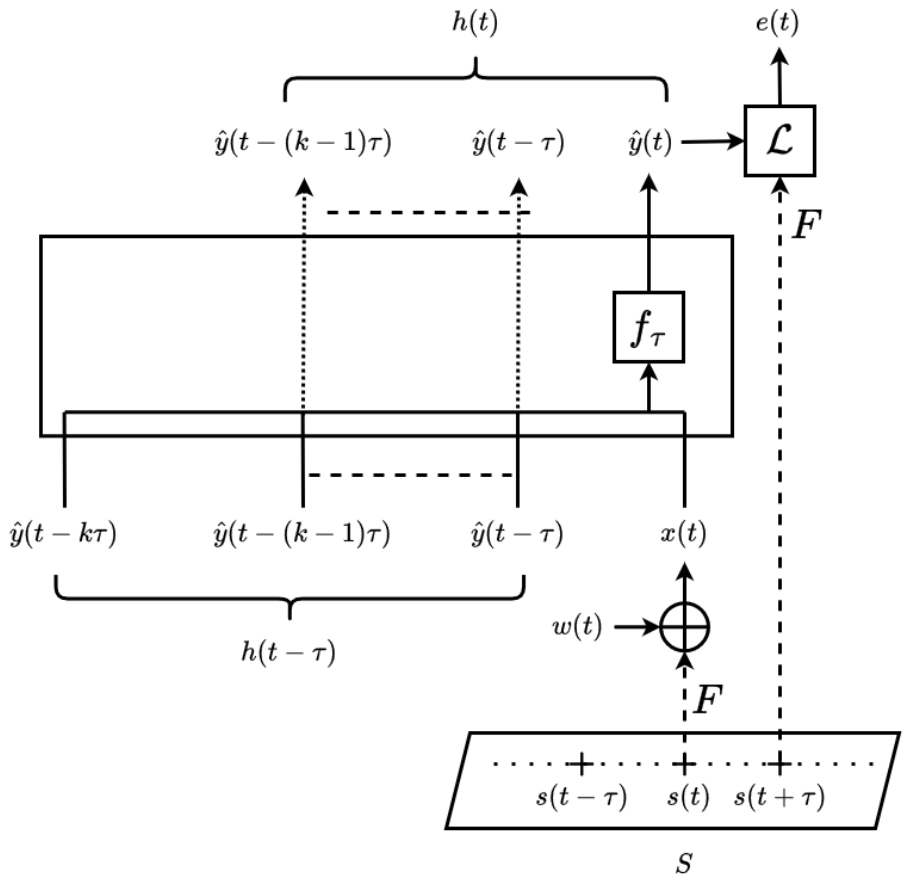


Figure 4.1: The objective is to learn the dynamic of a system living on a manifold S, given a measure function F. However, a random noise $w(t)$ is added to the measures. This noise can be a Gaussian noise or an adversarial attack. Thus, $x(t) = F(\boldsymbol{s}(t)) + w(t)$. The prediction of the model is $\hat{y}(t) = f_\tau(x(t), \boldsymbol{h}(t-\tau))$ where τ is a time step. When $\tau \longrightarrow 0$, the difference equation on $\boldsymbol{h}$ can be converted into a differential equation. A loss function is used to train the network by computing the error $e(t) = \mathcal{L}(\hat{y}(t), F(\boldsymbol{s}(t+\tau))$.

theorem (Theorem 4.2.1) states that the network is able to predict $y_{k+1} = F(\boldsymbol{s}_{k+1})$ from x_k and $\boldsymbol{h}_{k-1}$ if $m \geq 2d$. In the presence of noise, the objective is to know if the network will be able to produce a series of correct predictions or if it will be drowned in the noise of the input, for example if the variance of the predictions diverges. Moreover, the goal would be to quantify this level of robustness against noise and to know how this level of robustness varies as the initial conditions over $\boldsymbol{h}$ vary, or as the noiseless inputs vary. To address these questions, the recurrent neural network is modeled as stochastic differential equation. The initial condition is a random variable. Moreover, at each instant t, the network receives in input a random variable whose law is known, e.g., a Wiener process (to model a purely random noise). Then, the curve of the parameters of the output process can be studied. The output space is endowed with the Fisher information metric $\boldsymbol{g}$. To do so, the network is assumed to define a Gaussian law in each point of the output space. The general model is illustrated in Figure 4.1.

The distribution of $\hat{y}$ depends on f which is generally non-linear. In this section, a simpler model is studied for f. To approximate the true distribution of $\hat{y}$, the idea is to see $\hat{y}$ as a sequence of samples of a stochastic process which is the solution of a linear stochastic differential equation.

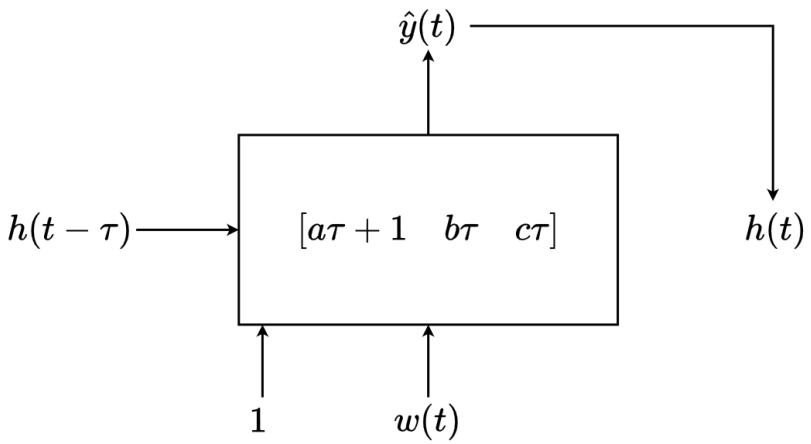


Figure 4.2: Simplified model. It is assumed that $m = 1$ hence, $\boldsymbol{h}(t) = x(t-1)$. Moreover, one has $\boldsymbol{h}(t) = f_\tau(w(t), \boldsymbol{h}(t-\tau)) = (a\tau+1)\boldsymbol{h}(t-\tau) + b\tau + c\tau w(t)$. A constant 1 is added in input to include the bias in the matrix of f_τ.

This can be interpreted as a “linearization" of the model. Thus, the following assumptions are made:

- The continuous case is considered i.e., $\boldsymbol{h}$ follows the differential equation $\dot{\boldsymbol{h}}(t) = f(x(t), \boldsymbol{h}(t-1))$ instead of the difference equation $\boldsymbol{h}_k - \boldsymbol{h}_{k-1} = f(x_k, \boldsymbol{h}_{k-1}) - \boldsymbol{h}_{k-1}$;
- f is purely recurrent i.e., the external input $x(t)$ is a Gaussian white noise ($x(t)dt = dw(t)$);
- $\boldsymbol{h} \in \mathbb{R}$ i.e., $m = 1$. Thus, $\boldsymbol{h}(t) = \hat{y}(t)$;
- f is linear.

The simplified model is shown in Figure 4.2.

In order to study the robustness of the network, $\boldsymbol{h}$ is assumed to be a stochastic process solution of the following differential equation:

$$d\boldsymbol{h}(t) = (a.\boldsymbol{h}(t) + b)dt + cdw(t),$$

where w is a standard Wiener process (i.e., Brownian motion). For a given t, one has $w(t) \sim \mathcal{N}(0, t^2)$.

It can be shown that $\boldsymbol{h}(t) \sim \mathcal{N}(\mu(t), \sigma^2(t))$. The objective of this section is to study the properties of the curves of $\boldsymbol{h}$ in the space (μ, σ^2) as a function of the initial conditions $\boldsymbol{h}_0 = (\mu(0), \sigma^2(0))$. Since $\boldsymbol{h}$ follows a Gaussian law, the space (μ, σ^2) is the Poincaré half-plane $\mathcal{H}_1$, which is a hyperbolic space. Since f is linear, the curves can be described explicitly. Two kinds of behaviors are expected depending on the eigenvalues of f. If f is a contraction mapping (non-positive eigenvalue), the curves will exponentially converge to a limit value σ^* proportional to the variance of $\boldsymbol{h}$. This means that the preceding noise has been attenuated and only the current noise affects the variance of $\boldsymbol{h}$. If f has a positive eigenvalue, the curves will diverge exponentially along σ. It is then impossible to make prediction on the value of $\boldsymbol{h}$ since its standard deviation diverges.

In the simplified case treated here, the input space has same dimension as the output space (since both spaces are identical). Hence, the pullback metric is not degenerate and there is no foliation in the input space.

4.2.2 Stochastic Differential Equations

Consider the following stochastic differential equation (SDE) scalar, linear, autonomous, and with additive noise:

$$d\boldsymbol{h}(t) = (a.\boldsymbol{h}(t) + b)dt + c.dw(t), \tag{4.2}$$
$$\boldsymbol{h}(0) = \mu_0 \text{ p.s.} \tag{4.3}$$

where $\boldsymbol{h}$ is a stochastic process, w is a standard Wiener process, and $a, b, c, \mu_0 \in \mathbb{R}$. Assume that $a \neq 0$. The SDE is a Langevin equation:

$$d\boldsymbol{h}(t) = K(\alpha - \boldsymbol{h}(t))dt + \beta dw(t),$$

with $K = -a, \alpha = -\frac{b}{a}, \beta = c$. Its solution is an Ornstein-Uhlenbeck (OU) process [139]:

$$\begin{aligned}\mu(t) &= \alpha + (\mu_0 - \alpha)e^{-Kt}, \\ &= -\frac{b}{a} + \left(\mu_0 + \frac{b}{a}\right)e^{at}, \\ \sigma^2(t) &= \frac{\beta^2}{2K}(1 - e^{-2Kt}), \\ &= \frac{c^2}{2a}(e^{2at} - 1).\end{aligned}$$

If $a < 0$, then $\sigma^2(t) \longrightarrow -\frac{c^2}{2a}$ and $\mu(t) \longrightarrow -\frac{b}{a}$.
If $a > 0$, then $\sigma^2(t) \longrightarrow +\infty$ and:

- $\mu(t) \longrightarrow +\infty$ if $\mu_0 > -\frac{b}{a}$.
- $\mu(t) \longrightarrow -\infty$ if $\mu_0 < -\frac{b}{a}$.
- $\mu(t) \longrightarrow -\frac{b}{a}$ if $\mu_0 = -\frac{b}{a}$.

Let $\boldsymbol{g}$ be the Fisher information metric associated to the family of univariate normal laws parameterized by their expectation μ and their variance σ^2. Consider the parameter space $(\mathbb{R} \times \mathbb{R}_+^*, \boldsymbol{g})$ and the global coordinate system $(\mathbb{R} \times \mathbb{R}_+^*, (\mu, \sigma^2))$. Let $\boldsymbol{\theta}(t) = (\mu(t), \sigma^2(t))$. The goal is to compute the length of the curves $L(t_0, t) = \int_{t_0}^{t} |\dot{\boldsymbol{\theta}}(s)| ds$ for all $t > t_0 > 0$ as well as the curvature of the curves. Using the coordinates $\boldsymbol{\theta} = (\mu, \sigma^2)$, one has:

$$p(y|\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right).$$

It can be shown[68] that:

$$\boldsymbol{G}_{\boldsymbol{\theta}} = \begin{pmatrix} \frac{1}{\sigma^2} & 0 \\ 0 & \frac{1}{2\sigma^4} \end{pmatrix}.$$

The corresponding metric tensor field is:

$$\boldsymbol{g}_{\boldsymbol{\theta}} = \frac{1}{\sigma^2}(d\mu)^2 + \frac{1}{2\sigma^4}(d\sigma^2)^2.$$

One has $\dot{\boldsymbol{\theta}}(t) = \begin{pmatrix} \dot{\mu}(t) & \dot{\sigma^2}(t) \end{pmatrix}$ with:

$$\begin{aligned}\dot{\mu}(t) &= (a\mu_0 + b)e^{at}, \\ \dot{\sigma^2}(t) &= c^2 e^{2at}.\end{aligned}$$

Assume that $c \neq 0$ (since the Fisher information is not defined if $\sigma^2 = 0$). The norm of the velocity is:

$$|\dot{\boldsymbol{\theta}}(t)| = \frac{\sqrt{2a((a\mu_0 + b)^2(e^{2at} - 1) + ac^2e^{2at})}}{|c||1 - e^{2at}|}e^{at},$$

hence,

$$L(t_0, t) = \int_{t_0}^{t} \frac{\sqrt{2a((a\mu_0 + b)^2(e^{2as} - 1) + ac^2e^{2as})}}{|c||1 - e^{2as}|}e^{as}ds.$$

Notice that the integration starts at $t_0 > 0$. Indeed, at $t = 0$, $\boldsymbol{\theta}(0)$ is almost surely constant. Thus, the Fisher information is not defined (and the velocity diverges).
The geodesic curvature of a curve $\boldsymbol{\theta}(t)$ is [82]:

$$\kappa(t) = \frac{\sqrt{|\dot{\boldsymbol{\theta}}(t)|^2|D_t\dot{\boldsymbol{\theta}}(t)|^2 - \langle D_t\dot{\boldsymbol{\theta}}(t), \dot{\boldsymbol{\theta}}(t)\rangle^2}}{|\dot{\boldsymbol{\theta}}(t)|^3}$$

The covariant derivative of $\dot{\boldsymbol{\theta}}(t)$ along $\boldsymbol{\theta}(t)$ has the following expression in coordinates:

$$D_t\dot{\boldsymbol{\theta}}(t) = (\ddot{\boldsymbol{\theta}}^k(t) + \dot{\boldsymbol{\theta}}^i(t)\dot{\boldsymbol{\theta}}^j(t)\Gamma_{ij}^k(\boldsymbol{\theta}(t)))\partial_k$$

The Christoffel symbols have the following expression:

$$\Gamma_{ij}^k = \frac{1}{2}\boldsymbol{g}^{kl}(\partial_i\boldsymbol{g}_{jl} + \partial_j\boldsymbol{g}_{il} - \partial_l\boldsymbol{g}_{ij})$$

After computation, one obtains $\Gamma_{11}^1 = \Gamma_{22}^1 = \Gamma_{12}^2 = \Gamma_{21}^2 = 0$, $\Gamma_{11}^2 = 1$, $\Gamma_{22}^2 = -\frac{1}{\sigma^2}$, and $\Gamma_{12}^1 = \Gamma_{21}^1 = -\frac{1}{2\sigma^2}$, hence :

$$D_t\dot{\boldsymbol{\theta}}(t) = -a(a\mu_0 + b)e^{at}\frac{e^{2at} + 1}{e^{2at} - 1}\partial_\mu + e^{2at}\left((a\mu_0 + b)^2 - \frac{2ac^2}{e^{2at} - 1}\right)\partial_{\sigma^2}$$

The norm of the covariant derivative is:

$$|D_t\dot{\boldsymbol{\theta}}(t)| = \frac{\sqrt{2}|a|e^{at}}{c^2(e^{2at} - 1)^2}\sqrt{ac^2(a\mu_0 + b)^2(e^{2at} + 1)^2(e^{2at} - 1) + e^{2at}\left((a\mu_0 + b)^2(e^{2at} - 1) - 2ac^2\right)^2}.$$

Furthermore, one has:

$$\langle D_t\dot{\boldsymbol{\theta}}(t), \dot{\boldsymbol{\theta}}(t)\rangle = -\frac{2a^2e^{2at}}{c^2(e^{2at} - 1)^2}\left((a\mu_0 + b)^2 + \frac{2ac^2e^{2at}}{e^{2at} - 1}\right).$$

The following expression for the curvature is obtained[1]:

$$\kappa(t) = |e^{2at} - 1||a\mu_0 + b|\sqrt{\frac{a(e^{2at} - 1)[(a\mu_0 + b)^2 + c^2a]^2}{2[a(e^{2at} - 1)(a\mu_0 + b)^2 + c^2a^2e^{2at}]^3}}$$

Thus, one has $\kappa(t) \geq 0$ (with $a \neq 0$ and $c \neq 0$):

- If $a < 0$ and $a\mu_0 + b \neq 0$, the following holds :

$$\lim_{t \to +\infty} \kappa(t) = \frac{|(a\mu_0 + b)^2 + c^2a|}{\sqrt{2}|a||a\mu_0 + b|^2}.$$

[1]Notice that $a(e^{2at} - 1) > 0$ for all $a \neq 0$ and $t > 0$.

- If $a > 0$, the following holds :

$$\lim_{t \to +\infty} \kappa(t) = \frac{|a\mu_0 + b|}{a\sqrt{2((a\mu_0 + b)^2 + c^2 a)}}.$$

- If $a\theta_0 + b = 0$ then $\kappa(t) = 0$ for all t.

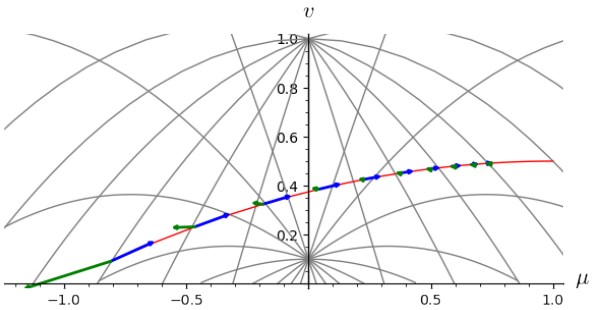


Figure 4.3: An example of a curve $\boldsymbol{\theta}(t) = (\mu(t), \sigma^2(t))$ solution of equation 4.2 in the Poincaré half-plane $\mathcal{H}_1$ (in red). The velocity tensor field $\dot{\boldsymbol{\theta}}(t)$ is represented with blue arrows. The covariant derivative $D_t\dot{\boldsymbol{\theta}}(t)$ is represented with green arrows. The parameters are $a = -1, b = 1, c = 1$ and $\mu_0 = -1$. The curve is plotted between $t = 0.01$ and $t = 2$. Some geodesics of $\mathcal{H}_1$ are plotted for better visualization (in gray). The Y-axis corresponds to the variance $v = \sigma^2$.

In future work, the assumption $m = 1$ can be relaxed. The case $\boldsymbol{h}(t) \in \mathbb{R}^m$ where m is arbitrary will be considered. It will no longer be possible to compute lengths and curvature explicitly. The goal will then be to bound the length and the curvature of the curve of the output process using known linear processes. A common method is to assume that the network is Lipschitz. Under this assumption, it is possible to bound the output of the network by two linear processes.

Another direction is to take into account the correlation between consecutive time steps of the input process $x(t)$. To do this, the curves of a network taking as input a sequence of consecutive time steps (as in Takens' theorem) have to be considered. In this setting, the dimension of the input space is strictly greater than the dimension of the output space, the pullback metric is then degenerated and the kernel and image foliations appear. The objective is then to study the interaction of these foliations with the curve $\boldsymbol{\theta}(t)$. By bounding the output process with known processes, it may be possible to quantify the robustness of the network to noise. In particular, it may be possible to show that the standard deviation does not diverge in finite time. The noise can come from a characteristic of the dynamical system that was not included in the model, or from an unknown variable (e.g., the wind). It may also come from measurement errors of transmission errors. It can also be intentionally injected (e.g., adversarial attack). If the standard deviation is asymptotically bounded then the network is robust: despite the noise, the observed output will always be sufficiently correlated with the expected output. In the next section, some visualizations of the kernel foliation are provided.

4.2.3 Visualizing the Kernel Foliation

To get insights into the behavior of the kernel foliation, this subsection will develop visualizations using a low-dimensional problem. In order to obtain a visualization that is can be easily interpreted, the dimension of the input space must be at most 3. Since Takens' theorem (theorem 4.2.1) guarantees the existence of an embedding of the state space with at least $2m + 1$ observations, the state space must have dimension $d = 1$.

4.2.3.1 A simple one-dimensional dynamical system

The goal is to find a one-dimensional dynamical system with interesting behaviors to be learn by a neural network. The following system is chosen for the experiments of this subsection:

$$\frac{dy}{dt} = \sin(\pi y)$$

This is a non-linear system whose equilibria are the integers. The odd integers are stable equilibria while the even integers are unstable equilibria. A time step $\delta t = 2.10^{-2}$ is chosen and a dataset is created in the following way for a supervised time series prediction:

- Each input is a vector of three successive points $(y(t), y(t+\delta t), y(t+2\delta t))$;
- The corresponding true output is the next point $y(t+3\delta t)$;
- No noise is added for this experiment.

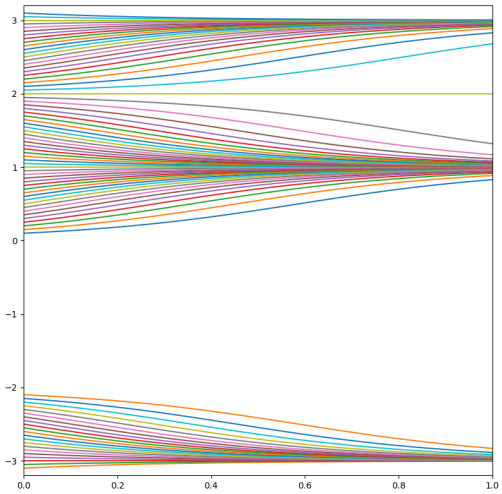


Figure 4.4: Dataset consisting of trajectories of a simple dynamical system (time t on the X-axis, $y(t)$ on the Y-axis).

Figure 4.4 shows the trajectories used to build the dataset. The dataset consists of 3,936 samples. The initial points are taken between -3 and 3. There is a gap in the data between -2 and 0 in order to study how the model will react to a lack of data. In particular, one of the objective is to know if the model is able to generalize and to infer the existence of the stable equilibria at -1. Another objective is to check if the model displays a high uncertainty when performing predictions in the interval $[-2, 0]$, and if the kernel foliation reflects the lack of data in this interval.

4.2.3.2 A simple neural network

Since this problem is very easy, this experiment uses a small recurrent neural network. The 3-dimensional input vectors are fed component by component to a LSTM layer (see section 5.4.1.2 for more details) with hidden dimension equal to 8. The hidden state of the last LSTM is fed to a fully connected layer that outputs a 2-dimensional vector:

- The first component of this 2-dimensional output vector is $\hat{y}$, the prediction for the true output $y = y(t+3\delta t)$;
- The second component is the standard deviation $\sigma > 0$.

To train this network, the negative log-likelihood of the univariate normal distribution is used as a loss function:

$$\mathcal{L}(\hat{y}, \sigma, y) = \log(\sigma) + \frac{(y-\hat{y})^2}{2\sigma^2},$$

The constant term $\frac{1}{2}\log 2\pi$ has been discarded.

The model is trained over 500 epochs with the Adam algorithm [140] (learning rate $= 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$). To evaluate the model, three test datasets are used:

- A validation dataset built from 51 trajectories whose initial points are uniformly sampled from the interval $[-3, 3]$;
- An in-distribution dataset built from 52 trajectories whose initial points are uniformly sampled from $[-3, -2] \cup [0, 3]$;
- An out-of-distribution dataset built from 31 trajectories whose initial points are uniformly sampled from $[-2, 0]$.

Two evaluation metrics are used:

- The relative error $r(\hat{y}, y) = \frac{|\hat{y}-y|}{y}$. The error range is $[0, 1]$ and smaller error is better;
- The likelihood defined as the probability of sampling from $\mathcal{N}(\hat{y}, \sigma^2)$ a point that is at least as far from the mean $\hat{y}$ as the true output y. The likelihood range is $[0, 1]$ and higher likelihood is better.

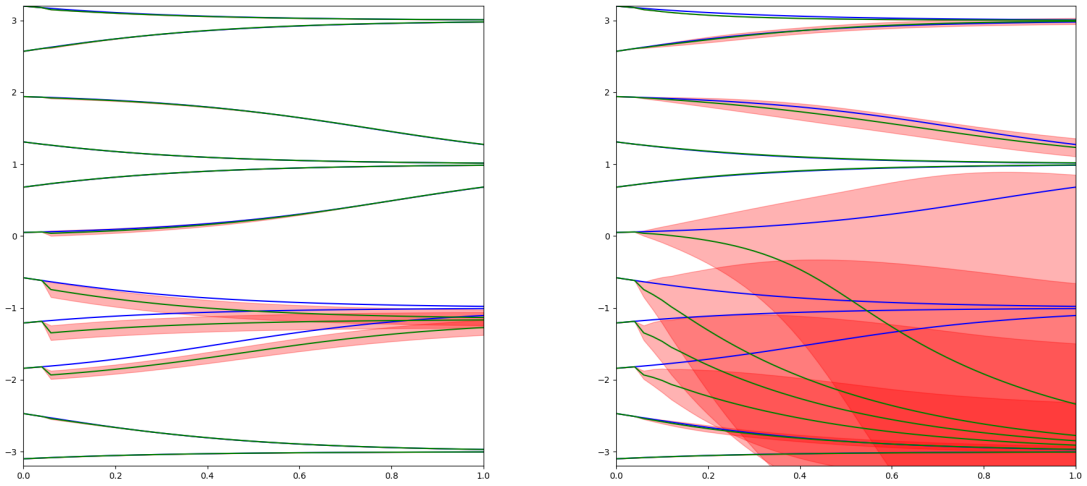
The metrics are averaged over each test datasets. The results are reported in Table 4.1 using percentages for better readability.

	Training	**Validation**	**In-distribution**	**Out-of-distribution**
Relative error	0.18	5.72	0.21	15.64
Likelihood	7.29	11.44	6.32	17.87

Table 4.1: Model evaluation over the training set and four test datasets. The relative error measures the accuracy of the model, while the likelihood measures the quality of the uncertainty quantification of the model. Both metrics are reported in %.

The model has very low relative error and low likelihood on the training and in-distribution datasets. This means that the predictions of the model are very close to the true outputs, but the predicted standard deviations are very small such that the model is overconfident even with respect to its good predictions. On the out-of-distribution dataset, a higher relative error is observed, associated with a higher likelihood. That means that the model predicts high standard deviations when confronted to out-of-distribution data such that, even if its predictions are poor, its level of confidence is correspondingly low.

To illustrate these remarks, Figure 4.5a shows the predicted trajectories and standard deviations along with the true trajectories for several initial points. The trajectories are predicted in a non-recursive mode, i.e., the model uses the last three points of the true trajectory for its next prediction, disregarding its own predictions for these three points. In Figure 4.5a, the error is higher on the out-of-distribution range $[-2, 0]$, but that the standard deviation is also higher in this range. Figure 4.5b shows the same initial points but the predicted trajectories are obtained in a recursive mode, which means that the model uses its own predictions as inputs for the next ones. In order to account for the propagation of error, the standard deviations are accumulated over time (contrary to the non-recursive mode where the standard deviation is displayed by assuming that the error in the inputs is zero). In Figure 4.5b, the model is not able to infer the existence of the stable equilibrium at -1. Instead, the trajectories starting in $[-2, 0]$ converge to the equilibrium at -3. This illustrates the inability of neural networks to infer even simple rules, and shows that these models are reliant on the representativeness of the data to achieve decent generalization performances.



(a) Using non-recursive prediction.

(b) Using recursive prediction.
The standard deviations are accumulated.

Figure 4.5: True trajectories (blue), predicted trajectories (green), and standard deviations (red).

4.2.3.3 The kernel foliation

This paragraph will focus on the input space $\mathbb{R}^3$. The state space $\mathbb{R}$ can be embedded in $\mathbb{R}^3$ using the map $y \mapsto (y(0), y(\delta t), y(2\delta t))$, resulting in the one-dimensional submanifold depicted in Figure 4.6a. The equilibria of the dynamical system correspond to the points of the diagonal X=Y=Z with integer coordinates. Since, the coordinates of $(y(0), y(\delta t), y(2\delta t))$ are close to each other, all the points of the embedded state space are close to the diagonal. To improve the visualization, the input space $\mathbb{R}^3$ is rotated such that the X-axis is sent to the diagonal $X = Y = Z$, and the space is scaled by $1/\sqrt{3}$ such that the X-axis keeps the same range. The result is shown in Figure 4.6b where the various equilibria corresponding to points $(y, 0, 0)$ can be seen.

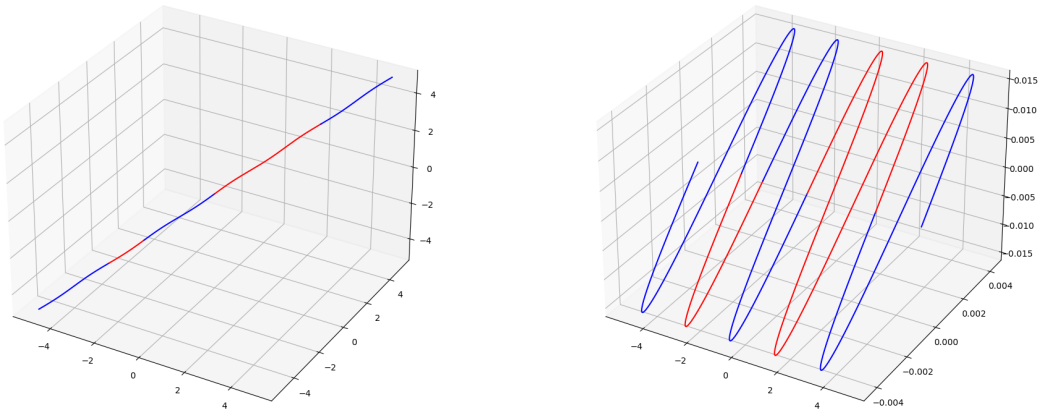
The (mean, standard deviation) coordinates are used on the output space of univariate normal distribution, while the canonical coordinates of $\mathbb{R}^3$ are used on the input space. Given a point $\boldsymbol{\theta} = (\hat{y}, \sigma)$ of the output space, the corresponding Fisher information matrix is

$$\boldsymbol{G}_{\boldsymbol{\theta}} = \begin{pmatrix} \frac{1}{\sigma^2} & 0 \\ 0 & \frac{2}{\sigma^2} \end{pmatrix}$$

Let $\boldsymbol{x} \in \mathbb{R}^3$ be a point of the input space. Let $\boldsymbol{J}(\boldsymbol{x})$ be the Jacobian matrix at $\boldsymbol{x}$ between the input space and the output space in the chosen coordinates. Denote the model by f such that $(\hat{y}, \sigma) = f(\boldsymbol{x})$. Then, the pullback metric on the input space is

$$\tilde{\boldsymbol{G}}_{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{G}_{f(\boldsymbol{x})} \boldsymbol{J}(\boldsymbol{x})$$

It is assumed that $\tilde{\boldsymbol{G}}$ has constant rank equal to 2. This is confirmed experimentally by sampling points in $\mathbb{R}^3$ and computing the rank of $\tilde{\boldsymbol{G}}$ at these points. No point was found with a rank smaller than 2. Hence, the kernel of $\tilde{\boldsymbol{G}}$ has constant rank equal to 1. It defines a distribution of dimension



(a) Using the canonical coordinates.

(b) After a rotation and a scaling. The Y- and Z-axis ranges have been modified.

Figure 4.6: Embedding of the state space in the input space. The red parts correspond to the training data while the blue parts were not seen by the model during training.

1 over $\mathbb{R}^3$. Since a one-dimensional distribution is always integrable, this guarantees the existence of the kernel foliation. The only relevant points are the ones that are on the embedded state space or close to it. Thus, several points are chosen along the embedded submanifold and the kernel leaf containing each point is computed. The result is shown in Figure 4.7 with two points of view.

Several remarks can be made:

- The kernel leaves are compact.
- The interval $[-2, 0]$ which was outside the distribution of training data does not reveals any abnormal structure in the leaves (at least qualitatively) compared with the rest of the foliation.
- The kernel foliation exhibits a symmetry around the image of 0 in the embedded manifold.
- Some leaves exhibit a chaotic behavior. It is not known if it is an artifact of the integration method used to compute the leaves, or if it is a property of the leaves themselves.

Future work could explore more formally the properties of the kernel leaves and how these properties can be linked to the behavior of the recurrent neural network.

4.3 Foliations

In this section, several experiments are performed around the data leaf hypothesis introduced in [1].

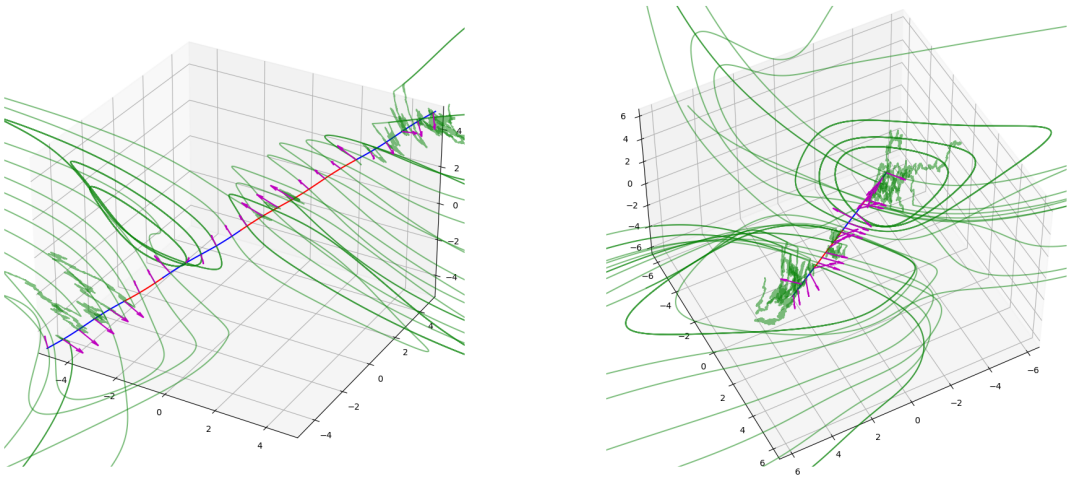


Figure 4.7: Kernel leaves (in green) of points belonging to the embedded state space, viewed from two different angles. The magenta arrows are basis of the kernel at each of these points.

4.3.1 Adversarial Attacks and Leaves

The behavior of the FGSM attack with respect to the data leaf is investigated[2]. In particular, the goal is to determine whether the adversarial examples lie on the data leaf or are far from it, and whether they are orthogonal to the data leaf.

4.3.1.1 Experiment setup

The MNIST dataset is used with LeNet[3]. The model is trained with SGD, batch size 64, and learning rate 0.01.

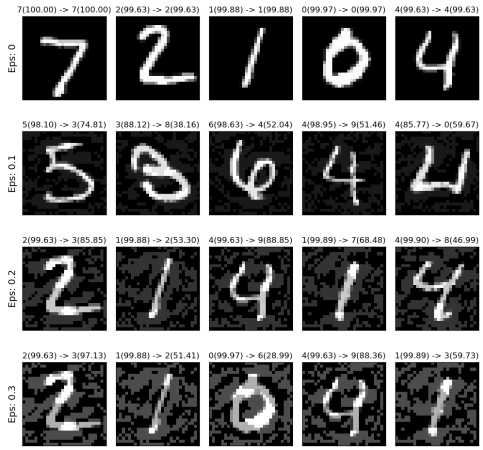
As in [1], the model from the 10th epoch is chosen for the experiments. Indeed, according to [1], the rank of the local data matrix shrank by the end of the training and the local data matrix does not have constant rank over the input space. Therefore, a partially trained model must be chosen to ensure that the rank is constant.

All the attacks will be performed on examples taken from the test set of MNIST. Figure 4.8a shows several FGSM examples for various attack budgets while Figure 4.8b provides the accuracy of the model over the test set when the test examples are replaced by FGSM examples for increasing levels of attack budget.

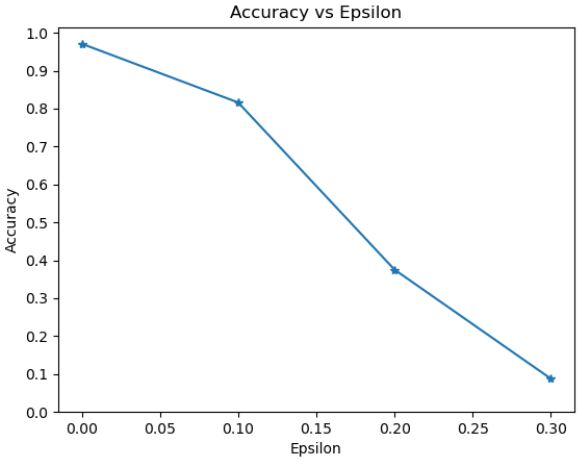
4.3.1.2 Projecting an FGSM example on the data leaf

Algorithm 1 from [1] is used to find a horizontal path between each image $\boldsymbol{x}_0$ and its adversarial example $\tilde{\boldsymbol{x}}$. If the algorithm does not converge (i.e., the adversarial example is not on the data leaf), the last iterate $\boldsymbol{x}_T$ is used as an estimate of the Euclidean projection of the adversarial example $\tilde{\boldsymbol{x}}$ on the data leaf.

[2]The code for this entire section is available here: `https://github.com/lshigarrier/nn_foliation`.
[3]32 and 64 channels respectively in the two CNN layers, and 128 units in the fully-connected layer.



(a) Each row corresponds to a certain attack budget in the L_∞ norm (denoted "Eps"). Five examples are given for each budget level. On top of each example, the predictions of the model on the original test example and on the adversarial example are provided along with the predicted probability. Note that the first row contains only original test examples since the budget is set to 0.



(b) Accuracy over the test set when the test examples are replaced by FGSM adversarial examples for increasing levels of attack budget.

Figure 4.8: FGSM attack

	Predicted class	**Predicted probability (%)**
Original example $\boldsymbol{x}_0$	5	98.10
FGSM example $\tilde{\boldsymbol{x}}$	3	74.81
Projection on the data leaf $\boldsymbol{x}_T$	3	70.57

Table 4.2: Predicted class and probability for the original example (Figure 4.9a), FGSM example (Figure 4.9d), and projection of the FGSM example on the data leaf (Figure 4.9c).

This experiment is performed on an example from the test set shown in Figure 4.9a.

As shown in Table 4.2, the projection of the FGSM example on the data leaf is also an adversarial example. Now, various distances and angles are computed to see how the adversarial examples behave with respect to the data leaf. For example, the Euclidean norm $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}_T\|_2$ gives the L_2 distance between $\tilde{\boldsymbol{x}}$ and the data leaf. As depicted in Table 4.3a, the projection $\boldsymbol{x}_T$ is closer to the original image $\boldsymbol{x}_0$ in L_2 norm but farther in L_∞ norm. The angle γ between $\tilde{\boldsymbol{x}} - \boldsymbol{x}_T$ and $(\ker \tilde{\boldsymbol{G}_{\boldsymbol{x}_T}})^\top$ can be obtained as follows. Let $\boldsymbol{v} = (\tilde{\boldsymbol{x}} - \boldsymbol{x}_T)/\|\tilde{\boldsymbol{x}} - \boldsymbol{x}_T\|_2$. One has $\cos\gamma = \sqrt{\sum_{i=1}(\boldsymbol{v}^\top \boldsymbol{e}_i)^2}$ where $(\boldsymbol{e}_i)$ is an orthonormal basis of $(\ker \tilde{\boldsymbol{G}_{\boldsymbol{x}_T}})^\top$. Table 4.3b shows that the angle between the data leaf and $\tilde{\boldsymbol{x}} - \boldsymbol{x}_T$ is (almost) 90° as expected.

4.3.1.3 Anti-adversarial examples

Experiments show that the entropy reaches a maximum along the horizontal path connecting $\boldsymbol{x}_0$ to $\boldsymbol{x}_T$ in the data leaf. When the entropy is maximum, the model is uncertain about the right class despite the image being easily classified by humans. In this paragraph, these images are

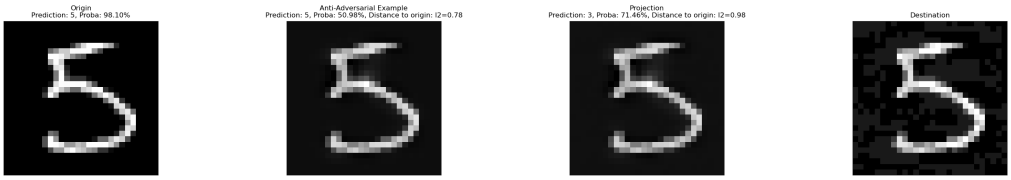
	L_2 norm	L_∞ norm
$d(x_0, \tilde{x})$	2.04	0.100
$d(x_0, x_T)$	1.03	0.174
$d(x_T, \tilde{x})$	1.75	-

(a) Distances between various examples.

Angle between the data leaf and:	
$\tilde{x} - x_0$	62.44
$\tilde{x} - x_T$	88.41

(b) Angles in degrees between the data leaf and various vectors.

Table 4.3: Distances and angles between various examples.



(a) Original image from the test set.

(b) Anti-adversarial example.

(c) Projection of the FGSM example on the data leaf.

(d) Adversarial image obtained with FGSM

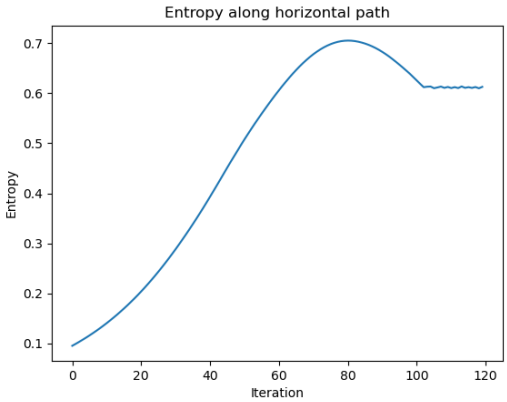
Figure 4.9: Original example, anti-adversarial example, projection on the data leaf, and FGSM example. An anti-adversarial example is characterized as follows: the model is unsure about how to classify the example despite being clearly recognizable by humans.

referred to as “anti-adversarial examples" (Figure 4.9b). The entropy and probabilities along the horizontal path are presented in Figure 4.10.

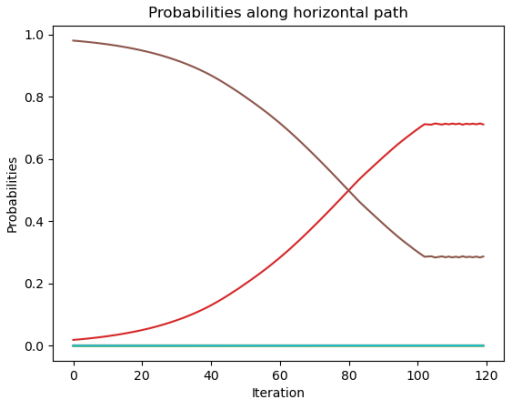
The behavior of the neural network along adversarial directions is further illustrated. An image x_0 is selected from the MNIST training set and the PGD attack is used to compute an adversarial perturbation $\tilde{x}$ of this image. Let $u_1 = \tilde{x} - x_0$ be the adversarial direction. Another vector u_2 is randomly sampled from the orthogonal complement of u_1. In figure 4.11, the plane of the input space $\mathbb{R}^d$ that is spanned by u_1 (Y-axis) and u_2 (X-axis) is plotted. The original example x_0 is in the center of each figure. The width of the image measured with the L_2 distance is equal to 84. In figure 4.11a, the spectral norm (i.e., the largest singular value) of the Jacobian matrix J_x of the network at each point of the (u_1, u_2) plane is plotted.

Figure 4.11a shows that the spectral norm is almost zero everywhere except around some boundaries where it becomes suddenly very large. Figure 4.11c allows to check that these boundaries are actually the decision boundaries of the network. Moreover, the original example is very close to a decision boundary when moving along the adversarial direction. This illustrates the fact that, in high dimensional spaces, almost every point is close to a decision boundary along a least one direction [87] (see also section 2.1.4.1). In figure 4.11b, the highest softmax probabilities predicted by the network at every point are plotted. The network assigns very large probabilities almost everywhere, except close to the decision boundaries where the “anti-adversarial examples" can be found. This suggests that local defense strategies may not be efficient, since the model is already almost constant close to the training points.

In [80], the authors also observe the sharp class transitions illustrated in figure 4.11. The authors claim that such sharp class can robustify a model by obfuscating the adversary, since the low gradient outside the decision boundaries does not contain a lot of information about the direction of the closest decision boundary. Future work can investigate this behavior further, and particularly why the model sensitivity increases dramatically when reaching a decision boundary.

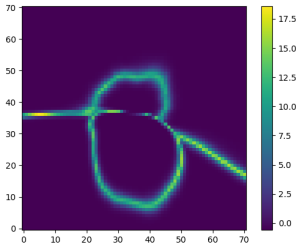


(a) Entropy along the horizontal path.

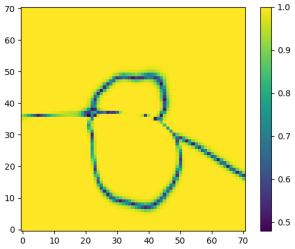


(b) Probabilities along the horizontal path.

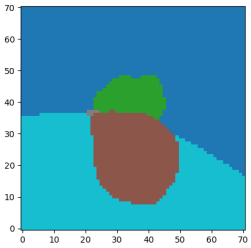
Figure 4.10: Entropy and predicted probabilities of each class with respect to the iterations of Algorithm 1 [1]. Note that the x-axis is the iteration not the distance. After the algorithm converges at the $\sim$ 100th iteration, the entropy and probabilities are constant because the distance between $\boldsymbol{x}_0$ and the current iterate does not increase anymore.



(a) The spectral norm of the Jacobian matrix of the network with respect to the input.



(b) The largest probabilities predicted by the network.



(c) The class predicted by the network.

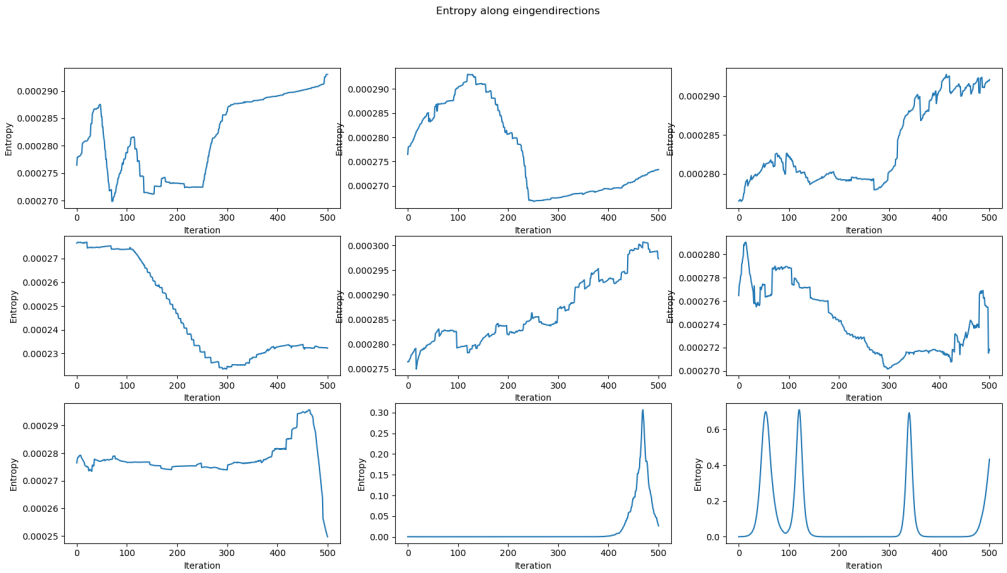
Figure 4.11: A plane of the input space where the Y-axis is an adversarial direction, while the X-axis is a random direction orthogonal to the adversarial direction.

This behavior could also be studied in constrained networks such as Lipschitz neural networks (section 2.2.2).

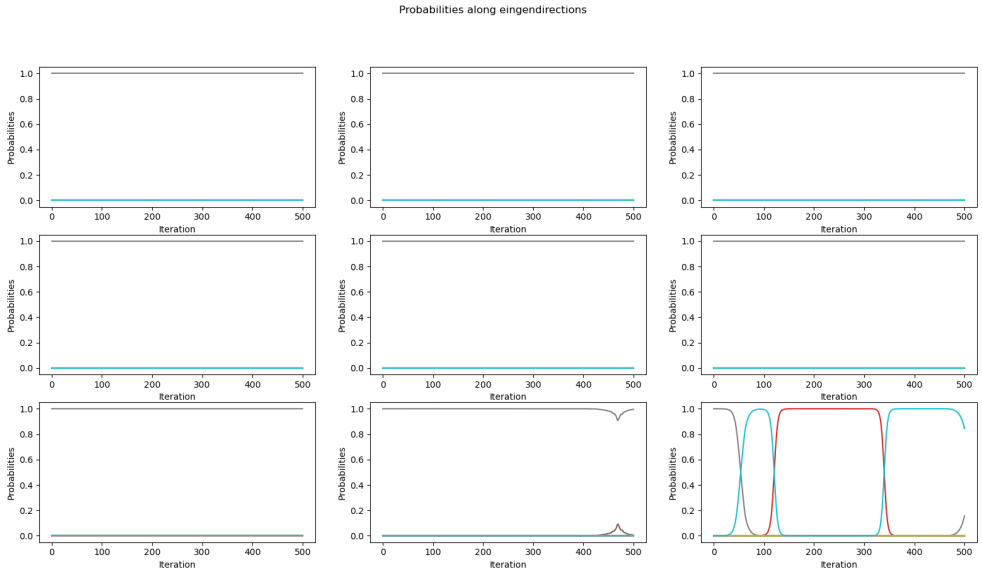
4.3.2 Trajectories on Leaves

This section explores the behavior of the model when moving along the data leaf by following eigenvectors of the local data matrix. Algorithm 1 is used with $T = 500$ and $\alpha = 0.05$. In the MNIST dataset, the number of classes is $c = 10$. The initial image $\boldsymbol{x}_0$ is shown in the first row of Figure 4.13b.

Class changes are only observed in the eigendirection associated to the highest eigenvalue.The distances plotted in figure 4.13a can be used to evaluate the “extrinsic curvature" of the data leaf i.e., how the data leaf is curved inside $\mathbb{R}^d$. The second derivative of the distances of figure 4.13a provides a measure of the extrinsic curvature along the corresponding eigendirection. Several observations can be made:

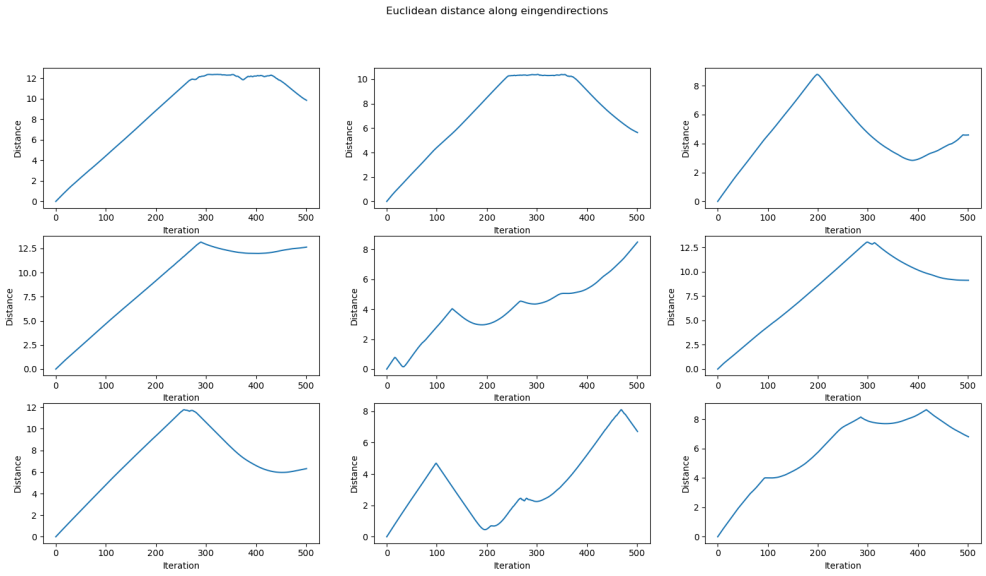


(a) Except for the two highest eigenvalues (bottom middle and right), the entropy stays almost constant and close to zero. For the highest eigenvalue (bottom right), three spikes corresponding to class changes can be seen (and it seems to be reaching a fourth spike). For the second highest eigenvalue (bottom middle), there is a smaller spike that does not lead to a class change.

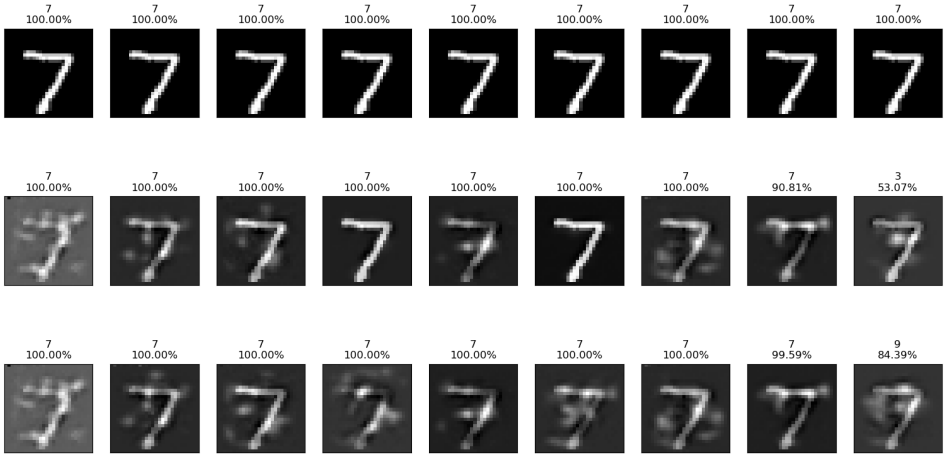


(b) Except for the two highest eigenvalues (bottom middle and right), the probabilities stays constant with the probability of class “7" almost equal to one and the others almost zero. For the highest eigenvalue, there are three class changes (from “7" to “9", then from “9" to “3", then from “3" to “9").

Figure 4.12: Entropy and probabilities along paths whose velocities are eigenvectors of the local data matrix. The associated eigenvalues are sorted with the smallest one in the top left corner and the highest one in the bottom right corner (the increase is along row first).



(a) For the first iterations, the distance increases almost linearly, indicating a low extrinsic curvature. Then, several sharp slope changes are observed, indicating areas of high extrinsic curvature.



(b) Each column corresponds to a different eigendirection. The first row is the original image. The second row is the image obtained at the point with maximum entropy. The third row is the final image obtained after T iterations. Only the highest eigenvalue (right most column) leads to class changes, but these are not adversarial examples because the perturbation is visible. For the other eigendirections, the model predicts the original class "7" with very high confidence despite the perturbations being clearly visible.

Figure 4.13: Distance along paths whose velocities are eigenvectors and examples of images along these paths.

Algorithm 1 Paths along eigendirections

Require: initial point $\boldsymbol{x}_0$, max iteration T, step size α, number of classes c
  $\tilde{\boldsymbol{G}} \leftarrow \texttt{localDataMatrix}(\boldsymbol{x}_0)$
  $\boldsymbol{v}_1, \dots, \boldsymbol{v}_{c-1} \leftarrow \texttt{eig}(\tilde{\boldsymbol{G}})$ ▷ the $c-1$ unit eigenvectors associated to the $c-1$ highest eigenvalues of $\tilde{\boldsymbol{G}}$ in decreasing order
  for $i = 1; i \leq c-1; i++$ **do**
    $\boldsymbol{d}_i \leftarrow \boldsymbol{v}_i$
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_0 - \alpha \boldsymbol{d}_i / \|\boldsymbol{d}_i\|_2$
  end for
  $k \leftarrow 1$
  while $k \leq T$ **do**
    for $i = 1; i \leq c-1; i++$ **do**
      $\tilde{\boldsymbol{G}} \leftarrow \texttt{localDataMatrix}(\boldsymbol{x}_i)$
      $\boldsymbol{v}_1, \dots, \boldsymbol{v}_{c-1} \leftarrow \texttt{eig}(\tilde{\boldsymbol{G}})$
      if $\boldsymbol{v}_i^\top \boldsymbol{d}_i < 0$ **then**
        $\boldsymbol{d}_i \leftarrow -\boldsymbol{v}_i$
      else
        $\boldsymbol{d}_i \leftarrow \boldsymbol{v}_i$
      end if
      $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \alpha \boldsymbol{d}_i / \|\boldsymbol{d}_i\|_2$
    end for
    $k \leftarrow k + 1$
  end while

- Three class changes that quickly follows each other were observed, and a fourth was going to happen when the last iteration was reached (figure 4.12b),
- The area of high extrinsic curvatures are not aligned with the area of high uncertainty (and class changes),
- The second highest eigenvalue seemed to start a class change but finally went back to the original class,
- All other eigendirections keep the same class with high confidence and low entropy (figures 4.12a and 4.12b),
- The extrinsic curvature seems to be low around the original point but becomes larger farther from it (figure 4.13a),
- The distance curves (figure 4.13a) seems almost piecewise linear, as if the extrinsic curvature was generally low with small areas of high extrinsic curvature,
- The model has a high confidence despite the images being very noisy,
- Some images seems to contain information from other digits. In the last row of Figure 4.13b, the third, fourth and sixth images seem to contain the shape of a “5". The fifth image contains an horizontal bar in the “7" that was not present in the original image. And the last image contains the shape of a “9" which is coherent with it being classified as a “9".

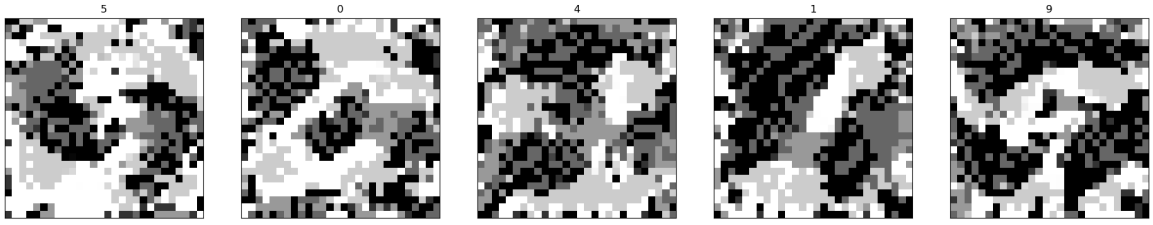


Figure 4.14: Examples of images from the robust MNIST training set. The true label is indicated above each image.

4.3.3 Robustified Dataset

This section investigates how the FIM and the foliations change when using different datasets or different models for the same task. In particular, the changes in the FIM when using a robustified dataset are studied.

4.3.3.1 Training a robust model with adversarial training

Using the same experiment setup as in subsection 4.3.1, a robust model is trained. The robust optimization criteria is approximated by choosing a random starting point in a L_∞ ball of radius $\epsilon = 0.3$, then performing 40 iterations of PGD with step size 0.01. PGD is an iterative attack where, at each iteration, a small gradient attack is performed (FGSM attack in this case) then the adversarial example is projected to the L_∞ ball of radius $\epsilon = 0.3$ and to the acceptable range of inputs (between 0 and 1 in this case).

In the following, the model trained on the standard MNIST dataset $\boldsymbol{D}$ is denoted by f_{std}, and the model trained by adversarial training is denoted by f_{adv}.

4.3.3.2 Constructing a robustified MNIST dataset

The method described in [50] is used to construct a robustified dataset $\boldsymbol{D}_r$ with the same size as $\boldsymbol{D}$ using a one-to-one function $\boldsymbol{D} \to \boldsymbol{D}_r, \boldsymbol{x} \mapsto \boldsymbol{x}_r$. The image $\boldsymbol{x}_r$ is built from $\boldsymbol{x}$ using an iterative process. Let g_{adv} be the composition of all layers until the penultimate layer of $f_{adv} = l_{adv} \circ g_{adv}$ (the last layer l_{adv} is a softmax regression layer i.e., $l_{adv}(\boldsymbol{z}) = s(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{b})$). The minimized loss is:

$$\mathcal{L}_{\boldsymbol{x}}(\boldsymbol{z}) = \|g_{adv}(\boldsymbol{z}) - g_{adv}(\boldsymbol{x})\|_2,$$

using normalized gradient descent i.e., the gradient is normalized and the iterates are clamped between 0 and 1 to be admissible images. The initial image is uniformly sampled from $\boldsymbol{D}$. For each image $\boldsymbol{x}$, 100 iterations are performed with step size 0.2. The image $\boldsymbol{x}_r$ is defined as follows:

$$\boldsymbol{x}_r = \arg\min_{\boldsymbol{z}} \mathcal{L}_{\boldsymbol{x}}(\boldsymbol{z}).$$

Figure 4.14 shows several examples from the robustified training set. It is difficult (while not impossible) for a human to recognize the correct label of these images. Nonetheless, they contain enough information to reach a decent accuracy while being more robust than the standard model f_{std}.

Now, a third model f_r is trained on $\boldsymbol{D}_r$ but without any adversarial training. The hyperpa-

Model	Standard accuracy	Adversarial accuracy
f_{std}	97.11 %	16.13 %
f_{adv}	97.72 %	85.61 %
f_r	73.96 %	38.30 %

Table 4.4: Standard and adversarial accuracy for the three models.

rameters are the same[4] as described in subsection 4.3.1. The three models f_{std}, f_{adv} and f_r are tested on the standard MNIST test set consisting of 10,000 images. To measure the robustness of each model, the three models are also tested on adversarial examples of each image of the test set. The adversarial samples are obtained using PGD with $\epsilon = 0.3$, 20 iterations, step size 0.01 and L_∞ norm.

As shown in Table 4.4, f_r has a higher adversarial accuracy than f_{std} but a lower standard accuracy.

4.3.3.3 Exploring the data leaf of the three models

The following conjectures will be tested:

- The training *and* test data lie on the same leaf (the data leaf). This is less true for the test set than for the training set. If true, the data leaf would be the approximation of the data manifold from the point of view of the model;
- The adversarial examples are not on the data leaf;
- It is possible to distinguish between the three models f_{std}, f_{adv} and f_r using only the distributions of the training, test, and adversarial examples in the foliation.

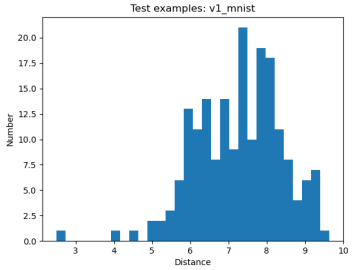
In order to test these conjectures, 20 images are randomly sampled from the training set and 20 images from the test set. For each of these images, an adversarial attack is crafted using PGD with the L_∞ norm, 100 iterations and a budget $\epsilon = 0.3$. Then, for each model, two experiments are performed:

1. For each pair of original (i.e., not adversarial) images, the horizontal path from the first image to the second image is computed using Algorithm 1 from [1]. The Euclidean distance between the last iterate and the second image is seen as the distance between the second image and the leaf of the first image[5]. If all images are in the same leaf, this distance is expected to be small,
2. For each original image, the horizontal path between this image and its associated adversarial image is computed. Once again, the Euclidean distance between the last iterate and the adversarial image gives the distance between the adversarial image and the leaf of the original image. Here, if the adversarial images are not in the data leaf, the distance are expected to be larger than in the former experiment.

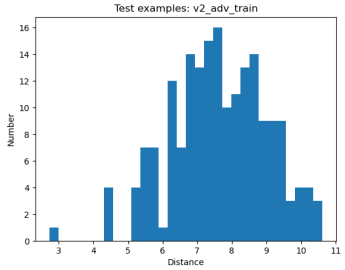
The horizontal path are computed with a step of 0.1, and a maximum number of iterations of 200. To avoid excessive computation time, a robust stopping criterion is used. Denoting the loss

[4] with the exception that the model after the second epoch is chosen instead of the tenth because the model seems to overfit the training set after the second epoch. This phenomenon was not observed for the other models.

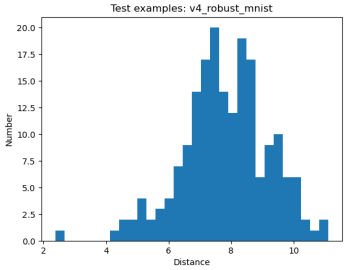
[5] Note that this relation is not symmetric: this is not the same as the distance from the first image to the leaf of the second image. To avoid excessive computation time, this other distance is not computed.



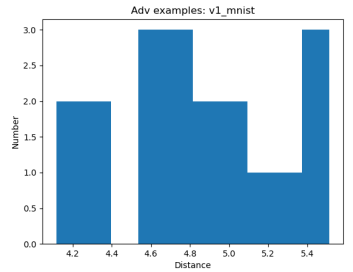
(a) Original images using f_{std}.



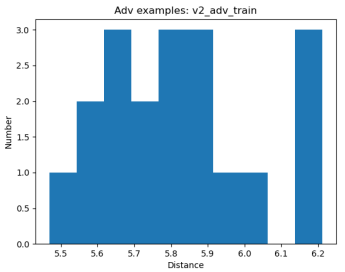
(b) Original images using f_{adv}.



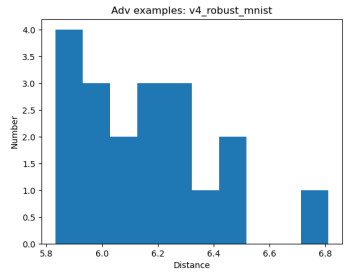
(c) Original images using f_r.



(d) Adversarial images using f_{std}.



(e) Adversarial images using f_{adv}.



(f) Adversarial images using f_r.

Figure 4.15: Histograms of distances using 20 images from the test set.

function by $\mathcal{L}$ (i.e., the distance between the current iterate and the destination), the following criterion is defined:

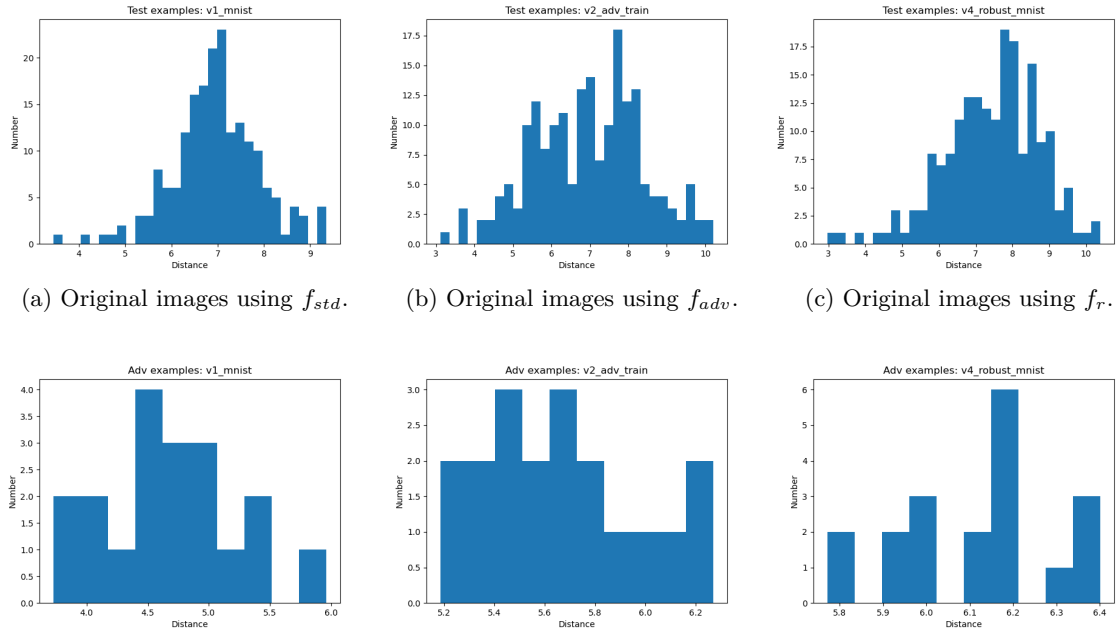
$$\frac{k|\mathcal{L}(\boldsymbol{x}_k) - \mathcal{L}(\boldsymbol{x}_{k-1})|}{|\mathcal{L}(\boldsymbol{x}_k)|} < \eta$$

where $\eta = 0.001$ is a tolerance.

In Figures 4.15 and 4.16, the histograms of the distances are plotted for each experiment. As depicted in these figures, all the conjectures are false. The distances for the original images are not close to zero and there is no difference between the training and test sets (first conjecture is false). The distance for the adversarial images are actually *smaller* than for the original images (second conjecture is false). The three models generate the same distributions for the original images as well as for the adversarial images (third conjecture is false). These results seem to indicate that the data leaf does not even exist, or at least not in the sense of being an approximation of the data manifold. This is in contradiction with the claims of [1].

In [1], the authors use at least 5000 iterations, up to 10000 iterations, to construct the horizontal paths. It might be possible that the horizontal paths were stuck in a local minimum. Because of the computation time, these experiments cannot be conducted with 5000 iterations. However, the horizontal path can be computed for a small number of examples. In Figure 4.17, the evolution of the distance between the current iterate and the destination is depicted for 5000 iterations. If the algorithm is stuck in a local minima, it does not seem to escape from it with more iterations. Moreover, the actual paths (also shown in 4.17) indicate that it is not possible to reach the destination. This is especially true for f_{adv} where it seems that the class of the image cannot be modified when moving along the leaf.

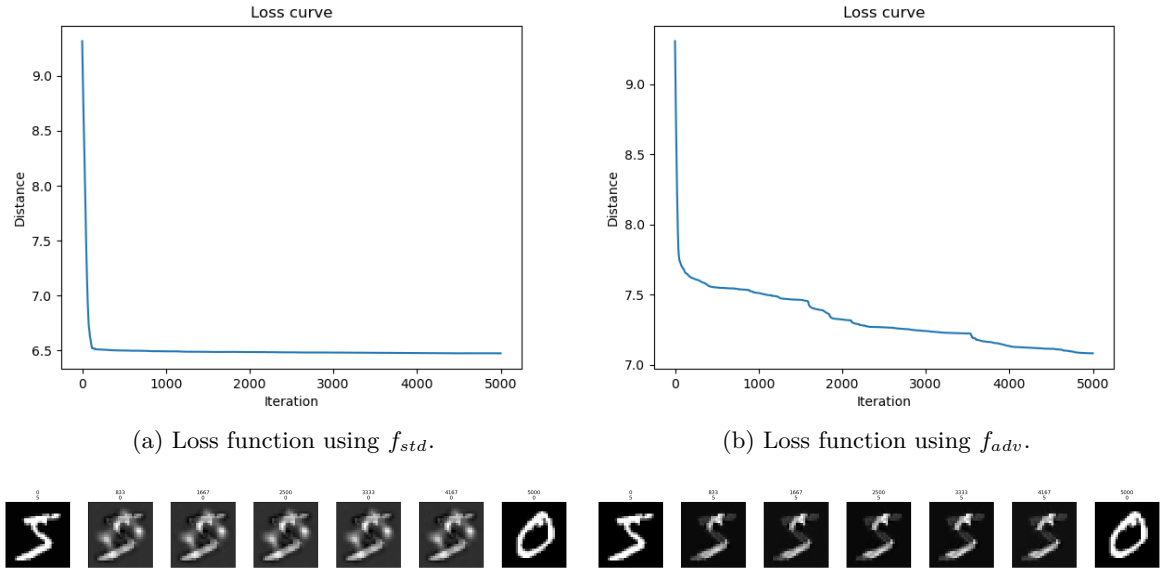
Now, one may conjecture that there is not a single data leaf, but one data leaf for each class. To test this conjecture, 100 images are sampled from the training set. As in Figure 4.16, the histogram of distances using f_{std} are plotted but considering only the horizontal paths between



(a) Original images using f_{std}. (b) Original images using f_{adv}. (c) Original images using f_r.

(d) Adversarial images using f_{std}. (e) Adversarial images using f_{adv}. (f) Adversarial images using f_r.

Figure 4.16: Histograms of distances using 20 images from the training set.



(a) Loss function using f_{std}. (b) Loss function using f_{adv}.

(c) Horizontal path using f_{std}. (d) Horizontal path using f_{adv}.

Figure 4.17: Loss function and horizontal path between two original images of the training set using f_{std} and f_{adv} with 5000 iterations. The first image is the origin, the last image is the destination, the five central images are sampled regularly along the horizontal path. Above each image, the iteration and the predicted class are indicated.

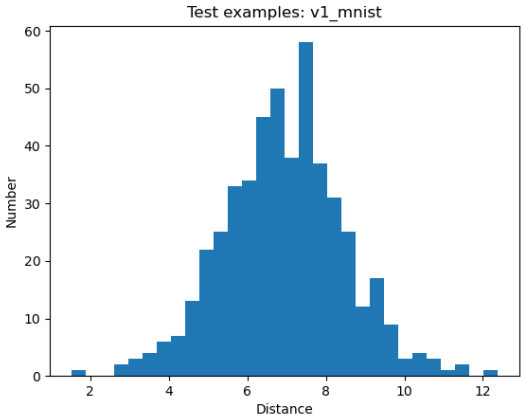


Figure 4.18: Histogram of distances between images of the same class using f_{std} and 100 images from the training set.

images classified in the same class. If images of the same class are in the same leaf, the distances are expected to be close to zero. Figure 4.18 shows that this is not the case.

These experiments suggest that the definition of the data leaf given in [1] cannot be achieved in practice. Future work should focus on studying the properties of the image foliation in order to understand how it should be interpreted.

4.4 Conclusion

In this chapter, several speculative topics linking information geometry with the robustness of neural networks against noise and adversarial attacks have been explored. After introducing the notion of kernel and image foliations, a framework for studying models for time series prediction was laid out by approximating a recurrent neural network with a stochastic differential equation. Then, a visualization of the kernel foliation of such recurrent network was provided in a simple case. In the second part of the chapter, several experiments were performed around the idea of the data leaf. However, the interpretation of the data leaf given in [1] could not be replicated. More generally, these experiments exhibit various behaviors that could be studied in more details in future work. At least, the experiments and illustrations provided in this chapter have demonstrated that studying machine learning robustness through an information geometric lens is a fruitful direction that should be explored further.

The next chapter leaves the information geometric world and come back to applications for civil aviation, in particular for the prediction of congested areas in the airspace. Moreover, a framework for uncertainty quantification is developed and applied to state-of-the-art models for time series prediction.

Chapter 5

Traffic Prediction and Uncertainty Quantification

In this chapter, machine learning techniques are applied to tackle traffic prediction problems. The chapter begins by discussing the context surrounding congestion and complexity in Air Traffic Management (ATM). Then, the literature about Air Traffic Flow Prediction (ATFP) is reviewed, with a focus on machine learning approaches. The aim of ATFP is to perform traffic prediction at a larger scale than classical trajectory prediction. The literature about complexity metrics that are used to quantify the complexity of a traffic situation is also reviewed.

In the next section, a new ATFP model based on a recurrent model called encoder-decoder LSTM is introduced. This work was presented at the Fourteenth USA/Europe Air Traffic Management Research and Development Seminar [24]. In the final part of the chapter, this model is improved using Transformer models that are the state-of-the-art framework for time series prediction. The variational moment propagation method for Transformers is derived and evaluated on a traffic prediction task using road data.

5.1 Context

With the continuous rise in air transportation demand (a 4.3% annual growth of the worldwide passenger demand is expected until 2035 [141]), the capacity of the 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 [142]). 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 [143], and was shown to negatively affect the controller's decision-making ability and increase the error rate [144]. 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 [145]. 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 chapter focuses on the first problem of predicting congested areas tens of minutes ahead of their formation.

5.2 Air Traffic Flow Prediction (ATFP)

Before presenting the model, the literature addressing the domain of application (Air Traffic Flow Management) is reviewed with a focus on the concepts and methods linked to congestion prediction at medium to short term. The literature to tackle the general problem of congestion prediction can be grouped 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.

5.2.1 Conflict detection tools

In the first approach of short-term conflict detection methods [146], 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 [146]. The MTCD applications, currently implemented in operational context, e.g., the Eurocontrol MTCD project [147], 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.

5.2.2 Classic ATFP Methods

Air traffic flow prediction (ATFP) focuses on aggregate models [148] 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 [149]. 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 [148, 150, 149] to more recent machine learning algorithms [151, 152, 153].

In [148], 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 [150], 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 [149] 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.

5.2.3 Machine Learning ATFP

Recently, several machine learning models were proposed for ATFP. In [151], 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 [152] using several 3D convolutional neural networks to extract spatial features for one timestep then recurrent layers to extract temporal features. In [153], 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.

5.3 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. First, two reviews that have been published on the topic [143, 154] are presented. Then, classic complexity metrics and intrinsic complexity metrics are discussed.

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 [155, 156, 157, 158, 159, 160, 161, 162]. 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 [157]. 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 [163] for one example) is not addressed in this review.

5.3.1 Reviews

In [143], 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 5.3.2 and 5.3.3):

- 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 [154], 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.

5.3.2 Classic Methods

In [155], 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 [156], 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 [157], B. Hilburn reviews the literature into cognitive complexity in air traffic control. Complexity can be defined as the state of being hard to separate, analyze, 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 consists of a huge number of parts). Complex systems are defined by: a large number of elements, which interact dynamically, which contain redundancy, which enjoy localized 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 behavior 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 behavior, 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 behaviors), the fact that humans do not “optimize" but “satisfy" (which is harder to model), and the fact that human behavior is very dependent on the context.

In [158], 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 [159].

Several versions of Dynamic Density have been proposed in the literature [160], [161]. For example, the interval complexity introduced in [162] 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 [164]. 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 consisting 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 [164].

In [165], 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 [166]. 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 [167], 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 [168] (see section 5.3.3). Despite its simplicity, this complexity metrics is able to account for the complexity in typical scenarios.

5.3.3 Intrinsic Complexity Metrics

As elaborated in [143], 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 [169], [168]. As developed in [154], 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 [169], 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 [168], an intrinsic complexity metric using linear and nonlinear dynamical

system models is defined. 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 [170], 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 $\boldsymbol{P}$ and $\boldsymbol{V}$, which denote, respectively, the positions and velocities of the n_{ac} aircraft, i.e.,

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

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 ith sample. A linear dynamical system $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ is fitted to the positions $\boldsymbol{P}$ and speeds $\boldsymbol{V}$ of these aircraft, such that $\boldsymbol{V} \simeq \boldsymbol{A}\boldsymbol{P} + \boldsymbol{b}$ (here, $\boldsymbol{b}$, representing the mean of the speed vector field, is broadcast to be added to the matrix $\boldsymbol{A}\boldsymbol{P}$). The matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$ are computed as the solutions of the Least Mean Squares problem:

$$\min_{\boldsymbol{A},\boldsymbol{b}} \|\boldsymbol{V} - (\boldsymbol{A}\boldsymbol{P} + \boldsymbol{b})\|_F^2. \tag{5.1}$$

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

$$\mathfrak{c}(\boldsymbol{A}) = \sum_{\text{Re}(\lambda(\boldsymbol{A}))<0} |\text{Re}(\lambda(\boldsymbol{A}))|. \tag{5.2}$$

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

5.4 Recurrent Model

In the last two sections, the literature related to the target application has been reviewed, including air traffic flow prediction and complexity metrics. Now, the first model used for

prediction of congested areas is presented. This model is based on recurrent networks i.e., models that process data in a sequential manner. After introducing the machine learning notions related to recurrent models, the architecture of the encoder-decoder LSTM network is described and evaluate it on synthetic data.

5.4.1 Review of Sequential Architectures

This section presents an overview of sequential models in machine learning. Sequential models are particularly well-suited for time series, such as aircraft trajectories. 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, encoder-decoder models are presented. The Transformer model, which is the current state-of-the-art machine learning sequential model, will be introduced in section 5.5.1. 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 [5].

5.4.1.1 Recurrent Neural Networks (RNN)

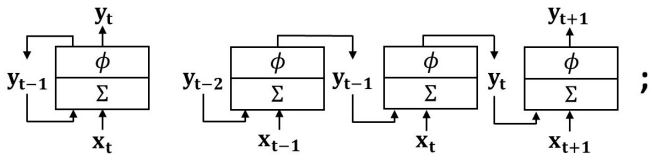


Figure 5.1: Basic RNN layer (left) and the same layer unrolled through time (right). Σ represents linear mapping and ϕ represents the activation function.

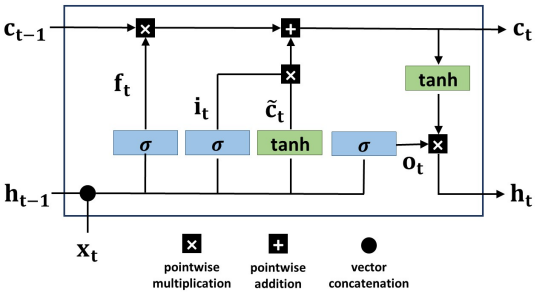


Figure 5.2: A Long Short-Term Memory (LSTM) layer. The current input $\boldsymbol{x}_t$ is combined with the previous hidden state $\boldsymbol{h}_{t-1}$ to compute the three gates used to update the cell state $\boldsymbol{c}_{t-1}$ and compute a new hidden state $\boldsymbol{h}_t$.

RNN is a class of networks 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 [171], or speech-to-text for example in [172]. They have also been applied in generative models, for example in image captioning [173].

The basic RNN layer (Figure 5.1) can be described as follows. At each time step t, the recurrent cell receives the input $\boldsymbol{x}_t$ as well as its own output from the previous time step $\boldsymbol{y}_{t-1}$. The output $\boldsymbol{y}_t$ is computed according to Equation (5.3), where $\boldsymbol{W}$ is the weights matrix, $\boldsymbol{b}$ is the bias vector and ϕ is an activation function, for example tanh or ReLU [174]. $\boldsymbol{x}_t$ and $\boldsymbol{y}_{t-1}$ have been concatenated in a single row vector denoted $[\boldsymbol{x}_t; \boldsymbol{y}_{t-1}]$.

$$\boldsymbol{y}_t = \phi(\boldsymbol{W} \cdot [\boldsymbol{x}_t; \boldsymbol{y}_{t-1}] + \boldsymbol{b}) \tag{5.3}$$

To train the network, the classical optimization algorithms (generally based on gradient descent) need the gradients of the parameters with respect to the outputs. The backpropagation method

is used to compute these gradients. It is often referred to as *backpropagation through time* (BPTT) when applied to RNN.

To prevent overfitting, a technique often used with RNN is dropout [175], [176]. 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 a hyperparameter.

In [177], 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.

5.4.1.2 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 [178]. To solve it, several architectures of RNN layer with long-term memory have been introduced. Two of these new type of layers will be presented: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU).

Long Short-Term Memory (Figure 5.2) were first introduced in [179], and then improved over the years, see for example [180] and [181]. 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 $\boldsymbol{h}_t$ (the hidden state) and $\boldsymbol{c}_t$ (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 (5.4) to (5.9). In this paragraph, σ is the logistic activation (to get outputs between 0 and 1). The current input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$ are used to compute the controllers of three *gates*: the input gate controller $\boldsymbol{i}_t$, the forget gate controller $\boldsymbol{f}_t$ and the output gate controller $\boldsymbol{o}_t$ (Equations (5.4) to (5.6)). These gate controllers are simply computed by a linear mapping followed by a logistic activation. In parallel, a candidate cell state $\tilde{\boldsymbol{c}}_t$ is computed using the current input and the previous hidden state (Equation (5.7)). Equation (5.7) can be seen as the LSTM equivalent of the equation of the basic RNN layer (Equation (5.3)). In Equation (5.8), the current cell state $\boldsymbol{c}_t$ is then updated by adding the previous cell state $\boldsymbol{c}_t$ (filtered by the forget gate) and the candidate cell state $\tilde{\boldsymbol{c}}_t$ (filtered by the input gate). The hidden state $\boldsymbol{h}_t$ is finally computed by activating the cell state $\boldsymbol{c}_t$ with the tanh function (or another activation function such as the ReLU), which is then passed though the output gate (Equation (5.9)). With the input gate $\boldsymbol{i}_t$, the LSTM is able to store information in the cell state $\boldsymbol{c}_t$, to extract it with the output gate $\boldsymbol{o}_t$ and to discard it with the forget gate $\boldsymbol{f}_t$.

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{x}_t; \boldsymbol{h}_{t-1}] + \boldsymbol{b}_i) \tag{5.4}$$
$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{x}_t; \boldsymbol{h}_{t-1}] + \boldsymbol{b}_f) \tag{5.5}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{x}_t; \boldsymbol{h}_{t-1}] + \boldsymbol{b}_o) \tag{5.6}$$
$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{x}_t; \boldsymbol{h}_{t-1}] + \boldsymbol{b}_c) \tag{5.7}$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{5.8}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{5.9}$$

A variant of LSTM introduced in [182] called *peephole connections* uses the cell states $\boldsymbol{c}_{t-1}$ and $\boldsymbol{c}_t$ to compute the gate controllers. Another variant of the LSTM is the Gated Recurrent Unit

(GRU) introduced in [183]. Its equations are detailed in Equations (5.10) to (5.13). The GRU is a simplified version of the LSTM, but seems to have the same performance as investigated in [184]. The GRU has a unique state vector $\boldsymbol{h}_t$, while the forget and input gates are merged into a single gate controller $\boldsymbol{z}_t$ (Equation (5.10) and (5.13)). The output gate is replaced by another gate controller $\boldsymbol{r}_t$ (Equation (5.11)), placed between the previous hidden state $\boldsymbol{h}_{t-1}$ and the candidate hidden state $\tilde{\boldsymbol{h}}_t$ (Equation (5.12)).

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z[\boldsymbol{x}_t; \boldsymbol{h}_{t-1}] + \boldsymbol{b}_z) \tag{5.10}$$
$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{x}_t; \boldsymbol{h}_{t-1}] + \boldsymbol{b}_r) \tag{5.11}$$
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_h[\boldsymbol{x}_t; \boldsymbol{r}_t \times \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h) \tag{5.12}$$
$$\boldsymbol{h}_t = \boldsymbol{z}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \odot \tilde{\boldsymbol{h}}_t \tag{5.13}$$

An improvement of the RNN structure is the Bidirectional Recurrent Neural Network (BRNN) introduced in [185]. In the basic RNN layer, the hidden state $\boldsymbol{h}_t$ is computed using only the sequence of inputs $\boldsymbol{x}_0, ..., \boldsymbol{x}_t$. 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 ($\boldsymbol{x}_0$ until $\boldsymbol{x}_T$ where T is the length of the input sequence), and the other one going through the input sequence backward ($\boldsymbol{x}_T$ until $\boldsymbol{x}_0$). Hence, two sequences of hidden states are obtained, 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.

5.4.1.3 Encoder-Decoder Architecture

The first introduction of an encoder-decoder model for automatic translation is provided in [171]. This model has also been applied to various tasks, such as speech recognition [186] or video captioning [187].

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 consists 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 does 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.

The beam search algorithm is illustrated with an example. Let $\mathfrak{e}$ and $\mathfrak{d}$ be respectively the trained encoder and the trained decoder. The encoder $\mathfrak{e}$ is a function that takes an input sequence and returns an encoding vector $\boldsymbol{v}$. The decoder $\mathfrak{d}$ is a function that takes the encoding vector $\boldsymbol{v}$ as well as the beginning of an output sequence $\boldsymbol{y}_1, \dots, \boldsymbol{y}_t$ and returns the distribution $p(\boldsymbol{y}_{t+1}|\boldsymbol{v}, \boldsymbol{y}_1, \dots, \boldsymbol{y}_t)$ of the next element of the output sequence $\boldsymbol{y}_{t+1}$ given the encoding vector and the first terms of the output sequence. Let $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T'}$ be an input sequence. For example, $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T'}$ can be seen as a sentence where each $\boldsymbol{x}_i$ is the representation of a word obtained with word embedding (see [5] for more details about word embedding). In this example, the output sequences $\boldsymbol{y}_1, \dots, \boldsymbol{y}_T$ are assumed to be built with elements $\boldsymbol{y}_t$ taken from a finite dictionary $\mathcal{D}$. In the case of the automatic translation task, each $\boldsymbol{y}_t$ is a word from the target language. The final objective would be to compute the distribution $p(\boldsymbol{y}_1, \dots, \boldsymbol{y}_T|\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T'})$ and then take the argmax to get the output sequence $\boldsymbol{y}_1, \dots, \boldsymbol{y}_T$ 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 $\boldsymbol{v} = \mathfrak{e}(\boldsymbol{x}_1, \dots, \boldsymbol{x}_{T'})$. Then, the distribution $p(\boldsymbol{y}_1|\boldsymbol{v}) = d(\boldsymbol{v})$ is computed, using a zero vector as input of the decoder network. A parameter $B \in \mathbb{N}^*$ corresponding to the number of sequences that will be kept at each timestamp is introduced here for the beam search. Therefore, using the distribution $p(\boldsymbol{y}_1|\boldsymbol{v})$, the B start-of-sequence elements $\boldsymbol{y}_1^1, \dots, \boldsymbol{y}_1^B$ with the highest partial likelihood are chosen. Then, each of these start-of-sequence element are used as inputs of the decoder: for every j from 1 to B, the distribution $p(\boldsymbol{y}_2|\boldsymbol{v}, \boldsymbol{y}_1^j)$ are computed. For every j from 1 to B and for every element $\boldsymbol{y}_2$ in the dictionary $\mathcal{D}$, the partial likelihood $p(\boldsymbol{y}_1^j, \boldsymbol{y}_2|\boldsymbol{v}) = p(\boldsymbol{y}_2|\boldsymbol{v}, \boldsymbol{y}_1) \times p(\boldsymbol{y}_1^j|\boldsymbol{v})$ can be computed. Once again, only the B partial sequence $\boldsymbol{y}_1, \boldsymbol{y}_2$ with the highest partial likelihood $p(\boldsymbol{y}_1^j, \boldsymbol{y}_2|\boldsymbol{v})$ are kept. By repeating this process for each timestamp, B complete output sequences $\boldsymbol{y}_1, \dots, \boldsymbol{y}_T$ are built. It is then possible to output the sequence with the highest likelihood among these B sequences. Note that there is no guarantee that this is the sequence that maximizes $p(\boldsymbol{y}_1, \dots, \boldsymbol{y}_T|\boldsymbol{v})$.

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 [171]. 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 [188] and [189]. 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 [4] which is currently the state-of-the-art architecture for most NLP tasks (see section 5.5.1).

5.4.2 Encoder-Decoder Long Short-Term Memory Network

This sections presents the sequential machine learning models that will process the series of aircraft state vectors. The model is an encoder-decoder network architecture as described in paragraph 5.4.1.3. Then, a one-dimensional convolutional layer used in the initial processing of aircraft state vectors is described.

The Encoder-Decoder LSTM is a recurrent neural network designed to address sequence-to-sequence problems. Sequence-to-sequence prediction problems are challenging because the number of items in the input and output sequences can vary. The Encoder-Decoder architecture is comprised of two models: one for encoding the input sequence into a fixed-length vector, and a second for decoding the fixed-length vector and outputting the predicted sequence. The innovation of this architecture is the use of a fixed-sized internal representation in the heart of the model, which may be referred to as *sequence embedding*. Both the encoder and decoder models are defined as recurrent neural networks, i.e., LSTMs.

The input of the decoder consists of two elements: the output of the encoder and the previously predicted (i.e., decoded) output sequence term. During training, the previously decoded sequence is provided as the ground-truth sequence at the previous timestep. During inference, 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 timestep as the input for the next one. However, this method may result in accumulation of prediction errors and lead to poor performance, especially 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 does not predict an element of the output sequence at each timestep, but rather a distribution over the possible outputs. Then, using a beam search algorithm [190], it is possible to maintain, at each timestep, a small number of possible output sequences using the partial probability of each sequence as the metric. At the last timestep, the model outputs the sequence with the highest probability.

This framework is more suited for classification tasks where the output distribution is discrete and finite. Even if it is possible to choose a family of continuous distributions to represent the set of possible output distributions of the decoder network for regression, a simpler approach has been retained here. The objective of the model is to find the output sequence $\boldsymbol{Y}$ that maximizes the conditional probability $\mathbb{P}[\boldsymbol{Y}|\boldsymbol{X}]$ given the input sequence $\boldsymbol{X}$. More formally, the following optimization problem is solved:

$$\arg\max_{\boldsymbol{Y}} \mathbb{P}[\boldsymbol{Y}|\boldsymbol{X}] = \arg\max_{\boldsymbol{Y}} \prod_{t} \mathbb{P}[\boldsymbol{y}_t|\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}, \boldsymbol{X}].$$

The network directly outputs a prediction $\boldsymbol{y}_\tau = \arg\max_{\boldsymbol{y}_\tau} \mathbb{P}[\boldsymbol{y}_\tau|\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{\tau-1}, \boldsymbol{X}]$ for the τ-th element of the output sequence. Hence, at time step τ, the current estimation of the prob-

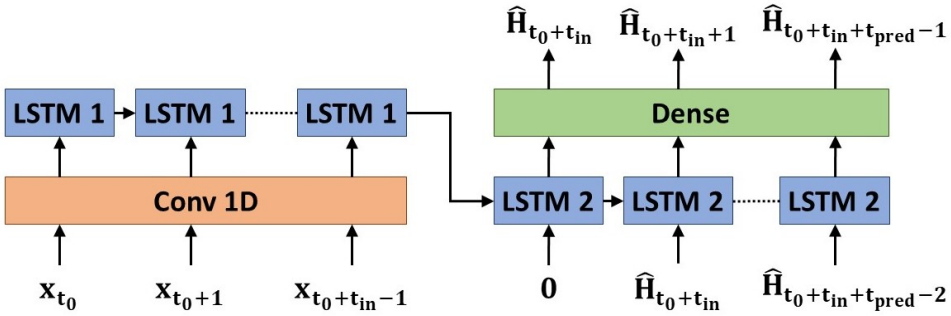


Figure 5.3: Encoder-decoder LSTM model for complexity prediction. The encoder network (left) uses a sequence of aircraft states $\boldsymbol{x}_{t_0}, \cdots, \boldsymbol{x}_{t_0+t_{in}-1}$ to compute an *encoding sequence*. The decoder network (right) uses the encoding sequence and the complexity matrix outputted at the last timestep $\hat{\boldsymbol{H}}_{t-1}$ to compute a prediction.

ability of the partial sequence $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{\tau-1}$ is $\prod_t^\tau \max_{\boldsymbol{y}_t} \mathbb{P}[\boldsymbol{y}_t | \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}, \boldsymbol{X}]$. This greedy approach can be seen as approximating $\max_{\boldsymbol{Y}} \mathbb{P}[\boldsymbol{Y}|\boldsymbol{X}] = \max_{\boldsymbol{Y}} \prod_t \mathbb{P}[\boldsymbol{y}_t | \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}, \boldsymbol{X}]$ by $\prod_t \max_{\boldsymbol{y}_t} \mathbb{P}[\boldsymbol{y}_t | \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}, \boldsymbol{X}]$. This approach is equivalent to the beam search algorithm with a beam width of 1.

One-dimensional (1-D) Convolutional Layer. The first layers of the encoder network are defined as 1-D convolutional layers to process the input sequence of aircraft states. A 1-D convolutional layer consists of n_f different filters. A filter contains a kernel of learnable parameters with dimension $h \times m$, where m is the dimension of the input sequence, and h, the width of the kernel, is a hyperparameter of the convolutional layer. Each filter aggregates a subsequence of the input sequence of length h such that each element of the output sequence of the layer contains information from several successive timesteps. More precisely:

$$y_{ij} = \phi\left(\sum_{k=-h}^{h} \sum_{l} w_{kl}^{j} x_{i+k,l}\right),$$

where y_{ij} is the j-th vector element of the i-th sequence term outputted by the 1-D convolutional layer, and x_{kl} is the l-th vector element of the k-th sequence term of the input; w_{kl}^{j} is the k-th weight of the kernel of the j-th filter associated with the l-th dimension of the input sequence, and ϕ is an activation function. If $i + k$ is lower than zero or greater than the input sequence's length, then $x_{i+k,l} = 0$, for all l.

The convolution operation is performed only along the time dimension of the input sequence; this is why this convolutional layer is referred to as "one-dimensional". With the 1-D convolutional layers, the encoder network is expected to identify dependencies in a very short time window, such that the LSTM layers could process an intermediate sequence in which each timestep contain dynamical information from the previous and following timesteps. This may facilitate the computation of the embedding for the whole sequence. Without this convolutional layers, the LSTM layers would process one timestep after the other without any information from the next timesteps.

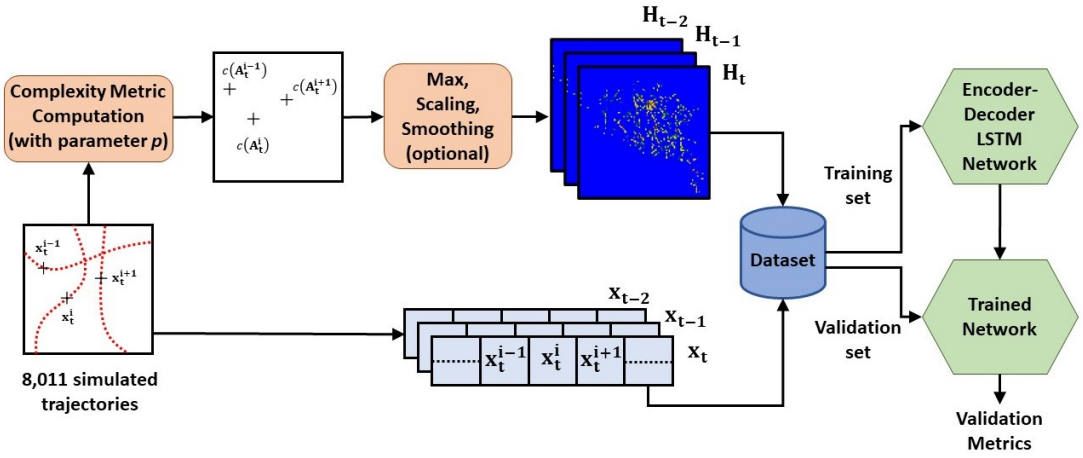


Figure 5.4: Depiction of the entire framework. The dataset consists of trajectories where $\boldsymbol{x}_t^i$ is the state vector of the ith trajectory at time t. The complexity values are computed as described in Section 5.3.3. The dataset, consisting of the aircraft states and the complexity matrices, is divided into training and validation sets. The encoder-decoder LSTM model is trained (validated) with the training (validation) sets.

5.4.3 Methodology and experiments

The implementation details of the model are discussed now, starting with the description of the dataset and the construction of the training set. Then, the architecture details of the encoder-decoder model are provided.

5.4.3.1 Data acquisition and preprocessing

In order to test the capacity of the model to predict the complexity of the traffic, a dataset of simulated trajectories is used. In the simulation, no deconflicting actions were applied, such that each flight follows its flight plan without interacting with the other flights. Therefore, the true complexity is obtained before mitigation, as opposed to historical data where the controllers have already applied measures to lower the complexity. The dataset represents a regular day of traffic over the French airspace. A trajectory is defined as a sequence of aircraft states. Each aircraft state consists of the following 8 elements.

$$\begin{aligned} \textit{State} = [&\text{timestamp, aircraft ID, latitude, longitude, altitude,} \\ &\text{ground speed, heading, rate of climb}]. \end{aligned}$$

The trajectories are created by inputting flight plans to a simulator, where the flight plans consist of a sequence of waypoints and a cruise flight level. The atmosphere is assumed to be in ISA conditions with no wind. The final dataset consists of $8,011$ simulated trajectories spread along $3,025$ timesteps (1 timestep corresponds to 15 seconds). A complexity value is then computed for each aircraft at each timestep as described in Section 5.3.3.

The objective of the supervised learning model is a sequence-to-sequence regression task. To achieve this, a training set must be built using pairs of training inputs/outputs, consisting of sequences of aircraft states paired with sequences of complexity values over the entire airspace. To build the training outputs, a $c \times c$ matrix $\boldsymbol{H}_t$ is defined for each time step t. The matrix $\boldsymbol{H}_t$ will be referred to as a *complexity matrix*. This matrix is superimposed over the airspace, such that each element of the matrix covers a small rectangle area whose dimensions depend on the

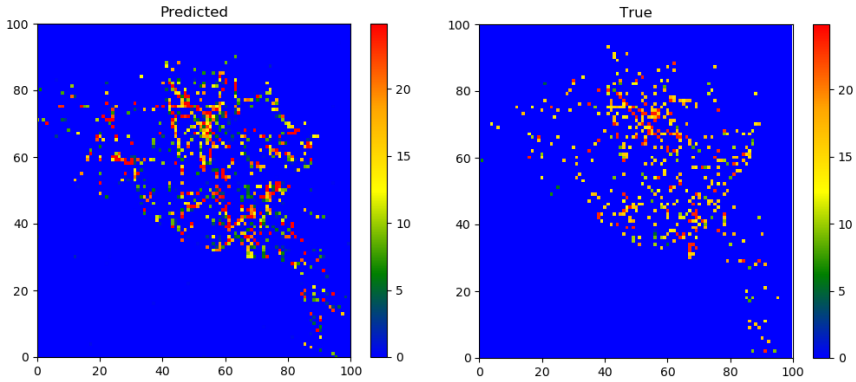


Figure 5.5: A heatmap of the complexity matrices of one sequence of trajectories from the training set. The longitude is represented horizontally and the latitude vertically. True values after 40 minutes (left) and predicted values (right).

airspace size and on the parameter c. To avoid a too high output dimension, the altitude axis is not considered. Hence, each area covers all flight levels and is only defined by the latitude and longitude of its center point. Then, for a given area, the corresponding element of the matrix takes the maximum value of the complexity metric of the aircraft inside this area at timestep t. The complexity values are then scaled by a logarithmic function $x \mapsto \log(1 + x/T_h)$, where T_h is a fixed threshold, to obtain a more uniform distribution. The $\boldsymbol{H}_t$ matrices are then flattened as c^2 vectors to be considered by the machine learning model.

At a given time t, the input vector $\boldsymbol{x}_t$ is built by concatenating all the aircraft states currently in the airspace, ordered by increasing longitude (and by increasing latitude if the longitude is the same). This means that a given flight will not occupy the same position in $\boldsymbol{x}_t$ for every t. However, a given position in $\boldsymbol{x}_t$ will be occupied by aircraft flying at roughly the same geographical coordinates such that the positional information is preserved in the structure of the inputs. To get a fixed-size input vector, the dimension of $\boldsymbol{x}_t$ is set to the maximum number of simultaneous aircraft states in the dataset multiplied by 8 (the length of an aircraft state). If the number of states is lower than the maximum, the remaining elements of $\boldsymbol{x}_t$ are padded with zeros.

Now, the training set can be defined as pairs of input sequences $\boldsymbol{X}_{t_0}$ and output sequences $\boldsymbol{Y}_{t_0}$ where $\boldsymbol{X}_{t_0} = \begin{bmatrix} \boldsymbol{x}_{t_0} & \cdots & \boldsymbol{x}_{t_0+t_{in}-1} \end{bmatrix}$ and $\boldsymbol{Y}_{t_0} = \begin{bmatrix} \boldsymbol{H}_{t_0+t_{in}} & \cdots & \boldsymbol{H}_{t_0+t_{in}+t_{pred}-1} \end{bmatrix}$, where t_{in} is the length of the input sequence, while t_{pred} is the time horizon of the prediction (length of the output sequence). In the following, the dataset will be randomly divided between training examples (90% of the dataset) and validation examples (10% of the dataset).

5.4.3.2 Architecture of the model

The proposed encoder-decoder LSTM model[1] predicts a sequence of complexity matrices using a sequence of concatenated aircraft states as input (see Figure 5.3). Consider the encoder network first. The input sequence is fed to one or several 1-D convolutional layers. The output of the 1-D convolutional layers is fed to several LSTM layers. The hidden state of the last LSTM layer is the *encoding sequence*, which is the only encoder information that will be passed to the decoder network. The decoder network consists of several LSTM layers (the hidden states of

[1]The code is available here: `https://github.com/lshigarrier/PFE`.

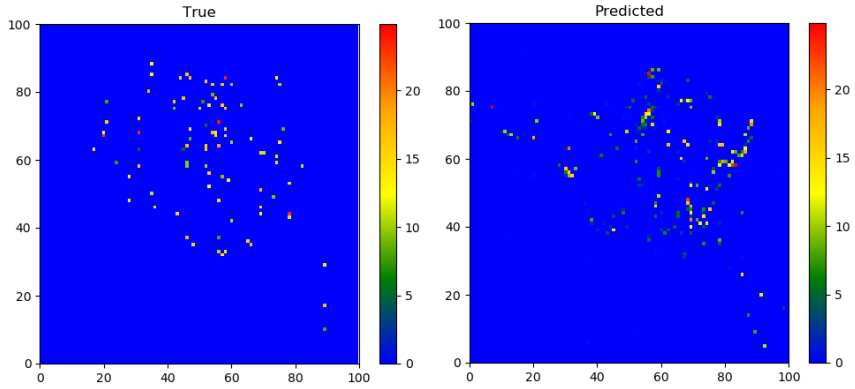


Figure 5.6: A heatmap of the complexity matrices of one sequence of trajectories from the validation set. The longitude is represented horizontally and the latitude vertically. True values after 40 minutes (left) and predicted values (right)

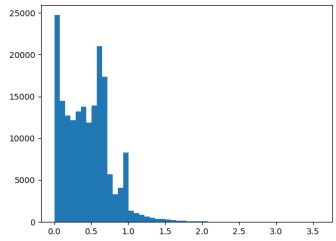
these layers are initialized with the encoding sequence) followed by several dense layers, which output a vector of dimension c^2 corresponding to the matrix of complexity values.

Another characteristic of the proposed model is that the decoder network is implementing a technique called “teacher forcing". This means that, during training, the decoder network takes as an additional input the true output sequence, but shifted by one timestep. In other words, the decoder network also receives as input the output it should have predicted at the previous timestep. During the inference process, the decoder network simply considers the prediction it has made at the previous timestep.

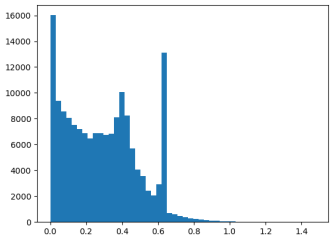
Several variations of the dataset and of the network are tested in this section. First, a dataset where the complexity matrices $\boldsymbol{H}_t$ are smoothed using a Gaussian kernel is considered. Gaussian smoothing was performed using a 3×3 approximate Gaussian kernel $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ to smooth the values of the complexity metric across the airspace. Smoothed data is then used to train the model. Second, the effect of the length of aircraft trajectory in the computation of the linear dynamical system metric is investigated. More precisely, p points corresponding to future positions are added as well as p points corresponding to past positions of each aircraft. This experiment aims at modelling the uncertainty of the aircraft's positions along the curvilinear abscissa. In particular, the effect of the choice of the parameter p on the performances of the model is studied. The entire framework is summarized in Figure 5.4.

The hyperparameters of the implemented model are as follows. The convolutional layer has a kernel of width $h = 3$ and $n_f = 512$ filters. The encoder's LSTM layer has a hidden dimension of 128. The number of time steps for the encoder network is fixed at $t_{in} = 160$. Since one time step is 5 seconds, this corresponds to 40 minutes, i.e., the model has access to the last 40 minutes of traffic to compute its prediction. The decoder's LSTM layer has a hidden dimension of 128 and is followed by one dense layer with output dimension $10,000$ (since the complexity matrix is a 100×100). The dataset (input and output) is linearly normalized between 0 and 1. The ReLU activation function is used for the last dense layer. The sequence predicted by the decoder network is set to $t_{pred} = 160$, which corresponds to the next 40 minutes of complexity values. For end-to-end model training, the Adam optimizer [140] is used with a learning rate $\epsilon = 10^{-3}$, and hyperparameters $\rho_1 = 0.9$, and $\rho_2 = 0.999$. The loss function is the Mean Squared Error defined as

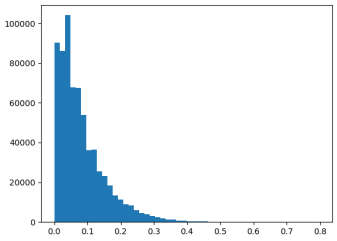
$$MSE(\hat{\boldsymbol{H}}, \boldsymbol{H}) = \frac{1}{c^2} \sum_{i=1}^{c^2} (\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i)^2. \tag{5.14}$$



(a) with trajectory extension of $p = 10$ states before and after the reference state (to take into account uncertainty in the measurements). No smoothing applied. 182,982 non-zero values with MAE = 0.4



(b) with trajectory extension of $p = 2$ states before and after the reference state. No smoothing applied. 161,130 values with MAE = 0.3



(c) with no trajectory extension and Gaussian smoothing. 678,240 values with MAE = 0.08

Figure 5.7: Distribution of non-zero absolute errors between prediction and ground truth over the validation set.

The mini-batch size is set to 128 and the model is trained over 100 epochs. The 8,011 simulated trajectories of the dataset are spread across 3,025 time steps. Since the model requires pairs of sequences of respective lengths t_{in} and t_{pred}, a set of $3,025 - t_{in} - t_{pred} = 2,705$ sequences was built. Among these 2,705 sequences, 2,434 were part of the training set, and 271 were used to validate the model.

5.4.4 Results and Discussion

Figure 5.5 shows the true and predicted complexity matrices, as heat maps, for a sequence from the training set. The horizontal axis corresponds to the longitude and the vertical axis is the latitude. The color indicates the complexity value associated to each area of the airspace. The complexity values have been rescaled to their original distribution. Figure 5.5 shows that the model is able to predict the future complexity of the airspace, 40 minutes ahead of time, with a reasonable accuracy. Figure 5.6 depicts the true and predicted complexity heat maps for a validation sequence, i.e., that the network did not encounter during training. It can be seen that the model is less accurate with validation examples. This may be due to overfitting, although the validation loss curve seems to converge to the same value as the training loss. Several regularization methods were tested, including L_2 regularization, dropout, and Batch Normalization, but they significantly decreased the overall performance. The small size of the dataset (2,705 sequences) may explain why the model struggles to generalize, as well as the absence of the vertical information in the target output.

To quantify the performance of the model, the absolute error between the predicted density and the true density over the validation set is used. Given that the validation set contains 271 examples, each example containing 10,000 density values, a total of 2,710,000 values of absolute errors are obtained, among which 182,982 errors are nonzero (which is expected since the airspace is mostly empty). After removing the points with zero error, the histogram of the non-zero absolute errors is plotted for three variants of the model, along with the corresponding Mean Absolute Error (MAE). Figures 5.7a and 5.7b show the histograms for the model with $p = 10$ (the same model used earlier in this Section) and $p = 2$, respectively. Figure 5.7c shows the distribution of the non-zero errors when the output matrices are smoothed with a Gaussian kernel. Comparing the two variants of the linear dynamical system metric (with $p = 2$

or $p = 10$), the two distributions feature two modes other than zero. The distributions have a decreasing overall trend but with a certain number of errors around or above 1. However, with $p = 2$, the errors are spread across a smaller range and the MAE is smaller. There is also a smaller number of non-zero errors overall. This result is coherent with the literature findings that smaller sequences are somewhat easier to predict that longer ones, notably due to known gradient vanishing or exploding issues. When the complexity measures are smoothed, the MAE is five times smaller than non-smoothed output, reaching a value of 0.08. The total number of non-zero errors is higher, which is due to the fact that there are more non-empty areas among the smoothed outputs. The prediction task with a smoothed dataset seems to be easier for the encoder-decoder LSTM model.

5.5 Transformer Model

This section expands the encoder-decoder LSTM model introduced in the previous section into a Transformer model. First, the Transformer framework is described. Then, the main contribution is elaborated in section 5.5.2: the variational moment propagation method is derived for the Transformer architecture, which is referred to as the Bayesian Transformer. Finally, the Bayesian Transformer is evaluated on a traffic prediction task. Preliminary results are provided and future work are discussed.

5.5.1 The Transformer framework

The Transformer framework is the state-of-the-art paradigm for processing sequential data. It was introduced by Vaswani *et al.* [4] in order to replace the recurrent encoder-decoder models previously used in automatic translation. The Transformer framework 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 attention blocks. The attention block has two 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 m is smaller than the representation dimension h;

The Transformer model also uses positional encoding (which provides 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.

Thereafter, the main architecture blocks of the Transformer are briefly described.

Attention head. The core of the Transformer framework is the attention head f_{head}. It takes in input a sequence $\boldsymbol{z}$ of length m. Each element of the sequence is a vector of dimension h, such that $\boldsymbol{z}$ is seen as a $h \times m$ matrix. The head f_{head} also takes in input three matrices of learnable parameters $\boldsymbol{\theta}_{head} = (\boldsymbol{w}_Q, \boldsymbol{w}_K, \boldsymbol{w}_V)$ where $\boldsymbol{w}_Q$, $\boldsymbol{w}_K$ and $\boldsymbol{w}_V$ are $h \times h$ matrices. In the literature, $\boldsymbol{w}_Q$ are called the “queries" parameters while $\boldsymbol{w}_K$ are the “keys" parameters. Similarly, $\boldsymbol{w}_V$ is often referred to as the “values" parameters. The attention head performs the

following operations:

$$f_{head}(\boldsymbol{z}, \boldsymbol{\theta}_{head}) = \boldsymbol{w}_V \boldsymbol{z} \quad s\left(\frac{\boldsymbol{z}^\top \boldsymbol{w}_K^\top \boldsymbol{w}_Q \boldsymbol{z}}{\sqrt{h}}\right) \tag{5.15}$$

The function s is the softmax function applied column-wise, such that each column of the $m \times m$ matrix $\boldsymbol{A}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K) = s\left(\frac{\boldsymbol{z}^\top \boldsymbol{w}_K^\top \boldsymbol{w}_Q \boldsymbol{z}}{\sqrt{h}}\right)$ adds up to 1. An attention head is basically a linear layer $\boldsymbol{w}_V \boldsymbol{z}$ whose columns are combined linearly in m different manners according the weights given by the columns of $\boldsymbol{A}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K)$. The so-called "attention mechanism" happens in $\boldsymbol{z}^\top \boldsymbol{w}_K^\top \boldsymbol{w}_Q \boldsymbol{z}$, where each element of the input sequence $\boldsymbol{z}$ is compared to every other element of the sequence (including itself). Thus, the i-th column of $\boldsymbol{A}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K)$ is a vector of m weights, such that $\boldsymbol{A}_{ji}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K)$ represents how much attention should be given to the j-th element of $\boldsymbol{z}$ from the point-of-view of the i-th element.

Note that the output $f_{head}(\boldsymbol{z}, \boldsymbol{\theta}_{head})$ is also a $h \times m$ matrix.

Attention block An attention block f_{att} is the composition of an attention head f_{head} with an affine layer f_{FC} and two layer normalization operations f_{norm}. The affine layer has parameters $\boldsymbol{\theta}_{FC} = (\boldsymbol{w}_{FC}, \boldsymbol{b}_{FC})$ such that:

$$f_{FC}(\boldsymbol{z}, \boldsymbol{\theta}_{FC}) = \mathrm{ReLU}(\boldsymbol{w}_{FC} \boldsymbol{z} + \boldsymbol{b}_{FC}), \tag{5.16}$$

where $\boldsymbol{w}_{FC}$ is a $h \times h$ matrix, and $\boldsymbol{b}_{FC}$ is a h-dimensional vector added to each column of $\boldsymbol{w}_{FC} \boldsymbol{z}$.

A layer normalization operation has parameters $\boldsymbol{\theta}_{norm} = (\boldsymbol{w}_{norm}, \boldsymbol{b}_{norm})$ where $\boldsymbol{w}_{norm}$ and $\boldsymbol{b}_{norm}$ have dimension h, such that:

$$f_{norm}(\boldsymbol{z}, \boldsymbol{\theta}_{norm}) = \boldsymbol{w}_{norm} \odot \left(\frac{\boldsymbol{z} - \langle \boldsymbol{z} \rangle}{\sqrt{\mathrm{var}[\boldsymbol{z}]}}\right) + \boldsymbol{b}_{norm}, \tag{5.17}$$

where $\langle \boldsymbol{z} \rangle$ is the m-dimensional vector of the mean values of $\boldsymbol{z}$ along each column, while $\mathrm{var}[\boldsymbol{z}]$ is the m-dimensional vector of the variances of $\boldsymbol{z}$ along each column. Every operation, including the element-wise product $\odot$, is applied independently to each column.

Finally, f_{att} can be written with parameters $\boldsymbol{\theta}_{att} = (\boldsymbol{\theta}_{head}, \boldsymbol{\theta}_{norm}^{(1)}, \boldsymbol{\theta}_{norm}^{(2)}, \boldsymbol{\theta}_{FC})$ as:

$$f_{att}(\boldsymbol{z}, \boldsymbol{\theta}_{att}) = f_{FC}\left(f_{norm}\left(f_{head}\left(f_{norm}\left(\boldsymbol{z}, \boldsymbol{\theta}_{norm}^{(1)}\right), \boldsymbol{\theta}_{head}\right) + \boldsymbol{z}, \boldsymbol{\theta}_{norm}^{(2)}\right), \boldsymbol{\theta}_{FC}\right) + \boldsymbol{z}, \tag{5.18}$$

where the input $\boldsymbol{z}$ is added after applying f_{head} and f_{FC} to avoid the vanishing gradient issue. This is known as residual connection. In practice, an attention block will contain several attention heads with independent parameters. The heads can be computed in parallel, then concatenated and projected back to a $h \times m$ matrix using a linear layer, before being fed to the second layer normalization operation.

Transformer Encoder A Transformer encoder is built by stacking L attention blocks. This architecture can be used for classification of sequential data, as well as for computer vision [191].

First, the input sequence $\boldsymbol{x}$ of dimension $d \times m$ can optionally be preprocessed to reduce the representation dimension and obtain a $h \times m$ matrix $\boldsymbol{z}_0$, using one or several affine layers for example. In the case of Natural Language Processing, this can be done with word embedding. Since the Transformer architecture is not recurrent, a positional embedding must be added to $\boldsymbol{z}_0$ to provide the model with the temporal information (in the case of time series) [4]. Then, the matrix $\boldsymbol{z}_0$ can be fed to the L attention blocks, such that $\boldsymbol{z}_k = f_{att}\left(\boldsymbol{z}_{k-1}, \boldsymbol{\theta}_{att}^{(k)}\right)$. Finally, a matrix $\boldsymbol{z}_L$ is obtained.

To perform classification, the average of each row of $\boldsymbol{z}_L$ is computed, thus obtaining a h-dimensional vector $\boldsymbol{z}$. The vector $\boldsymbol{z}$ is used in input of an affine layer yielding a c-dimensional vector of logits $\hat{\boldsymbol{y}}$, where c is the number of classes. The softmax function can be applied on $\hat{\boldsymbol{y}}$ to provide the probabilities $s(\hat{\boldsymbol{y}})$ of each class.

If a decoder is appended after the encoder, the “keys" and “values" matrices of the last attention block are stored i.e., $\boldsymbol{w}_K^{(L)}\boldsymbol{z}_{L-1}$ and $\boldsymbol{w}_V^{(L)}\boldsymbol{z}_{L-1}$, as the encoding vectors to be used by the decoder.

Transformer Encoder-Decoder In order to perform sequence-to-sequence prediction or classification, one can use an encoder-decoder architecture, similar to the LSTM encoder-decoder described in section 5.4.2.

The Transformer decoder has the same architecture of the encoder, with minor differences. The decoder consists of an optional embedding, a positional encoding, L attention blocks, and a fully-connected layer to perform the prediction or the classification. During training, the decoder will receive in input the ground truth sequence. The decoder goal is to predict this sequence with a delay of one timestep. In other words, if the decoder receives $\boldsymbol{x}^0, \boldsymbol{x}^1$ in input, it will predict $\boldsymbol{x}^2$. Then, the ground truth $\boldsymbol{x}^2$ is given in input along $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$, and the decoder will predict $\boldsymbol{x}^3$. The first element of the sequence $\boldsymbol{x}^0$ is a “start-of-sequence token". During inference, the decoder uses its own predictions as inputs in order to predict the next element of the sequence.

The differences with the encoder occur in the attention blocks, which are no longer “self-attention". Indeed, the decoder's attention blocks only compute the “queries" $\boldsymbol{w}_Q^{(k)}\boldsymbol{z}_{k-1}$, then reuse the “keys" and “values" $\boldsymbol{w}_K^{(L)}\boldsymbol{z}_{L-1}$ and $\boldsymbol{w}_V^{(L)}\boldsymbol{z}_{L-1}$ from the encoder. The computation is the same as described in equation 5.15.

Another difference comes from the fact that, when predicting the i-th timestep, the decoder should not be able to use information from the subsequent timesteps. This is because, during inference, the decoder will predict the output sequence in an auto-regressive manner thus, when computing the i-th timestep, the subsequent timesteps have not been computed yet. To prevent the decoder to access this information, a mask is applied to the weights $\boldsymbol{A}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K)$ computed in equation 5.15. In the i-th column of $\boldsymbol{A}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K)$, the weights corresponding to timesteps strictly larger than i are set to 0, i.e., $\boldsymbol{A}_{ji}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K) = 0$ for $j > i$. The other weights are re-normalized to add up to 1.

This concludes the presentation of the Transformer framework. Now, the extension of this framework with the variational moment propagation method can be laid out. In the following, the Transformer framework described in this section will be referred to as the *deterministic Transformer*, while the Transformer extended with moment propagation will be named the *Bayesian Transformer*.

5.5.2 Moment propagation for Transformers

Deploying machine learning solutions to critical systems, such as air traffic control or healthcare, hinges on building trustworthy models. Trustworthiness requires the robustness of the models to noisy and unexpected input and the model's ability to justify its decisions. These two properties depend on the ability of a model to quantify its own uncertainty. A natural measure of uncertainty is obtained by estimating the posterior distribution over the learnable parameters using Bayesian inference. This section develops the variational moment propagation framework for Transformer models. The moment propagation method has already been successfully applied to convolutional neural networks [192] and to recurrent neural networks [193].

Since the variational inference method has been described in section 2.3.2, it will be directly applied to Transformers in this section. The learnable parameters $\boldsymbol{\theta}$ of the Transformer model are now seen as random variables. Let $\boldsymbol{D}$ be the training dataset. The goal is to compute the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{D})$ of the parameters given the data. To achieve this goal, $p(\boldsymbol{\theta}|\boldsymbol{D})$ is approximated by a variational family $q(\boldsymbol{\theta}|\boldsymbol{\xi})$ parameterized by $\boldsymbol{\xi}$. Variational inference consists in minimizing the negative evidence lower bound (ELBO):

$$-\text{ELBO}(\boldsymbol{\xi}) = \mathbb{E}_{\boldsymbol{\theta}|\boldsymbol{\xi}}\left[-\log p(\boldsymbol{D}|\boldsymbol{\theta})\right] + \text{KL}\left[q(\boldsymbol{\theta}|\boldsymbol{\xi})||p(\boldsymbol{\theta})\right]. \tag{5.19}$$

The moment propagation method consists in assuming that all distributions are Gaussian. Thus, the prior $p(\boldsymbol{\theta})$, the variational family $q(\boldsymbol{\theta}|\boldsymbol{\xi})$ and the likelihood $p(\boldsymbol{D}|\boldsymbol{\theta})$ are all Gaussian. With this assumption, the distributions are entirely determined by their mean vector and their covariance matrix. Thus, the variational parameters $\boldsymbol{\xi}$ are the mean and variance of $\boldsymbol{\theta}$, which will be written $\boldsymbol{\xi} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

In practice, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ will be the learnable parameters of the Bayesian Transformer that will be optimized with gradient descent. If $\boldsymbol{\mu}$ has dimension h then $\boldsymbol{\Sigma}$ has dimension h^2. To avoid the quadratic increase in the number of learnable parameters and keep the memory impact of the moment propagation method reasonable, the covariance matrix of every distribution is assumed to be diagonal. Thus, the Bayesian Transformer has two times more parameters than the deterministic Transformer. In the following, $\boldsymbol{\Sigma}$ is considered to have the same shape[2] as $\boldsymbol{\mu}$, and corresponds to the diagonal elements of the covariance matrix. More generally, the parameters $\boldsymbol{\theta}$ are all assumed to be independent from each other, even when they belong to the same layer.

In section 2.3.1, it was shown that the negative log-likelihood can be written as:

$$-\log p(\boldsymbol{D}|\boldsymbol{\theta}) = -\sum_{i=1}^{n} \log p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta}). \tag{5.20}$$

Since $p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta})$ is Gaussian, it follows that[3]:

$$-\log p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{2}\left((\boldsymbol{y}_i - \boldsymbol{\mu}_{\boldsymbol{y}}(\boldsymbol{x}_i, \boldsymbol{\theta}))^\top \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1}(\boldsymbol{x}_i, \boldsymbol{\theta})\,(\boldsymbol{y}_i - \boldsymbol{\mu}_{\boldsymbol{y}}(\boldsymbol{x}_i, \boldsymbol{\theta})) + \log|\boldsymbol{\Sigma}_{\boldsymbol{y}}(\boldsymbol{x}_i, \boldsymbol{\theta})|\right), \tag{5.21}$$

where $\boldsymbol{\mu}_{\boldsymbol{y}}(\boldsymbol{x}_i, \boldsymbol{\theta})$ and $\boldsymbol{\Sigma}_{\boldsymbol{y}}(\boldsymbol{x}_i, \boldsymbol{\theta})$ are the moments of the conditional distribution over the output $\boldsymbol{y}_i$ given the input $\boldsymbol{x}_i$ and the parameters $\boldsymbol{\theta}$. Since $\boldsymbol{\theta}$ is entirely defined by $\boldsymbol{\xi} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$, so are $\boldsymbol{\mu}_{\boldsymbol{y}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{y}}$ for a given $\boldsymbol{x}_i$. To compute $\boldsymbol{\mu}_{\boldsymbol{y}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{y}}$ as function of $\boldsymbol{\xi}$, the two moments $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are propagated from the parameters $\boldsymbol{\theta}$ to the output $\boldsymbol{y}$.

In the following paragraphs, the moment propagation is described for every layer of the Transformer model. For every layer, the mean and variance of the input are denoted by $\boldsymbol{\mu}_{\boldsymbol{z}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{z}}$, while $\boldsymbol{\mu}_f$ and $\boldsymbol{\Sigma}_f$ are the mean and variance of the output of the layer. The square operations are always computed element-wise.

Affine layer An affine layer has two learnable parameters $\boldsymbol{w}_{FC}$ and $\boldsymbol{b}_{FC}$, with mean and variance $\boldsymbol{\mu}_w$, $\boldsymbol{\Sigma}_w$ and $\boldsymbol{\mu}_b$, $\boldsymbol{\Sigma}_b$ respectively. Note that $\boldsymbol{\mu}_w$, $\boldsymbol{\Sigma}_w$ are matrices, while $\boldsymbol{\mu}_b$, $\boldsymbol{\Sigma}_b$ are vectors. The mean $\boldsymbol{\mu}_f$ of the output of the layer is given by:

$$\boldsymbol{\mu}_f = \boldsymbol{\mu}_w \boldsymbol{\mu}_{\boldsymbol{z}} + \boldsymbol{\mu}_b. \tag{5.22}$$

To compute the variance $\boldsymbol{\Sigma}_f$ of the output, Proposition 2 from [192] is applied and yields:

$$\boldsymbol{\Sigma}_f = \boldsymbol{\Sigma}_w \boldsymbol{\Sigma}_{\boldsymbol{z}} + \boldsymbol{\mu}_w^2 \boldsymbol{\Sigma}_{\boldsymbol{z}} + \boldsymbol{\Sigma}_w \boldsymbol{\mu}_{\boldsymbol{z}}^2 + \boldsymbol{\Sigma}_b. \tag{5.23}$$

[2] which can be either a vector or a matrix.

[3] The additive terms that do not depend on $\boldsymbol{\theta}$ have been removed.

Attention block In the attention head computation given in equation 5.15, the multiplication of two random matrices happens twice: when computing $(\boldsymbol{w}_K \boldsymbol{z})^\top \boldsymbol{w}_Q z$ and when computing $(\boldsymbol{w}_V \boldsymbol{z})\boldsymbol{A}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K)$. This operation is similar to a linear layer. Consider two random matrices $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ with moments $\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1$ and $\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2$. The moments $\boldsymbol{\mu}_f, \boldsymbol{\Sigma}_f$ of the product $\boldsymbol{z}_1 \boldsymbol{z}_2$ are given by:

$$\boldsymbol{\mu}_f = \boldsymbol{\mu}_1 \boldsymbol{\mu}_2, \tag{5.24}$$

and

$$\boldsymbol{\Sigma}_f = \boldsymbol{\Sigma}_1 \boldsymbol{\Sigma}_2 + \boldsymbol{\mu}_1^2 \boldsymbol{\Sigma}_2 + \boldsymbol{\Sigma}_1 \boldsymbol{\mu}_2^2, \tag{5.25}$$

where Proposition 2 from [192] has been used again.

Scaling by the constant $1/\sqrt{h}$ amounts to multiplying the mean by $1/\sqrt{h}$ and the variance by $1/h$.

To compute $\boldsymbol{A}(\boldsymbol{z}, \boldsymbol{w}_Q, \boldsymbol{w}_K)$, the moments have to be propagated through the softmax function. Let $\boldsymbol{z'}$ be the input to the softmax function, with moments $\boldsymbol{\mu}_{z'}$ and $\boldsymbol{\Sigma}_{z'}$. The mean of $s(\boldsymbol{z})$ is:

$$\boldsymbol{\mu}_f = s(\boldsymbol{\mu}_{z'}). \tag{5.26}$$

To propagate the variance, the first-order Taylor approximation of the softmax function is used. Since the off-diagonal terms of the covariance matrix are assumed to be zero, the only required elements of the Jacobian matrix of the softmax are the diagonal elements which are equal to $s(\boldsymbol{\mu}_{z'}) \odot (1 - s(\boldsymbol{\mu}_{z'}))$. Then, the covariance of $s(\boldsymbol{z})$ is:

$$\boldsymbol{\Sigma}_f = \boldsymbol{\Sigma}_{z'} \odot \left(s(\boldsymbol{\mu}_{z'}) \odot (1 - s(\boldsymbol{\mu}_{z'}))\right)^2. \tag{5.27}$$

The moments are propagated through other non-linear activation functions in a similar manner. In the implementation, the ReLU activation function is used for the hidden layers, and the sigmoid function is used for the output layer of the decoder. The mean is propagated by applying the activation function to the mean of the input. Using the Jacobian matrix of the ReLU, the variance is propagated as:

$$\boldsymbol{\Sigma}_f = \boldsymbol{\Sigma}_z \odot \mathbb{1}_{\mathbb{R}_+}(\boldsymbol{\mu}_z),$$

where the indicator function is applied element-wise. For the sigmoid (denoted by f here), the variance is propagated as:

$$\boldsymbol{\Sigma}_f = \boldsymbol{\Sigma}_z \odot \left(f(\boldsymbol{\mu}_z) \odot (1 - f(\boldsymbol{\mu}_z))\right)^2.$$

Layer normalization and residual connection Now, consider the layer normalization operation (equation 5.17). Let $\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}}$ be the moments of $\boldsymbol{w}_{norm}$, and $\boldsymbol{\mu}_{\boldsymbol{b}}, \boldsymbol{\Sigma}_{\boldsymbol{b}}$ be the moments of $\boldsymbol{b}_{norm}$. To simplify the computation, the empirical mean $\langle \boldsymbol{z} \rangle$ and variance $\mathrm{var}[\boldsymbol{z}]$ are treated as constants equal to $\langle \boldsymbol{\mu}_z \rangle$ and $\mathrm{var}[\boldsymbol{\mu}_z]$ respectively. Remember that $\langle \cdot \rangle$ and $\mathrm{var}[\cdot]$ are applied to each column independently. The propagated mean is:

$$\boldsymbol{\mu}_f = \boldsymbol{\mu}_{\boldsymbol{w}} \odot \left(\frac{\boldsymbol{\mu}_z - \langle \boldsymbol{\mu}_z \rangle}{\sqrt{\mathrm{var}[\boldsymbol{\mu}_z]}} \right) + \boldsymbol{\mu}_{\boldsymbol{b}}, \tag{5.28}$$

and the propagated variance is:

$$\boldsymbol{\Sigma}_f = \boldsymbol{\Sigma}_{\boldsymbol{w}} \odot \left(\frac{\boldsymbol{\Sigma}_z}{\mathrm{var}[\boldsymbol{\mu}_z]} \right) + \boldsymbol{\Sigma}_{\boldsymbol{b}}. \tag{5.29}$$

Finally, consider a residual connection around a layer g, i.e., the goal is to compute the mean and variance of $f(\boldsymbol{z}) = g(\boldsymbol{z}) + \boldsymbol{z}$. The mean is:

$$\boldsymbol{\mu}_f = g(\boldsymbol{\mu}_z) + \boldsymbol{\mu}_z.$$

To propagate the variance through the residual connection, the first-order Taylor approximation can be used. Let $\boldsymbol{J}_g$ be the Jacobian matrix of g, while $\boldsymbol{\mu}_g$ and $\boldsymbol{\Sigma}_g$ are the moments of $g(\boldsymbol{z})$. The following holds:

$$\boldsymbol{\Sigma}_f = \boldsymbol{\Sigma}_z + \boldsymbol{\Sigma}_g + 2\left(\boldsymbol{J}_g(\boldsymbol{z}) \odot \left(\boldsymbol{\Sigma}_z + \boldsymbol{\mu}_z^2\right) - \boldsymbol{\mu}_z \odot \boldsymbol{\mu}_g\right).$$

However, this requires to compute the Jacobian matrix of g, which can consist of many operations. To reduce the computational complexity, the extra assumption that $\boldsymbol{z}$ and $f(\boldsymbol{z})$ are independent is made. In this case, the variance is simply:

$$\boldsymbol{\Sigma}_f = \boldsymbol{\Sigma}_z + \boldsymbol{\Sigma}_g. \tag{5.30}$$

In the implementation, equation 5.30 will be used.

Computation of the ELBO and training of the Bayesian Transformer Using the previous paragraphs, it is possible to compute $\boldsymbol{\mu}_{\boldsymbol{y}}(\boldsymbol{x}_i, \boldsymbol{\theta})$ and $\boldsymbol{\Sigma}_{\boldsymbol{y}}(\boldsymbol{x}_i, \boldsymbol{\theta})$ as a function of $\boldsymbol{\xi} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Therefore, the negative log-likelihood can be computed using equations 5.20 and 5.21, which is the first term of the negative ELBO. The second term is the Kullback-Leibler divergence between the variational family and the prior. In this chapter, the standard Gaussian is chosen as prior. Since the variational family is also Gaussian, the KL divergence can be computed explicitly. According to equation 21 in [192], it follows that[4]:

$$\mathrm{KL}\left[q(\boldsymbol{\theta}|\boldsymbol{\xi})||p(\boldsymbol{\theta})\right] = \frac{1}{2}\sum_j \left(\boldsymbol{\mu}_j^2 + \boldsymbol{\Sigma}_j - \log \boldsymbol{\Sigma}_j\right), \tag{5.31}$$

where the index j runs over all learnable parameters in $\boldsymbol{\xi} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. In practice, since the KL divergence acts as a regularization term, it is weighted with a hyperparameter $\lambda > 0$. Let $\hat{\boldsymbol{y}} = (\boldsymbol{\mu}_{\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{y}})$ be the moments of the predicted output. The final loss function used to train the Bayesian Transformer is:

$$\mathcal{L}(\boldsymbol{\mu}_{\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{y}}, \boldsymbol{y}, \xi) = \frac{1}{n}\sum_{i=1}^{n}\sum_k \left(\frac{\left(\boldsymbol{y}_{i,k} - \boldsymbol{\mu}_{\boldsymbol{y}_{i,k}}\right)^2}{\boldsymbol{\Sigma}_{\boldsymbol{y}_{i,k}}} + \log \boldsymbol{\Sigma}_{\boldsymbol{y}_{i,k}}\right) + \lambda \sum_j \left(\boldsymbol{\mu}_j^2 + \boldsymbol{\Sigma}_j - \log \boldsymbol{\Sigma}_j\right), \tag{5.32}$$

where the index i runs over the training set (or the batch), the index j runs over the learnable parameters, and the index k runs over the element of the output y.

Let $\boldsymbol{\xi}^*$ be the variational parameters after training. During inference, the predictive distribution $p(\boldsymbol{y}|\boldsymbol{x}^*, \boldsymbol{D})$ is approximated as a Gaussian whose mean and variance are given by $\boldsymbol{\mu}_{\boldsymbol{y}}(\boldsymbol{x}^*, \boldsymbol{\xi}^*)$ and $\boldsymbol{\Sigma}_{\boldsymbol{y}}(\boldsymbol{x}^*, \boldsymbol{\xi}^*)$. This moments are obtained by propagating $\boldsymbol{\xi}^*$ through the model as during training. The mean $\boldsymbol{\mu}_{\boldsymbol{y}}(\boldsymbol{x}^*, \boldsymbol{\xi}^*)$ is the prediction of the model for the input $\boldsymbol{x}^*$, while the variance $\boldsymbol{\Sigma}_{\boldsymbol{y}}(\boldsymbol{x}^*, \boldsymbol{\xi}^*)$ provides a quantitative measure of uncertainty in the output decision. Larger variance corresponds to higher uncertainty. An intuitive explanation is the following. Assume that the regularization term is removed (i.e., $\lambda = 0$) and that the model squared error $\left(\boldsymbol{y}_{i,k} - \boldsymbol{\mu}_{\boldsymbol{y}_{i,k}}\right)^2$ is fixed and the same for all training examples. Then, the optimal value for $\boldsymbol{\Sigma}_{\boldsymbol{y}_{i,k}}$ is precisely

[4] The additive terms that do not depend on $\boldsymbol{\xi}$ have been removed.

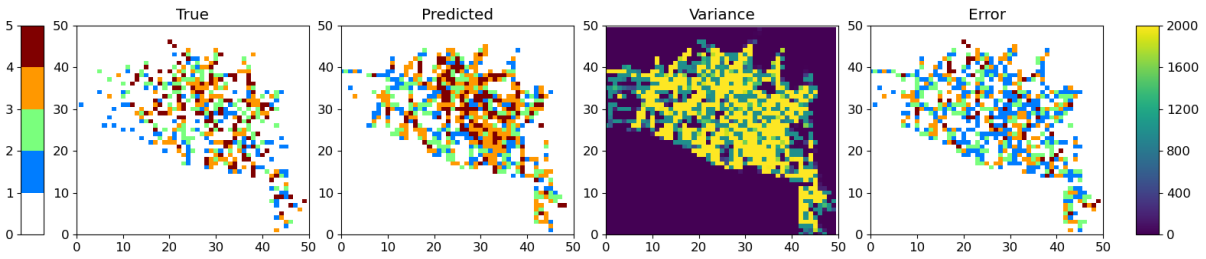


Figure 5.8: Prediction of the model on one example of the test set of the simulated trajectories dataset. From left to right: ground truth for each cell of the airspace, predicted classes, predictive variance, absolute error between ground truth and predicted classes.

$\left(\boldsymbol{y}_{i,k} - \boldsymbol{\mu}_{\boldsymbol{y}_{i,k}}\right)^2$. Thus, during training, the model is learning to calibrate its uncertainty $\boldsymbol{\Sigma}_{\boldsymbol{y}_{i,k}}$ to its actual performance on the training set given by the squared error. During inference, $\boldsymbol{y}_{i,k}$ is unknown, but $\boldsymbol{\Sigma}_{\boldsymbol{y}_{i,k}}$ provides an estimate of the squared error. A large variance $\boldsymbol{\Sigma}_{\boldsymbol{y}_{i,k}}$ means that the model is expected to have provided a poor prediction and should not be trusted. Conversely, a small variance means that the model is confident in its prediction.

5.5.3 Experiments

In this section, some preliminary results are provided about the Bayesian Transformer. The model has been evaluated on three datasets: the simulated trajectories introduced in section 5.4.3, MNIST, and the PeMS-SF dataset. Overall, the Bayesian Transformer is harder to train than the encoder-decoder LSTM. It has more hyperparameters and tends to be more unstable during training.

Encoder-decoder Transformer for airspace congestion prediction A Bayesian encoder-decoder Transformer model[5] is trained on the same dataset of synthetic trajectories used for the encoder-decoder LSTM model in section 5.4.3. To simplify the problem, the regression task is replaced by a classification task among five classes: no complexity, low, medium, high, and very high complexity. The thresholds between the classes are chosen such that the classes low, medium, high and very high are balanced in the training set. However, the “no complexity" class remains by far the largest class in the training set. Thus, to alleviate the effects of the unbalanced dataset, the classes are weighted in the loss function and a focal term is used [194].

Nonetheless, the Bayesian Transformer struggles to learn on this dataset. The predictions and the variance tend to be almost constant accross the testing set. This may be due to the unbalance of the dataset as well as the small numbers of trajectories in the training set (around 8,000) compared with the number of learnable parameters in the Bayesian Transformer (around 18 million). An example of a prediction over the test set is provided in figure 5.8. This figure has been obtained with a model consisting of 2 attention blocks in the encoder and the decoder, 2 heads per attention block, a representation dimension of 512, no regularization in the ELBO ($\lambda = 0$), Adam optimizer [140] (learning rate $= 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$) and 30 epochs.

[5]The code is available here: `https://github.com/lshigarrier/atfp_vdp`.

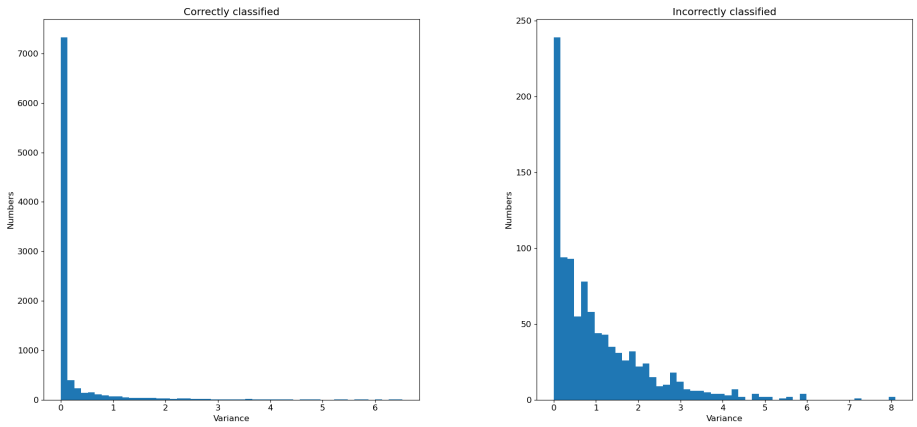


Figure 5.9: Histograms of the predictive variances for the correctly (left) and incorrectly (right) classified examples over the test set of MNIST.

5.5.3.1 Encoder Transformer

Now, a Bayesian encoder Transformer is trained on MNIST[6]. A simple Visual Transformer architecture is used [191]. The model consists of 2 attention blocks in the encoder and the decoder, 2 heads per attention block, a representation dimension of 256, a regularization factor in the ELBO equal to $\lambda = 10^{-8}$, Adam optimizer (learning rate $= 10^{-2}$) and 50 epochs. The model is much smaller since it contains 18,740 parameters. The model achieves an accuracy of 90.00%. To evaluate the uncertainty quantification, the predictive variance for the correctly classified examples of the test set is compared with the predictive variance of the incorrectly classified examples. The results are provided in figure 5.9, where the variance is observed to be significantly larger on incorrectly classified examples than on correctly classified examples.

Finally, the encoder Transformer model is trained on the PeMS-SF dataset[7]. This dataset contains traffic data from San Francisco collected by the Californian Performance Measurement System (PeMS). The data comes from 963 captors that measure the occupancy rate (between 0 and 1) of various freeways every 10 minutes. The PeMS-SF dataset contains 15 months of traffic, where each input sequence corresponds to one day. Thus, an input sequence has dimension 963×144 (144 measures per day for each captor). The goal of the model is to predict the day of the week given an input sequence corresponding to one day.

Two models are trained[8]: a deterministic Transformer and a Bayesian Transformer. The deterministic model is trained with 2 attention blocks in the encoder and the decoder, 9 heads per attention block, the representation dimension is the same as the input (963), Adam with learning rate of 10^{-3} and 30 epochs. The model contains 6.5 million parameters and achieves an accuracy of 78.03%. The Bayesian model is trained with the same hyperparameters, a regularization factor of $\lambda = 10^{-7}$ and 50 epochs. The model contains 13 million parameters and achieves an accuracy of 75.72%.

Both models are evaluated against increasing levels of Gaussian noise. The predictive variance is computed by averaging the variance of the predicted class over the test set. The results are presented in figure 5.10. First, it can be seen that the Bayesian model exhibits the same robust-

[6]The code is available here: `https://github.com/lshigarrier/atfp_vdp`.
[7]`http://www.timeseriesclassification.com/description.php?Dataset=PEMS-SF`
[8]The code is available here: `https://github.com/lshigarrier/transformer_vmp`.

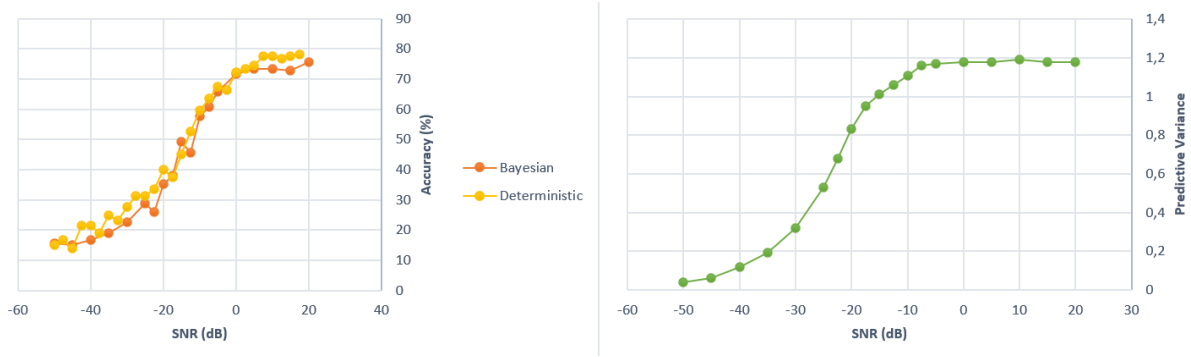


Figure 5.10: Left: accuracy of the deterministic and Bayesian models on the test set of PeMS-SF as a function of the signal-to-noise ratio. Right: predictive variance of the Bayesian model on the test set of PeMS-SF as a function of the signal-to-noise ratio.

ness against noise as the deterministic model. Contrary to what was expected, the predictive variance increases with SNR. The cause of this phenomenon is not known. Nonetheless, a clear monotonic behavior is observed between the SNR and the predictive variance, which shows that the model is aware of the level of noise it is subjected to. In this case, the variance measures the confidence of the model in its prediction instead of its uncertainty.

Due to difficulties to train the Bayesian Transformer, this section only contains preliminary results. Ablation studies should be conducted to find the best set of hyperparameters of the Bayesian model and to improve the trainability of the model. Then, the model should be evaluated against adversarial noise, and its predictive variance should be compared with the variance obtained against Gaussian noise. Finally, the performance of the Bayesian Transformer could be compared with other methods of uncertainty quantification such as Bayes-by-Backprop [127] or the methods relying on dropout [128].

5.6 Conclusion

In this chapter, several aspects of Air Traffic Management have been reviewed: flow prediction, complexity metrics, and machine learning models for time series prediction. Then, two models have been presented: an encoder-decoder LSTM model for airspace congestion prediction, and a Bayesian Transformer relying on the variational moment propagation method.

Even if the encoder-decoder LSTM was able to achieve satisfactory results, the synthetic dataset used for congestion prediction was certainly too small and did not exhibit clear patterns to be learned, while the task was very challenging since the model had to predict a complexity value for every cell of the airspace. Moreover, the Bayesian Transform was found to be difficult to train, even if some preliminary results have been achieved on simpler datasets.

In a broader perspective, a future work could try to bridge the variational moment propagation with the methods developed in the previous chapters based on concepts from information geometry. As for now, the moment propagation method has only been validated experimentaly. Even if the variational inference method is well-understood, the exact interpretation of the predictive variance of the Bayesian models remains unclear. In particular, there are no guarantees that the predictive variance can be trusted and cannot be targeted by adversaries to instill overconfidence or underconfidence. The various behaviors of the predictive variance against Gaussian noise, adversarial attacks, or out-of-distribution examples could also be studied, both empirically and theoretically, to understand how neural networks perceive these different threats. Providing a

trustworthy and guaranteed quantification of the uncertainty will be required to deploy machine learning solutions with greater autonomy, in particular in safety-critical aspects of Air Traffic Management.

Chapter 6

Conclusion and Perspectives

6.1 General Conclusion

In this thesis, the adversarial robustness of machine learning models has been studied with the aim to contribute to the development of trustworthy machine learning for civil aviation and other safety-critical fields. The thesis focused on the information geometry theory and its application to adversarial robustness.

In chapter 3, an empirical defense against L_2 attacks relying on information geometry was proposed. The Fisher metric between the predicted distributions of the model is pulled back to the input space. This pullback metric quantifies the distance between input examples as seen by the model itself. The model is regularized to make the pullback metric as close as possible to the Euclidean metric, which is the distance used by the L_2 adversary.

Chapter 4 explored several research avenues connecting information geometry and machine learning robustness. A model for recurrent neural network as stochastic differential equation was introduced, and the curvature of the predicted trajectory of a simple linear recurrent network was computed. Then, the kernel foliation of a recurrent network was visualized on a simple nonlinear regression task. In a second part, the data leaf hypothesis [1] was investigated with three experiments about how adversarial examples behave with respect to the data leaf, about the behavior of trajectories following the eingenvalues of the local data matrix, and about the behavior of the data leaf when the same model is trained with a normal dataset, a robustified dataset, and with adversarial training. All these experiments lead to question the relevance of the data leaf hypothesis. Nonetheless, the experiments of this chapter motivate the continuation of research to apply information geometry for machine learning robustness.

Finally, two applications are developed in chapter 5. First, a prediction framework for airspace complexity using an encoder-decoder recurrent model was introduced. This framework is improved, both in terms of accuracy and trustworthiness, by using the Transformer architecture and the variational moment propagation method for uncertainty quantification.

Industries, and the society at large, are becoming more and more dependent on machine learning tools, and this trend is accelerating. As discussed in chapter 2, the performance of machine learning models should not hide the fact that their foundations are not understood. To this day, there is still no consensus on how machine learning models learn and why they are so brittle when subjected to small well-chosen perturbations. The tension between the hype about machine learning performance on one hand, and the lack of in-depth understanding about these models on the other hand, beg several questions. How is it possible to trust a technology that becomes

more and more opaque, and where small areas of *catastrophic instability* are hiding? Since "opening the black-box" of machine learning models is impossible, how to produce quantified safety assessments?

Verification methods can partially address these questions, by relying on the field of formal methods, or by developing new approaches specific to machine learning, such as randomized smoothing or Lipschitz neural networks. Another solution lies in the *resilience* of systems containing machine learning items. Such systems should be designed to avoid or limit safety incidents in the event of a catastrophic failure from the machine learning items. In the absence of a satisfactory understanding of machine learning robustness, resilience remains the best strategy for safety-critical systems relying on machine learning to be trustworthy.

6.2 Future Research Directions

6.2.1 Rethinking Robustness

A predictor is robust if its prediction is *stable* under *irrelevant* perturbations of its input. For classification, the *stability* of the prediction corresponds to class change. However, class change is hard to study because the $0-1$ loss is non-differentiable, this is why surrogate losses are used in practice. The *irrelevancy* of the perturbation depends on the choice of a dissimilarity measure on the input space $\mathcal{X}$. The frequently used L_p norms are bad dissimilarity measures because they do not reflect the intuitive "indistinguishability" [195] (among L_p norms, the least worse dissimilarity measure is the L_∞ norm). Importantly, there exist:

1. Perturbations with large L_p norms but still indistinguishable [47];
2. Perturbations with small L_p norms but not indistinguishable [86, 195, 100].

As argued in [86], both accuracy and robustness are defined by reference to an *oracle*[1]. In practice, this oracle is a human. One can note that:

1. The humans label the dataset, so they are responsible of the accuracy;
2. The humans decide if a perturbation is *adversarial* or not (i.e., indistinguishable or not), so they are responsible of the robustness.

But surprisingly, the formal definition of adversarial robustness does not refer to the oracle, while the formal definition of accuracy does. Thus, the authors of [86] argue that the accuracy-robustness trade-off does not exist. If one imposes that a predictor should give the same prediction in some L_p ball, then it is almost tautological to say that there will be a trade-off between robustness and accuracy. Indeed, there may be an example in this L_p ball with a different label (as given by the oracle), so one either has to change the predicted label of this example (sacrifice robustness), or either has to missclassify this example (sacrifice accuracy).

As a side note, a better dissimilarity measure on $\mathcal{X}$ may be given by the Wasserstein distance [196, 197, 198].

Dissimilarity is an ill-defined notion. For example, two pictures containing the same object may be different in terms of lighting or colors, but still similar because they both contain this object. They are not indistinguishable, but one can hardly label them as dissimilar. In

[1] Here, an oracle is some deterministic or stochastic function that gives a label $\boldsymbol{y} \in \mathcal{Y}$ to any input $\boldsymbol{x} \in \mathcal{X}$.

fact, indistinguishability is not a relevant concept. There exist small perturbations that can be distinguished while the predictor is expected to be stable against them, therefore these perturbations are still irrelevant.

Now, imagine another task that consists in classifying the background and not the object. In this case, the images may be dissimilar because the backgrounds are different. Finding a good or a “perfect" dissimilarity measure is already a learning task. It amounts to finding the relevant features to discriminate two inputs. Dissimilarity measures are task-dependent thus, the definition of robustness is task-dependent. In [100], the authors show that finding the right dissimilarity measure is equivalent to solving the task. Thus, any other dissimilarity measure used to “certify" the robustness of a network will create either sensitivity attack (beyond the certified radius) or invariance attack, and often both.

How to define the process of “solving a task"? First, an *oracle* $\boldsymbol{h} : \mathcal{X} \to \mathcal{Y}$ is assumed to exist. A task is said to be ϵ-solved by a model f if $\|\boldsymbol{h} - f\|_\infty \leq \epsilon$. The definition of adversarial examples from section 2.1.5 is recalled here. Assume that there is some “neighborhood" of $\boldsymbol{x}$ called $\eta(\boldsymbol{x})$. A point $\boldsymbol{x}'$ is said to be an adversarial example for $\boldsymbol{x}$ if:

1. $\boldsymbol{x}' \in \eta(\boldsymbol{x})$;
2. $f(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x})$;
3. $f(\boldsymbol{x}') \neq \boldsymbol{h}(\boldsymbol{x}')$.

This definition include both sensitivity and invariance attacks. For any $\eta(\boldsymbol{x})$, there exists points $\boldsymbol{x}'$ that are adversarial examples for $\boldsymbol{x}$, unless $\eta(\boldsymbol{x})$ contains only points that have the class $\boldsymbol{h}(\boldsymbol{x})$ according to $\boldsymbol{h}$. The classical criteria for choosing $\eta(\boldsymbol{x})$ is that the perturbed point $\boldsymbol{x}'$ should be “almost indistinguishable" from $\boldsymbol{x}$ according to humans, which is an ill-defined criteria. If a clearly visible perturbation is added to $\boldsymbol{x}$ such that $f(\boldsymbol{x}') \neq h(\boldsymbol{x}')$ but the class $h(\boldsymbol{x})$ is unchanged, it will still be seen as a classification error and not as an adversarial attack. This is because one wants to stick to the scenario where an adversary slightly perturb the input. However, this scenario almost never happens in real-life applications, and even if an adversary could modify the input of a model, it may not be limited to small perturbations. Since there is no objective mechanism to separate normal from perturbed inputs, there is no constraint for the adversary.

Therefore, “robustness" is conceptually close to “generalization". Is it possible to imagine a model with very good generalization but poor robustness? Yes, because generalization is measured with the expectation over the true distribution $p(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})$. If a model is good for likely $\boldsymbol{x}$ (large $p(\boldsymbol{x})$) but bad for unlikely $\boldsymbol{x}$ (small $p(\boldsymbol{x})$), it will have good generalization but poor robustness. To train a robust model, the probability of unlikely $\boldsymbol{x}$ (i.e., the data points that are “closer to adversarial examples") should be increased. In the training data, unlikely inputs correspond to “isolated" data points. During training, these isolated data points should be given more weight.

6.2.2 Bayesian Priors, Simplicity, and Robustness

In this paragraph, several hypothesis that could drive future research are stated. The main idea is that adversarial vulnerability may be due to an imperfect notion of simplicity.
Hypothesis 1. In high dimension and for any natural dataset, for almost any point, the true boundary is very close to the chosen point. Closeness is measured using natural distances (L_p norms, Wasserstein distance etc.) in a natural coordinate system (e.g., pixel-coordinates for images). This is a consequence of the *curse of dimensionality*.

The first cause of adversarial vulnerability is *overfitting*, which can be seen even in low dimension. However, if there is no overfitting, adversarial vulnerability is still possible in high dimension. Indeed, if Hypothesis 1 is true, then any discrepancy in the generalization between the model and the true distribution will lead to regions of sensitivity examples and regions of invariance examples (section 2.1.5). It is possible to solve this issue by sampling more data in the problematic regions.
Hypothesis 2. The problematic regions corresponds to very unlikely regions, i.e., $p(\boldsymbol{x}) \approx 0$.
Thus, it is required to sample far more data in order to have sufficient data points in the problematic regions. Another solution is to augment the training set by other means such as adversarial training, semantic perturbations (i.e., rotation, brightness, reflection), or noise. Each of these solutions will change $p(\boldsymbol{x})$ in order to sample from a problematic region. Both adversarial training and semantic perturbations only sample from a small subset of all problematic regions, depending on the choice of attacks and perturbations, while noise can sample everywhere but without focusing on very unlikely problematic regions, which is inefficient. The existence of the human brain which is able to learn from few data shows that there must exists another solution.
Hypothesis 3. Solving adversarial vulnerability is possible using Bayesian machine learning if the correct prior is used.
When a model learns in high dimension, the training data are necessarily sparse. The model has many choices about how to choose the decision boundary in regions where $p(\boldsymbol{x}) \approx 0$ (i.e., regions with almost no training points). What is the correct generalization? It depends on the chosen notion of simplicity. The notion of simplicity corresponds to the regularization, or equivalently, the prior in Bayesian inference. There are two questions concerning the correct prior:
Question 1. Is there an optimal prior for any natural data, or is the prior dependent on the type of data?
Question 2. How to find the “correct" prior?

6.2.2.1 Bayesian Priors

The choice of a prior in Bayesian inference depends on the end goal of the researcher. Here is a typology of different types of priors [199]:

- **Informative priors** contain knowledge. This knowledge can be subjective, it can be expert knowledge, it can come from previous data, or other sources of information. *Prior elicitation* aims at transforming this knowledge into priors. Current data can also be used to learn the prior. However, this is a deviation from the Bayesian paradigm, since the prior should not depend on the data, and there is a risk of overfitting. *Model selection* aims at using data to choose the best prior. It relies on tools such that Bayesian information criterion (BIC) or marginal likelihood (i.e., Bayesian evidence $p(\boldsymbol{z}|\boldsymbol{M}) = \int p(\boldsymbol{z}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{M})d\boldsymbol{\theta}$);
- **Regularization priors** enforce some pre-determined properties, such as smoothness or sparsity;
- **Conjugate priors** reduce the computational complexity of the posterior analysis;
- **Uninformative priors** assume as little as possible about the data in order to minimize its effect on the posterior.
  Jeffreys prior [200] is the most famous uninformative prior. It is *invariant under reparametrization of the parameter space* $\boldsymbol{\theta}$. However, it has several drawbacks. It is hard to compute, it may be improper (i.e., normalization is infinite), it is not defined for singular models, and it may not be as uninformative as it claims to be. Jeffreys prior is uninformative only in the limit of infinite data. In practice, the prior choice problem is overlooked

by choosing Gaussian priors, which are often claimed to be “uninformative" without clear justifications.

A first idea to find the correct prior is to focus on uninformative priors [200]. Using uninformative priors may produce models that avoid overfitting and overconfidence, that favor “simple" models while avoiding model unstability (i.e., adversarial attacks), and that can learn from limited data. One promising direction is the so-called *reference prior* [201, 202, 203].

In [201], the authors review a body of work showing that parametric predictive models in every field of science (including machine learning models) have an *hyperribbon structure*: when looking at the parameter space with an information distance (such as the Fisher-Rao distance), some directions have a low impact on the predictions, while other directions have a large impact. Thus, it is possible to obtain a simpler model by discarding the directions with low impact. When inferring the parameter from data, the prior should focus on the directions with high impact. This is possible when using a *reference prior*, which is obtained by maximizing the mutual information between the prior and the distribution of the data, such that the information learned from the data is maximized.

In [203], the authors use the reference prior to improve transfer learning. However, the impact of reference prior to the adversarial robustness has not been studied yet. Exploring the impact of Bayesian priors on adversarial robustness, in particular using the underexplored field of Bayesian information geometry, constitutes a promising research direction.

Bibliography

[1] L. Grementieri and R. Fioresi, "Model-Centric Data Manifold: The Data Through the Eyes of the Model," *SIAM Journal on Imaging Sciences*, vol. 15, no. 3, pp. 1140–1156, 2022.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems* (P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), (Lake Tahoe, NV, USA), pp. 1106–1114, 3-6 December 2012.

[3] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Las Vegas, NV, USA), pp. 779–788, IEEE Computer Society, 27-30 June 2016.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), (Long Beach, CA, USA), pp. 5998–6008, 4-9 December 2017.

[5] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent Trends in Deep Learning Based Natural Language Processing," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, 2018.

[6] D. AG, "Neural Network Based Runway Landing Guidance for General Aviation Autoland," tech. rep., Federal Aviation Administration William J. Hughes Technical Center, Atlantic City International Airport, NJ, USA, 2021.

[7] European Union Aviation Safety Agency, "EASA Concept Paper: First Usable Guidance for Level 1 Machine Learning Applications," 2021.

[8] N. Minaskan, C. Dormoy, A. Pagani, J. André, and D. Stricker, "Human Intelligent Machine Teaming in Single Pilot Operation: A Case Study," in *International Conference on Augmented Cognition* (D. D. Schmorrow and C. M. Fidopiastis, eds.), vol. 13310 of *Lecture Notes in Computer Science*, (Virtual Event), pp. 348–360, Springer, 26 June - 1 July 2022.

[9] G. Jarry, N. Couellan, and D. Delahaye, "On the Use of Generative Adversarial Networks for Aircraft Trajectory Generation and Atypical Approach Detection," in *Air Traffic Management and Systems IV* (Electronic Navigation Research Institute, ed.), (Singapore), pp. 227–243, Springer Singapore, 2021.

[10] P. Juntama, S. Chaimatanan, and D. Delahaye, "Air Traffic Structuration based on Linear Dynamical Systems," in *SESAR Innovation Days*, (Virtual Event), 7-10 December 2020.

[11] S. Badrinath and H. Balakrishnan, "Automatic Speech Recognition for Air Traffic Control Communications," *Transportation Research Record*, vol. 2676, no. 1, pp. 798–810, 2022.

[12] D. Vukadinovic and D. Anderson, "X-ray baggage screening and artificial intelligence (AI) – A technical review of machine learning techniques for X-ray baggage screening," tech. rep., Publications Office of the European Union, Luxembourg, 2022.

[13] R. Roberts, F. Menant, G. Di Mino, and V. Baltazart, "Optimization and sensitivity analysis of existing deep learning models for pavement surface monitoring using low-quality images," *Automation in Construction*, vol. 140, 2022.

[14] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *International Conference on Computer Aided Verification* (R. Majumdar and V. Kuncak, eds.), vol. 10426 of *Lecture Notes in Computer Science*, (Heidelberg, Germany), pp. 97–117, Springer, 24-28 July 2017.

[15] G. Vidot, C. Gabreau, I. Ober, and I. Ober, "Certification of embedded systems based on Machine Learning: A survey," *CoRR*, 2021.

[16] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks," in *International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (Sydney, NSW, Australia), pp. 1321–1330, PMLR, 6-11 August 2017.

[17] European Union Aviation Safety Agency, "Artificial Intelligence Roadmap. A human-centric approach to AI in aviation," 2020.

[18] T. Kistan, A. Gardi, and R. Sabatini, "Machine learning and cognitive ergonomics in air traffic management: Recent developments and considerations for certification," *Aerospace*, vol. 5, no. 4, 2018.

[19] European Union Aviation Safety Agency, "Concepts of Design Assurance for Neural Networks (CoDANN) II," 2020.

[20] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Comput. Sci. Rev.*, vol. 37, 2020.

[21] B. Li, P. Qi, B. Liu, S. Di, J. Liu, J. Pei, J. Yi, and B. Zhou, "Trustworthy AI: From Principles to Practices," *ACM Comput. Surv.*, vol. 55, no. 9, 2023.

[22] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, and K. A. Roundy, ""Real Attackers Don't Compute Gradients": Bridging the Gap Between Adversarial ML Research and Practice," in *IEEE Conference on Secure and Trustworthy Machine Learning*, (Raleigh, NC, USA), pp. 339–364, IEEE, 8-10 February 2023.

[23] L. Shi-Garrier, N. C. Bouaynaya, and D. Delahaye, "Adversarial Robustness with Partial Isometry," *Entropy*, vol. 26, no. 2, 2024.

[24] L. Shi-Garrier, D. Delahaye, and N. C. Bouaynaya, "Predicting Air Traffic Congested Areas with Long Short-Term Memory Networks," in *USA/Europe Air Traffic Management Research and Development Seminar*, (New Orleans (Virtual Event), LA, USA), September 2021.

[25] N. N. Dalvi, P. M. Domingos, Mausam, S. K. Sanghai, and D. Verma, "Adversarial classification," in *International Conference on Knowledge Discovery and Data Mining* (W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel, eds.), (Seattle, WA, USA), pp. 99–108, ACM, 22-25 August 2004.

[26] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations* (Y. Bengio and Y. LeCun, eds.), (Banff, AB, Canada), 14-16 April 2014.

[27] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into Transferable Adversarial Examples and Black-box Attacks," in *International Conference on Learning Representations*, (Toulon, France), OpenReview.net, 24-26 April 2017.

[28] S. Cheng, Y. Dong, T. Pang, H. Su, and J. Zhu, "Improving Black-box Adversarial Attacks with a Transfer-based Prior," in *Advances in Neural Information Processing Systems* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), (Vancouver, BC, Canada), pp. 10932–10942, 8-14 December 2019.

[29] F. Tramèr, N. Papernot, I. J. Goodfellow, D. Boneh, and P. D. McDaniel, "The Space of Transferable Adversarial Examples," *CoRR*, 2017.

[30] L. Wu and Z. Zhu, "Towards Understanding and Improving the Transferability of Adversarial Examples in Deep Neural Networks," in *Asian Conference on Machine Learning* (S. J. Pan and M. Sugiyama, eds.), vol. 129 of *Proceedings of Machine Learning Research*, (Bangkok, Thailand), pp. 837–850, PMLR, 18-20 November 2020.

[31] C. Sitawarin, A. N. Bhagoji, A. Mosenia, M. Chiang, and P. Mittal, "DARTS: Deceiving Autonomous Cars with Toxic Signs," *CoRR*, 2018.

[32] R. Jia and P. Liang, "Adversarial Examples for Evaluating Reading Comprehension Systems," in *Conference on Empirical Methods in Natural Language Processing* (M. Palmer, R. Hwa, and S. Riedel, eds.), (Copenhagen, Denmark), pp. 2021–2031, Association for Computational Linguistics, 9-11 September 2017.

[33] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell, "Adversarial Policies: Attacking Deep Reinforcement Learning," in *International Conference on Learning Representations*, (Addis Ababa, Ethiopia), OpenReview.net, 26-30 April 2020.

[34] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust Adversarial Reinforcement Learning," in *International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (Sydney, NSW, Australia), pp. 2817–2826, PMLR, 6-11 August 2017.

[35] X. Pan, D. Seita, Y. Gao, and J. F. Canny, "Risk Averse Robust Adversarial Reinforcement Learning," in *International Conference on Robotics and Automation*, (Montreal, QC, Canada), pp. 8522–8528, IEEE, 20-24 May 2019.

[36] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," in *International Conference on Learning Representations* (Y. Bengio and Y. LeCun, eds.), (San Diego, CA, USA), 7-9 May 2015.

[37] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," in *International Conference on Learning Representations*, (Toulon, France), OpenReview.net, 24-26 April 2017.

[38] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting Adversarial Attacks With Momentum," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Salt Lake City, UT, USA), pp. 9185–9193, Computer Vision Foundation / IEEE Computer Society, 18-22 June 2018.

[39] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," in *IEEE European Symposium on Security and Privacy*, (Saarbrücken, Germany), pp. 372–387, IEEE, 21-24 March 2016.

[40] C. Zhao, P. T. Fletcher, M. Yu, Y. Peng, G. Zhang, and C. Shen, "The Adversarial Attack and Detection under the Fisher Information Metric," in *Conference on Artificial Intelligence*, (Honolulu, Hawaii, USA), pp. 5869–5876, AAAI Press, 27 January - 1 February 2019.

[41] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," in *International Conference on Learning Representations*, (Vancouver, BC, Canada), OpenReview.net, 30 April - 3 May 2018.

[42] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Las Vegas, NV, USA), pp. 2574–2582, IEEE Computer Society, 27-30 June 2016.

[43] H. Xu, Y. Ma, H. Liu, D. Deb, H. Liu, J. Tang, and A. K. Jain, "Adversarial Attacks and Defenses in Images, Graphs and Text: A Review," *Int. J. Autom. Comput.*, vol. 17, no. 2, pp. 151–178, 2020.

[44] N. Carlini and D. A. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *IEEE Symposium on Security and Privacy*, (San Jose, CA, USA), pp. 39–57, IEEE Computer Society, 22-26 May 2017.

[45] L. Engstrom, D. Tsipras, L. Schmidt, and A. Madry, "A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations," *CoRR*, 2017.

[46] A. Joshi, A. Mukherjee, S. Sarkar, and C. Hegde, "Semantic Adversarial Attacks: Parametric Transformations That Fool Deep Classifiers," in *IEEE/CVF International Conference on Computer Vision*, (Seoul, South Korea), pp. 4772–4782, IEEE, 27 October - 2 November 2019.

[47] C. Xiao, J. Zhu, B. Li, W. He, M. Liu, and D. Song, "Spatially Transformed Adversarial Examples," in *International Conference on Learning Representations*, (Vancouver, BC, Canada), OpenReview.net, 30 April - 3 May 2018.

[48] A. Athalye, N. Carlini, and D. A. Wagner, "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples," in *International Conference on Machine Learning* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm, Sweden), pp. 274–283, PMLR, 10-15 July 2018.

[49] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks," in *IEEE Symposium on Security and Privacy*, (San Jose, CA, USA), pp. 582–597, IEEE Computer Society, 22-26 May 2016.

[50] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial Examples Are Not Bugs, They Are Features," in *Advances in Neural Information Processing Systems* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), (Vancouver, BC, Canada), pp. 125–136, 8-14 December 2019.

[51] F. Tramèr, A. Kurakin, N. Papernot, I. J. Goodfellow, D. Boneh, and P. D. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses," in *International Conference on Learning Representations*, (Vancouver, BC, Canada), OpenReview.net, 30 April - 3 May 2018.

[52] S. Gu and L. Rigazio, "Towards Deep Neural Network Architectures Robust to Adversarial Examples," in *International Conference on Learning Representations* (Y. Bengio and Y. LeCun, eds.), (San Diego, CA, USA), 7-9 May 2015.

[53] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense Against Adversarial Attacks Using High-Level Representation Guided Denoiser," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Salt Lake City, UT, USA), pp. 1778–1787, Computer Vision Foundation / IEEE Computer Society, 18-22 June 2018.

[54] M. Cissé, P. Bojanowski, E. Grave, Y. N. Dauphin, and N. Usunier, "Parseval Networks: Improving Robustness to Adversarial Examples," in *International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (Sydney, NSW, Australia), pp. 854–863, PMLR, 6-11 August 2017.

[55] H. Xu and S. Mannor, "Robustness and generalization," *Mach. Learn.*, vol. 86, no. 3, pp. 391–423, 2012.

[56] R. Müller, S. Kornblith, and G. E. Hinton, "When does label smoothing help?," in *Advances in Neural Information Processing Systems* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), (Vancouver, BC, Canada), pp. 4696–4705, 8-14 December 2019.

[57] C. Shen, Y. Peng, G. Zhang, and J. Fan, "Defending Against Adversarial Attacks by Suppressing the Largest Eigenvalue of Fisher Information Matrix," *CoRR*, 2019.

[58] M. Picot, F. Messina, M. Boudiaf, F. Labeau, I. B. Ayed, and P. Piantanida, "Adversarial Robustness Via Fisher-Rao Regularization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 2698–2710, 2023.

[59] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan, "Theoretically Principled Trade-off between Robustness and Accuracy," in *International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, CA, USA), pp. 7472–7482, PMLR, 9-15 June 2019.

[60] Y. Wang, D. Zou, J. Yi, J. Bailey, X. Ma, and Q. Gu, "Improving Adversarial Robustness Requires Revisiting Misclassified Examples," in *International Conference on Learning Representations*, (Addis Ababa, Ethiopia), OpenReview.net, 26-30 April 2020.

[61] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On Detecting Adversarial Perturbations," in *International Conference on Learning Representations*, (Toulon, France), OpenReview.net, 24-26 April 2017.

[62] D. Hendrycks and K. Gimpel, "Early Methods for Detecting Adversarial Images," in *International Conference on Learning Representations*, (Toulon, France), OpenReview.net, 24-26 April 2017.

[63] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. D. McDaniel, "On the (Statistical) Detection of Adversarial Examples," *CoRR*, 2017.

[64] N. N. Čencov, *Statistical decision rules and optimal inference*, vol. 53 of *Translations of Mathematical Monographs*. Providence, RI, USA: American Mathematical Society, 1982.

[65] C. R. Rao, *Information and the Accuracy Attainable in the Estimation of Statistical Parameters*, pp. 235–247. New York, NY, USA: Springer New York, 1992.

[66] C. Atkinson and A. F. S. Mitchell, "Rao's Distance Measure," *Sankhyā: The Indian Journal of Statistics, Series A*, vol. 43, no. 3, pp. 345–365, 1981.

[67] S. I. Costa, S. A. Santos, and J. E. Strapasson, "Fisher information distance: A geometrical reading," *Discrete Applied Mathematics*, vol. 197, pp. 59–69, 2015.

[68] O. Calin and C. Udrişte, *Geometric Modeling in Probability and Statistics*. Springer International Publishing, 2014.

[69] N. N. Čencov, "Algebraic foundation of mathematical statistics 2," *Series Statistics*, vol. 9, no. 2, pp. 267–276, 1978.

[70] S.-I. Amari and H. Nagaoka, *Methods of Information Geometry*. American Mathematical Society, 2000.

[71] J. d'Eon, "Applications of Information Geometry to Machine Learning," Master's thesis, University of Waterloo, 2019.

[72] A. Soen and K. Sun, "On the Variance of the Fisher Information for Deep Learning," in *Advances in Neural Information Processing Systems* (M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, eds.), (Virtual Event), pp. 5708–5719, 6-14 December 2021.

[73] J. Pennington and P. Worah, "The Spectrum of the Fisher Information Matrix of a Single-Hidden-Layer Neural Network," in *Advances in Neural Information Processing Systems* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), (Montréal, QC, Canada), pp. 5415–5424, 3-8 December 2018.

[74] Z. Liao, T. Drummond, I. Reid, and G. Carneiro, "Approximate Fisher Information Matrix to Characterize the Training of Deep Neural Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 1, pp. 15–26, 2020.

[75] A. Achille, M. Lam, R. Tewari, A. Ravichandran, S. Maji, C. C. Fowlkes, S. Soatto, and P. Perona, "Task2Vec: Task Embedding for Meta-Learning," in *IEEE/CVF International Conference on Computer Vision*, (Seoul, South Korea), pp. 6429–6438, IEEE, 27 October - 2 November 2019.

[76] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems* (D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, eds.), (Barcelona, Spain), pp. 3981–3989, 5-10 December 2016.

[77] E. Tron, N. Couellan, and S. Puechmorel, "Canonical foliations of neural networks: application to robustness," *CoRR*, 2022.

[78] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii, "Distributional Smoothing with Virtual Adversarial Training," *CoRR*, 2015.

[79] J. Martin and C. Elster, "Inspecting adversarial examples using the Fisher information," *Neurocomputing*, vol. 382, 2020.

[80] A. Nayebi and S. Ganguli, "Biologically inspired protection of deep networks from adversarial attacks," *CoRR*, 2017.

[81] Y. Shi, B. Liao, G. Chen, Y. Liu, M. Cheng, and J. Feng, "Understanding Adversarial Behavior of DNNs by Disentangling Non-Robust and Robust Components in Performance Metric," *CoRR*, 2019.

[82] J. M. Lee, *Riemannian Manifolds: An Introduction to Curvature*. No. 176 in Graduate Texts in Mathematics, New York: Springer, 1997.

[83] F. Nielsen, "An Elementary Introduction to Information Geometry," *Entropy*, vol. 22, no. 10, 2020.

[84] S. Bubeck, Y. T. Lee, E. Price, and I. P. Razenshteyn, "Adversarial examples from computational constraints," in *International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, CA, USA), pp. 831–840, PMLR, 9-15 June 2019.

[85] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness May Be at Odds with Accuracy," in *International Conference on Learning Representations*, (New Orleans, LA, USA), OpenReview.net, 6-9 May 2019.

[86] A. S. Suggala, A. Prasad, V. Nagarajan, and P. Ravikumar, "Revisiting Adversarial Risk," in *International Conference on Artificial Intelligence and Statistics* (K. Chaudhuri and M. Sugiyama, eds.), vol. 89 of *Proceedings of Machine Learning Research*, (Naha, Okinawa, Japan), pp. 2331–2339, PMLR, 16-18 April 2019.

[87] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. J. Goodfellow, "Adversarial Spheres," in *International Conference on Learning Representations*, (Vancouver, BC, Canada), OpenReview.net, 3 May - 30 April 2018.

[88] S. Mahloujifar, D. I. Diochnos, and M. Mahmoody, "The Curse of Concentration in Robust Learning: Evasion and Poisoning Attacks from Concentration of Measure," in *AAAI Conference on Artificial Intelligence*, (Honolulu, HI, USA), pp. 4536–4543, AAAI Press, 27 January - 1 February 2019.

[89] A. Fawzi, H. Fawzi, and O. Fawzi, "Adversarial vulnerability for any classifier," in *Advances in Neural Information Processing Systems* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), (Montréal, QC, Canada), pp. 1186–1195, 3-8 December 2018.

[90] A. Shafahi, W. R. Huang, C. Studer, S. Feizi, and T. Goldstein, "Are adversarial examples inevitable?," in *International Conference on Learning Representations*, (New Orleans, LA, USA), OpenReview.net, 6-9 May 2019.

[91] T. Tanay and L. D. Griffin, "A Boundary Tilting Persepective on the Phenomenon of Adversarial Examples," *CoRR*, 2016.

[92] A. Shamir, I. Safran, E. Ronen, and O. Dunkelman, "A Simple Explanation for the Existence of Adversarial Examples with Small Hamming Distance," *CoRR*, 2019.

[93] L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Madry, "Adversarially Robust Generalization Requires More Data," in *Advances in Neural Information Processing Systems* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), (Montréal, QC, Canada), pp. 5019–5031, 3-8 December 2018.

[94] F. Tramèr, N. Papernot, I. J. Goodfellow, D. Boneh, and P. D. McDaniel, "The Space of Transferable Adversarial Examples," *CoRR*, 2017.

[95] N. Carlini and D. A. Wagner, "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods," in *ACM Workshop on Artificial Intelligence and Security* (B. Thuraisingham, B. Biggio, D. M. Freeman, B. Miller, and A. Sinha, eds.), (Dallas, TX, USA), pp. 3–14, ACM, 3 November 2017.

[96] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal Adversarial Perturbations," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Honolulu, HI, USA), pp. 86–94, IEEE Computer Society, 21-26 July 2017.

[97] K. Sadeghi, A. Banerjee, and S. K. S. Gupta, "A System-Driven Taxonomy of Attacks and Defenses in Adversarial Machine Learning," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 4, pp. 450–467, 2020.

[98] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognit.*, vol. 84, pp. 317–331, 2018.

[99] B. Biggio, G. Fumera, and F. Roli, "Security Evaluation of Pattern Classifiers under Attack," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 4, pp. 984–996, 2014.

[100] F. Tramèr, J. Behrmann, N. Carlini, N. Papernot, and J. Jacobsen, "Fundamental Tradeoffs between Invariance and Sensitivity to Adversarial Perturbations," in *International Conference on Machine Learning*, vol. 119 of *Proceedings of Machine Learning Research*, (Virtual Event), pp. 9561–9571, PMLR, 13-18 July 2020.

[101] J. Liu, B. Lu, M. Xiong, T. Zhang, and H. Xiong, "Adversarial Attack with Raindrops," *CoRR*, 2023.

[102] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *International Conference on Machine Learning*, vol. 119 of *Proceedings of Machine Learning Research*, (Virtual Event), pp. 2206–2216, PMLR, 13-18 July 2020.

[103] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. S. Fitzgerald, "Formal methods: Practice and experience," *ACM Comput. Surv.*, vol. 41, no. 4, pp. 19:1–19:36, 2009.

[104] C. Cheng, G. Nührenberg, and H. Ruess, "Maximum Resilience of Artificial Neural Networks," in *Automated Technology for Verification and Analysis* (D. D'Souza and K. N. Kumar, eds.), vol. 10482 of *Lecture Notes in Computer Science*, (Pune, India), pp. 251–268, Springer, 3-6 October 2017.

[105] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners," in *Advances in Neural Information Processing Systems* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds.), (New Orleans, LA, USA), 28 November - 9 December 2022.

[106] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree, "Regularisation of neural networks by enforcing Lipschitz continuity," *Mach. Learn.*, vol. 110, no. 2, pp. 393–416, 2021.

[107] K. Leino, Z. Wang, and M. Fredrikson, "Globally-Robust Neural Networks," in *International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, (Virtual Event), pp. 6212–6222, PMLR, 18-24 July 2021.

[108] M. Hein and M. Andriushchenko, "Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation," in *Advances in Neural Information Processing Systems* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), (Long Beach, CA, USA), pp. 2266–2276, 4-9 December 2017.

[109] M. Mirman, T. Gehr, and M. T. Vechev, "Differentiable Abstract Interpretation for Provably Robust Neural Networks," in *International Conference on Machine Learning* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm, Sweden), pp. 3575–3583, PMLR, 10-15 July 2018.

[110] E. Wong and J. Z. Kolter, "Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope," in *International Conference on Machine Learning* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm, Sweden), pp. 5283–5292, PMLR, 10-15 July 2018.

[111] M. Lécuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified Robustness to Adversarial Examples with Differential Privacy," in *IEEE Symposium on Security and Privacy*, (San Francisco, CA, USA), pp. 656–672, IEEE, 19-23 May 2019.

[112] H. Liang, E. He, Y. Zhao, Z. Jia, and H. Li, "Adversarial Attack and Defense: A Survey," *Electronics*, vol. 11, no. 8, 2022.

[113] J. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified Adversarial Robustness via Randomized Smoothing," in *International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, CA, USA), pp. 1310–1320, PMLR, 9-15 June 2019.

[114] Z. Hao, C. Ying, Y. Dong, H. Su, J. Song, and J. Zhu, "GSmooth: Certified Robustness against Semantic Transformations via Generalized Randomized Smoothing," in *International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, (Baltimore, MD, USA), pp. 8465–8483, PMLR, 17-23 July 2022.

[115] H. Salman, J. Li, I. P. Razenshteyn, P. Zhang, H. Zhang, S. Bubeck, and G. Yang, "Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers," in *Advances in Neural Information Processing Systems* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), (Vancouver, BC, Canada), pp. 11289–11300, 8-14 December 2019.

[116] J. Teng, G.-H. Lee, and Y. Yuan, "l1 Adversarial Robustness Certificates: a Randomized Smoothing Approach," *OpenReview.net*, 2020.

[117] G. Yang, T. Duan, J. E. Hu, H. Salman, I. P. Razenshteyn, and J. Li, "Randomized Smoothing of All Shapes and Sizes," in *International Conference on Machine Learning*, vol. 119 of *Proceedings of Machine Learning Research*, (Virtual Event), pp. 10693–10705, PMLR, 13-18 July 2020.

[118] R. Ettedgui, A. Araujo, R. Pinot, Y. Chevaleyre, and J. Atif, "Towards Evading the Limits of Randomized Smoothing: A Theoretical Analysis," *CoRR*, 2022.

[119] L. Li, J. Zhang, T. Xie, and B. Li, "Double Sampling Randomized Smoothing," in *International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, (Baltimore, MD, USA), pp. 13163–13208, PMLR, 17-23 July 2022.

[120] M. Alfarra, A. Bibi, P. H. S. Torr, and B. Ghanem, "Data dependent randomized smoothing," in *Uncertainty in Artificial Intelligence* (J. Cussens and K. Zhang, eds.), vol. 180 of *Proceedings of Machine Learning Research*, (Eindhoven, The Netherlands), pp. 64–74, PMLR, 1-5 August 2022.

[121] H. Hong and Y. Hong, "Certified Adversarial Robustness via Anisotropic Randomized Smoothing," *CoRR*, 2022.

[122] F. Eiras, M. Alfarra, P. H. S. Torr, M. P. Kumar, P. K. Dokania, B. Ghanem, and A. Bibi, "ANCER: Anisotropic Certification via Sample-wise Volume Maximization," *Trans. Mach. Learn. Res.*, vol. 2022, 2022.

[123] P. Labarbarie, H. Hajri, and M. Arnaudon, "Riemannian data-dependent randomized smoothing for neural networks certification," *CoRR*, 2022.

[124] P. Súkeník, A. Kuvshinov, and S. Günnemann, "Intriguing Properties of Input-Dependent Randomized Smoothing," in *International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, (Baltimore, MD, USA), pp. 20697–20743, PMLR, 17-23 July 2022.

[125] C. Anil, J. Lucas, and R. B. Grosse, "Sorting Out Lipschitz Function Approximation," in *International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, CA, USA), pp. 291–301, PMLR, 9-15 June 2019.

[126] L. Béthune, T. Boissin, M. Serrurier, F. Mamalet, C. Friedrich, and A. González-Sanz, "Pay attention to your loss : understanding misconceptions about Lipschitz neural networks," in *Advances in Neural Information Processing Systems* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds.), (New Orleans, LA, USA), 28 November - 9 December 2022.

[127] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Network," in *International Conference on Machine Learning* (F. R. Bach and D. M. Blei, eds.), vol. 37 of *JMLR Workshop and Conference Proceedings*, (Lille, France), pp. 1613–1622, JMLR.org, 6-11 July 2015.

[128] Y. Gal and Z. Ghahramani, "Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference," *CoRR*, 2015.

[129] D. Dera, G. Rasool, and N. Bouaynaya, "Extended Variational Inference for Propagating Uncertainty in Convolutional Neural Networks," in *IEEE International Workshop on Machine Learning for Signal Processing*, (Pittsburgh, PA, USA), pp. 1–6, IEEE, 13-16 October 2019.

[130] 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," in *IEEE International Radar Conference*, (Washington, DC, USA), pp. 362–367, 28-30 April 2020.

[131] 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 *IEEE International Conference on Acoustics, Speech and Signal Processing*, (Toronto, ON, Canada), pp. 7378–7382, IEEE, 6-11 June 2021.

[132] L. Kong, J. Sun, and C. Zhang, "SDE-Net: Equipping Deep Neural Networks with Uncertainty Estimates," in *International Conference on Machine Learning*, vol. 119 of *Proceedings of Machine Learning Research*, (Virtual Event), pp. 5405–5415, PMLR, 13-18 July 2020.

[133] J. Hoffman, D. A. Roberts, and S. Yaida, "Robust Learning with Jacobian Regularization," *CoRR*, 2019.

[134] S.-i. Amari, *Differential-Geometrical Methods in Statistics*, vol. 28 of *Lecture Notes in Statistics*. Springer New York, 1985.

[135] A. Shafahi, M. Najibi, A. Ghiasi, Z. Xu, J. P. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!," in *Advances in Neural Information Processing Systems* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), (Vancouver, BC, Canada), pp. 3353–3364, 8-14 December 2019.

[136] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," in *International Conference on Learning Representations*, (Addis Ababa, Ethiopia), OpenReview.net, 26-30 April 2020.

[137] L. T. Skovgaard, "A Riemannian Geometry of the Multivariate Normal Model," *Scandinavian Journal of Statistics*, vol. 11, no. 4, pp. 211–223, 1984.

[138] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence* (D. Rand and L.-S. Young, eds.), (Berlin, Heidelberg), pp. 366–381, Springer Berlin Heidelberg, 1981.

[139] S. Särkkä and A. Solin, *Applied Stochastic Differential Equations*. Institute of Mathematical Statistics Textbooks, Cambridge University Press, 2019.

[140] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations* (Y. Bengio and Y. LeCun, eds.), (San Diego, CA, USA), 7-9 May 2015.

[141] ICAO, "Long-Term Traffic Forecasts, Passenger and Cargo," 2018.

[142] SESAR Joint Undertaking, "Airspace Architecture Study," 2019.

[143] M. Prandini, L. Piroddi, S. Puechmorel, and S. L. Brazdilova, "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.

[144] E. M. Pfleiderer, C. A. Manning, and S. Goldman, "Relationship of complexity factor ratings with operational errors," tech. rep., FAA Civil Aerospace Medical Institute, Oklahoma City, OK, USA, 2007.

[145] SESAR Joint Undertaking, "SESAR Solutions Catalogue, Third Edition," p. 61, 2019.

[146] 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.

[147] S. Kauppinen, C. Brain, and M. Moore, "European medium-term conflict detection field trials," in *Digital Avionics Systems Conference*, (Irvine, CA, USA), 27-31 October 2002.

[148] B. Sridhar, T. Soni, K. Sheth, and G. Chatterji, "Aggregate Flow Model for Air-Traffic Management," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 4, pp. 992–997, 2006.

[149] 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*, vol. 50, no. 8, pp. 2174–2192, 2016.

[150] 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.

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

[152] 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.

[153] G. Gui, Z. Zhou, J. Wang, F. Liu, and J. Sun, "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.

[154] 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.

[155] I. Laudeman, S. Shelden, R. Branstrom, and C. Brasil, "Dynamic density: An air traffic management metric," tech. rep., NASA, Moffett Field, CA, USA, 1998.

[156] B. Sridhar, K. S. Sheth, and S. Grabbe, "Airspace Complexity and its Application in Air Traffic Management," in *USA/Europe Air Traffic Management Research and Development Seminar*, (Orlando, FL, USA), December 1998.

[157] B. Hilburn, "Cognitive Complexity in Air Traffic Control: A Literature Review," tech. rep., EUROCONTROL Experimental Center, Brétigny-sur-Orge, France, 2004.

[158] G. Chatterji and B. Sridhar, "Neural network based air traffic controller workload prediction," in *American Control Conference*, vol. 4, (San Diego, CA, USA), pp. 2620–2624, 2-4 June 1999.

[159] G. Chatterji and B. Sridhar, "Measures for air traffic controller workload prediction," in *Aircraft, Technology Integration, and Operations Forum*, (Los Angeles, CA, USA), 16-18 October 2001.

[160] P. Kopardekar and S. Magyarits, "Dynamic density: Measuring and predicting sector complexity," in *Digital Avionics Systems Conference*, vol. 1, (Irvine, CA, USA), 27-31 October 2002.

[161] A. J. Masalonis, M. B. Callaham, and C. R. Wanke, "Dynamic density and complexity metrics for real-time traffic flow management: Quantitative analysis of complexity indicators," in *USA/Europe Air Traffic Management Research and Development Seminar*, (Budapest, Hungary), 23-27 June 2003.

[162] 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.

[163] J. Lygeros and M. Prandini, "Aircraft and weather models for probabilistic collision avoidance in air traffic control," in *IEEE Conference on Decision and Control*, vol. 3, (Las Vegas, NV, USA), pp. 2427–2432, 10-13 December 2002.

[164] S. Mondoloni and D. Liang, "Airspace fractal dimension and applications," in *USA/Europe Air Traffic Management Research and Development Seminar*, (Santa Fe, NM, USA), 4-7 December 2001.

[165] K. Lee, E. Feron, and A. Pritchett, "Air traffic complexity: An input-output approach," in *American Control Conference*, (New York City, NY, USA), pp. 474–479, 11-13 July 2007.

[166] H. Wang, Z. Song, and R. Wen, "Modeling Air Traffic Situation Complexity with a Dynamic Weighted Network Approach," *Journal of Advanced Transportation*, vol. 2018, no. 1, 2018.

[167] H. J. Wee, S. W. Lye, and J.-P. Pinheiro, "A Spatial, Temporal Complexity Metric for Tactical Air Traffic Control," *Journal of Navigation*, vol. 71, no. 5, p. 1040–1054, 2018.

[168] D. Delahaye and S. Puechmorel, "Air traffic complexity based on dynamical systems," in *IEEE Conference on Decision and Control*, (Atlanta, GA, USA), pp. 2069–2074, 15-17 December 2010.

[169] M. Ishutkina, E. Feron, and K. Bilimoria, "Describing Air Traffic Complexity Using Mathematical Programming," in *AIAA Aviation, Technology, Integration, and Operations Conference*, (Arlington, VA, USA), 26-28 September 2005.

[170] D. Delahaye, A. García, J. Lavandier, S. Chaimatanan, and M. Soler, "Air Traffic Complexity Map Based on Linear Dynamical Systems," *Aerospace*, vol. 9, no. 5, 2022.

[171] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), (Montreal, QC, Canada), pp. 3104–3112, 8-13 December 2014.

[172] G. Saon, G. Kurata, T. Sercu, K. Audhkhasi, S. Thomas, D. Dimitriadis, X. Cui, B. Ramabhadran, M. Picheny, L. Lim, B. Roomi, and P. Hall, "English Conversational Telephone Speech Recognition by Humans and Machines," in *Annual Conference of the International Speech Communication Association* (F. Lacerda, ed.), (Stockholm, Sweden), pp. 132–136, ISCA, 20-24 August 2017.

[173] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Boston, MA, USA), pp. 3156–3164, IEEE Computer Society, 7-12 June 2015.

[174] Q. V. Le, N. Jaitly, and G. E. Hinton, "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units," *CoRR*, 2015.

[175] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout Improves Recurrent Neural Networks for Handwriting Recognition," in *International Conference on Frontiers in Handwriting Recognition*, (Crete, Greece), pp. 285–290, IEEE Computer Society, 1-4 September 2014.

[176] Y. Gal and Z. Ghahramani, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," in *Advances in Neural Information Processing Systems* (D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, eds.), (Barcelona, Spain), pp. 1019–1027, 5-10 December 2016.

[177] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *CoRR*, 2016.

[178] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, vol. 28 of *JMLR Workshop and Conference Proceedings*, (Atlanta, GA, USA), pp. 1310–1318, JMLR.org, 16-21 June 2013.

[179] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[180] H. Sak, A. W. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," *CoRR*, 2014.

[181] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *CoRR*, 2014.

[182] F. A. Gers and J. Schmidhuber, "Recurrent Nets that Time and Count," in *International Joint Conference on Neural Networks*, vol. 3, (Como, Italy), pp. 189–194, IEEE Computer Society, 24-27 July 2000.

[183] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Conference on Empirical Methods in Natural Language Processing* (A. Moschitti, B. Pang, and W. Daelemans, eds.), (Doha, Qatar), pp. 1724–1734, ACL, 25-29 October 2014.

[184] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, 2017.

[185] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997.

[186] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A Comparison of Sequence-to-Sequence Models for Speech Recognition," in *Annual Conference of the International Speech Communication Association* (F. Lacerda, ed.), (Stockholm, Sweden), pp. 939–943, ISCA, 20-24 August 2017.

[187] S. Venugopalan, M. Rohrbach, J. Donahue, R. J. Mooney, T. Darrell, and K. Saenko, "Sequence to Sequence - Video to Text," in *IEEE International Conference on Computer Vision*, (Santiago, Chile), pp. 4534–4542, IEEE Computer Society, 7-13 December 2015.

[188] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *International Conference on Learning Representations* (Y. Bengio and Y. LeCun, eds.), (San Diego, CA, USA), 7-9 May 2015.

[189] T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," in *Conference on Empirical Methods in Natural Language Processing* (L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton, eds.), (Lisbon, Portugal), pp. 1412–1421, The Association for Computational Linguistics, 17-21 September 2015.

[190] A. Graves, "Sequence Transduction with Recurrent Neural Networks," *CoRR*, 2012.

[191] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, (Virtual Event, Austria), OpenReview.net, 3-7 May 2021.

[192] D. Dera, N. C. Bouaynaya, G. Rasool, R. Shterenberg, and H. M. Fathallah-Shaykh, "PremiUm-CNN: Propagating Uncertainty Towards Robust Convolutional Neural Networks," *IEEE Trans. Signal Process.*, vol. 69, pp. 4669–4684, 2021.

[193] D. Dera, S. Ahmed, N. C. Bouaynaya, and G. Rasool, "TRustworthy Uncertainty Propagation for Sequential Time-Series Analysis in RNNs," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 2, pp. 882–896, 2024.

[194] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *IEEE International Conference on Computer Vision*, (Venice, Italy), pp. 2999–3007, IEEE Computer Society, 22-29 October 2017.

[195] M. Sharif, L. Bauer, and M. K. Reiter, "On the Suitability of Lp-Norms for Creating and Preventing Adversarial Examples," in *IEEE Conference on Computer Vision and Pattern Recognition*, (Salt Lake City, UT, USA), pp. 1605–1613, Computer Vision Foundation / IEEE Computer Society, 18-22 June 2018.

[196] E. Wong, F. R. Schmidt, and J. Z. Kolter, "Wasserstein Adversarial Examples via Projected Sinkhorn Iterations," in *International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, CA, USA), pp. 6808–6817, PMLR, 9-15 June 2019.

[197] A. Levine and S. Feizi, "Wasserstein Smoothing: Certified Robustness against Wasserstein Adversarial Attacks," in *International Conference on Artificial Intelligence and Statistics* (S. Chiappa and R. Calandra, eds.), vol. 108 of *Proceedings of Machine Learning Research*, (Palermo (Virtual Event), Sicily, Italy), pp. 3938–3947, PMLR, 26-28 August 2020.

[198] A. T. Bui, T. Le, Q. H. Tran, H. Zhao, and D. Q. Phung, "A Unified Wasserstein Distributional Robustness Framework for Adversarial Training," in *International Conference on Learning Representations*, (Virtual Event), OpenReview.net, 25-29 April 2022.

[199] J. Arbel, K. Pitas, M. Vladimirova, and V. Fortuin, "A Primer on Bayesian Neural Networks: Review and Debates," *CoRR*, 2023.

[200] R. E. Kass and L. Wasserman, "The Selection of Prior Distributions by Formal Rules," *Journal of the American Statistical Association*, vol. 91, no. 435, pp. 1343–1370, 1996.

[201] K. N. Quinn, M. C. Abbott, M. K. Transtrum, B. B. Machta, and J. P. Sethna, "Information geometry for multiparameter models: new perspectives on the origin of simplicity," *Reports on Progress in Physics*, vol. 86, no. 3, 2022.

[202] H. H. Mattingly, M. K. Transtrum, M. C. Abbott, and B. B. Machta, "Maximizing the information learned from finite data selects a simple model," *Proceedings of the National Academy of Sciences*, vol. 115, no. 8, pp. 1760–1765, 2018.

[203] Y. Gao, R. Ramesh, and P. Chaudhari, "Deep Reference Priors: What is the best way to pretrain a model?," in *International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, (Baltimore, MD, USA), pp. 7036–7051, PMLR, 17-23 July 2022.

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Université de Toulouse

Titre : Contributions à la Robustesse de l'Apprentissage Machine avec Applications à la Prédiction de Trafic

Mots clés : Apprentissage machine robuste, Métrique de l'information de Fisher, Prédiction de trafic, Quantification de l'incertitude

Résumé : L'émergence de l'apprentissage profond est porteur de nombreuses opportunités pour l'aviation civile. Les méthodes d'apprentissage peuvent automatiser des tâches pénibles (e.g., transcription de la parole vers le texte pour les rapports d'incidents) ou introduire de nouvelles fonctions visant à améliorer la sécurité, l'efficacité, ou l'impact environnemental du transport aérien (e.g., réduction de la complexité du trafic aérien). Cependant, les caractéristiques propres aux modèles d'apprentissage machine (fondés sur les données, stochastiques, manque de transparence) limitent leur introduction dans les applications où la sécurité est un enjeu central. Au-delà de la performance en termes de précision, ces applications exigent des modèles fiables, capables de fournir des garanties en termes de robustesse et d'explicabilité.
Dans cette thèse, nous explorons divers aspects de la robustesse des réseaux de neurones contre les attaques par exemples contradictoires en s'appuyant sur des concepts empruntés à la géométrie de l'information. Dans une première partie, nous introduisons un terme de régularisation visant à améliorer la robustesse des réseaux de neurones contre les attaques L2. Ce terme pousse le modèle à être localement isométrique par rapport à la métrique euclidienne (en entrée) et à la métrique d'information de Fisher (en sortie). Cette méthode est évaluée sur les jeux de données MNIST et CIFAR-10. Dans la seconde partie, on introduit une méthode pour étudier la robustesse des réseaux de neurones récurrents basée sur les équations différentielles stochastiques. Plusieurs expériences sont ensuite menées sur des données synthétiques et sur MNIST afin d'étudier l'interaction entre la vulnérabilité aux exemples contradictoires et les foliations induites par le noyau de la métrique d'information de Fisher dans l'espace d'entrée. Dans la troisième partie, nous appliquons les réseaux de neurones récurrents à la prédiction de la congestion dans l'espace aérien. Finalement, nous développons une méthode fondée sur l'inférence variationnelle pour la quantification de l'incertitude dans les modèles de type Transformer. Cette méthode est évaluée sur des données de trafic routier.

Title: Contributions to Machine Learning Robustness with Applications to Traffic Prediction

Key words: Machine Learning Robustness, Fisher information metric, Traffic prediction, Uncertainty quantification

Abstract: The advent of deep learning brings many opportunities to the civil aviation community. Data-driven methods can automatize tedious tasks (e.g., speech-to-text for incident reporting) or introduce new functions to improve the safety, efficiency, or environmental impact of the air transportation system (e.g., reduction of the complexity of air traffic). However, the unique features of machine learning models (data-driven, stochastic, lack of transparency) prevent their introduction into safety-critical applications. Beyond accuracy, safety-critical applications require the model to be trustworthy by providing guarantees on robustness and explainability.
In this work, we explore several aspects of neural networks robustness against adversarial noise using concepts borrowed from information geometry. In the first part, we introduce a regularization term to improve the robustness of neural networks against L2 attacks. This term pushes the model to be locally isometric with respect to the Euclidean metric (in input) and the Fisher information metric (in output). The method is evaluated on MNIST and CIFAR-10 datasets. In the second part, we introduce a method to study the robustness of recurrent neural networks using stochastic differential equations. Then, we conduct several experiments on synthetic data and on MNIST in order to study the interaction between adversarial vulnerability and the foliations induced by the kernel fisher information metric in the input space. In the third part, we apply recurrent neural networks for the prediction of airspace congestion. Finally, we derive a method for uncertainty quantification for Transformer models based on variational inference. This method is evaluated on road traffic data.