# THÈSE

En vue de l'obtention du

**DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE**

**Délivré par l'Université Toulouse 2 - Jean Jaurès**

**Présentée et soutenue par**

**Guillaume VIDOT**

Le 15 décembre 2022

**Vers la certification des systèmes avioniques basés sur l'apprentissage automatique : Exploitation des preuves mathématiques pour garantir la fiabilité**

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
**IRIT : Institut de Recherche en Informatique de Toulouse**

Thèse dirigée par
**Ileana OBER et Iulian OBER**

Jury
**M. Stéphane CANU,** Rapporteur
**M. Jun SUN,** Rapporteur
**Mme Caterina URBAN,** Examinatrice
**M. David DELAHAYE,** Examinateur
**Mme Mélanie DUCOFFE,** Examinatrice
**Mme Ileana OBER,** Directrice de thèse
**M. Iulian OBER,** Co-directeur de thèse
**M. Christophe GABREAU,** Co-directeur de thèse du monde socio-économique

## Abstract

Lately, Machine Learning (ML) algorithms have proven their high efficiency in solving tasks (computer vision, speech recognition, object detection, etc.) that were considered hard to solve with other coding paradigms. As a result, numerous industries take the opportunity to improve their products and services using ML algorithms. However, for some of these industries, such as the automotive or the aeronautic industry, it is complicated to use ML algorithms since they deal with critical embedded systems. An embedded system is critical when its failure leads to products damages and death or injury of people. Thus, critical embedded systems are subject to a certification process. Still, it is impossible to embed ML-based systems because some certification objectives are not reachable. The aeronautic industry has got many new applications using ML algorithms without the possibility of using them since the certification of ML-based systems is not yet achievable. This thesis first discusses the gaps that prevent from certifying ML-based systems. Then, we elaborate on the challenges raised by these gaps before diving into the trustworthiness considerations of ML-based systems. Amongst the issues related to the certification, we will focus on two topics: adversarial robustness and formal verification. Adversarial robustness is the capacity of an algorithm to be resilient to adversarial examples. An *adversarial example* is defined as an example that looks like a "normal" example but fools the ML algorithm. Formal verification brings proof that the ML model complies with the requirements. We propose in this thesis first to enhance the safety by design (during the learning process) by studying the adversarial robustness of an ML algorithm by using the PAC-Bayesian theory. Instead of deriving a worst-case analysis of the risk of a hypothesis over all the possible perturbations, we leverage the PAC-Bayesian framework to bound the averaged risk on the perturbations for majority votes (over the whole class of hypotheses). Then, we propose to improve the safety by verification (post learning) by analyzing the monotony property of a surrogate neural network using a MILP solver. This monotony analysis provides a lower and upper bound of the space volume where the property does not hold, which we denote "Non-Monotonic Space Coverage". Both contributions comply with the first guidance provided by the European Union Aviation Safety Agency (EASA) about ML-based systems certification.

# Remerciements

Tout d'abord, je remercie Ileana et Iulian Ober (directrice et co-directeur) pour m'avoir donner l'opportunité de réaliser cette thèse. Je les remercie également pour leur investissement tout au long de cette thèse malgré le contexte difficile que nous avons eu à traverser ces dernières années. Je remercie ensuite Christophe Gabreau pour m'avoir tant appris au sujet de la certification et pour m'avoir accompagné et donné de précieux conseils pendant ces trois années de thèse. Pour finir le tour des personnes qui m'ont encadrées, un très grand merci à Mélanie Ducoffe pour son aide incommensurable, ses conseils et ses critiques toujours justes sur mes travaux qui m'ont permis de faire toujours mieux.

Merci aux rapporteurs, Jun Sun et Stéphane Canu pour la relecture de ma thèse et pour leurs retours. Merci également à Caterina Urban et David Delahaye d'avoir accepté de faire partie de mon jury de thèse.

L'écriture de ce manuscrit marque la fin de trois années de travail intense et il est donc temps de remercier toutes les personnes qui ont été impliquées directement ou indirectement. Je commence par une dédicace spéciale pour Paul Viallard sans qui cette thèse aurait été radicalement différente. Au delà des années de thèse, merci Paul pour toutes ces années de fac, toutes ces soirées, toutes les séances d'escalade, de patinage et toutes les discussions "philosophiques" sur la vie qu'on a pu avoir. On peut dire que d'écrire un papier ensemble a vraiment été la consécration de tout notre cursus passé ensemble. Encore un grand merci pour tout ! Je n'oublie pas Emilie Morvant et Amaury Habrard pour leur aide précieuse pendant l'écriture du papier avec Paul, merci.

Merci à Luc, sans qui je n'aurais pas eu l'opportunité de rejoindre Airbus après ma thèse. Merci également pour tous les "one2one" qui apportaient toujours un bol d'air frais et des conseils avisés. Merci globalement à toute mon équipe à Airbus pour ces trois années passées qu'on a pu terminer par un excellent team booster. Merci à ma nouvelle équipe à Airbus pour votre accueil chaleureux.

Merci au Bapz, excellent salon de thé à Toulouse, dans lequel j'ai passé des après-midis entiers à travailler sur ce manuscrit dans les meilleures conditions possibles entre les coupes de glace, les crumbles, les scones, les thés glacés et chocolats chauds.

On entend souvent dire que les études supérieures sont des années de sa vie que l'on n'oublie pas. C'est probablement grâce aux personnes avec qui on les a passé qui rendent le tout inoubliable. Merci à Adam pour son humour extraordinaire qui me fera toujours rire, à Lucas pour ces discussions post-soirée à 3h du matin, à Clément, Noémie, Chloé, Nina, Romain C. et Paul pour ces séances de badminton et/ou escalade. Merci à Romain P. pour toutes ces soirées qu'on a pu passer chez lui avec Emilie et Paul. De manière plus générale, je vous remercie tous, sans oublier Chris, Dimitrie et Hugo. Enfin bref, je ne vais malheureusement pas pouvoir citer ici toutes les belles choses vécues pendant

ces années à l'université mais je suis sans aucun doute reconnaissant pour chacun de ces moments.

Je tenais à remercier Jean-Claude Barthelemy avec qui j'ai eu la chance de travailler pendant mes stages de master.

Viens ensuite les vacances qui, de mon point de vue, font partie intégrante de la thèse car très régulièrement, la thèse revient au galop et on a toujours des petites choses à terminer. Donc merci à toutes ces personnes avec qui j'ai pu me détendre pendant les week-ends, les vacances d'été, les fêtes et qui ont également supporté ce travail incessant ainsi que toutes les discussions qu'on a pu avoir autour de ma thèse. Merci à Nicolas pour les séjours à Biarritz et pour tout le reste. Merci à Marie que je ne vois clairement pas assez mais qui est toujours disponible dès qu'on a l'occasion de se voir. Merci à Elodie, Rémi, et Laurent pour ces multiples vacances dans les Pyrénées accentuées par les randonnées et les jeux de sociétés. Merci à Marik et Baudouin pour m'avoir fait découvrir la Normandie. Un grand merci à Baudouin de m'avoir donné la chance de pouvoir conduire sa Donk ! Ces quelques mots sont bien légers par rapport aux remerciements que vous méritez tous, mais on aura, sans doute, l'occasion d'en parler de vive voix.

Finalement, de manière plus générale, je remercie toute ma famille car sans eux, je n'en serai peut-être pas là aujourd'hui. Plus particulièrement, je souhaite remercier Kévin et David pour tout ce qu'on a pu vivre ensemble ; Christelle, ma chère soeur avec qui j'ai grandi et son mari Timothée. Je donne mon infinie reconnaissance à mes parents qui ont toujours été d'un soutien sans faille et ont toujours été présents. Merci pour tout du fond du coeur.

J'ai finalement gardé le meilleur pour la fin... enfin la meilleure. Merci à Ludivine Damour, celle avec qui je partage ma vie depuis maintenant bientôt cinq années. Merci d'avoir été là chaque jour à me supporter, merci d'avoir écouté mes explications qui pour toi n'ont aucun sens, merci de m'avoir soutenu... Pour faire très court, merci d'être là tout simplement.

# Table of content

# List of Figures

# List of Tables

# List of Notations

**Acronym**

| | |
|---|---|
| AI | Artificial Intelligence |
| AMC | Acceptable Means of Compliance |
| AT | Adversarial Training |
| CNF | Conjunctive Normal Form |
| CRI | Certification Review Item |
| DAL | Design Assurance Level |
| DNN | Deep Neural Network |
| EASA | European Union Aviation Safety Agency |
| ERM | Empirical Risk Minimization |
| MBE | Models Based Engineering |
| MILP | Mixed Integer Linear Programming |
| ML | Machine Learning |
| ODD | Operational Design Domain |
| RBE | Requirements Based Engineering |

**Mathematical Notation**

| | |
|---|---|
| $\mathcal{D}$ | Unknown distribution on $\mathbb{X}\times\mathbb{Y}$ |
| $\mathbb{E}_{(x,y)\sim\mathcal{D}}(\cdot)$ | Expectation according to $(x,y)\sim\mathcal{D}$ |
| $\mathcal{H}$ | Set of hypothesis |
| $\ell(\cdot,\cdot)$ | The loss function |
| $\|\cdot\|_p$ | $L_p$-norm distance |
| $\mathbb{N}$ | Set of natural numbers |
| $\mathbb{R}$ | Set of real numbers |

| | |
|---|---|
| $\mathcal{R}_{\mathcal{D}}(h)$ | Risk of the hypothesis $h \in \mathcal{H}$ according to the distribution $\mathcal{D}$ |
| $S$ | Set of training examples drawn from $\mathcal{D}$ |
| $\mathbb{X}$ | Set of inputs |
| $\mathbb{Y}$ | Set of outputs – or label |
| $T$ | Set of testing examples drawn from $\mathcal{D}$ |
| $W_i$,$b_i$ | Neural Network parameters (weight and bias) |
| | **Adversarial Robustness** |
| $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(h)$ | True adversarial risk |
| $\boldsymbol{\mathfrak{D}}$ | Distribution on $(\mathbb{X}{\times}\mathbb{Y}) \times \mathbb{B}$ (perturbed example) |
| $\omega_{(x,y)}$ | Distribution on the set of possible noises $\mathbb{B}$ |
| $\mathbb{e}_i$ | Set of perturbations drawn from $\omega_{(x_i,y_i)}$ |
| $\mathbf{S}$ | Set of perturbed training examples |
| $\mathbb{B}$ | Set of possible noises |
| | **Formal Verification** |
| $\alpha$ | Subset of the discrete features $\mathbb{V}_d$ on which the monotony applies. |
| $\mathbb{V}$ | Set of input features |
| $\overline{\Omega}$ | Upper bound on the Non-Monotonic Space Coverage |
| $\underline{\Omega}$ | Lower bound on the Non-Monotonic Space Coverage |
| $\mathcal{P}$ | Partition of the input space $\mathbb{X}$ |
| $\boldsymbol{\hat{x}^i}$,$\boldsymbol{x^i}$,$\boldsymbol{a^i}$ | MILP variables |
| $C^h(p)$ | MILP constraint representing the neural network $h$ |
| $C^{mon}$,$C^{\neg mon}$ | MILP encoding of the monotony property |
| $M^i$ | Large upper bound s.t. $-M^i \leq \boldsymbol{\hat{x}^i}$ and $\boldsymbol{x^i} \leq M^i$ |
| | **PAC-Bayes** |
| $\mathbf{B}_{\mathcal{Q}}$ | $\mathcal{Q}$-weighted majority vote or Bayes classifier |
| $\mathbf{G}_{\mathcal{Q}}$ | Gibbs classifier (stochastic) |
| $\mathrm{KL}(\mathcal{Q}\\|\mathcal{P})$ | Kullback-Leibler divergence (KL-divergence) between $\mathcal{Q}$ and $\mathcal{P}$ |
| $\mathcal{P}$ | Prior distribution on $\mathcal{H}$ |
| $\mathcal{Q}$ | Posterior distribution on $\mathcal{H}$ |

# Introduction

## Environnement de travail de la thèse

Cette thèse synthétise les résultats d'une recherche conjointe entre la société Airbus et l'IRIT (Institut de Recherche en Informatique de Toulouse), rendue possible par l'ANRT (Association Nationale Recherche Technologie) et Airbus (Projet CIFRE). Cette collaboration a été initiée pour tirer profit de l'expertise en certification de systèmes détenue par Airbus et de l'expérience en vérification formelle des systèmes cyber-physiques détenue par l'IRIT.

## Contexte

Les systèmes de sûreté critiques, et l'avionique en particulier, représentent un domaine d'application récalcitrant à l'application de nouvelles méthodes de développement logiciel (Krodel, 2008). Par exemple, certains aspects de la programmation orientée objet, tels que le polymorphisme et la liaison dynamique n'ont jamais fait leur chemin vers les systèmes de sûreté critiques, principalement en raison du déséquilibre entre la valeur ajoutée et l'augmentation des coûts de développement et de validation. Néanmoins, les récentes avancées dans le domaine de l'apprentissage automatique ont suscité un véritable intérêt, car celui-ci offre des résultats préliminaires prometteurs et ouvre la voie à un large éventail de nouvelles fonctions pour les systèmes avioniques visant à développer de nouvelles expériences de vol : assistance à la navigation/surveillance (*e.g.*, navigation par vision, détection d'obstacles, détection virtuelle), applications de conversion de la parole en texte, vol autonome, maintenance prédictive, assistance au poste de pilotage, etc.

Contrairement à la programmation classique, qui peut difficilement prendre en charge ces fonctions, l'apprentissage automatique (Machine Learning, ML), qui est un sous-domaine de l'intelligence artificielle (IA), se démarque par ses bons résultats pour la plupart d'entre elles (*e.g.*, Bahdanau *et al.*, 2015; Devlin *et al.*, 2019; Redmon *et al.*, 2016). Les techniques ML introduisent un nouveau paradigme dans le développement avionique : la conception des modèles *pilotée par les données* (supervisée, non supervisée et apprentissage par renforcement). Les caractéristiques de la conception pilotée par les données sont principalement liées à la forme complètement nouvelle que prend la spécification des exigences : au lieu que celles-ci soient entièrement exprimées par l'ingénieur système, ce sont les données qui les expriment (en partie).

L'expression "pilotée les données" fait référence au fait que les données déterminent le comportement de l'algorithme par le biais d'une phase d'apprentissage. Les directives industrielles actuelles (*e.g.*, DO-178x) sont orientées sur les technologies établies dans les applications aéronautiques, avec un fort accent sur la spécification des exigences et la façon dont elles sont capturées et respectées par le produit final. Cependant, ces directives ne sont pas adaptées à ce changement de paradigme.

## Objectifs et contributions

Avec cette thèse, nous aspirons à l'exploration du champ des possibles en matière de démonstration de conformité des systèmes à base de ML face aux contraintes réglementaires. Cela se décompose en plusieurs objectifs :

O1 **Identifier les impacts de l'utilisation d'un système basé sur le ML sur l'approche de certification actuelle.**
Cet objectif consiste à définir comment l'approche de certification actuelle n'est pas adaptée aux systèmes basés sur le ML et à définir les écarts introduits par l'utilisation de ceux-ci par rapport à la pratique de certification actuelle.

O2 **Identifier les défis soulevés par les impacts précédemment identifiés.**
Cet objectif consiste à étudier comment il est possible de réduire ou de surmonter les écarts identifiés afin que le processus de certification puisse être appliqué aux systèmes contenant des éléments ML.

O3 **Proposer de nouveaux moyens pour pallier certains défis précédemment identifiés.**
En raison de la durée limitée de cette recherche, nous ne pouvons pas relever tous les challenges. Pour faire une sélection, nous considérons les besoins de la certification. Comme l'objectif principal de la démonstration de conformité est d'assurer la sûreté, nous nous concentrons sur les défis liés aux questions de sûreté.

Notre recherche répond à ces objectifs à travers plusieurs contributions. Ci-après un résumé de celles-ci.

**Contribution aux objectifs O1 et O2:** afin d'identifier les impacts de l'utilisation d'un système à base de ML sur l'approche de certification actuelle, nous effectuons une analyse des lacunes de l'approche de certification actuelle pour la démonstration de conformité des systèmes à base de ML et une revue de la littérature sur les défis soulevés.

Pour identifier les lacunes notables entre la vérification de logiciels classiques et la vérification de logiciels contenant des parties obtenues par du ML, nous devons d'abord définir le cadre du domaine ML considéré. En prenant comme référence le point de vue des autorités actuelles sur cette question (EASA, 2020, 2021b), cette recherche se concentrera sur *l'apprentissage supervisé hors ligne*, c'est-à-dire que l'entraînement est effectué avec des données étiquetées (supervisé) et avant l'intégration du logiciel ML (hors ligne). Même avec ce périmètre contraint, trois manques majeurs, par rapport à la pratique actuelle, peuvent être identifiés :

- **La spécificabilité.** Cette question est liée à la façon dont la spécification des exigences est transformée en présence de parties ML. Par exemple, si l'on considère le domaine de la conduite automobile autonome, la détection d'un piéton est difficilement spécifiable, car la simple notion de "piéton" n'a jamais été réellement définie.

- La **traçabilité** est une caractéristique d'une importance capitale lors d'une certification classique, car elle conditionne l'ensemble du processus de certification. Dans le contexte des modèles ML, la phase d'apprentissage pouvant être considérée comme une boîte noire, les paramètres finaux du modèle ML ne peuvent être rattachés à aucune exigence - qui sont elles-mêmes affectées par le problème de spécificabilité mentionné précédemment.

- Certains **comportements indésirables** peuvent survenir à cause de la phase d'apprentissage, où les paramètres du modèle sont appris. Cependant, les stratégies classiques utilisées pour traiter les comportements indésirables (*e.g.*, tests logiciels ou revue de code, ce qui n'est plus possible aujourd'hui) peuvent ne pas être suffisantes ou adaptées pour les prévenir.

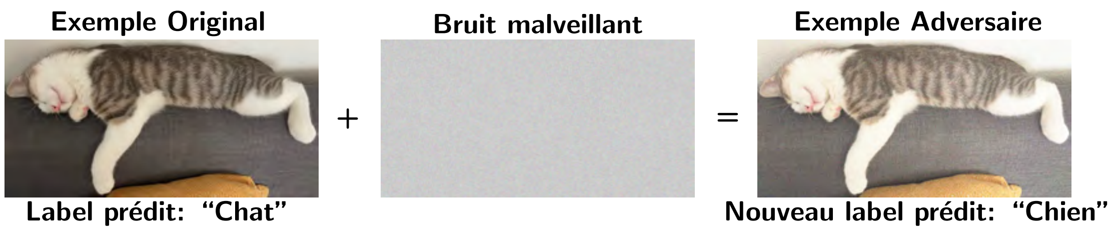

Figure 1: Exemple d'une attaque adversaire. L'image de gauche est l'original labélisé "chat" par le modèle ML. L'image du milieu est le bruit intentionnellement généré pour changer le label original. L'image de droite est l'exemple adversaire, *i.e.*, l'image originale plus le bruit mais le modèle prédit maintenant "chien".

Les impacts identifiés précédemment sur l'approche de certification actuelle soulèvent des défis à différents niveaux. L'adaptation de la méthodologie actuelle et la garantie que les données utilisées possèdent les propriétés appropriées (à déterminer) auront un impact sur la *spécificabilité*. L'explicabilité fait référence à la capacité de comprendre et d'interpréter les résultats du modèle ML et contribuera à la question de la traçabilité. Enfin, la robustesse et la prouvabilité se concentrent sur le comportement indésirable. Nous donnons un aperçu de chacun de ces défis et proposons une revue de la littérature sur l'explicabilité, la robustesse et la prouvabilité, qui sont directement liées aux caractérisques du modèle ML.

**Contribution à l'objectif O3 (challenge de la robustesse):** nous nous concentrons sur la robustesse adversaire pour aborder la question de la robustesse. Ce phénomène consiste à perturber légèrement un exemple d'entrée pour tromper le modèle ML. En d'autres termes, les exemples situés dans la plage des entrées entraînent un comportement indésirable (voir Figure 1). Ce type de robustesse n'est pas abordé par la démonstration actuelle de la conformité des systèmes classiques.

Dans notre approche, nous fournissons une technique qui limite la probabilité d'être trompé par un exemple adversaire avec un taux de confiance élevé. Pour cela, nous proposons des bornes de généralisation sur le risque robuste adversaire (c'est-à-dire la probabilité d'être trompé par un exemple adversaire) en utilisant le cadre PAC-Bayes.

**Contribution à l'objectif O3 (défi de la prouvabilité):** pour ouvrir la voie à la prouvabilité, nous nous attaquons à la vérification des exigences.

Dans la programmation classique, le développement d'un logiciel commence par l'écriture d'exigences qui offrent une description complète de son comportement. Ces exigences sont ensuite utilisées pour vérifier que le logiciel développé répond à ses exigences. Dans ce contexte, la démonstration actuelle de la conformité consiste (entre autres) à écrire des tests par rapport aux exigences.

Dans un contexte de conception pilotée par les données, ce flux de travail est modifié en raison de changements dans les techniques de spécification des exigences, notamment parce que certains des comportements attendus sont "appris" (paradigme piloté par les données).

Dans la littérature, on trouve des méthodes de vérification des propriétés des modèles ML, connues sous le nom de vérificateurs, qui prennent un modèle ML et une propriété en entrée et indiquent si la propriété est vérifiée ou non (voir Figure 2). Dans ces travaux, nous proposons une nouvelle méthode pour analyser formellement la propriété de monotonie (une propriété de sûreté) d'un réseau neuronal de substitution à l'aide d'un solveur MILP (le vérificateur).

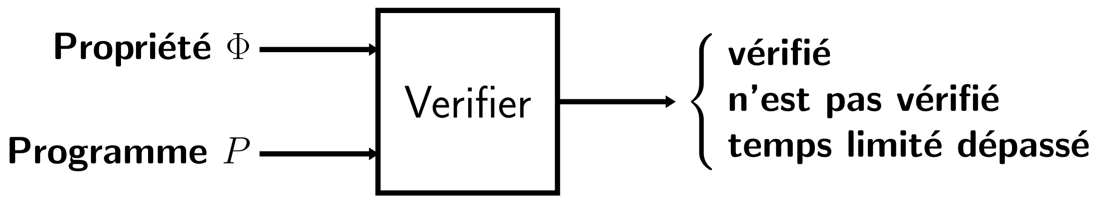

Figure 2: Principe de la vérification

Figure 3 résume notre approche descendante pour aborder la démonstration de conformité des systèmes ML. Cette figure montre les manques introduits par les systèmes à base de ML dans l'approche de certification actuelle : spécificabilité, traçabilité, et comportement indésirable. Pour chacun de ces manques, nous présentons les challenges qui y sont associés (méthodologie, données, explicabilité, robustesse, et prouvabilité). Enfin, cette illustration met en évidence le positionnement de nos principales contributions.

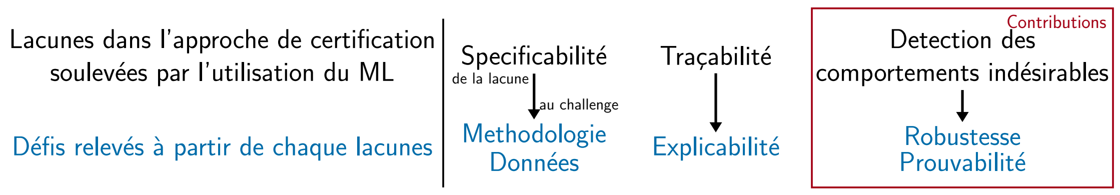
Lacunes dans l'approche de certification soulevées par l'utilisation du ML
Défis relevés à partir de chaque lacunes
Specificabilité
de la lacune
au challenge
Methodologie
Données
Traçabilité
Explicabilité
Contributions
Detection des comportements indésirables
Robustesse
Prouvabilité

Figure 3: Vue globale des lacunes et challenges abordés dans cette thèse. Le carré rouge montre où se situent nos contributions.

## Plan

Cette thèse comprend trois parties, et chaque partie est composée de chapitres.

Le chapitre Part I fournit les connaissances préliminaires pour comprendre le reste de la thèse : dans les chapitres Chapter 1 et Chapter 2 sont présentés les principes de base du domaine ML et du domaine de la vérification formelle.

Dans Part II, nous abordons les objectifs O1 et O2 en détaillant les aspects de la certification : le Chapter 3 expose la non adéquation de l'approche de la certification aux systèmes basés ML, nous investiguons les lacunes introduites par l'utilisation de systèmes basés sur le ML par rapport à la pratique de certification actuelle. En plus de cela, nous présentons les cinq challenges identifiés, et nous passons en revue la littérature existante sur les challenges dans Chapter 4.

Dans la Part III, nous nous concentrons sur l'objectif O3 et présentons nos contributions sur la robustesse et la prouvabilité (défis liés à la sûreté). Pour cela, dans le Chapter 5, nous effectuons une analyse PAC-Bayes de la robustesse adversaire. Au-delà de l'analyse, nous fournissons des bornes de généralisation sur le risque de robustesse adversaire, qui donne une garantie probabiliste de ne pas se tromper lorsque le modèle embarqué dans l'avion reçoit des entrées perturbées. Dans le chapitre Chapter 6, nous développons un algorithme pour effectuer l'analyse de monotonie d'un réseau neuronal de substitution et l'appliquons à une étude de cas avionique (estimation de la distance de freinage à l'aide d'un réseau neuronal) où la vérification de la propriété de monotonie est essentielle du point de vue de la sûreté.

# Synthèse des chapitres

## Chapitre 1

Dans ce chapitre, nous abordons la théorie de l'apprentissage automatique, en commençant par les définitions des termes de base liés au domaine de l'apprentissage automatique. Ces termes seront utilisés tout au long de la thèse. Ensuite, conformément aux contraintes réglementaires actuelles de l'aviation, cette thèse se concentrera principalement sur le paradigme de l'apprentissage dit supervisé. Nous terminons par quelques notions sur la généralisation des algorithmes ML. Nous fournissons les aspects théoriques du domaine de l'apprentissage automatique nécessaires à la compréhension du reste de ce document.

## Chapitre 2

Dans ce chapitre, nous commençons par expliquer pourquoi la vérification formelle est pertinente pour le domaine aéronautique. Ensuite, nous définissons les notions de base nécessaires au domaine de la vérification ML. Enfin, nous détaillons les principales techniques présentes dans la littérature qui effectuent de la vérification formelle sur des algorithmes ML. En particulier, nous présentons la méthode MILP que nous mettrons en pratique dans notre contribution dans le contexte de Chapter 6.

## Chapitre 3

Dans ce chapitre, l'objectif est de réduire la portée de notre travail. Pour ce faire, nous commençons par l'approche de certification actuelle, puis nous identifions les lacunes soulevées par l'utilisation de systèmes à base de ML et enfin nous développons les différents challenges pour combler ces lacunes. Ce travail préliminaire permet de déterminer la direction à prendre pour progresser vers la certification des systèmes critiques utilisant du ML.

## Chapitre 4

Ce chapitre est consacré à la revue de la littérature sur la robustesse adversaire, la prouvabilité et l'explicabilité. Avec ces trois challenges, nous couvrons une partie de l'aspect "confiance" de la certification. Notre revue de la littérature permet de résumer plusieurs pistes existantes vers la certification de systèmes à base de ML tout en mettant en évidence des axes de recherche prometteurs.

## Chapitre 5

Nous proposons les premières bornes de généralisation PAC-Bayes pour la robustesse adversaire, qui estiment, au moment du test, dans quelle mesure un modèle sera invariant aux perturbations imperceptibles de l'entrée. Au lieu de dériver une analyse du pire cas du risque d'une hypothèse sur toutes les perturbations possibles, nous tirons parti du cadre PAC-Bayes pour borner le risque moyen sur les perturbations pour le vote de majorité (sur toute la classe d'hypothèses). Notre analyse, fondée sur la théorie, a l'avantage de fournir des bornes générales *(i)* qui sont valables pour tout type d'attaques (par exemple, les attaques adversaires), *(ii)* qui sont précises grâce au cadre PAC-Bayes, *(iii)* qui peuvent être directement minimisées pendant la phase d'apprentissage pour obtenir un modèle robuste sur différentes attaques au moment du test.

## Chapitre 6

L'utilisation de la technologie ML pour concevoir des systèmes critiques nécessite une compréhension complète des propriétés d'un réseau neuronal. Parmi les propriétés pertinentes dans un contexte industriel, la vérification de la monotonie partielle peut devenir obligatoire pour certains modèles ML. Ce chapitre propose une méthode pour évaluer la propriété de monotonie à l'aide d'un solveur de programmation linéaire partiellement en nombre entier (MILP). Contrairement à la littérature existante, cette analyse de la monotonie fournit une borne inférieure et une borne supérieure du volume de l'espace où la propriété ne tient pas, que nous dénommons "couverture de l'espace non monotone". Ce travail présente plusieurs avantages : *(i)* notre formulation de la propriété de monotonie fonctionne sur des entrées discrètes, *(ii)* la nature itérative de notre algorithme permet de raffiner l'analyse au besoin, et *(iii)* d'un point de vue industriel, les résultats de cette évaluation sont précieux pour le domaine aéronautique où ils peuvent soutenir la démonstration de certification. Nous avons appliqué cette méthode à un cas d'étude avionique (estimation de la distance de freinage à l'aide d'un réseau neuronal) où la vérification de la propriété de monotonie est d'un intérêt primordial du point de vue de la sûreté.

## Conclusion

Dans cette thèse, nous nous sommes intéressés à la certification des systèmes à base de ML. Nous évoluions dans un environnement riche où plusieurs acteurs abordaient le même sujet (*e.g.*, EASA, le projet DEEL[1], ou Daedalean[2]). Malgré le contexte exigeant abordé par cette recherche, nous avons pu obtenir plusieurs innovations théoriques qui s'avèrent avoir un véritable intérêt industriel. Nous avons abordé cette tâche ambitieuse en analysant et en définissant le champ de notre recherche principale (objectifs O1 et O2 du chapitre d'introduction), puis en nous concentrant sur la question de la robustesse et de la prouvabilité dans le domaine spécifique de la démonstration de conformité (objectif O3).

En effet, dans le chapitre Chapter 3, nous effectuons une analyse de la différence entre la démonstration de conformité d'un système classique et celle d'un système basé sur le ML, ce qui soulève plusieurs challenges. A la lumière de notre revue de littérature sur les différents défis et les attentes des autorités, nous avons décidé de nous concentrer sur les aspects de sûreté.

Nous avons abordé la question de la sûreté de deux manières complémentaires : du point de vue de la conception — le pré-apprentissage (challenge de la robustesse), et du point de vue de la vérification — le post-apprentissage (challenge de la prouvabilité). Dans Chapter 5, nous fournissons des bornes de généralisation sur les risques adversaires relaxés des modèles ML, qui peuvent être optimisés pendant une phase d'apprentissage robuste. En d'autres termes, nous fournissons une méthode pour obtenir des modèles ML robustes aux attaques adversaires avec des garanties probabilistes. Cette méthode est conforme aux exigences de robustesse et de généralisation spécifiées dans EASA (2021b). Dans Chapter 6, nous traitons de la vérification de la propriété de monotonie d'un réseau neuronal de substitution. Nous proposons une nouvelle méthode d'analyse de la propriété de monotonie avec des entrées mixtes (*i.e.*, entières et réelles). Cette nouvelle méthode s'appuie sur un solveur MILP et fournit des bornes inférieures et supérieures sur la proportion de l'espace d'entrée qui viole la propriété de monotonie. Avec cette technique, nous répondons partiellement à l'objectif concernant la vérification d'un modèle ML énoncé dans EASA (2021b).

En bref, chacune de nos contributions a été motivée par le besoin de démonstration de la conformité. En particulier, nous nous attaquons aux exigences de fiabilité (telles que la généralisation, la robustesse ou la vérification des propriétés de sûreté), qui reflètent l'essence même de la certification, à savoir garantir la sûreté. Pour chacune d'entre elles, nous avons proposé des approches innovantes qui abordent la question de la sûreté tout en répondant à certaines exigences de la première directive publiée par l'EASA (EASA, 2021b).

[1]`https://www.deel.ai/`
[2]`https://daedalean.ai/`

## Perspectives

Les travaux sur les bornes de généralisation des risques robustes adversaires relaxés (Chapter 5) donnent lieu à de nombreuses questions et pistes de recherche intéressantes. Une idée serait de se concentrer sur l'extension de nos résultats à d'autres contextes de classification, tels que multiclasse ou multilabel.

Une autre ligne de recherche consisterait à tirer parti d'autres outils de la littérature PAC-Bayes. Parmi eux, on pourrait utiliser d'autres bornes sur le risque du vote de majorité qui prennent en considération la diversité entre chacun des votants. Par exemple, la C-borne (Lacasse *et al.*, 2006), ou plus récemment la perte tandem (Masegosa *et al.*, 2020c). Une autre borne PAC-Bayes très récente pour les votes de majorité qui doit être étudiée dans le cas de la robustesse adversaire est celle proposée par Zantedeschi *et al.* (2021) qui a l'avantage d'être directement optimisable avec la perte 0-1. Dans les applications de la vie réelle, on souhaite souvent combiner différentes sources d'entrée (provenant de différents capteurs, caméras, etc.). Être capable de combiner ces sources de manière efficace est alors un problème crucial.

Dans le domaine aéronautique, la distribution du jeu de données d'entraînement peut différer du domaine de conception opérationnelle (ODD). En effet, les ingénieurs peuvent augmenter l'occurrence des cas extrêmes pendant la phase d'entraînement afin d'atteindre des performances suffisantes et donc de maintenir un haut niveau de sécurité pour ces cas extrêmes. Ainsi, une perspective intéressante serait de considérer l'adaptation de domaine dans notre cadre PAC-Bayes robuste afin d'être capable de gérer différentes distributions tout en ayant des garanties sur la robustesse du modèle.

En ce qui concerne le travail sur la vérification formelle (Chapter 6), la méthode proposée laisse place à quelques améliorations, comme des valeurs big-M plus précises en utilisant des méthodes incomplètes fournissant des bornes pour les couches intermédiaires du réseau neuronal (Xu *et al.*, 2020) ou l'utilisation de bornes asymétriques (Tjeng *et al.*, 2019). Comme extension de cette analyse de la monotonie, nous prévoyons d'estimer l'intégrale sous la courbe de $h(x_1) - h(x_2)$ dans les sous-espaces où la monotonie est violée en exploitant notre définition de la propriété de monotonie. Cela donnerait un indicateur clé du niveau de violation de la propriété de monotonie qui pourrait contribuer à la performance de la phase d'entraînement. Une autre suite naturelle de notre travail serait d'étendre l'analyse que nous avons effectuée dans un contexte discret, à des caractéristiques continues en utilisant la formulation de la monotonie basée sur le gradient. Notre approche a prouvé l'adéquation de l'analyse de la monotonie à des problèmes industriels réels. Ainsi, une perspective serait d'abord de développer un outil industriel avec la méthode actuelle qui sera, dans le futur, complétée avec les nouvelles fonctionnalités mentionnées ci-dessus.

A notre connaissance, la mise à l'échelle des méthodes complètes reste un défi dans la communauté de la vérification et est principalement utilisée avec des réseaux neuronaux "peu profonds". Par conséquent, il reste beaucoup à faire pour que la vérification s'étende aux modèles profonds. Une perspective pour accélérer le solveur MILP pour la vérification des réseaux neuronaux est de considérer la dépendance entre les activations ReLU pour élaguer l'arbre de recherche du solveur MILP comme proposé dans Botoeva *et al.* (2020).

Il est clair qu'il reste beaucoup à faire afin de prendre en compte la présence d'éléments basés sur du ML dans les systèmes certifiés. Cela pose des questions de recherche stimulantes, entraine des pratiques industrielles innovantes et ouvre la voie à de nouvelles pratiques dans le contexte des systèmes avioniques et pas seulement. La présente contribution est un modeste pas dans cette direction et vise à montrer le réalisme d'une telle vision. Enfin, cette thèse est la preuve tangible que l'une des clés pour avancer vers la démonstration de la conformité des systèmes basés sur du ML est la collaboration étroite entre le monde universitaire et l'industrie.

# Introduction

## Thesis work environment

This thesis synthesizes the results of joint research between the Airbus company and IRIT (Institut de Recherche en Informatique de Toulouse - Toulouse Computer Science Research Institute), made possible by an ANRT (Association Nationale Recherche Technologie) and Airbus founding (CIFRE - Project). This collaboration was initiated to take advantage of the expertise on systems certification held by Airbus and the experience in the formal verification of cyber-physical systems held by IRIT.

## Context

Safety critical systems, and avionics, in particular, represent an application field unenthusiastic in applying new software development methods (KRODEL, 2008), as shown, for instance, by the fact that some aspects of the object-oriented programming, such as polymorphism and dynamic binding, never made their way to safety-critical systems, mostly because of the inconvenient balance between added value and increased development and validation costs. Nevertheless, the recent advances in Machine Learning (ML) triggered genuine interest, as machine learning offers promising preliminary results and opens the way to a wide range of new functions for avionics systems aiming at developing new flight experiences: navigation/surveillance assistance (*e.g.*, vision-based navigation, obstacle sensing and virtual sensing), speech-to-text applications, autonomous flight, predictive maintenance, cockpit assistance, etc.

Contrary to *classical programming*, which can hardly support these functions, Machine Learning, which is a sub-domain of Artificial Intelligence (AI), is well known to show good results for most of them (*e.g.*, BAHDANAU *et al.*, 2015; DEVLIN *et al.*, 2019; REDMON *et al.*, 2016). ML techniques introduce a new paradigm in avionic development: *the data-driven design* of models (supervised, unsupervised, and reinforcement learning). The characteristics of data-driven design are mainly related to the completely new form that the requirements specification takes. Instead of having the requirements provided entirely by the system engineer, the data will characterize (parts of) it in data-driven design.

*Data-driven* refers to the fact that the data determines the algorithm behavior through a learning phase. Current industrial guidances (*e.g.*, DO-178x) are oriented on established technologies in aeronautical applications, with a strong focus on requirement specification and how these requirements are captured and satisfied by the final product. However, this current practice does not fit the development paradigm change.

## Objectives and Contributions

The high-level goal of this thesis is to explore the range of possibilities in terms of compliance demonstration of ML-based systems to regulation constraints. This goal can be decomposed into several objectives:

O1 **Identify the impacts of using an ML-based system on the current certification approach.** This objective consists in defining how the current certification approach is not adapted to ML-based systems and defining the gaps introduced by using ML-based systems with respect to the current certification practice.

O2 **Identify the challenges raised by the previously identified impacts.** The objective consists in investigating how it is possible to reduce or overcome the identified gaps so that the certification process can be applied to systems containing ML parts.

O3 **Propose new means to overcome some previously identified challenges.** Due to the limited timeline of this research, we cannot address all the challenges. To make a selection, we consider the needs of the certification. As the major purpose of the demonstration of conformity is to ensure safety, we focus on challenges related to safety issues.

Our research addresses these objectives through several contributions detailed later in this document.

**Contribution to objective O1 and O2:** In order to identify the impacts of using an ML-based system on the current certification approach, we perform a gap analysis of the current certification approach for the compliance demonstration of ML-based systems and a literature review on the raised challenges.

To identify noteworthy gaps between the verification of classical software and the verification of software containing parts obtained through ML, we must first define the scope of the ML domain considered for embedding. Taking as a reference the current authorities standpoint on this issue (EASA, 2020, 2021b), this research will focus on *offline supervised learning*, *i.e.*, the training is performed with *labelled data* (supervised) and *before embedding the ML software* (offline). Even with this constrained perimeter, three majors gaps, with respect to current practice, can be identified:

i The **specifiability**. This issue relates to the way requirement specification is transformed in the presence of ML parts. For example, if we consider the autonomous automotive driving domain, the detection of a pedestrian is hardly specifiable, as the mere notion of "a pedestrian" was never actually defined.

ii The **traceability** is a characteristic of paramount importance during classical certification, driving the entire certification process. In the context of ML models, as the learning phase can be seen as a black box, the final parameters of the ML model cannot be traced back to any requirements - which themselves are affected by the specifiability issue mentioned before.

iii Some **unintended behavior** may arise because of the training phase, where the model's parameters are learned. However, the classical strategies used to deal with unintended behavior (*e.g.*, software tests or code review, which is not possible anymore) may neither be sufficient nor adapted to prevent it.

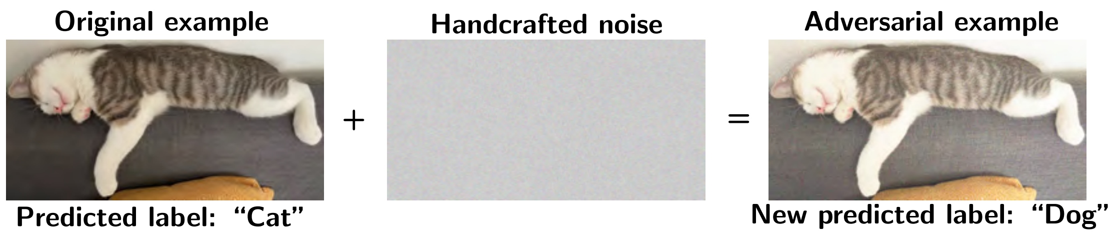

Figure 4: Example of an adversarial attack. The left image is the original one predicted as a "cat" by the ML model. The middle image is the purposefully generated noise that is maliciously inserted to change the original label. The right image is the adversarial example, *i.e.*, the original image plus the noise; the ML model now predicts "dog".

The previously identified impacts on the current certification approach raise challenges on various levels. Adapting the current *methodology* and ensuring the used *data* has the suitable properties (to be identified) will impact the specifiability issue. *Explainability* refers to the capacity to understand and interpret ML results and will help with the traceability issue. Finally, the *robustness* and *provability* focus on unintended behavior. We provide an overview of each of these challenges and propose a literature review on explainability, robustness, and provability, which are directly linked to the characteristics of the ML model.

**Contribution to objective O3 (robustness challenge):** We focus on *adversarial robustness* to address the robustness. This phenomenon consists in slightly perturbing an input example to fool the ML model. That is, examples lying within the range of the inputs lead to unintended behavior (see Figure 4). This kind of robustness is not tackled by the current demonstration of conformity of classical systems.

In our approach, we provide a technique that bounds the chance of being fooled by an adversarial example with a high confidence rate. For this, we propose generalization bounds on the adversarial robust risk (*i.e.*, the probability of being fooled by an adversarial example) using the PAC-Bayesian framework.

**Contribution to objective O3 (provability challenge):** To open the way for provability, we tackle the requirement verification.

In classical programming, the development of software starts with the writing of requirements that offer a complete description of its behavior. These requirements are later used to verify that the developed software meets its requirements. In this context, the current demonstration of conformity consists of (among others) writing tests with respect to the requirements.

In a data-driven design context, this workflow is altered due to modifications in the requirement specification techniques, particularly, since some of the expected behavior is "learned" (data-driven paradigm).

In the literature, one can find methods to verify ML model properties known as verifiers that take an ML model and a property as input and output whether or not the property holds (see Figure 5). In this research, we propose a new method to formally analyze the monotony property (a safety property) of a surrogate neural network using MILP solver (*i.e.*, the verifier).

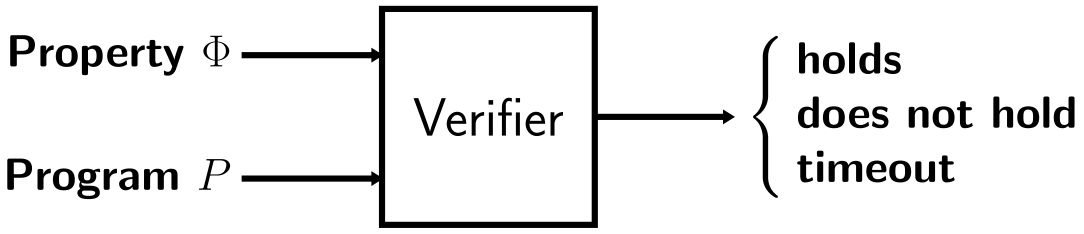

Figure 5: Verification principle

Figure 6 summarizes our top down approach for addressing compliance demonstration ML-systems. This figure shows the gaps introduced by ML-based systems in the current certification approach: specifiability, traceability, and unintended behavior. For each of these gaps we explicit the challenges associated to it (methodology, data, explainability, robustness, and provability). Finally, this illustration highlights the positioning of our main contributions.

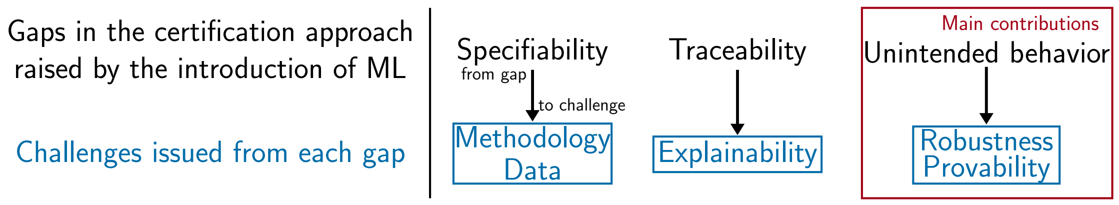

Figure 6: Overview of the gaps and challenges discussed in this thesis. The red square highlights where stand out main contributions.

## Outline

This thesis comprises three parts, and each part is composed of chapters.

Part I provides the preliminary knowledge to understand the rest of the thesis: in Chapter 1 and Chapter 2 introduces the basic principles of the ML domain and the formal verification domain.

In Part II, we address the objectives O1 and O2 by detailing the certification aspects: in Chapter 3 we elaborate on the fact that the certification approach is not adapted to ML-based systems, we investigate on the gaps introduced by the use of ML-based systems with respect to the current certification practice. Furthermore, we present the five challenges, and we review the existing literature on challenges in Chapter 4.

In Part III, we focus on the objective O3 and present our contributions on the robustness and provability (safety-related challenges). For this, in Chapter 5, we perform a PAC-Bayes analysis of the adversarial robustness. Beyond the analysis, we provide generalization bounds on the adversarial robust risk, which gives a probabilistic guarantee of not failing when the embedded model in the aircraft receives perturbed inputs. In Chapter 6, we develop an algorithm to perform the monotony analysis of a surrogate neural network and apply it to an avionic case study (braking distance estimation using a neural network) where the verification of the monotony property is essential from a safety perspective.

# Part I

# Preliminaries

# 1 Machine Learning

## Contents

**Abstract**

In this chapter, we tackle the fundamental machine learning theory, starting with the definitions of basic terms related to the Machine Learning domain. Then, in accordance with the current aviation regulatory constraints, this thesis will mainly focus on the so-called supervised learning paradigm. We finish with some notions on the generalization of ML algorithms. We provide the necessary theoretical aspects of the machine learning domain required to understand the remaining of this document.

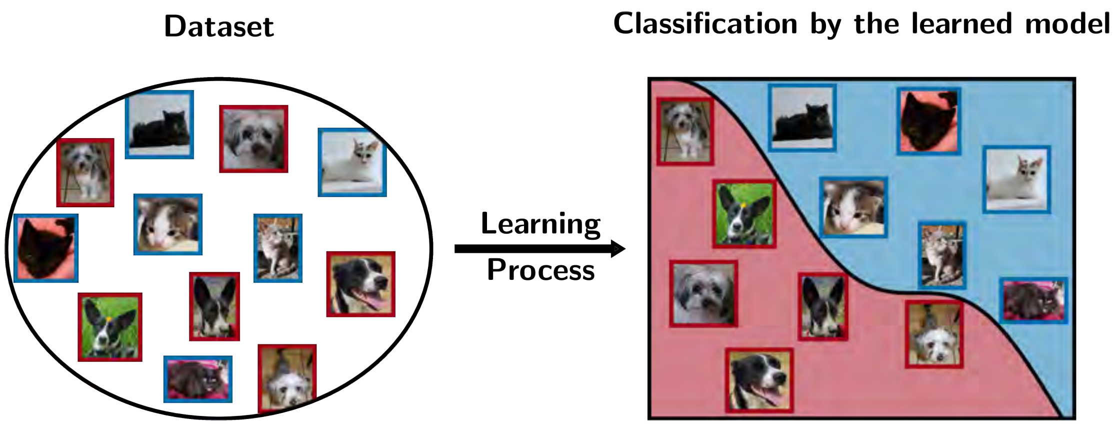

Figure 1.1: Sketch of a dog and cat images classification. Starting from the dataset to the learned machine learning model. The border color on the images represents the true label (red for dog and blue for cat). On the right, the background color represents the model's prediction with the same color code.

## 1.1 Introduction

Machine Learning (ML) is very attractive for the aeronautical domain since it allows the development of functions that are not conceivable considering the "classical" software process methods (*e.g.*, obstacle detection, speech-to-text, or autonomous taxiing). However, ML introduces an entirely new development paradigm into the domain of safety-critical embedded systems. The development of systems using "classical" software is based on explicit programming, whereas this is no longer the case when developing ML-based systems: we speak of data-driven development. Thus, this chapter presents the fundamental theory behind ML. This knowledge will support understanding the different challenges raised in the development of safety-critical ML-based systems (see Chapter 3).

## 1.2 Basic definitions

Machine Learning aims to make a computer capable of solving a "real-life" problem. Contrary to classical software (*i.e.*, software explicitly programmed by the developer), machine learning software is self-made by learning from data to solve problems that classical software cannot make (*e.g.*obstacle detection on a runway).

Figure 1.1 gives the intuition of what "learning from data" means by depicting an example of cats' and dogs' image recognition tasks. The left side of Figure 1.1 shows a collection of dog and cat images, namely the dataset. The knowledge to recognize cats from dogs must be contained in the dataset to provide a well-trained ML algorithm. If this assumption is fulfilled, the learning process should provide a machine learning model that correctly separates the dog images from the cat images, as shown on the right side of Figure 1.1.

Figure 1.2 gives an overview of a standard Machine Learning development pipeline. During all these steps, we may encounter different terms specific to the ML process (defined hereafter). Figure 1.2 details the process pictured by Figure 1.1. To ease the reading of the thesis, we provide some basic notions:

- **Problem and Task**: problem refers to the real-world issue the machine learning model will solve, while the task refers to what the model does to solve the problem. In Figure 1.1, the real-life problem consists in recognizing whether it is a dog or a cat in images, and the model performs a classification task to tackle the problem.
- **Data collection**: step in which the data related to the problem and the task is collected. Note that this step is crucial since the model performance relies on the available data.
- **Dataset**: the data have been collected. During the development of machine learning models, at least three types of datasets are encountered; the training dataset for "learning" the model, the validation dataset to select the best hyper-parameters (see definition below), and the testing dataset to assess the model's performance. In Figure 1.1, the dataset corresponds to the left part component.
- **Example and features**: An example denotes the data itself and is composed of features. In Figure 1.1, an example is an image of a cat or a dog, and the features are the pixels of the image.
- **Label**: the annotation comes with an example. In Figure 1.1, the labels are cat and dog. Note that having labels is not mandatory; it depends on the machine learning paradigm used, which is described later.
- **Model/Hypothesis**: the mathematical function optimized/learned to solve the problem. Through this thesis, we will use both terms interchangeably.
- **Set of hypothesis**: the family of mathematical functions selected for solving the problem. For instance, a family of hypotheses could be all linear functions.
- **Learning**: the optimization algorithm used to select the best parameters among the model parameters.
- **Loss function**: the mathematical function used during the learning to guide the optimization. It measures the error made by the ML model.
- **Hyper-parameters**: the parameters not learned by the optimization algorithm and fine-tuned by the developer.

Once the problem and task are defined, the ML development pipeline can be split into two processes: data management and design. If the data is not yet available, they must be collected. Usually, the data might be cleaned, labeled, and then preprocessed (*i.e.*, give the data a suitable form for the selected algorithm). During the design process, we iterate between the training and validation, which permits tuning the hyper-parameters and yields the best possible algorithm. The final step is the testing phase, where the ML model's performance is assessed to determine whether the model is ready for deployment.

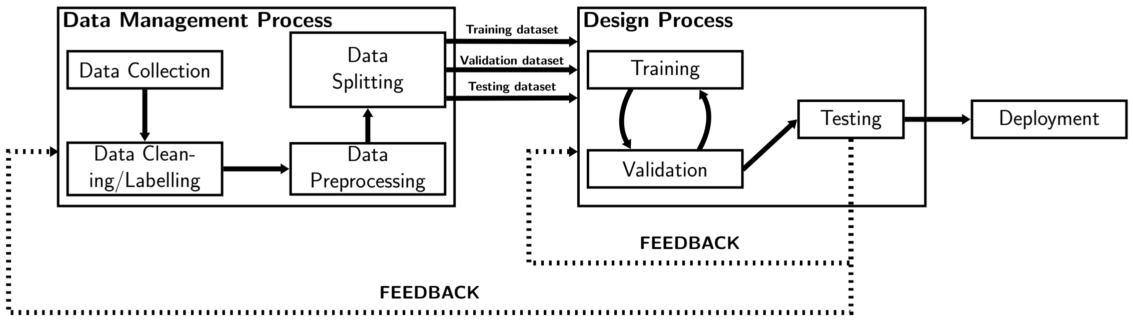

Figure 1.2: Machine Learning development pipeline.

## 1.3 Supervised Learning

Machine Learning gathers three paradigms: supervised, unsupervised, and reinforcement learning. However, the actual guidance of the authorities for embedded ML-based systems is applicable for *offline* supervised learning.(EASA, 2021b). In this context, offline means the ML model is frozen before embedding. Thus, we focus in this section on the supervised paradigm.

### Setting

Supervised learning is a sub-category of machine learning algorithms where the problem is solved by learning a model from labeled data. More formally, we have an input space $\mathbb{X} \subseteq \mathbb{R}^d$, defining $d$-dimensional examples, and an output space $\mathbb{Y} \subseteq \mathbb{R}$ describing the labels. We assume a fixed but unknown distribution $\mathcal{D}$ on $\mathbb{X}\times\mathbb{Y}$ that describes the probability of drawing a specific example. We denote by $S = \{(x_i, y_i)\}_{i=1}^m \subseteq \mathbb{X}\times\mathbb{Y}$ the dataset comprising $m$ examples independently and identically distributed *(i.i.d.)* from $\mathcal{D}$; We denote the distribution of this m-sample by $\mathcal{D}^m$ and thus $S \sim \mathcal{D}^m$. Let $\mathcal{H}$ be the set of hypotheses $h \in \mathcal{H}$ constituted of real-valued functions $h : \mathbb{X} \mapsto \mathbb{Y}$.

### Learning Process

#### Learning the model parameters

First, we suppose that a function $f$ that perfectly maps $\mathbb{X}$ to $\mathbb{Y}$ exists: $y = f(x)$. We say that $f$ is the oracle for the given task. Then, the goal of the learning process is to find the hypothesis $h \in \mathcal{H}$, which approximates $f$ as well as possible. We need a criterion $\ell : \mathcal{H} \times (\mathbb{X}\times\mathbb{Y})$, that measures the error made by $h$. The loss function $\ell(\cdot,\cdot)$ takes an hypothesis $h \in \mathcal{H}$ and a tuple of example and label $(x, y) \in \mathbb{X}\times\mathbb{Y}$ as input, compares the predicted label $h(x)$ and $y$ and return a real value reflecting the error made by the hypothesis $h$. The typical loss functions used for classification and regression tasks are presented in Section 1.3. The *true risk* — or *true error* — of a model $h$ is defined by taking the expectation (denoted $\mathbb{E}$) of the loss (denoted $\ell$) considering the distribution $\mathcal{D}$ of the data, *i.e.*, it is equivalent to considering that all existing data is available.

**Definition 1.3.1.** *True Risk.*
Let $\ell : \mathcal{H} \times (\mathbb{X}\times\mathbb{Y}) \mapsto \mathbb{R}$ be the loss function. Given $h \in \mathcal{H}$ and the distribution $\mathcal{D}$ on $\mathbb{X}\times\mathbb{Y}$, we have the true risk defined as

$$\mathcal{R}_{\mathcal{D}}(h) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \ell(h,(x,y)) \tag{1.1}$$

However, the distribution $\mathcal{D}$ is unknown. Thus, Equation (1.1) is not computable and the *true risk* of a model $h$ remains unknown. Since the *true risk* $\mathcal{R}_{\mathcal{D}}(h)$ is not computable, we compute an unbiased estimator of it using the training dataset, known as the *empirical risk*.

**Definition 1.3.2.** *Empirical Risk*
Let $\ell : \mathcal{H} \times (\mathbb{X}\times\mathbb{Y}) \mapsto \mathbb{R}$ be the loss function. Given $h \in \mathcal{H}$ and the training dataset $S \sim \mathcal{D}$, we have the empirical risk defined as

$$\mathcal{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h,(x_i,y_i)) \tag{1.2}$$

Finally, to learn the best hypothesis $h$ with respect to the training data $S$ and the loss function $\ell(\cdot,\cdot)$, we have to minimize the *empirical risk* $\mathcal{R}_S(h)$. This principle is known as Empirical Risk Minimization (ERM).

**Definition 1.3.3.** *Empirical Risk Minimization*

$$h^* \in \operatorname*{argmin}_{h\in\mathcal{H}} \mathcal{R}_S(h) \tag{1.3}$$

In this case, $\operatorname{argmin}_{h\in\mathcal{H}}$ represents the set of all hypotheses that minimize $\mathcal{R}_S(h)$. Hence, to obtain the hypothesis $h^*$, the $\mathcal{R}_S(h)$ is minimized during the optimization. With the empirical risk minimization, we obtain a model that performs well on the training dataset $S$ for a given task and a set of hypotheses $\mathcal{H}$, *i.e.*, the selected family of models. However, nothing guarantees to perform well on data that does not belong to the training dataset (*i.e.*, data not known by the model) or that the set of hypotheses $\mathcal{H}$ is well suited for the problem. These uncertainties potentially enable two phenomena known as overfitting and underfitting. Intuitively, overfitting means that the model learned the training dataset "by heart". Overfitting is detected when the empirical risk $\mathcal{R}_S(h)$ (the risk at training time) is low and the risk a testing time, denoted $\mathcal{R}_T(h)$ is high (where $T$ is the testing dataset different from the training dataset). Underfitting is the opposite of overfitting: intuitively, the model does not fit the data and thus does not solve the problem. When underfitting occurs, the training and testing errors (respectively, $\mathcal{R}_S(h)$ and $\mathcal{R}_T(h)$) are high. Figure 1.3 illustrates both phenomena and details the training and testing risks for each case. When the hypothesis performs well

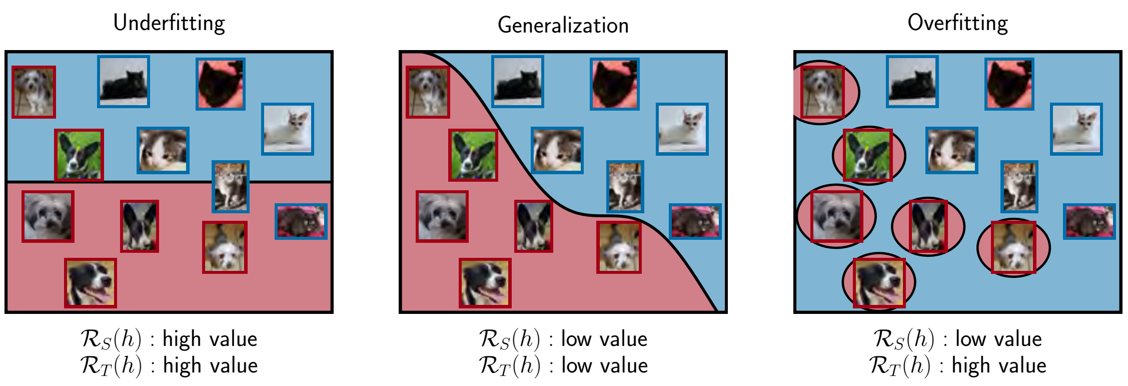
Underfitting
Generalization
Overfitting
$\mathcal{R}_S(h)$ : high value
$\mathcal{R}_T(h)$ : high value
$\mathcal{R}_S(h)$ : low value
$\mathcal{R}_T(h)$ : low value
$\mathcal{R}_S(h)$ : low value
$\mathcal{R}_T(h)$ : high value

Figure 1.3: Graphic representation of underfitting (left plot), overfitting (right plot), and generalization (center plot). $\mathcal{R}_S(h)$ is the training error, and $\mathcal{R}_T(h)$ is the testing error.

on unseen data, as illustrated on Figure 1.3's center plot, we conclude that the model can generalize (see Section 1.4).

In a nutshell, the key criterion involved in both phenomena is the complexity of the set of hypotheses: overfitting often occurs when the set of hypotheses $\mathcal{H}$ could learn complex behavior, whereas underfitting occurs when the set of hypotheses learns only simple behavior (*e.g.*, linear functions). Following the *empirical risk minimization* paradigm, a family of models too simple will lead to underfitting, while a too complex one will lead to overfitting. Consequently, to learn the task, there is no choice but to take the risk of overfitting and to choosing a set of hypotheses composed of sufficiently complex functions.Fortunately, a simple way to prevent overfitting consists of penalizing the model's complexity and is known as *regularization*. We talk about *regularized empirical risk minimization*, which is formally stated as:

**Definition 1.3.4.** *Regularized Empirical Risk Minimization*

$$h^* \in \underset{h\in\mathcal{H}}{\operatorname{argmin}} \left(\mathcal{R}_S(h) + \mu\texttt{penalization}(h)\right) \tag{1.4}$$

The *regularized empirical risk minimization* differs from the *empirical risk minimization* by the penalization term $\mu\texttt{penalization}(h)$. The penalization function takes as input a model $h$ and return the corresponding penalization value. For example, an $L_1$-norm (denoted $\|\cdot\|_1$ and defined as $\|x\|_1 = \sum_i |x_i|$) on the model's parameters, could be used as a penalization function. Intuitively, the hyper-parameter $\mu$ calibrates the penalization. On the one hand, an arbitrarily large value for $\mu$ indirectly makes the risk negligible for minimization. Therefore, the only way to minimize Equation (1.4) is to have a small complexity penalization value for the hypothesis. Note that the larger $\mu$ is, the higher the chance to underfit. On the other hand, a very small value for $\mu$ makes $\mu\texttt{penalization}(h)$ close to $0$ and then has no impact on the minimization. Although regularization helps to avoid overfitting, the hyper-parameter $\mu$ requires fine-tuning.

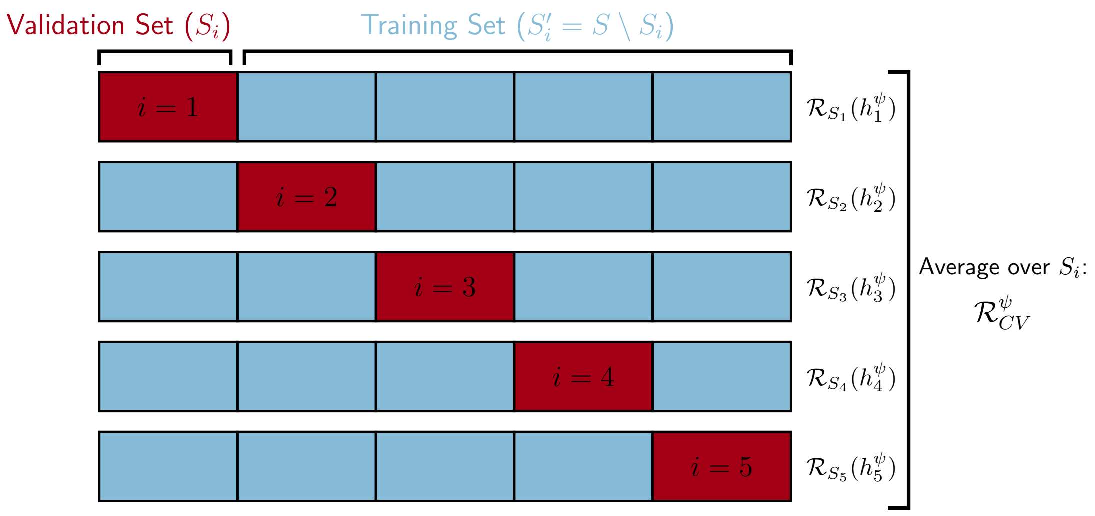
Validation Set ($S_i$)
Training Set ($S'_i = S \setminus S_i$)
$i = 1$
$i = 2$
$i = 3$
$i = 4$
$i = 5$
$\mathcal{R}_{S_1}(h_1^\psi)$
$\mathcal{R}_{S_2}(h_2^\psi)$
$\mathcal{R}_{S_3}(h_3^\psi)$
$\mathcal{R}_{S_4}(h_4^\psi)$
$\mathcal{R}_{S_5}(h_5^\psi)$
Average over $S_i$:
$\mathcal{R}_{CV}^\psi$

Figure 1.4: Illustration of 5-fold cross-validation. Each row represents a different split of the dataset. The red rectangle is the validation set in each split, and the blues are the training set.

### Hyper-parameters tuning

Hyper-parameters may strongly impact on the resulting machine learning model, *e.g.*, the parameter $\mu$ of the regularized empirical risk minimization. So, in this example, tuning means testing different values of $\mu$ and selecting the value that results in the model with the best performance. Usually, there is more than one hyper-parameter; hence numerous combinations of hyper-parameters should be tested to know which configuration provides the best performance. For instance, when learning a neural network, the hyper-parameters set comprises the number of layers, the number of neurons per layer, the activation function used, the learning rate, the batch size, etc. K-fold cross-validation is a well-known method to fine-tune the hyper-parameters (Bishop, 2006).

Figure 1.4 informally describes the principle of cross-validation. First, we consider a set $\Psi$ composed of the possible combinations of hyper-parameters instances. Then, the training dataset $S$ is partitioned into $k$ subset[1]: $k$ different training and validation dataset is constructed by taking $S \setminus S_i$ and $S_i$, for $i \in [1, k]$, respectively. For each hyper-parameter instance, $k$ models are learned from each dataset, and finally, the cross-validation error is obtained by averaging the validation error of the $k$ models. The selected combination of hyper-parameters instances is the one that yields the best cross-validation error. Algorithm 1 formalizes the cross-validation method.

Intuitively, the cross-validation technique creates diversity in the dataset to train several models to obtain a better estimation of the training error, which provides more confidence in fine-tuning hyper-parameters. This method is even more relevant when the training dataset is small because it allows using the few amounts of data available efficiently. Note that selecting the hyper-parameters using cross-validation may be time-consuming since $k \times |\Psi|$ (where $|\cdot|$ means "cardinality of") models must be learned.

[1] "partitioned" means that we have $k$ disjoint subset of the training dataset $S$.

**Algorithm 1** Hyper-parameters selection trough Cross Validation

```
Require: the set of combination of hyper-parameter instances $\Psi$, the training dataset
    $S$, and the number of split $k$
1: $\{S_1, ..., S_k\} \leftarrow$ partition $S$ in $k$ subsets.
2: for each $\psi \in \Psi$ do
3:     for each $S_i \in \{S_1, ..., S_k\}$ do
4:         $h_i^\psi \leftarrow$ train the model with $S \setminus S_i$ with respect to $\psi$
5:         $\mathcal{R}_{S_i}(h_i^\psi) \leftarrow$ compute the validation error with $S_i$
6:     end for each
7:     $\mathcal{R}_{CV}^\psi \leftarrow \frac{1}{k}\sum_{i=1}^{k}\mathcal{R}_{S_i}(h_i^\psi)$          ▷ Average of the risks for a given $\psi$
8: end for each
9: $\psi^* \leftarrow \operatorname{argmin}_{\psi\in\Psi}\mathcal{R}_{CV}^\psi$
Return: The best combination of hyper-parameter instances $\psi^*$
```

## What tasks can be tackled ?

### Classification

Classification denotes the task where the examples are ordered into different categories. The formalisation in Section 1.3 is the general setting; Example 1.3.1 shows an example of a binary classification task setting based on the task described in Figure 1.1.

> **Example 1.3.1.** *Toy Example: cats and dogs classification (see Figure 1.1)*
> Let $\mathbb{X} = [0, 1]^d$ be the input space describing the RGB images of cats and dogs. Let $\mathbb{Y} = \{-1, 1\}$ be the output space, where $-1$ and $1$ mean cat and dog, respectively. We assume a fixed but unknown distribution $\mathcal{D}$ on $\mathbb{X}\times\mathbb{Y}$, *i.e.*, the space representing the labeled images of cat and dog, that describes the probability of drawing a specific image. We have $S \sim \mathcal{D}^m$ the dataset comprises $m$ examples *i.i.d.* from $\mathcal{D}$. Finally, let $\mathcal{H}$ be the set of hypotheses gathering the real-valued functions mapping $\mathbb{X}$ to $\mathbb{Y}$ and $h \in \mathcal{H}$ be the optimized function that classifies the images.

We based this example of formalization on the general setting (Section 1.3), and we only need to set up the definitions of $\mathbb{X}$ and $\mathbb{Y}$ more precisely. Having $\mathbb{Y} = \{-1, 1\}$ is typical for binary classification task; if not specified otherwise, we assume in the rest of the thesis that the labels for binary classification are $-1$ and $1$. Note that at this stage, a stronger assumption would have been made on the hypotheses set — or family — but we chose not to.

The intuitive way to measure the efficiency of a model performing a classification task is to count the number of mistakes; a wrong prediction is a mistake, while a correct one is not. It is mathematically expressed with the 0-1 loss: when the model makes a wrong prediction, the loss returns 1 and 0 otherwise.

$$\ell_{\mathbf{01}}(h, (x, y)) = \mathbf{I}(h(x) \neq y) \tag{1.5}$$

where $\mathbf{I}(c) = 1$ when $c$ is true and $0$ otherwise.

Then, the model's performance is computed by taking the average over the data, and according to the ERM, this averaged loss must be minimized. However, optimizing the 0-1 loss is intractable due to its non-differentiability. Hence it can not be used for the learning process. A solution is to use a surrogate loss function approximating the 0-1 loss behavior. Popular surrogates exist, like the hinge loss or exponential loss used for Support Vector Machine and Boosting, respectively. The surrogate used in this thesis is standard in the context of generalization and is known as the linear loss:

$$\ell_{linear} = \frac{1}{2}(1 - y * h(x)) \tag{1.6}$$

$y * h(x)$ is positive when the model predict correctly (*i.e.*, $y$ and $h(x)$ are both $1$ or both $-1$) and negative otherwise. Hence, the linear loss, $\ell_{linear}$ is bounded by $0$ and $1$; the further $h(x)$ is from $y$, the closer the loss will be to $1$ (see Figure 1.5).

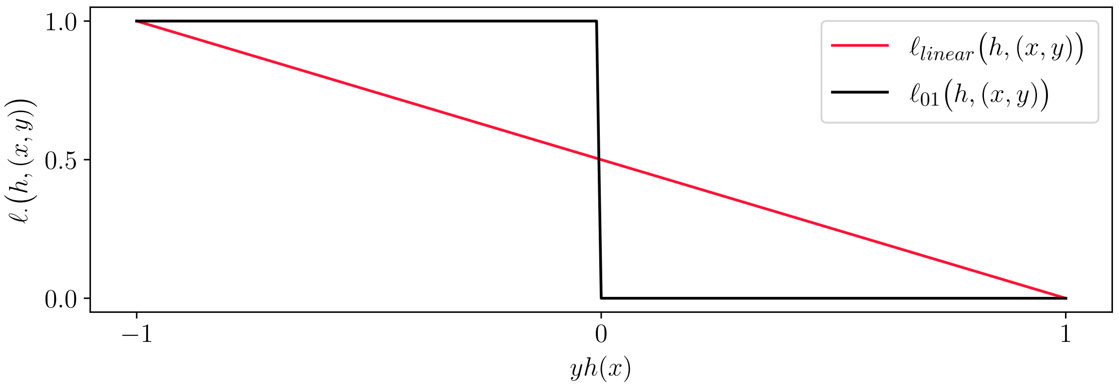

Figure 1.5: Representation of the 0-1 loss and the surrogate linear loss

### Regression

The regression is the task where the model predicts a continuous value based on the different features from the input space, *e.g.*, predicting the price of a house based on the number of bedrooms and its global surface or predicting the braking distance of an airplane based on its speed, weight, and altitude (see Example 1.3.2).

> **Example 1.3.2.** *Toy Example: Braking distance estimation*
> Let $\mathbb{X} = [0,1]^3$ be the input space describing the speed, weight, and altitude normalized between 0 and 1. Let $\mathbb{Y} = \mathbb{R}^+$, the positive real value, be the output space representing the aircraft's braking distance. We assume a fixed but unknown distribution $\mathcal{D}$ on $\mathbb{X}\times\mathbb{Y}$. We have $S \sim \mathcal{D}^m$ the dataset comprises $m$ examples *i.i.d.* from $\mathcal{D}$. Finally, let $\mathcal{H}$ be the set of hypotheses gathering the real-valued functions mapping $\mathbb{X}$ to $\mathbb{Y}$ and $h \in \mathcal{H}$ be the optimized function predicting a braking distance.

Learning the hypothesis that fits the data would result in minimizing the average distance between the predicted and actual values. For example, in Figure 1.6, it corresponds to taking the difference between the expected values (y-axis of the red points) and the predicted one (projection of the red point onto the blue line as depicted by the black dashed lines).

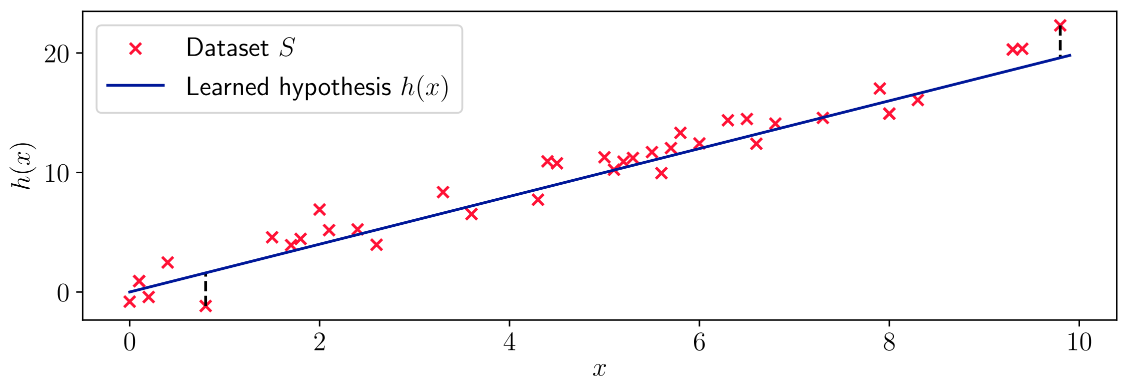

Figure 1.6: Crafted regression example. The red crosses are the dataset points, and the blue line represents the learned hypothesis. The dashed lines show examples of the deviation of the predicted value from the expected one.

A popular loss, known as *square loss*, is to take the square of the difference instead of only considering the difference:

$$\ell_{squared} = (y - h(x))^2 \tag{1.7}$$

There are several advantages in considering the square loss: *(i)* the errors does not compensate (*e.g.*, in Figure 1.6, the errors depicted by the dashed line do not cancel each other out), *(ii)* a large gap will be severely penalized while a smaller gap will be lightly penalized.

If the data contains outliers, the square loss might be over-sensitive, then another similar loss exists, the *absolute error*:

$$\ell_{absolute} = |y - h(x)| \tag{1.8}$$

Similarly to the square loss, the errors do not cancel each other out, and due to the absolute value instead of the square, the outliers will not be overvalued. Indeed, with the absolute loss, all errors are considered equally.

We explained how to compute the error for one example. By extending this principle to the whole dataset (*i.e.*, averaging the errors over the dataset), these two losses are known as the *Mean Square Error* (MSE) and the *Mean Absolute Error* (MAE).

### Neural Network

Among the numerous Machine Learning algorithms that exist (*e.g.*, SVM, Decision Trees, Boosting, or k-NN), neural networks deserve special attention because, in the literature on embedded critical ML-based systems, neural networks are the most used ML algorithm. Therefore all experiments of this thesis use neural networks (or neural networks alike).

Let us dive into the detail of a neural network. A neural network is composed of neurons that are structured into layers. A neuron is a mathematical function denoted as

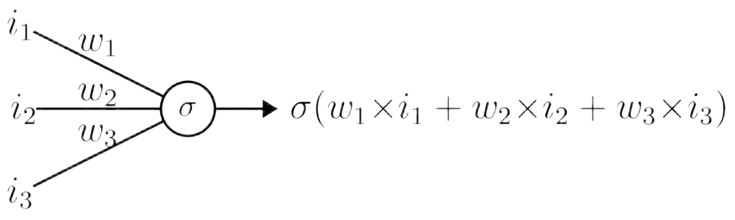

Figure 1.7: Explanation of a neuron internal working

the activation function; it takes inputs, applies the activation function, and returns an output (see Figure 1.7). A neuron's input is either the actual input of the neural network or the previous layer's output multiplied by the corresponding weights (see Figure 1.7).

We use different terms to denote the layers according to their place within the neural network: the first layer is the *input layer*, the last layer is the *output layer*, and the others are the *hidden layers* (see Figure 1.8).

The output of a neural network corresponds to what we denote until now $h(x)$. As described in Figure 1.8, a neural network has several layers. The depth of the network is defined by its number of layers. Each layer approximates a function $h^{(i)}$. Hence, the function $h$ is a composition of different functions. Then a neural network with $L$ layers could be described as

$$h(x) = h^{(L)} \circ h^{(L-1)} \circ \cdots \circ h^{(1)} \tag{1.9}$$

$$\text{where} \qquad h^{(i)}(x) = \sigma(W_i X_{i-1} + b_i) \quad \text{and} \quad i \in \{1; L\}. \tag{1.10}$$

where $W_i$ is a matrix, $X_i$ and $b_i$ are vectors, $\sigma$ is the activation function, and finally, the operator $\circ$ is the classical operator for the composition of function. For Figure 1.8 we get:

$$h(x) = h^{(3)} \circ h^{(2)} \circ h^{(1)}. \tag{1.11}$$

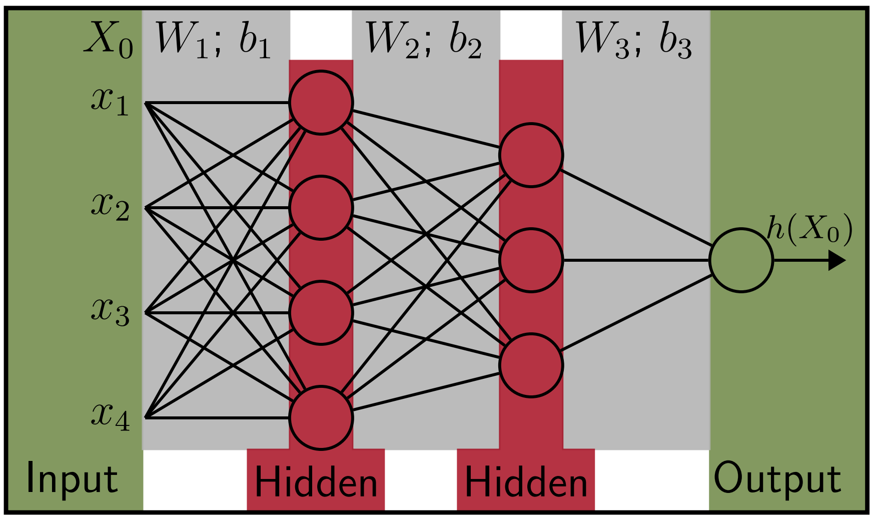

Figure 1.8: Neural Network example.

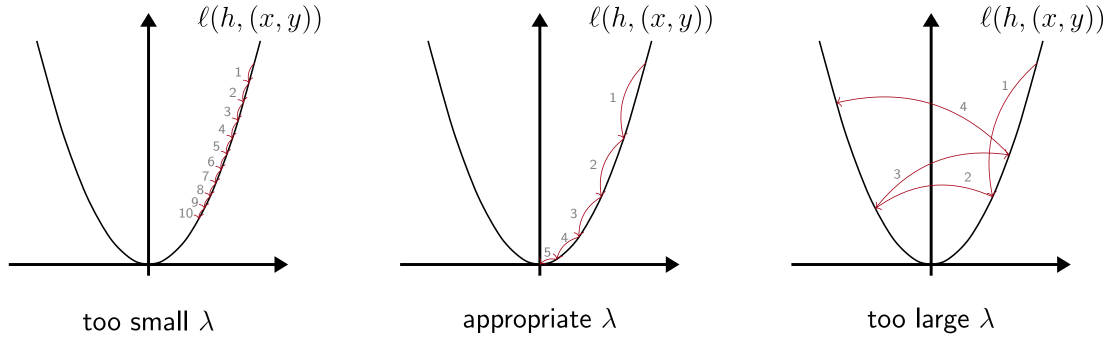

Figure 1.9: Illustration of the impact of the learning rate $\lambda$ on the convergence of the gradient-descent algorithm

This type of neural network is known as *feedforward neural network*. Other types of neural networks exist, but they are not studied in this thesis (*e.g.*, Recurrent Neural Network, Convolutional Neural Network, or AutoEncoder).

Learning a neural network means finding the weights and biases (see Figure 1.8) such that the neural network makes few or no errors, corresponding to applying the ERM principle. However, it is usually intractable to explicitly find the best values for the weights and biases that minimize the empirical risk. One way to find the minimum is to leverage the gradient of the function w.r.t to its parameters. For example, if we take the square function: $x^2$, due to its convexity, the minimum of the function occurs when its gradient is null. The derivative is $2x$, and by analytically studying $2x = 0$, we deduce that the minimum is reached when $x = 0$.

For neural networks, the analytical study is not possible because of the complexity of the function. Instead, the gradient descent method is used. It consists of iteratively computing the gradient of the loss w.r.t its weights and biases where at each step, the weights and biases are updated such that the loss becomes smaller by applying the following rule:

$$w^{(t)} = w^{(t-1)} - \lambda \nabla_w \ell(h,(x,y)) \tag{1.12}$$

where $(x,y) \in S$ and $\lambda \in \mathbb{R}^+$ is the parameter controlling the descent known as the *learning rate*. The choice of the value of $\lambda$ will determine the speed of convergence of the minimization. Figure 1.9 gives the intuition of the impact of the learning rate on the gradient descent method. A too-small value will end up with slow learning, *i.e.*, numerous iterations will be needed to converge to a minimum (left plot of Figure 1.9). Picking a too-large value may lead to divergence and hence fails the minimization (right plot of Figure 1.9). The learning rate is part of the hyper-parameters and can be tuned.

Moreover, a neural network is usually not convex. So then, theoretically we can only say that the gradient descent will find a local minimum; there is no way to ensure convergence towards the global minimum. Therefore, tuning the learning rate may have several advantages: (i) accelerating the learning time, (ii) finding better (local) minima resulting potentially in a better model.

## 1.4 Generalization

The generalization is closely linked to the overfitting and underfitting phenomena. Indeed, an algorithm generalizes well when it maintains its performance on unseen data, *i.e.*, the theoretical error, $\mathcal{R}_{\mathcal{D}}(h)$, is close to the empirical error $\mathcal{R}_S(h)$. In other words, if we know that an algorithm generalizes, it is ensured to perform well on unseen data. It boils down to having a small *generalization gap* defined as:

$$|\mathcal{R}_D(h) - \mathcal{R}_S(h)|. \tag{1.13}$$

The error (or generalization) bound guarantees the algorithm's ability to generalize. The idea is to bound the generalization gap (Equation (1.13)) using probabilistic bounds (Bartlett and Mendelson, 2003; Dziugaite and Roy, 2017; Germain *et al.*, 2015; Masegosa *et al.*, 2020a; McAllester, 1999; Parrado-Hernández *et al.*, 2012; Shawe-Taylor and Williamson, 1997; Valiant, 1984; Vapnik and Chervonenkis, 1971).

However, $\mathcal{R}_{\mathcal{D}}(h)$ is not computable since it is the error on all possible data. Hence the need for probabilistic bounds, which give an upper bound on the gap between $\mathcal{R}_{\mathcal{D}}(h)$ and $\mathcal{R}_S(h)$. The first generalization bounds appear with the Probably Approximately Correct (PAC) theory (Valiant, 1984) and have the following form:

$$\Pr_{S\sim\mathcal{D}^m}\left(|\mathcal{R}_D(h) - \mathcal{R}_S(h)| \leq \epsilon(\text{model's complexity}, \#S, \delta)\right) \geq 1-\delta, \tag{1.14}$$

where $\epsilon(\cdot) \geq 0$ and $\delta \in ]0,1]$. $\epsilon$ is a function modeling the upper bound that usually relies on the complexity of the model, the number of samples in $S$ (denoted by $\#S$), and the probability $\delta$. The ideal scenario is to have the gap between $\mathcal{R}_D(h)$ and $\mathcal{R}_S(h)$ small while having a high probability that the inequality holds, *i.e.*, having $\epsilon(\cdot)$ and $\delta$ as small as possible.

In this thesis we focus on the PAC-Bayesian theory (McAllester, 1999; Shawe-Taylor and Williamson, 1997) in a binary classification setting Section 1.3. Note that it is not restricted to binary classification and different setting, such as multi-class classification (see, *e.g.*, Masegosa *et al.*, 2020b; Zantedeschi *et al.*, 2021) or regression (see, *e.g.*, Germain *et al.*, 2016; Shalaeva *et al.*, 2020), has been studied. The PAC-Bayesian theory is two folded: *(i)* the PAC theory, which provides generalization guarantees, like in Equation (1.14), and *(ii)*, the Bayesian approach, which involves, in the learning algorithm, *prior* and *posterior* distributions over the family of classifiers[2]. The *prior* distribution denoted as $\mathcal{P}$ represents a priori knowledge of what a *good* classifier could be (before seeing the data). In contrast, the *posterior* distribution denoted as $\mathcal{Q}$ is obtained from the learning algorithm.

The PAC-Bayesian theory is well suited to handle *majority votes* based on a set of classifiers $\mathcal{H}$ (as defined earlier) where each classifier is a voter (see Figure 1.10).

Intuitively, in the PAC-Bayesian setting, the majority vote considered can be seen as the weighted sum of several models where the weights of the sum constitute a distribution $\mathcal{Q}$ learned from the training dataset $S$ and the prior distribution $\mathcal{P}$ (see Example 1.4.1).

[2]Since we consider only binary classification for the PAC-Bayesian theory, we may use "classifier" instead of hypothesis or model to add more semantics

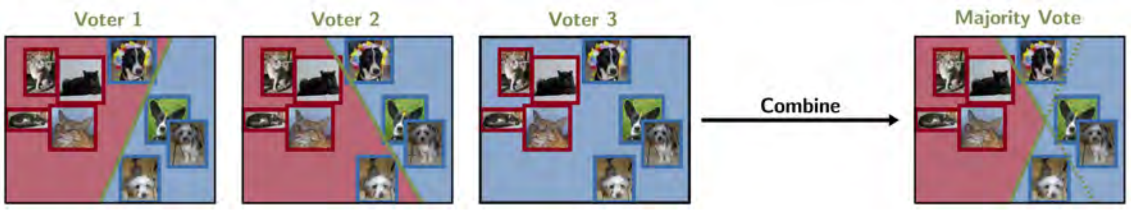

Figure 1.10: Example of a Majority Vote with three voters

The Q-weighted majority vote is also known as the *Bayes classifier* and is formally defined in Definition 1.4.1.

**Definition 1.4.1.** *Q-weighted majority vote a.k.a. Bayes Classifier*
For any hypotheses set $\mathcal{H}$ and any posterior distribution $\mathcal{Q}$ on $\mathcal{H}$, we have the Bayes classifier defined as

$$\mathbf{B}_{\mathcal{Q}}(x) = \operatorname{sign}\left[\mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x)\right] \tag{1.15}$$

where $\operatorname{sign}(\cdot) = 1$ when $\cdot > 0$, $\operatorname{sign}(\cdot) = -1$ when $\cdot < 0$ and $\operatorname{sign}(\cdot) = 1$ when $\cdot = 0$.

**Example 1.4.1.** *Toy example of the Q-weighted Majority Vote (based on Figure 1.10).*
We are in a binary classification setting: $S \sim \mathcal{D}^m$ is the training dataset drawn from the fixed but unknown distribution $\mathcal{D}$ composed of $m$ tuples $(x, y) \in \mathbb{X}\times\mathbb{Y}$. For this toy example, we restrict the set of hypothesis $\mathcal{H}$ to 3 linear classifiers $\{h_1, h_2, h_3\}$. We have a prior distribution $\mathcal{P}$ arbitrarily set to the uniform distribution over the set of hypotheses $\mathcal{H}$ and a posterior distribution $\mathcal{Q}$ over the set of hypotheses $\mathcal{H}$ learned based on $\mathcal{P}$ and $S$.

The Equation (1.16) shows the impact of the learning on the majority vote.

$$\overbrace{\operatorname{sign}\left[\frac{1}{3}h_1(x)+\frac{1}{3}h_2(x)+\frac{1}{3}h_3(x)\right]}^{\mathbf{B}_{\mathcal{P}}(x)} \xrightarrow[\text{with } S]{\text{Learning}} \overbrace{\operatorname{sign}\left[\frac{1}{4}h_1(x)+\frac{2}{4}h_2(x)+\frac{1}{4}h_3(x)\right]}^{\mathbf{B}_{\mathcal{Q}}(x)} \tag{1.16}$$

Before the learning, the majority vote follows the prior distribution $\mathcal{P}$, represented by the uniform distribution. After the learning, we end up with the posterior distribution $\mathcal{Q}$, which gives a more significant impact to the prediction of the classifier $h_2$ ($\frac{2}{4}$ versus $\frac{1}{4}$). In other words, the knowledge extracted from the training dataset $S$ constructs $\mathcal{Q}$ so that the classifier $h_2$ strongly impacts the output of the majority vote $\mathbf{B}_{\mathcal{Q}}$.

On the one hand, the Bayes Classifier is deterministic since it computes the expectation of $h(x)$ over the set of hypotheses $\mathcal{H}$ following the posterior distribution $\mathcal{Q}$. On the other hand, the Gibbs Classifier $\mathbf{G}_{\mathcal{Q}}$ is a stochastic classifier: to classify an example $x$, the Gibbs Classifier draws a classifier $h \in \mathcal{H}$ with respect to the posterior distribution $\mathcal{Q}$ and predicts the label $h(x)$.

The Gibbs classifier and the Bayes Classifier are closely related by their associated risks.

**Definition 1.4.2.** *Bayes Risk.*
For any distribution $\mathcal{Q}$ on the hypotheses set $\mathcal{H}$, the Bayes risk is the probability that the majority vote $\mathbf{B}_\mathcal{Q}$ fails to correctly classify the sample (x,y) drawn from $\mathcal{D}$.

$$\mathcal{R}_\mathcal{D}(\mathbf{B}_\mathcal{Q}) = \Pr_{(x,y)\sim\mathcal{D}} (\mathbf{B}_\mathcal{Q}(x) \neq y) \tag{1.17}$$

$$= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \ell_{\mathbf{01}}(\mathbf{B}_\mathcal{Q}, (x,y)) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathbf{I}(\mathbf{B}_\mathcal{Q}(x) \neq y) \tag{1.18}$$

$$= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathbf{I}( \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} y * h(x) \leq 0) \tag{1.19}$$

**Definition 1.4.3.** *Gibbs Risk*
For any distribution $\mathcal{Q}$ on the hypotheses set $\mathcal{H}$, the Gibbs risk is the probability that the (stochastic) Gibbs classifier $\mathbf{G}_\mathcal{Q}$ fails to correctly classify the sample (x,y) drawn from $\mathcal{D}$.

$$\mathcal{R}_\mathcal{D}(\mathbf{G}_\mathcal{Q}) = \Pr_{\substack{(x,y)\sim\mathcal{D}\\ h\sim\mathcal{Q}}} (h(x) \neq y) \tag{1.20}$$

$$= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \ell_{\mathbf{01}}(h, (x,y)) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathbf{I}(h(x) \neq y) \tag{1.21}$$

$$= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} \mathbf{I}(y * h(x) \leq 0) \tag{1.22}$$

As said in Section 1.3, the 0-1 loss is hardly optimizable and surrogate loss can be used instead. In the PAC-Bayesian setting, the linear loss $\ell_{linear}$ is of common use. Then, applying the linear loss to the Bayes Risk and Gibbs Risk, we have

$$\mathcal{R}_\mathcal{D}(\mathbf{B}_\mathcal{Q}) = \frac{1}{2}\left[1 - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \operatorname{sign}\left(\mathop{\mathbb{E}}_{h\sim\mathcal{Q}} y * h(x)\right)\right] \tag{1.23}$$

$$\mathcal{R}_\mathcal{D}(\mathbf{G}_\mathcal{Q}) = \frac{1}{2}\left[1 - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} y * h(x)\right] \tag{1.24}$$

The Bayes risk $\mathcal{R}_\mathcal{D}(\mathbf{B}_\mathcal{Q})$ is bounded by twice the Gibbs risk $\mathcal{R}_\mathcal{D}(\mathbf{G}_\mathcal{Q})$ (see Remark 6 and Proposition 10 of GERMAIN *et al.*, 2015, for the demonstration of Proposition 1.4.1), the Proposition 1.4.1 presents this result.

**Proposition 1.4.1.** *Bayes risk and Gibbs risk relation*
For any distribution $\mathcal{Q}$ on the set of hypotheses $\mathcal{H}$ and for any distribution $\mathcal{D}$ on $\mathbb{X}\times\mathbb{Y}$, we have

$$\mathcal{R}_\mathcal{D}(\mathbf{B}_\mathcal{Q}) \leq 2\mathcal{R}_\mathcal{D}(\mathbf{G}_\mathcal{Q}) \tag{1.25}$$

Since bounds are hard to derive directly for the Bayes risk (Definition 1.4.2), the PAC-Bayesian bounds are usually derived using the Gibbs risk (Definition 1.4.3) and the Proposition 1.4.1. This way, guarantees are provided for the Bayes risk, *i.e.*, the majority vote.

Without diving into the detail, we present in Theorem 1.4.1 a classic PAC-Bayesian generalization bound (Germain *et al.*, 2009; McAllester, 1999).

**Theorem 1.4.1.** *PAC-Bayesian Bound of (*Germain et al.*, 2009;* McAllester*, 1999)*
For any distribution $\mathcal{D}$ over $\mathbb{X}\times\mathbb{Y}$, for any hypotheses set $\mathcal{H}$, for any distribution $\mathcal{P}$ on $\mathcal{H}$, for any $\delta \in (0,1]$, we have

$$\Pr_{S\sim\mathcal{D}^m}\left(\begin{array}{l}\forall \text{ posteriors } \mathcal{Q} \text{ on } \mathcal{H},\\ \mathcal{R}_D(\mathbf{G}_\mathcal{Q}) \leq \mathcal{R}_S(\mathbf{G}_\mathcal{Q}) + \sqrt{\frac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) + \ln\frac{2\sqrt{m}}{\delta}\right]}\end{array}\right) \geq 1-\delta.$$

One can notice that the PAC bound (Equation (1.14)) and the PAC-Bayes bound (Theorem 1.4.1) have a similar structure. First, this PAC-Bayes bound is "pessimistic" because the bound is valid for all posterior distribution $\mathcal{Q}$ on $\mathcal{H}$, which means that the bound is valid at every moment of the learning. Thus,, this type of bounds (*i.e.*, pessimistic bounds) can be transformed into objective functions that can be optimized, and the resulting model should provide the best possible generalization bound.

One of the specificities of PAC-Bayes bounds compared to classical PAC bounds is the use of the Kullback-Leibler divergence (KL-divergence) between $\mathcal{Q}$ and $\mathcal{P}$ to quantify the complexity of the model.

**Definition 1.4.4.** *Kullback-Leibler divergence*

$$\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) = \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} \ln\frac{\mathcal{Q}(h)}{\mathcal{P}(h)} \tag{1.26}$$

where $\mathcal{Q}(h)$ and $\mathcal{P}(h)$ represent the probability of drawing $h$ either following the posterior or the prior distribution, respectively.

Intuitively, the KL-divergence measures the distance between the posterior distribution $\mathcal{Q}$ and the prior distribution $\mathcal{P}$; The KL-divergence, to a certain extent, controls how much we learn, and the penalization (*i.e.*, the value of the KL-divergence) grows with the learning; hence, it prevents the overfitting phenomenon.

## 1.5 Conclusion

In this chapter, we defined the general setting for ML models used in the thesis. We explained the Empirical Risk Minimization principle by introducing the notion of risks (theoretical and empirical). We saw through the cross-validation technique and the generalization aspects how to increase the confidence one can have in the performance of an ML model when deployed. We will see in Chapter 3 that the reliability of the ML algorithms is questioned when it comes to introducing it in critical embedded software. Indeed, the model might be fooled by only adding imperceptible noises to its input. Additional means will be necessary to consider the embeddability of ML. Formal verification that we will be presented in Chapter 6 is an excellent example of an additional means that should be included in the ML development pipeline.

# 2 Formal Verification

## Contents

**Abstract**

In this chapter, we start by explaining why Formal Verification is relevant for the aeronautical domain. Then, we define the basic notions needed for the ML verification domain. Finally, we detail the major techniques found in the literature that perform formal verification of ML algorithms. In particular, we overview the MILP method that we will put into practice in our contribution in the context of Chapter 6.

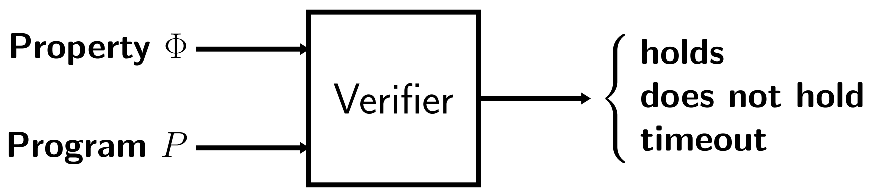

Figure 2.1: Verification principle

## 2.1 Introduction

Developing embedded critical software requires a rigorous process to comply with the authorities' regulations in the aeronautical field. A means of compliance is to follow the EUROCAE/RTCA standard ED-12x/DO-178x (see Section 3.3). Roughly speaking, the goal is to demonstrate that the software correctly performs as intended and is free of unintended behaviors that could challenge the safety requirements. In addition to the rigorous implementation process (see left part of the V-cycle in Figure 3.3), testing (right part of the V-cycle in Figure 3.3) is the part where the software is verified against its requirements. This activity requires the identification of all the possible scenarii (up to tens of thousands for highly critical systems), checking that the execution of the software on the target board is compliant and robust with all the specified requirements. Writing and executing these tests on specific test means is very time-consuming. Formal Methods play a significant role in helping to reduce the time spent on testing while providing mathematical proofs.

In the past few decades, the usefulness of formal methods has been proven through the rise of several industrial tools developed using formal methods (*e.g.*, Astrée, AbsInt or CompCert)[1]. Airbus is already using formal methods in its systems development (BRAHMI *et al.*, 2018; MOY *et al.*, 2013; SOUYRIS *et al.*, 2009). However, their use remains isolated to a few verification tasks (*e.g.*, the analysis of the Worst Case Execution Time using AbsInt.)

Finally, using formal methods to verify the conformity of embedded critical software to its specification could even improve its level of safety while reducing the testing time. Experience gained using formal methods in classical software makes them promising candidates for machine learning validation.

We will look into more details on machine learning verification in Chapter 6. In this current chapter, we present a broader view of the different formal verification techniques of the literature. This overview does not aim at being exhaustive, rather, it is made having in mind the techniques related to our approach.

## 2.2 Basic Definition

Formal verification in software engineering consists in checking whether a *program* complies with its *specification*. It implies that the properties are specified in a way the verifier understands.

[1]Here are the links to each of the tools mentioned: `https://www.astree.ens.fr/`, `https://www.absint.com/index.htm` and `https://compcert.org/publi.html`

Figure 2.1 gives a schematic view of formal verification: a *program* $P$ and a *property* $\Phi$ are the inputs of a verification method, called *verifier*, allowing to verify that the program $P$ preserves the property $\Phi$. The verifier outputs either *holds*, *does not hold* or *timeout*. However, according to the verifier's characteristics, the interpretation of these outputs may differ, as we will see below.

Let us give more detail on the three elements of Figure 2.1 by making the parallel with Example 2.2.1.

**Example 2.2.1.** *Aircraft's Braking Distance Estimation*
Let $P$ be a program aiming at estimating the braking distance of an aircraft based on its speed, weight, and the state of the runway (either dry or wet). Let $\Phi_i$ with $i \in \{1, 2, 3\}$ be the properties the program $P$ should satisfy.

- $\Phi_1$ When (the speed increases)$_1$ and (the weight and the runway's state remain unchanged)$_2$, then (the braking distance should increase)$_3$.
- $\Phi_2$ When (the weight increases)$_1$ and (the speed and the runway's state remain unchanged)$_2$, then (the braking distance should increase)$_3$.
- $\Phi_3$ When (the runway's state goes from dry to wet)$_1$ and (the speed and the weight remain unchanged)$_2$, then (the braking distance should increase)$_3$.

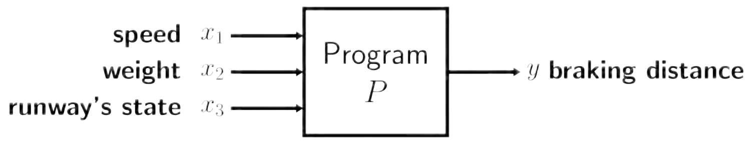

Figure 2.2: Schematic view of the program $P$

**Program.** The code implemented to meet the need. For example, in Example 2.2.1, the need is to estimate the braking distance, and the program $P$ is implemented with this purpose. In the aeronautical context, the program is developed following a standard such as the DO-178x.

**Property.** A property is, in general, a formula expressed in some logic (*e.g.*, predicate logic, modal or temporal). The choice of the logic to be used depends on the targeted verifier. A particular form of property can be stated as precondition(s) and postcondition(s). For example, in $\Phi_1$, we check that the braking distance increases (postcondition) if the speed increases AND the weight and the runway's state do not change (precondition). Example 2.2.2 shows what a formalization of this particular form of property looks like.

**Example 2.2.2.** *Example of pre/post condition property using $\Phi_1$ of Example 2.2.1* Let $X = \{x_1, x_2, x_3\}$ and $X' = \{x'_1, x'_2, x'_3\}$ be two sets of inputs that the program $P$ may encounter. $y$ and $y'$ are the outputs of $P$ when $X$ and $X'$ are inputs, respectively. The property $\Phi_1$ is formalized as follows:

$$\underbrace{x_1 < x'_1 \quad \wedge \quad x_2 = x'_2 \quad \wedge \quad x_3 = x'_3}_{\text{precondition}} \implies \overbrace{y < y'}^{\text{postcondition}} \tag{2.1}$$

We find back the three elements of $\Phi_1$ in Equation (2.1):

- (the speed increases)$_1$: $x_1 < x'_1$
- (the weight and the runway's state remain unchanged)$_2$: $x_2 = x'_2 \quad \wedge \quad x_3 = x'_3$
- (the braking distance should increase)$_3$: $y < y'$

With this form of property, we have the postcondition, which must be satisfied only when the precondition is also satisfied (implication operator).

**Verifier.** This term encompasses all the formal verification techniques that may be used to verify a property of a program. Although many paradigms exist, we focus on linear programming in this thesis. The reader may refer to Monin and Hinchey (2003) to get a deeper knowledge of formal methods. Dealing with critical systems requires the answer given by a verifier to be reliable. We find this reliability with *sound* verifiers. Intuitively, *soundness* means that when the verifier says a property is verified, it is always true. Another essential criterion for verifiers is the *completeness*. Intuitively, *completeness* means that if a property is satisfied, the *complete* verifier will eventually output that the property holds. Then, when using verifier leveraging approximation (*e.g.*, abstract interpretation or linear relaxation), we end up with *sound* but *incomplete* verifiers, which may trigger false alarms because, with these two characteristics, the verifiers will be conservative, *i.e.*, it may say that a property does not hold when it actually does. Concerning the aeronautical domain, it is not desirable to get too many false alarms, otherwise, the verification is useless. Hence incomplete methods should be chosen carefully regarding the approximation that is performed.

Figure 2.3 summarizes the *soundness* and *completeness* notions. Since this thesis takes place in an aeronautical context, all the verifiers presented are at least *sound*. In Section 2.3, we dive into the details of complete and incomplete verifiers and highlight the advantages and drawbacks of both types of techniques.

## 2.3 Complete and Incomplete Verifiers

For verifying critical embedded systems, the *soundness* of the verifiers is necessary. However, though it might be desirable, the *completeness* is not essential. This section describes complete (Satisfiability Modulo Theory Solver and Mixed Integer Linear

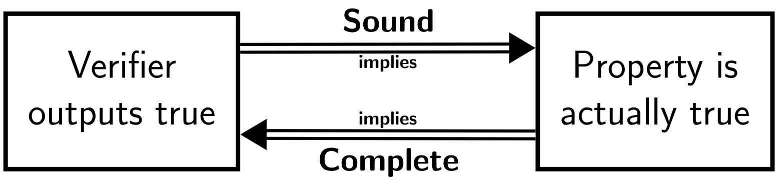

Figure 2.3: Graphic explanation of *soundness* and *completeness*. Whenever the verifier outputs "true", then the property is true: the verifier is sound. Whenever a property is "true", then the verifier outputs true: the verifier is complete.

Programming) and incomplete (Abstract Interpretation) verifiers and highlights their advantages and drawbacks.

## Satisfiability Modulo Theory (SMT)

Satisfiability Modulo Theory is a generalization of the SAT problem. The SAT problem consists in finding whether a *boolean formula* is *satisfiable*; this problem was the first problem proved to be NP-Complete (COOK, 1971).

**Formula** SAT problems are expressed in propositional logic. Hence, a formula is composed of literals, *i.e.*, boolean variables and boolean operators such that $\wedge$ (and — "conjonction"), $\vee$ (or — disjunction) and $\neg$ (not — negation). A valid formula respects the following grammar:

$$\begin{array}{rll}
\langle literal \rangle & ::= \text{"any boolean variable"} & \\
\langle formula \rangle & ::= \langle literal \rangle & \text{r1} \\
 & \mid \langle formula \rangle \wedge \langle formula \rangle & \text{r2} \\
 & \mid \langle formula \rangle \vee \langle formula \rangle & \text{r3} \\
 & \mid \neg \langle formula \rangle & \text{r4} \\
 & \mid (\langle formula \rangle) & \text{r5}
\end{array}$$

These few rules are sufficient to create any formula of propositional logic. $\langle literal \rangle$ represents a boolean variable that belongs to the set of variables of the problem.

**Satisfiability** A formula $f$ is satisfiable if there exists an *assignment* such that the formula is true. An assignment $A$ — or interpretation — consists in giving a boolean value (true or false) to each variable in the formula. If a formula is true with a given assignment $A$, we say that $A$ is a model of $f$ denoted as $A \models f$ otherwise, $A$ is not a model of $f$ denoted as $A \not\models f$. In the first case, we say that $f$ is satisfiable (SAT). A formula $f$ is unsatisfiable (UNSAT) if there exists no assignment which is a model of $f$.

**DPLL and CNF** The Davis-Putnam-Logemann-Loveland (DPLL) algorithm (DAVIS *et al.*, 1962), or variants of it, are widely used to check the satisfiability of formulas. However, the DPLL algorithm does not take any arbitrary form of formula as input; it only takes Conjunctive Normal Form (CNF) formulas. Fortunately, every formula can be transformed into a CNF using the tseytin's transformation (Tseitin, 1983). We do not dive into the detail of the algorithm, but the reader may refer to Biere *et al.* (2021). A Conjunctive Normal Form is a formula composed as a conjunction ($\wedge$) of clauses — disjunction ($\vee$) of literals. A CNF formula observes the following grammar:

$$\begin{aligned}\langle clause\rangle &::= \langle literal\rangle | \neg\langle literal\rangle \\ &\quad | \langle clause\rangle \vee \langle clause\rangle \\ \langle CNF\rangle &::= (\langle clause\rangle) \\ &\quad | \langle CNF\rangle \wedge \langle CNF\rangle\end{aligned}$$

Contrary to the SAT problems, SMT problems are not expressed with propositional logic but with First Order Logic (FOL). More precisely, an SMT problem boils down to using FOL where the *predicate* symbols (define hereafter) and variables are instantiated according to a specific theory (*e.g.*, the theory of Linear Real Arithmetic dealing real number ($\mathbb{R}$) and the operators set $\{+, -, <, >, =\}$ as predicates)[2] (see Biere *et al.*, 2021, for more details).

**Predicate symbol** A predicate symbol has an associated arity ($\geq 0$), *i.e.*, a number indicating the number of arguments of the predicate, and represents a relation between variables.

**Example 2.3.1.** *Predicate example*

$$x \underbrace{<}_{\text{Predicate}} y \tag{2.2}$$

The predicate symbol is $<$ and, as in linear arithmetic, means that $x$ is lower than $y$. In the case of SMT, this predicate will keep the same interpretation as in linear arithmetic theory.

Example 2.3.1 shows an example of a predicate from the linear arithmetic theory. Then, by considering all the real numbers and the symbols to express Linear Real Arithmetic as predicates, we obtain a formula in first-order logic that depicts the linear real arithmetic theory.

Note that FOL can be seen as a generalization of the propositional logic since, with a particular setting, we can do propositional logic with FOL. In the general setting of FOL, there is also *quantifier*. However, we do not detail this term because many SMT solvers do not support quantifiers, which will not be relevant for the rest of the thesis.

In short, SMT solvers are any methods that deal with the search for satisfiable models for SMT problems (*e.g.*, DPLL(T) (Ganzinger *et al.*, 2004), which is an extension of DPLL). The solvers are sound, and according to the theory applied, they can be complete or incomplete.

[2]We gave an example of linear real arithmetic, but other theories can be applied, such as "arrays" or "strings" theory.

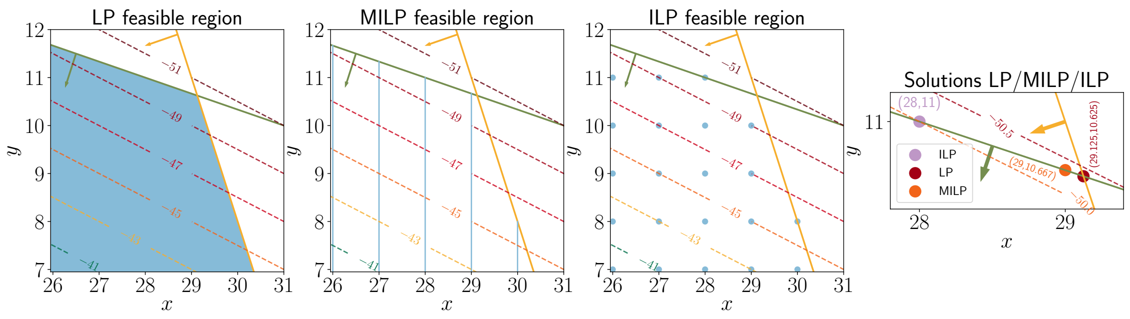

Figure 2.4: Feasible region of the optimization problem in Example 2.3.2. The feasible regions are in blue. The most-right plot shows the solution for the ILP, MILP, and LP problems. The dashed lines represent the possible values of the objective function w.r.t $x$ and $y$.

## Mixed Integer Linear Programming (MILP)

A Mixed Integer Linear Programming (MILP) problem is an *optimization problem* where the *objective function* and the *constraints* are linear and some of the variables are restricted to integers. A MILP problem is formalized as follows:

$$\begin{aligned} \textbf{minimize} \quad & f(x,y) = c^T x + d^T y && \text{objective function} \\ \textbf{subject to} \quad & Ax \leq b,\ x + y \leq g && \text{constraints} \\ & x \geq 0, y \geq 0 \\ & x \in \mathbb{Z}^n,\ y \in \mathbb{R}^n && \text{variables and their type} \end{aligned}$$

where $x$, $y$, $b$, $c$, $d$ and $g$ are vectors and $A$ is a matrix. We can textually express this MILP problem as *a minimization of the objective function* $f$ *with respect to the constraints and the ranges of the variables*. Within this section we will consider only minimization problem without loss of generality because a maximization problem can be transformed as a minimization problem: **maximizing** $f(x)$ is equivalent to **minimizing** $-f(x)$. Note that MILP is a particular setting where integers and real values are involved in the same problem. An optimization problem with only integers is an Integer Linear Programming problem (ILP), while one with only real numbers is a Linear Programming problem (LP). Transforming a MILP problem into an LP problem by removing the integrality constraint and considering all variables as reals is called *linear relaxation*.

An assignment to the variables is *feasible* if it satisfies all the constraints. Compared to its linear relaxation, the feasible region of a MILP problem is significantly reduced since integers are involved. However, the feasible region is still larger than in an ILP problem. We crafted a small optimization problem in Example 2.3.2 and adapted the types of the variables to get an LP, an ILP, and a MILP problem. Then, with the Figure 2.4, we graphically highlight the difference between their feasible regions and show the impact on the solution.

**Example 2.3.2.** Crafted toy example. Feasible Solution: LP vs ILP vs MILP

$$\begin{aligned} \textbf{minimize} \quad & -x - 2y \\ \textbf{subject to} \quad & x + 3y \leq 61, \; 3x + y \leq 98 \\ & x \geq 0, y \geq 0 \\ & x \in \mathbb{Z}, \; y \in \mathbb{Z} && \text{ILP} \\ & x \in \mathbb{Z}, \; y \in \mathbb{R} && \text{MILP} \\ & x \in \mathbb{R}, \; y \in \mathbb{R} && \text{LP} \end{aligned}$$

We find here the same structure of the optimization problem presented earlier. For this example purpose, we instantiate an ILP problem (both variables are integers), a MILP problem ($x$ is an integer and $y$ is a real), and an LP problem (both variables are reals). The colors of the lines in Figure 2.4 match the constraints' colors.

**LP problem** Every values (w.r.t the constraints) are possible; the feasible region is the blue area of the most-left plot of Figure 2.4.

**MILP problem** The feasible region is reduced to lines as illustrated in the second most-left plot of Figure 2.4.

**ILP problem** The feasible region is reduced to dots that correspond to the cartesian products between the two variables restricted by the constraints (third plot from the left of Figure 2.4).

We can graphically notice (see Figure 2.4) the significant drop in the feasible region size from an LP problem to an ILP problem for the same objective function and constraints. The most-right plot of Figure 2.4 shows the solution for each problem. In this example, each problem has a unique and distinct solution: $(29.125, 10.625)$, $(28, 11)$ and $(29, 10.667)$ for the LP, ILP and MILP problems respectively.

Contrary to LP problems, MILP and ILP problems are known to be NP-Hard (Martello and Toth, 1990). We do not dive into the details of complexity be the reader may refer to Sipser (2013).

For solving MILP problems, the "branch-and-cut" algorithm, which is sound and complete, is widely used. First, a linear relaxation is made to leverage the efficiency of LP solvers, *e.g.*, Simplex or Barrier algorithms (see Boyd and Vandenberghe (2004) for more detail). Then, the search for the optimal solution combines a branch-and-bound algorithm with the cutting plane algorithm that improves the sub-problems handle at each node of the branch-and-bound. The reader may refer to Conforti *et al.* (2014) and Pisaruk (2019) for more advanced details on each algorithm.

In Chapter 6, we leverage the MILP solver named Gurobi (Gurobi Optimization, LLC, 2022) to perform the verification of a neural network's property.

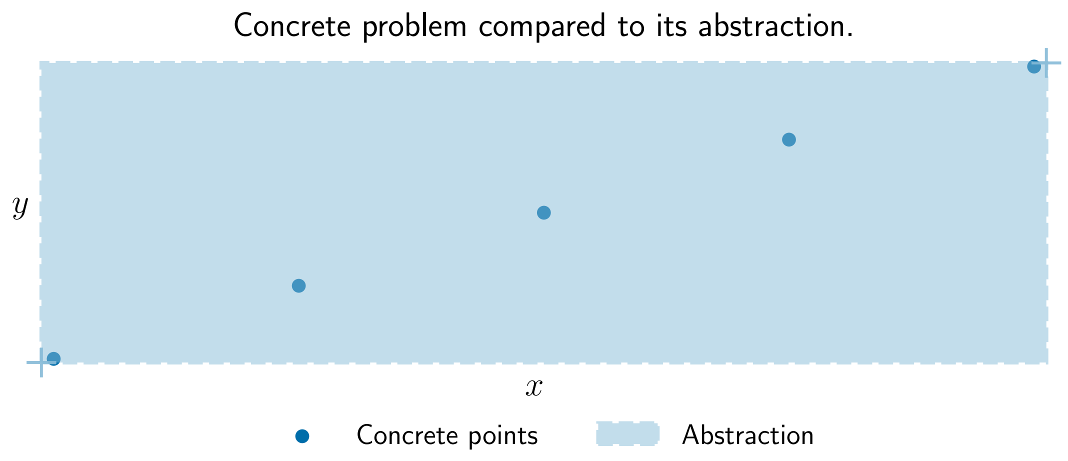

Figure 2.5: The values of the concrete problem are represented in dark blue: `concrete_domain` $= \{$blue points $\bullet\}$. A possible abstraction is the blue surface describing intervals on $x$ and $y$: `abstract_domain` $= [x_{min}, x_{max}]$ and $[y_{min}, y_{max}]$.

## Abstract Interpretation

Abstract Interpretation is a vast theory, hence, we focus here only on the basics principles that allow understanding the verification of ML algorithms based on abstract interpretation (see Chapter 4). To go further, the reader may refer to Cousot (2021), Cousot and Cousot (2010), and Miné (2017).

Abstract Interpretation is a formal verification theory which relies on the *abstraction* of the problem to make its verification more tractable: the original problem is referred to as the *concrete* problem while its abstraction as the *abstract* problem which simplifies the representation of the concrete one (see Figure 2.5).

The abstraction is made to be sound: Any property that turns out to be true for the *abstract* problem will always be true for the *concrete* problem. However, due to the over-approximation (*i.e.*, the abstraction), we lose the completeness aspect. Thus, abstract interpretation may raise false alarms: the property is actually true, but the verifier can not prove it and hence outputs that the property does not hold.

Consequently, Abstract Interpretation can be used for verifying critical systems. However, one needs to pay attention to the abstraction used so that it will not raise too many false alarms. Static analysis by abstract interpretation is already used on critical avionics software to verify the proper behavior of the latter (Moy *et al.*, 2013; Souyris *et al.*, 2009).

Basically, to perform abstract interpretation one needs to define *abstract domain* and *abstract operation*. We will explain both terms through the use of the following toy example:

**Example 2.3.3.** Toy program[3]

```
Require: $x \in \{15, 16, 17\}$
1: $y = 7$
2: while $x \geq 0$ do
3:     $x = x - 2$
4:     $y = y + 252$
5: end while
```

The algorithm takes as input $x$, which may take as input 15, 16, or 17 and perform an operation on $y$ regarding the value of $x$. An example of execution with $x = 15$:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $x$ | 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 | -1 |
| $y$ | 7 | 259 | 511 | 763 | 1015 | 1267 | 1519 | 1771 | 2023 |

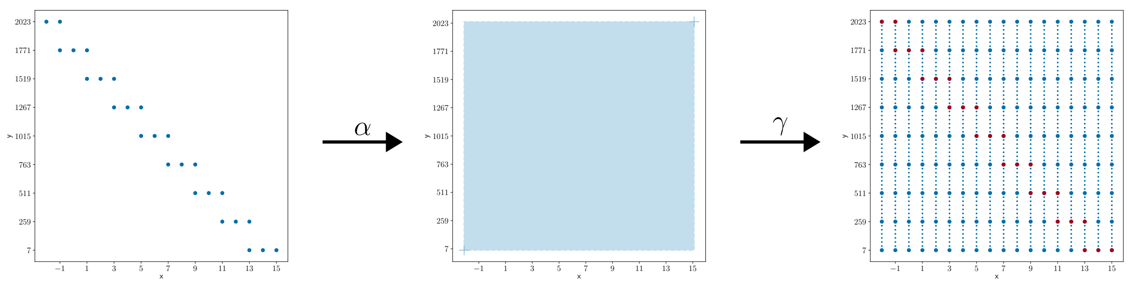

Figure 2.6: Example of an abstraction and a concretization based on Example 2.3.3

**abstract domain** Let $\mathbb{D}$ denotes the concrete domain and $\mathbb{D}^{\#}$ the abstract domain. On the one hand, $\alpha$ is the function that transforms any element from $\mathbb{D}$ into an element from $\mathbb{D}^{\#}$ (*i.e.*, $\alpha : \mathbb{D} \mapsto \mathbb{D}^{\#}$). On the other hand, $\gamma$ is the function that transforms any element from $\mathbb{D}^{\#}$ into an element from $\mathbb{D}$ (*i.e.*, $\gamma : \mathbb{D}^{\#} \mapsto \mathbb{D}$). Figure 2.6 illustrates the concrete domain of Example 2.3.3 (the left plot), its abstraction (the middle plot) and the concretization of the abstraction (the right plot). The concrete domain, for this example, is the powerset of the integers, then all the possible execution of the program is an element of this concrete domain. We can see that the value of $x$ decreases while $y$ increases. For the example, we abstract this concrete element using abstract interval domain by retrieving the min and max of both variable: $x_{min} = -2$, $x_{max} = 17$ and $y_{min} = 7$, $y_{max} = 2023$. The concretization of the abstraction corresponds to the largest possible concrete element such that applying an abstraction again does not change the previous abstraction: $\{(x, y) | (x, y) \in [-2, 15] \times [7, 2023]\}$.

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[dr, dash] & \\
\leq 0 \arrow[dr, dash] & & \geq 0 \arrow[dl, dash] \\
 & 0 \arrow[d, dash] & \\
 & \bot &
\end{tikzcd}$$

(a) Lattice of the sign domain

| $+$ | $\top$ | $\leq 0$ | $\geq 0$ | 0 | $\bot$ |
|---|---|---|---|---|---|
| $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\bot$ |
| $\leq 0$ | $\top$ | $\leq 0$ | $\top$ | $\leq 0$ | $\bot$ |
| $\geq 0$ | $\top$ | $\top$ | $\geq 0$ | $\geq 0$ | $\bot$ |
| $0$ | $\top$ | $\leq 0$ | $\geq 0$ | $0$ | $\bot$ |
| $\bot$ | $\bot$ | $\bot$ | $\bot$ | $\bot$ | $\bot$ |

(b) Binary operation "+" on the abstract sign domain

Figure 2.7: abstract sign domain and an example of abstract operation.

**abstract operation** Abstract operation is used to manipulate abstract elements. An abstract operation has an arity — the number of parameters the operation takes as input. However, one must pay attention that the abstract operation applied to the abstract element are still sound. In other words, the abstract operation on abstract elements must "include" the abstraction of the concrete operation of the concrete elements. For a binary operation, we have:

$$\texttt{binary}^{\#}(x, y) = \alpha(\texttt{binary}(\gamma(x), \gamma(y))).$$

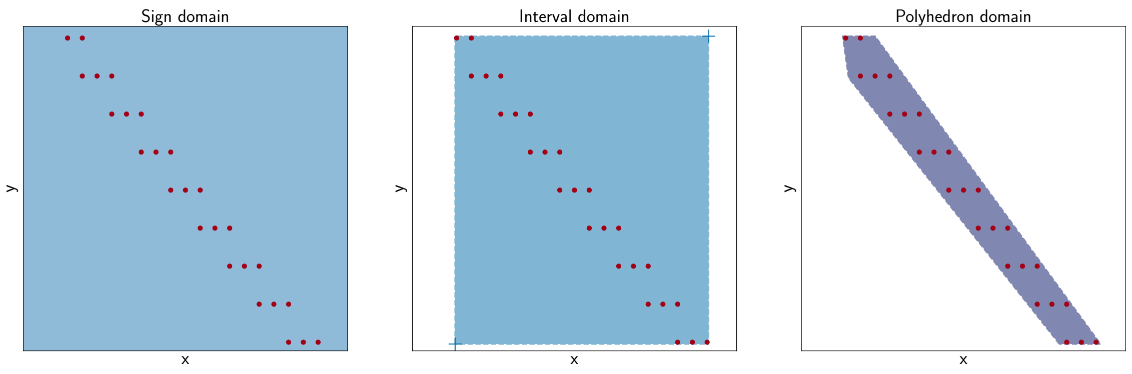

Figure 2.8: Example of different abstract domains applied on Example 2.3.3

Example 2.3.4 shows a simple example with the sign domain.

**Example 2.3.4.** The sign domain can be represented with the lattice of Figure 2.7a: $\top$ is the greatest abstract element while $\perp$ is the smallest. Then we have the negative element ($\leq 0$), the positive element ($\geq 0$), and zero ($0$). The concretizations of those abstract elements are the following:

$$\begin{aligned}\gamma(\top) &= \mathbb{Z}\\ \gamma(\leq 0) &= \{x \mid x \leq 0\}\\ \gamma(\geq 0) &= \{x \mid x \geq 0\}\\ \gamma(0) &= \{0\}\\ \gamma(\perp) &= \emptyset\end{aligned}$$

The results of the binary operation $+$ on the abstract element are stated in the Figure 2.7b. Here an example of how to read the table: $\leq 0 + \leq 0 = \leq 0$. Then, the "+" operation is defined between all abstracts element, and finally, analyses can be performed using this operation.

Other types of abstraction exist, as shown in Figure 2.8. The left plot illustrates the abstract sign domain, the middle plot the interval domain, and the right plot the polyhedron domain. The different abstract domains come with their advantages and drawbacks. On the one hand, the sign domain is easy to handle but is not precise. On the other hand, the polyhedron domain is precise but it is a more complex task to do the abstraction.

## 2.4 Conclusion

In this chapter, we first present the basic terms used in the formal verification field. Then, we explained two essential criteria that we must pay attention to in the aeronautical domains: soundness and completeness. The soundness of the verifiers is required to ensure reliable verification, but completeness is not. We finally present complete and incomplete formal techniques highlighting the influence of the property regarding the aeronautical domain. Among the presented techniques, we leverage a MILP solver in Chapter 6 to formally verify a monotony property of a neural network.

# Part II

# Certification of Machine Learning

# 3 Certification meets Machine Learning

**This chapter is based on the following paper**

Guillaume Vidot, Christophe Gabreau, Ileana Ober, and Iulian Ober,
arXiv, https://arxiv.org/abs/2106.07221,
Certification of embedded systems based on Machine Learning: A survey

## Contents

**Abstract**

In this chapter, the goal is to reduce the scope of our work. To do so, we start with the current certification approach, then identify the gaps raised by using ML-based systems and finally elaborate on the different challenges to fill these gaps. This preliminary work allows choosing the relevant direction to progress toward the certification of safety-critical systems using ML.

## 3.1 Introduction

The demonstration of conformity to the regulation constraints (*i.e.*, the certification) of critical embedded systems is a well-established process defined during the last few decades, especially through creating and improving industrial standards that comply with the regulatory text. The change of paradigm raised by the use of ML is questioning the current certification approach. Thus, this chapter is dedicated to identifying the gaps that invalidate this approach. We describe the current demonstration of conformity by giving the big picture in Section 3.2 and by focusing on the software standard DO-178x in Section 3.3. Then we discuss the impact that ML has on the certification approach (Section 3.4) and elaborate on the challenges deduced from the identified gaps (Section 3.5). We end this chapter with the existing standardization initiative on ML development and certification by both authorities and industry (Section 3.6).

## 3.2 Current Certification Approach

The aeronautical domain comprises the production of different aircraft types, such as planes, helicopters, or drones. Even if this thesis focuses on the software aspect of the certification of embedded systems for large airplanes (because of the industrial environment in which this thesis took place), most of its content applies to all aircraft types. Nowadays, a large airplane cockpit offers many complex avionic functions: flight controls, navigation, surveillance, communications, displays, etc. Hence, software plays a significant role in aircraft safety. Since the aircraft can carry up to several hundred passengers (*e.g.*, airplanes) and their safety relies on the detection of systems failures where software faults can possibly contribute, a specific development process is necessary to ensure the correctness of the embedded software. The regulatory authorities are in charge of setting up the rules that the aircraft's manufacturer and suppliers must comply with to deliver any aircraft.

The design of these complex functions has required a top-down iterative approach from the aircraft level downward. Thus the functions are performed by systems of systems, with each system decomposed into subsystems (or equipment) that may contain a collection of software and hardware items. Therefore, any avionic development considers 3 levels of engineering: *(i)* Function, *(ii)* System/Subsystem and *(iii)* Item.

Figure 3.1 summarizes the stakeholders, the regulatory material, and the standards involved in the demonstration of conformity. The European Union Aviation Safety Agency (EASA), *i.e.*, the regulatory authorities of the European Union (top of the pyramids in Figure 3.1), provides regulatory materials that define and explain all the requirements due for developing safe avionic products EASA, 2021a. Regarding software/hardware items embedded in avionic safety-related systems, the regulatory requirements are described in *Certification Specification* documents (CS 2x.1301 and CS 2x.1309) which apply to a large range of aircraft (planes, helicopters, and even some drones categories). Since we focus on large airplanes, we will refer to the CS-25 (EASA, 2021a), which is the regulatory text for large airplane. CS-25.1301 states that the system must perform exactly as specified in the requirements (intended function), and CS-25.1309 states

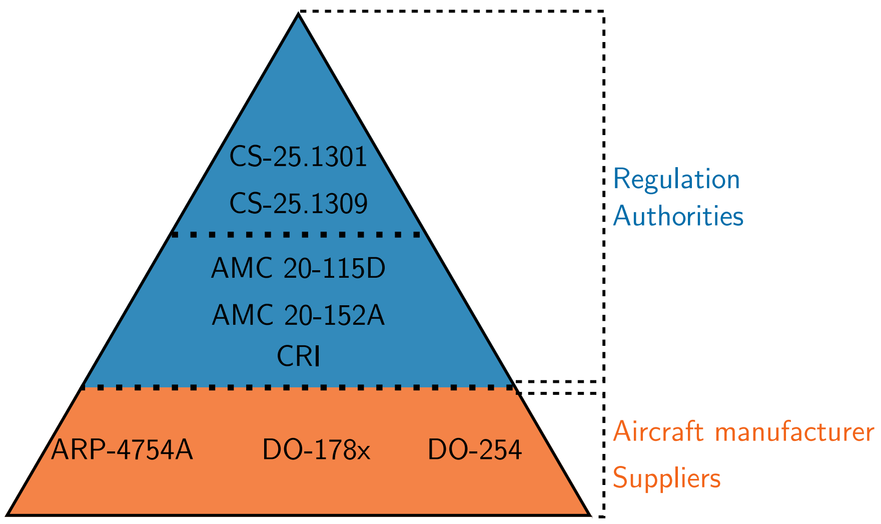

Figure 3.1: Overview of the stakeholders and the regulatory material involved in the demonstration of conformity.

that the system must safely perform the function to ensure the passengers' safety. We could summarize these paragraphs as follows: *avionic systems should safely perform their intended function under all foreseeable operating and environmental conditions.*

Furthermore, the different levels of rigor required (Design Assurance Level — DAL) within the development process depends on the failure conditions the improper functioning of the system may contribute to.

## Design Assurance Level (DAL)

The regulation text CS-25 EASA, 2021a classifies the failure conditions into five level of severity which matches with five levels of DAL:

- *No safety effect* — The failure conditions have no impact on the airplane safety: DAL E.
- *Minor* — The failure conditions slightly impact the airplane safety: DAL D
- *Major* — The failure conditions significantly impact the safety of the airplane and may possibly cause injuries: DAL C
- *Hazardous* — The failure conditions severely impact the safety of the airplane and would cause serious or fatal injury to a small number of the passenger: DAL B
- *Catastrophic* — The failure conditions would cause a significant number of death and the loss of the airplane: DAL A

Then, to comply with the regulation text (EASA, 2021a), the occurrence of each category of failure condition must be stated. The qualitative and quantitative probability terms are defined by the regulatory authorities as follows:

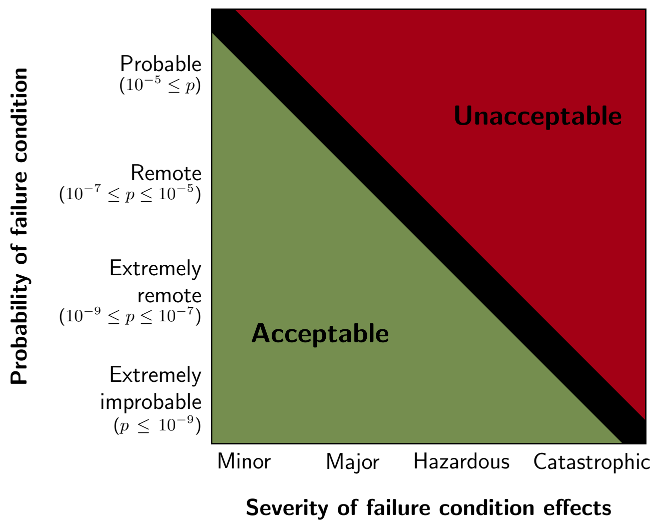
Probability of failure condition
Probable ($10^{-5} \leq p$)
Remote ($10^{-7} \leq p \leq 10^{-5}$)
Extremely remote ($10^{-9} \leq p \leq 10^{-7}$)
Extremely improbable ($p \leq 10^{-9}$)
Unacceptable
Acceptable
Minor
Major
Hazardous
Catastrophic
Severity of failure condition effects

Figure 3.2: Relationship between Probability and Severity of Failure Condition Effects (adapted from EASA, 2021a, Figure 1 AMC 20.1309, p.779)

- *Probable* - The failure conditions are expected to occur at least one time during the entire life of each airplane. The average probability per flight hour is greater than $10^{-5}$.
- *Remote* - The failure condition that should unlikely occur during the entire life of each airplane but may occur several times when considering the total life of a number of airplanes. The average probability per flight hour is between $10^{-7}$ and $10^{-5}$.
- *Extremely remote* - The failure conditions are not expected to occur during the entire life of each airplane but may occur a few times when considering the total life of all airplanes. The average probability per flight hour is between $10^{-9}$ and $10^{-7}$.
- *Extremely improbable* - The failure conditions are so unlikely to occur that they are not expected to occur during the entire life of all airplanes. The average probability per flight hour is less than $10^{-9}$.

Figure 3.2 shows the relationship between the severity category and the probability. Intuitively, the more severe a failure condition is, the less probable the occurrence should be. Note that no probability is required for the failure conditions with *no safety effect*. A *minor* failure condition may be *probable*; a *major* failure condition must be at most *remote*; an *hazardous* failure condition must be at most *extremely remote*; a *catastrophic* failure condition must be at most *extremely improbable*.

The development process of each engineering level relies on several decades of experience and good practices that keep on being adapted today. These methods have been standardized through EUROCAE/SAE standards for system development (ED-79A/ARP4754A), EUROCAE/RTCA standards for software items (ED-12C/DO-178C and supplements), and hardware items (ED-80/DO-254). Through supplemental documents known as *Acceptable Means of Compliance* (AMC) and, more precisely, the

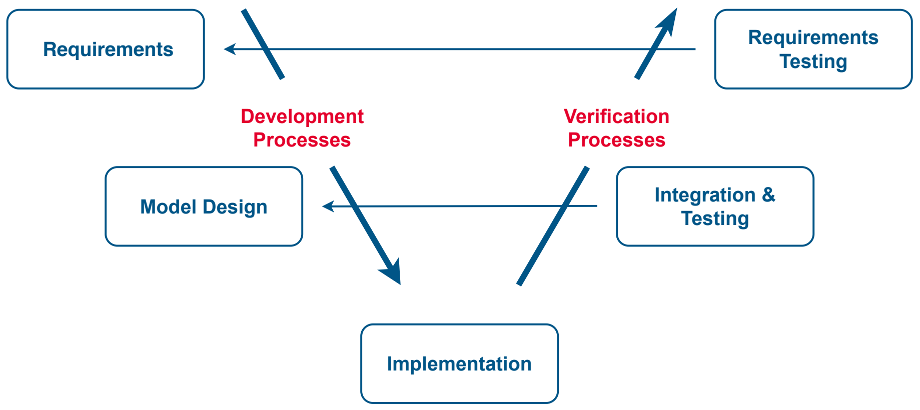

Figure 3.3: Item development workflow

AMC20-115D and AMC20-152A, EASA recognizes the current avionic standards as acceptable means of compliance to the regulation specifications (see Section 3.3). Hence, applying those standards when developing aircraft systems ensures compliance with the regulation specifications.

Whenever the standards do not cover new technology, a discussion is open between the regulatory authorities and the aircraft manufacturer to determine the requirements for embedding the new technology. The outcomes of the discussions are stated in a document named *Certification Review Item* (CRI), which is considered as a supplemental guidance to applicable standards. The CRI previously written may be included within the standard when a new issue is released. For example, there are already three issues of the DO-178 where CRIs have been progressively included.

## 3.3 Software Standards: DO-178x

Contrary to the hardware assessment, no metric exists to measure the software quality. Then, the development process's quality directly impacts the software's quality. The standard DO-178x can be defined as a set of software considerations in airborne systems. It defines requirements in terms of objectives (*e.g.*, Source Code complies with low-level requirements), activities (*e.g.*, code review), and evidence (*e.g.*, document summarizing the code review results). As for the DAL applied to the system, five levels —from A (highest safety required) to E (lowest safety required) — of the development process are defined for the DO-178x. Note that the system's DAL might differ from the standard's development level because of the architecture and the mitigation applied within the system. Hence the development level of the DO-178x of the software is specified according to the system architecture.

### Development Workflow

Concerning the DO-178C standard, the complete standardized workflow comprises the planning process, four development processes (requirement, design, coding, integration), and four integral processes (verification, configuration management, quality assurance, and certification liaison). In this thesis, only the development and verification processes will be considered in the contributions. The item's development workflow is usually represented by the V-cycle (schematically represented in Figure 3.3). Today, the usual development paradigm is the Requirement-Based Engineering — RBE (either textual or model-based): it corresponds to the activities link to the top left square of Figure 3.3. This step might be referred to as the definition of the High-Level requirements (HLRs). Then, we have the model design phase: it encompasses the software architecture and the definition of the Low-Level Requirements (LLRs). LLRs are the last level of requirements before the code; they are closer to the implementation, while HLRs are more related to the intended function. Actually, an LLR is a refinement of an HLR, and each LLR can be traced back to an HLR. Once HLRs, LLRs, and software architecture are well defined, the implementation can start (bottom of the V-cycle). For the sake of the complete description of the software behavior, the specification can be completed with derived requirements, *i.e.*non-traceable requirements to higher-level requirements. In parallel, specific verification activities interfere with each development activity. To summarize, each line of code implemented in a software must be *(i)* developed and traceable from the LLRs, *(ii)* reviewed against the LLRs, and *(iii)* covered by requirement-based tests (either HLR and/or LLR). Several activities might be performed with independence with respect to the defined software level (*e.g.*, review or verification activities).

## 3.4 Impact of ML on the software qualification approach

Assuming the ML techniques are reduced to non-adaptive learning and used to develop human-assistance function as per AI level 1 defined in EASA, 2021b, we do not anticipate any change to the regulation requirements from EASA, 2021a. However, even if regulation requirements are unchanged, the current standards do not provide sufficient guidance to make a complete demonstration of conformity for an ML-based system, leaving open some gaps schematically represented in Figure 3.4. Indeed, some fundamentals of the usual techniques (like RBE or MBE) are jeopardized, challenging the classical safety guarantee argumentation.

Three gaps can be directly identified: the lack of *specifiability*, the lack of *traceability*, and the introduction of *unintended behavior* that traditional methods cannot handle. We elaborate on these gaps that drive this thesis hereafter. Readers interested in a more thorough overview of the challenges of certifying ML-based items can refer to Mamalet *et al.* (2021).

**Specifiability.** One of the fundamentals of the RBE (or MBE) relies on the correct and complete capture of the ML-based item requirements: either they come from system-allocated requirements (intent) or from their own behavior (emerging functions). In this context, the item can be verified to safely perform the intended function under all foreseeable operating and environmental conditions.

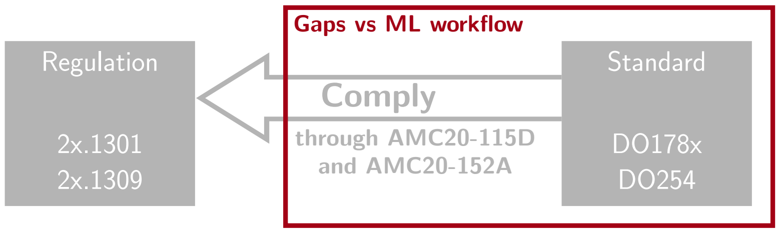

Figure 3.4: Certification Process Overview

ML-based items are often used when it is impossible to specify the function entirely with a classical requirement process. For instance, the pedestrian detection use case in the automotive industry illustrates well the limits of textual or model requirements. Indeed, the description of a pedestrian is not well defined, which makes its complete specification impossible. We can make the parallel with the runway detection in the aeronautical industry; all the possible ways to describe a runway, whatever the environmental and operational conditions (*e.g.*, weather, light conditions, sensor bias), cannot be defined using textual requirements or a modeling technique. For this reason, an ML-based approach will be preferred, with a training dataset of thousands to millions of images whose role would be to complete the requirements capture and allow the development of an acceptable runway detection function.

The absence of a *complete* specification inherently decreases the confidence that the model behavior will always fit the functional intent and, therefore, will be free of unintended behaviors that may jeopardize safety. It is, however, possible to mitigate this problem by approaching it from two angles:

- By ensuring the completeness and the quality of the dataset. For example, for a runway detection function, the idea is to describe as thoroughly as necessary the Operational Design Domain (ODD) in which the detection function is supposed to operate and then validate that the collected data correctly and sufficiently represents this ODD.

- By formally specifying safety properties that the ML model should satisfy for any combination of inputs. For example, for a braking distance estimation function constructed by ML, the idea is to formally specify specific properties, such as bounded variation or monotony, with respect to certain parameters (weight, speed). The satisfaction of such properties of the ML model can, in some cases, be verified, *e.g.*, using an SMT or MILP solver, as we will see in Chapter 6.

**Traceability.** The current standard for software item development — DO-178C — requires traceability between the requirements and the code (in both senses) so that each line of embedded code can be traced and then justified as implementing captured requirements. In an ML development context, this relationship that makes the design a white-box process is lost. Actually, the trained algorithm is a complex parametric mathematical expression where the values of the *learned parameters* are not traceable to any upward functional requirement. Thus it becomes infeasible to demonstrate the completeness and the correctness of implementation using traceability.

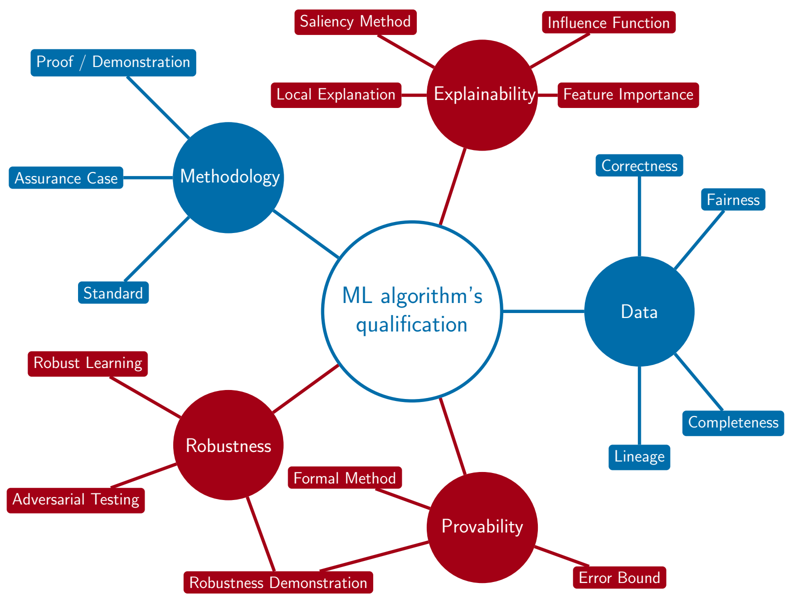

Figure 3.5: Key domains involved in ML model qualification. The domains we address in detail in Chapter 4 are represented in red, and those we present briefly are in blue.

This lack of traceability results in the loss of transparency of the model, making the link from the input data to the output predictions not understandable by a human. It lowers the confidence that the safety properties of the intended function are preserved in its operational domain. In this context, explainable AI is seen as a means to enhance confidence in these algorithms when safety is highly critical. (see Section 4.4).

**Unintended behavior.** All the life-cycle processes (requirements capture, model design, implementation, integration, and requirement-based testing) of the item development workflow (see Figure 3.3) are used to demonstrate the consistency with the intended function. In addition to the problems mentioned above, the learning phase could introduce some unexpected behavior, such as high sensitivity to perturbations, that we cannot measure or prevent by usual verification and validation activities. Consequently, one cannot guarantee the absence of unintended behavior during item operation, specifically those affecting aircraft safety. Unintended behaviors may be due to several factors, such as a low-quality data management process (*e.g.*, biased or mislabeled data) or an inadequate learning process. We will see in Sections 4.2 and 4.3 that there are methods to limit or to formally verify the absence of such effects.

These three gaps are raising even more challenges to tackle; we present in the next section the key domains that will help to reduce the existing gaps.

## 3.5 Key domains involved in ML software qualification

The qualification of ML-based items in the frame of avionic developments offers lots of research challenges in various domains, as summarised by Figure 3.5. We do not aim to provide an exhaustive overview of possible techniques to solve these challenges. We present these challenges through the lens of the avionic domain with the perspective of conformity to the regulation requirements and provide a literature survey. In 2021, the EASA released its first usable guidance regarding the certification of machine learning application (EASA, 2021b): this document gathers the first set of objectives in order to anticipate future EASA guidance and requirements and frame any future development of an ML-based function as a part of a safety-critical system.

Due to the large number of domains involved in the qualification of an ML-based item, we chose to reduce our scope to the software level of engineering and address some novel issues raised by the introduction of Machine Learning. Thus, embeddability and resiliency issues, such as fault tolerance, are not addressed, which may be handled at the system level. We spotlight five different domains which are promising to contribute significantly to support the qualification of ML-based items: explainability, provability, adversarial robustness, data, and methodology (see Figure 3.5).

**Explainability.** Explainability is a topic inherent to ML techniques. It is not needed for classical programming since the item development can be fully explained through the joint requirement and traceability processes which enable the interpretation of each line of embedded code. ML development breaks this understanding chain and makes room for an undesirable black box effect. This is why one can expect that, when required by the safety level of the implemented function, explainability will be requested to add the necessary confidence to support the demonstration of conformity EASA, 2021b. Phillips *et al.* (2020) introduce four principles of explainable AI: explanation, meaningfulness, explanation accuracy, and knowledge limits. Basically, these principles state that an explainable AI must: *(i)* provide an "evidence or reason for all outputs", *(ii)* be meaningful with respect to the audience, *(iii)* be representative of the way that the algorithm produces the output, and *(iv)* be aware of the domain of usage of the algorithm, *i.e.*, it should not give an explanation when the input is out of scope since the algorithm is not designed to work with it.

In the avionic context, four stakeholders could need explanation: the designers, the authorities, the end user (*e.g.*, pilots), and the forensic investigators. Considering the "meaningfulness" principle, each could receive a different explanation. Indeed, the need for an explanation would be different for an engineer who knows the system and for an end-user who has no prior or inner knowledge of the system. The explanations could also differ in their nature whether it is used for debugging purposes during the development (for the designer), for investigation purposes in case of in-flight issues (for authorities or investigator), or for a user assessment of the model predictions when the system is deployed, *e.g.*, a pilot who requests justification of the decision to enhance his confidence in the system before acting. In the literature, we find two kinds of explanation: local explanation Dabkowski and Gal, 2017; Fong and Vedaldi, 2017; Guidotti *et al.*, 2018; Hendricks *et al.*, 2016; Kendall and Gal, 2017; Ribeiro *et al.*, 2016, 2018; Zeiler and Fergus, 2014 and global explanation Lundberg and Lee, 2017; Shrikumar *et al.*, 2017; Zhao and Hastie, 2019.

**Adversarial Robustness.** Robustness comprises several sub-domains (adversarial examples, distributional shift, unknown classes, *physical* attacks, etc.). However, we focus only on *adversarial robustness*, which is defined as the capability of algorithms to give the same outputs considering some variation of the inputs in a region of the state space. The issues addressed by the adversarial robustness domain are at the heart of the software qualification process; it partially addresses the demonstration that there is no unintended function. The aim is to assess and enhance the algorithm's behavior when dealing with *perturbed inputs* (*e.g.*, noises, corner cases, or sensor malfunction). The literature is two-fold: on the one hand, it works at enhancing the adversarial robustness of the algorithm (Carlini *et al.*, 2019; Gilmer *et al.*, 2019; Goodfellow *et al.*, 2015; Kurakin *et al.*, 2017b; Madry *et al.*, 2018; Papernot and McDaniel, 2018; Papernot *et al.*, 2016; Szegedy *et al.*, 2014; Zhang *et al.*, 2019a), on the other hand, at improving its verification (Cissé *et al.*, 2017; Gehr *et al.*, 2018; Huang *et al.*, 2017; Katz *et al.*, 2017; Koh and Liang, 2017; Mirman *et al.*, 2018; Salman *et al.*, 2019b; Singh *et al.*, 2019b; Tjeng *et al.*, 2019; Tsuzuku *et al.*, 2018). One way to improve the adversarial robustness is finding robust learning procedures that are resilient against crafted examples made to defeat the ML algorithm.

**Provability.** On the one side, the provability aspect is the capability to formally demonstrate that model's properties are preserved. Formal methods provide mathematical evidence to support such a demonstration (see Chapter 2). Since robustness, as presented above, can be expressed as a property, the provability comprises the adversarial robustness demonstration. Indeed, formal methods are already used to verify well-defined properties of models, such as the robustness (Gehr *et al.*, 2018; Huang *et al.*, 2017; Mirman *et al.*, 2018; Singh *et al.*, 2019b; Wang *et al.*, 2018) but also safety properties (Katz *et al.*, 2017, 2019). Regarding the aeronautical context, verifying models' properties, either functional or safety-related (like adversarial robustness), is an asset for its qualification since it may avoid time and cost-consuming testing while providing formal proof that the property holds.

On the other side, provability aspect encompasses the methods providing generalization proof of the models. Indeed, the error (or generalization) bound gives guarantees on the ability of the algorithm to generalize (see Section 1.4). An algorithm generalizes well when it maintains its performance on unseen data, *i.e.*, the theoretical error, $r$, is close to the empirical error $\hat{r}_S$ (computed from a training set $S$). The idea is to bound the gap between the theoretical error $r$ and its empirical counterpart $\hat{r}_S$ using probabilistic bounds (Bartlett and Mendelson, 2003; Dziugaite and Roy, 2017; Germain *et al.*, 2015; Masegosa *et al.*, 2020a; McAllester, 1999; Parrado-Hernández *et al.*, 2012; Shawe-Taylor and Williamson, 1997; Valiant, 1984; Vapnik and Chervonenkis, 1971).

**Data management.** In the avionic context, data has been used for a long time, with systems making heavy use of databases or configuration files. However, Machine Learning techniques are bringing a new aspect to the certification approach. Contrary to *classical programming*, ML design techniques are data-driven. Thus, the data management process is even more essential to the demonstration of conformity. As already stated, building a good ML algorithm requires good quality data (*e.g.*, no erroneous or mislabelled data), but this is not sufficient. Indeed, data representativeness also plays a significant role. Representativeness comes from statistics and is quite a challenging problem: it allows checking if the data is an accurate snapshot of the phenomenon to be learned. In other words, it tries to measure if there are sufficient data to learn the intended function, if the data contain the correct information regarding the needs, and if the data are similarly distributed w.r.t. the ODD.

The quality of the data can be measured using metrics (Cai and Zhu, 2015; Picard *et al.*, 2020) such as: accuracy, consistency, integrity, timeliness, traceability, and fairness. **Accuracy** checks if the data is well measured and stored. **Consistency** checks for the expected relationship between parameters regarding the ODD. For example, we could never have an aircraft with zero speed at 40,000 feet. **Integrity** guarantees that data are not corrupted or removed from the source to the final user. **Timeliness** concerns the availability and correctness of the data over time. **Traceability** verifies the reliability of the data source. **Fairness** is about avoiding undesirable bias in the dataset.

**Methodology.** The methodology structures the assurance activities needed to support the qualification aspects, *i.e.*, the demonstration to the Authorities that the item development complies with the recognized and standardized guidance. The red square in Figure 3.4 highlights that changes are required in the AMCs and/or the industrial standards. Therefore, it will be necessary to find alternative means to comply with the regulation material, either by adapting the current qualification approach or by building a new one.

It will be crucial for the future AMCs and/or the industrial standards to clearly define the development process of ML-based items, starting from existing best practices (Amershi *et al.*, 2019; Ashmore *et al.*, 2019; Baylor *et al.*, 2017; Studer *et al.*, 2020), in order to *(i)* ease their development and maintainability, *(ii)* identify the validation and verification activities and *(iii)* guarantee their certifiability. Figure 3.6 describes this development process: requirements from the system/subsystem level are refined into ML-based item requirements to fit the three main stages of the workflow:

- **Data Management Process**: First, data are collected regarding the problem to be solved. Then, the data should be cleaned, and labeled[1], *i.e.*, inaccurate data samples are removed, and each remaining data sample is assigned a true label (known as "ground truth"). Finally, data are preprocessed and split. Preprocessing encompasses all the tasks that transform the data into a more suitable format for the design phases (*e.g.*, feature engineering or data normalization). The preprocessing effort may vary depending on the data collection phase (*e.g.*, data comes from different sources) and the type of data (*e.g.*, images, time series, tabular, sound, text). After the preprocessing step, the data is split into a training, a validation, and a testing dataset.

[1]as the scope is reduced to non-adaptive supervised learning, labeling the data is necessary.

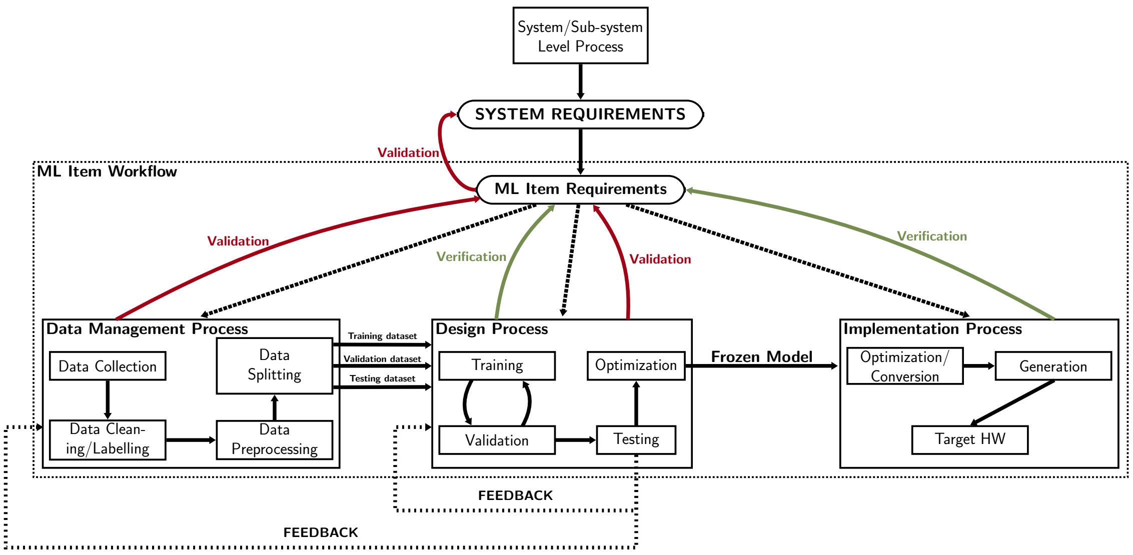

Figure 3.6: Machine Learning development process. The red arrows show the validation activities, while the green ones show the verification activities.

- **Design Process**: As illustrated in Figure 3.6, the design process takes as input the three datasets and outputs a frozen ML model, which means no retraining afterward. The processes in this stage are rather iterative than sequential. Indeed, iteration between the training and the validation phase is done to tune the hyper-parameters of a model. After tuning, the model is tested. However, suppose the performances are lower than expected (based on the ML-based item requirements). In that case, it is possible to loop back either to the data management process or the design process to improve the model. The model is frozen when it attains the performance criteria before implementation.

- **Implementation Process**: Within the implementation stage, the specificity of the hardware's specificity is considered: the frozen model is optimized and converted with respect to the host platform constraints and the operational requirements cascaded from the system level. Finally, a binary code is generated and loaded into the hardware target.

Since its creation in the 80's, the ED-12/DO-178 standard issues gather the *industrial best practices* that impose the necessary rigor of development to avoid introducing errors that may lead to a system failure, and thus increasing the safety of the system. Historically, methodologies undeniably brought efficient results in terms of safety. Thus methodological considerations are key to build a correct certification approach. An essential element in building relevant safety argumentation is the concept of safety case (Rushby, 2015) or assurance case (Martin, 2017). Indeed, Rushby (2015) argues that introducing this kind of methodology (*i.e.*, not prescriptive approaches) in the industries helps to significantly reduce accidents and deaths. Assurance cases, which are a generalization of safety cases, are defined by MITRE (Martin, 2017) as follows: *a documented body of evidence that provides a convincing and valid argument that a specified set of critical claims regarding a system's properties are adequately justified* *for a given application in a given environment*. Hence, assurance cases could be used during the development process of an ML-based item to build, share and discuss sets of structured arguments to support the demonstration of conformity based on outcomes of specific assurance techniques (DAMOUR *et al.*, 2021).

## 3.6 Authorities standpoint and Standardization

Until now, we stated the current status of the certification of "classical" (not ML) software, and then we present some of the problems that ML software raises with respect to its certification and the corresponding challenges that address those problems. Actually, the identification of the gaps and challenges is discussed among groups of scientists and engineers as well as the authorities.

Among all the challenges cited above, one of high importance for both authorities and industries is the standardization of the development methods (methodology aspect in Figure 3.5). Indeed, standardizing is gathering the best industrial practices into a guideline document. The creation of this standard must be done with respect to the authorities' regulatory text: the goal is to provide a standardized process of development and associated assurance objectives that gives the necessary confidence that both the intent and the safety of the function are respected according to its criticality. On the one hand, and as mentioned above, EASA has already released the first guidance to the certification of ML-based systems (EASA, 2021b). On the other hand, many industries and academics are working together to tackle the technical challenges raised but also to provide a standard that would, in the future, be recognized as a means of compliance by the authorities. The group working on the ML-based system certification is the EUROCAE WG-114 group[2].

## 3.7 Conclusion

This chapter contributes to addressing three major gaps regarding the certification of ML-based systems: the specifiability, the traceability, and the risk of unintended behavior. We elaborated on five challenges that result from the gaps: the methodology, the data, the explainability, the robustness, and the provability. All these challenges contribute to taking a step towards the demonstration of conformity of ML-based systems. As this thesis focuses on the ML model's intrinsic properties, the next chapter will perform a literature review on explainability, robustness, and provability (red elements in Figure 3.5).

[2]This group has merged with the SAE G-34 to form a worldwide working group gathering certification and AI scientist experts from the whole aeronautical ecosystem.

# 4 Trustworthiness consideration: Literature review

**This chapter is based on the following paper**

Guillaume Vidot, Christophe Gabreau, Ileana Ober, and Iulian Ober,
arXiv, https://arxiv.org/abs/2106.07221,
Certification of embedded systems based on Machine Learning: A survey

## Contents

**Abstract**

This chapter is dedicated to the literature review of adversarial robustness, provability, and explainability. With these three challenges, we cover a part of the trustworthiness aspect of certification. Our literature review allows summarizing several existing leads toward the certification of ML-based systems while highlighting promising lines of research.

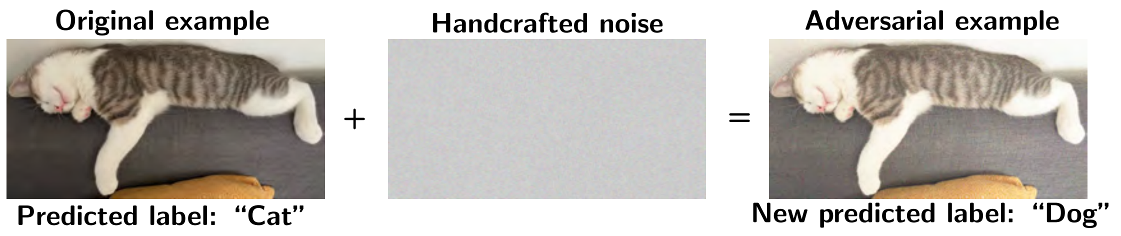

Figure 4.1: Example of an adversarial attack. The left image is the original one predicted as a "cat" by the ML model. The middle image is the purposefully generated noise that is maliciously inserted to change the original label. The right image is the adversarial example, *i.e.*, the original image plus the noise; the ML model now predicts "dog".

## 4.1 Introduction

The previous chapter developed the key domains involved in the certification of ML-based systems (see Figure 3.5). In this chapter, we tackle a subpart of the trustworthiness aspect by diving into the state-of-the-art of three domains: adversarial robustness, provability, and explainability. The purpose of this literature review is to *(i)* illustrate what is already possible in terms of compliance demonstration and *(ii)* identify promising lines of research. Adversarial robustness and provability address safety issues by undermining unintended behaviors. Provability can also verify the intended function by guaranteeing the systems' properties. Explainability addresses the black-box issues, *i.e.*, the fact that ML models are not interpretable by a human, by providing explanations about the internal working of the ML model.

## 4.2 Adversarial Robustness

SZEGEDY *et al.* (2014) was one of the first to point out adversarial examples as a weakness of ML. An *adversarial example* is defined as a perturbed example that looks like its original counterpart (*i.e.*, non-perturbed) but whose prediction is different.Figure 4.1 illustrates this phenomenon; Back to the example of cat and dog classification, we have on the left of Figure 4.1 an image of a cat correctly classified as a "cat". The image on the right results from adding the purposefully generated noise and the original image, which fools the ML model into predicting "dog" instead of "cat".

The highlighting of this weakness gave rise to numerous research results around adversarial attacks and defenses (GILMER *et al.*, 2019; GOODFELLOW *et al.*, 2015; GOURDEAU *et al.*, 2019; KOH and LIANG, 2017; KURAKIN *et al.*, 2017a,b; MADRY *et al.*, 2018; PAPERNOT and MCDANIEL, 2018; PAPERNOT *et al.*, 2016; SZEGEDY *et al.*, 2014; ZHANG *et al.*, 2019a) that will be discussed in this section. Concerning the fundamentals of aviation regulation, which mainly require demonstrating that the system safely performs the intended function, the key role of robustness is twofold. On the one hand, according to ED-12C/DO-178C, it is *the extent to which software can continue to operate correctly despite abnormal inputs and conditions*. On the other hand, and more specifically to ML applications, EASA (2021b) states that an ML system is robust when it *produces the same outputs for an input varying in a region of the state space*.

Table 4.1: Summary of robustness methods: the table shows the papers that provide an attack and/or a defense. The $L_p$-norm column refers to the norm originally considered in the papers to compute the distance between $x$ and $x_{adv}$[1]. "-" means the method does not rely on an $L_p$-norm. The color code indicate whether the norm is used for attack or defense; black means it is used for both.

| Method Name | **Attack/Defense** Papers | Attack | Defense | $\ell_p$-norm |
|---|---|---|---|---|
| C&W | Carlini and Wagner (2017) | ✓ | | $L_0/L_2/L_\infty$ |
| Parseval Training | Cissé *et al.* (2017) | | ✓ | $L_2$ |
| FGSM | Goodfellow *et al.* (2015) | ✓ | ✓ | $L_\infty$ |
| IFGSM | Kurakin *et al.* (2017b) | ✓ | ✓ | $L_\infty$ |
| Gradient-based attack+ILA (Huang *et al.*, 2019) | Li *et al.* (2020) | ✓* | | $L_\infty$ |
| PGD | Madry *et al.* (2018) | ✓ | ✓ | $L_\infty/L_2$ |
| DeepFool | Moosavi-Dezfooli *et al.* (2016) | ✓ | | $L_2$ |
| JSMA | Papernot *et al.* (2016) | ✓ | | $L_0$ |
| DkNN | Papernot and McDaniel (2018) | ✓* | ✓ | $L_\infty/L_2$ |
| L-BFGS | Szegedy *et al.* (2014) | ✓ | ✓ | $L_2$ |
| LMT | Tsuzuku *et al.* (2018) | | ✓ | $L_2$ |
| GDA | Zantedeschi *et al.* (2017) | | ✓ | - |
| YOPO | Zhang *et al.* (2019a) | | ✓ | $L_\infty$ |

* The authors adapt existing attacks to target the weakness of their defense.

Perturbations can be natural (*e.g.*, sensor noise or bias), variations due to failures (*e.g.*, invalid data from degraded sensors), or maliciously inserted to fool the model predictions (*e.g.*, pixels modified in images). When perturbed examples fool the ML algorithm, we talk about *adversarial examples*. We state below two possible manners of generating adversarial examples, *i.e.*, attacking an ML model.

Let $(x, y) \in \mathbb{X}\times\mathbb{Y}$ be a benign example and the true label, $\epsilon$ be a perturbation, $b$ be the maximum allowed perturbation, $\ell$ be a loss function, and $h$ be the trained ML model. An adversarial example that fools $h$ for a given example $(x, y)$ is defined as

$$x + \epsilon^*(x, y) \tag{4.1}$$

where $\epsilon^*(x, y)$ is obtained by solving one of the following optimizations problems:

$$\begin{aligned} &\min_\epsilon \|\epsilon\|_p \\ \text{such that}\quad & h(x+\epsilon) \neq y, \\ & x + \epsilon \in \mathbb{X} \end{aligned} \quad (4.2) \qquad \text{or} \qquad \begin{aligned} &\max_\epsilon \ell(h, (x+\epsilon, y)) \\ \text{such that}\quad & \|\epsilon\|_p \leq b, \\ & x + \epsilon \in \mathbb{X} \end{aligned} \quad (4.3)$$

where $\|\cdot\|_p$ is an $L_p$-norm.

On the one side, Equation (4.2) is the optimization problem where the goal is to find the smallest adversarial example. Indeed, the objective function is the minimization of the $L_p$-norm of $\epsilon$ constrained by the change of label. On the other side, Equation (4.3) is the optimization problem where the goal is to find an adversarial example that maximizes the current loss value constrained by the "size" of the perturbation $\epsilon$. Note that these are the objective functions to find the optimal adversarial example. However, in practice, to speed up the search, one might stop the optimization as soon as the label of the example changes.

[1]It is important to note that the techniques can be adapted with different $L_p$-norm

The optimization problems (Equations (4.2) and (4.3)) lead to get an adversarial example with a different label than the original one; we refer to it as "untargeted attack". A few modifications of the optimization problems allow having "targeted attacks", *i.e.*, the possibility to choose the label of the adversarial examples. Let $t \in \mathbb{Y}$ be the targeted label; we have

$$\begin{aligned} &\min_{\epsilon} \|\epsilon\|_p \\ \text{such that}\quad & h(x+\epsilon) = t, \\ & x+\epsilon \in \mathbb{X} \end{aligned} \qquad \text{or} \qquad \begin{aligned} &\min_{\epsilon} \ell(h,(x+\epsilon,t)) \\ \text{such that}\quad & \|\epsilon\|_p \le b, \\ & x+\epsilon \in \mathbb{X} \end{aligned} \quad .$$

Moreover, the attacks could be done in either a white-box or black-box setting. In a white-box setting, the *attacker* has full access to the model, its parameters, and the training dataset. In a black-box setting, the *attacker* can only query the model.

Finally, the means deployed to overcome adversarial attacks are known as adversarial defenses. One of the most efficient defenses is the Adversarial Training (AT) (Goodfellow *et al.*, 2015; Kurakin *et al.*, 2017b; Madry *et al.*, 2018). It consists in augmenting the training dataset with adversarial examples. However, it is often observed that the adversarial training based on a particular $L_p$-norm is less effective against attacks based on a different $L_p$-norm.

We review adversarial attacks/defenses and summarize the information in Table 4.1. It is important to highlight that, for the attack/defense methods, we report the $L_p$-norm used in the paper. However, the methods could be adapted to other norms.

## Adversarial attacks

The following presented methods are known as *gradient-based methods*. The principle is illustrated in Figure 4.2: The goal is to maximize the loss value to fool the model. Figure 4.2 pictures the maximization of the loss value (the arrow that goes into the high value of the contour plot) while crossing the decision boundary, which means that the model is fooled.

Goodfellow *et al.* (2015) develop an efficient method to find adversarial examples called Fast Gradient Sign Method (FGSM). The attack consists in crafting the adversarial example $x_{adv}$, by adding a fraction, $\iota$, of the loss gradient's sign with respect to the input to the original example $x$:

$$x_{adv} = x + \iota \times \text{sign}(\nabla_x \ell(h,(x,y))).$$

Besides, the authors show that adversarial examples are invariant to the learning and the architectures, *i.e.*, a different architecture trained on different subsets of the dataset tends to misclassify the same adversarial example. Later, Kurakin *et al.* (2017b) proposed an iterative version of FGSM, denoted IFGSM, where at each iteration, a perturbation is added to $x$ by applying FGSM to finally obtain $x_{adv}$ after the desired number of iterations:

$$x^0_{adv} = x \qquad\qquad x^i_{adv} = x^{i-1}_{adv} + \iota \times \text{sign}(\nabla_x \ell(h,(x^{i-1}_{adv},y))).$$

Since perturbations are added several times to $x$ in IFGSM, the $\epsilon$ chosen is, therefore, smaller than in FGSM. The attack introduced by Madry *et al.* (2018) based on Pro-

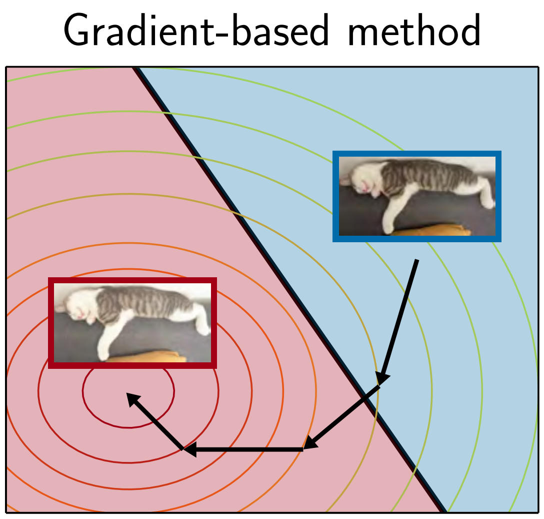

Figure 4.2: Representation of gradient-based method that finds adversarial example. The contour plot shows the loss value; it goes from red (high loss value) to green (low loss value). The arrows illustrate the search for the adversarial example while maximizing the loss.

jected Gradient Descent (PGD) is similar to IFGSM except that PGD randomly initialize $x$ before the optimization:

$$x_{adv}^{0} = x + noise \qquad x_{adv}^{i} = x_{adv}^{i-1} + \iota \times \text{sign}(\nabla_x \ell(h, (x_{adv}^{i-1}, y))).$$

Carlini and Wagner (2017) also develop a gradient-based method but with a slightly different formulation of the optimization problem (see Equations (4.2) and (4.3)). It is similar to Equation (4.2) but at the same time they minimize the model's margin. Besides, the authors provide the formulation of their method for three distance metrics ($L_0$, $L_2$, and $L_\infty$). Moosavi-Dezfooli *et al.* (2016) tailor their attack framework, called DEEPFOOL, for finding the minimum perturbation necessary with respect to the $L_2$-norm[2] to fool the algorithm. They first simplify the optimization problem to a linear classifier, derive the optimal solution for it, and finally, adapt the optimal solution for the linear classifier to a neural network. They provide the algorithm of DEEPFOOL, which describes their iterative approach to estimate the smallest perturbation to create an adversarial example (see Algorithm 2 of Moosavi-Dezfooli *et al.*, 2016). Their results show that the adversarial example crafted by DEEPFOOL is often closer to the original example than the ones crafted by FGSM and L-BFGS (Szegedy *et al.*, 2014).

Papernot *et al.* (2016) leverage the "forward derivative" of a network $h$, to design the Jacobian-based Saliency Map Attack (JSMA). To obtain the forward derivative, they compute, from the input layer to the output layer, the derivative of the network $h$, instead of the derivative of its loss function, with respect to the input $x$. It corresponds to the Jacobian of the function learned by the network $h$. Then, the authors derive an "adversarial saliency map" based on the Jacobian of the network, which points out the input features that should be modified to significantly impact the network's output.

[2]Note that the authors explain how to adapt their method to other $L_p$-norm

Kurakin *et al.* (2017a) assess the adversarial robustness of models deployed in "real-world conditions", *i.e.*, where the only way to communicate with the systems is through its sensors. Indeed, this is a legitimate question since an attacker will not necessarily have access to the ML model. In their experiments, the authors feed an image classification algorithm through a camera. However, they still use the model to generate adversarial examples. They demonstrate that models are still vulnerable against adversarial attacks even in "real-world conditions".

In the avionic context, security measures will be taken to restrict access to the models and the datasets to undermine the white-box and black-box attacks. Nevertheless, the threat still exists since Li *et al.* (2020) propose an attack in a more restricted setting than black-box known as "no-box" setting (Chen *et al.*, 2017) where the attacker only has a very small number of examples that are not training examples. They successfully fooled the models trained on the imagenet dataset by developing an auto-encoder called "prototypical reconstruction" that manages to learn well with very few data. An auto-encoder consists of two parts: an encoder and a decoder. The encoder encodes the input by learning a new representation of the original input. The decoder strives to reconstruct the input from the encoding as close as possible to the original input. The authors introduce a new loss for their auto-encoder that is well-suited for gradient-based attacks. Then, their attack consists in using ILA (Huang *et al.*, 2019), which improves the transferability of the attack, in addition to the gradient-based method.

## Adversarial defense

As stated earlier, one of the most effective defenses, called Adversarial Training (AT), consists in replacing the original training dataset with an "adversarial dataset" crafted with an attack. However, this method is time-consuming since it requires generating adversarial examples at each step of the learning phase. Nevertheless, Goodfellow *et al.* (2015) overcome this issue with FGSM and propose an adversarial training where they replace a part of the training dataset with perturbed examples. Following this principle, other works with stronger attacks propose adversarial training based on their attacks (Kurakin *et al.*, 2017b; Madry *et al.*, 2018). Besides, Madry *et al.* (2018) show that the adversarial training principle boils down to solving the following min-max optimization (or saddle-point problem):

$$\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[ \max_{\|\epsilon\|_p \le b} \ell(h_\theta, (x+\epsilon, y)) \right], \tag{4.4}$$

where $\mathcal{D}$ is the unknown distribution of the dataset, $h_\theta$ is the model parameterized by $\theta$, $\ell$ is the loss, $\epsilon$ is the applied perturbation, and $b$ is the maximum allowed perturbation.

Actually, Equation (4.4) can be rewritten w.r.t the adversarial risk defined as

**Definition 4.2.1.** *True adversarial risk*
For any distribution $\mathcal{D}$ on $\mathbb{X}\times\mathbb{Y}$ and any $h \in \mathcal{H}$, we have

$$\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(h) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \max_{\|\epsilon\|_p \le b} \ell\left(h, (x+\epsilon, y)\right). \tag{4.5}$$

The inner part of Equation (4.5) is the maximization of the loss $\ell$ regarding the noises $\epsilon$: this corresponds to the objective function of the optimization problem that finds the optimal noise to create an adversarial example (see Equation (4.3)). Equation (4.5) is the expected value of this objective function; intuitively, we are measuring the average error of the model $h_\theta$ under attack. Finally, the formalization of the Adversarial Training proposed by MADRY *et al.* (2018) can be rewritten as the minimization of the true adversarial risk:

$$\min_{\theta} \mathcal{R}_{\mathcal{D}}^{\texttt{ROB}}(h_\theta). \tag{4.6}$$

Since $\mathcal{D}$ is unknown, $\mathcal{R}_{\mathcal{D}}^{\texttt{ROB}}(h)$ cannot be directly computed, and then one usually deals with the empirical adversarial risk

**Definition 4.2.2.** *Empirical adversarial risk*
For any dataset $S \sim \mathcal{D}$ and any $h \in \mathcal{H}$, we have

$$\mathcal{R}_{S}^{\texttt{ROB}}(h) = \frac{1}{m}\sum_{i=1}^{m} \max_{\|\epsilon\|_p \le b} \ell\left(h, (x_i{+}\epsilon, y_i)\right). \tag{4.7}$$

We can directly note the parallel between ERM and AT. In ERM, the empirical risk (Definition 1.3.2) is minimized while in AT, the empirical adversarial risk (Equation (4.7)) is minimized. To the best of our knowledge, AT based on PGD is one of the most efficient defenses against attacks using the $L_\infty$-norm.

Though the methods presented above make AT feasible, it still increases the learning time of a model. ZHANG *et al.* (2019a) aim at improving the computation cost of AT by reformulating it as a differential game and then deriving the Pontryagin's Maximum Principle (PMP)[3]. The PMP reveals that AT is closely linked with the first layer of neural networks. Therefore, they develop YOPO (You Only Propagate Once), which leverages this fact by limiting the number of forward and backward propagation without worsening the network's performance. Their experiments show that YOPO learns four to five times faster to achieve as good results as adversarial training based on PGD.

PAPERNOT and MCDANIEL (2018) find another way to enhance ML algorithms' robustness. The authors develop a method called Deep k-Nearest Neighbors (DKNN), *i.e.*, a hybrid classifier that mixes Deep Neural Network (DNN) and kNN. Their motivation is to improve the confidence estimation, the model interpretability, and the robustness. The principle of DKNN is to find the nearest neighbors (from the training set) of an input $x$ at each layer of the DNN and find the classes of each neighbor. This procedure allows the analysis of the evolution of classes in the neighborhood of $x$ throughout the network. That is why the authors introduce several metrics such as *nonconformity* and *credibility*. The nonconformity metric is used to measure the discrepancy between the labels of the neighborhood and the predicted label of an input $x$. A high nonconformity value means that the labels predicted for the neighborhood of $x$ differ from those predicted for $x$. The computation of the credibility measure is based on a "calibration dataset"

[3]PMP is used in optimal control theory to find the best solution regarding input and constraint.

whose examples were not used for the training. Then, the credibility of an input $x$ is the ratio of nonconformity measures of the examples from the calibration dataset that are greater than the input's nonconformity measure. DKNN increases the *confidence* in predictions thanks to the *credibility measure*, which is used to assess and select the model's predictions. Moreover, this algorithm becomes more interpretable because the neighborhood it generates provides insight into how the model works. PAPERNOT and MCDANIEL (2018) claim that DKNN would prevent adversarial examples by assigning a low credibility measure to them. Their empirical results show particularly encouraging results against the attack method proposed by CARLINI and WAGNER (2017).

The above techniques' drawback lies in their dependence on the $L_p$-norm. Indeed, even if it brings robustness against attacks relying on the same metric distance (like PGD), the defense could become ineffective when the attacks depend on a different metric distance. Instead of searching for the perturbation that maximizes the loss function of a model (Equation (4.4)), ZANTEDESCHI *et al.* (2017) propose to consider all the local perturbations (drawn from a gaussian distribution) around each example:

$$\min_{\theta} \underset{(x,y)\sim\mathcal{D}}{\mathbb{E}} \ \underset{\delta\sim\mathcal{N}(0,\sigma^2)}{\mathbb{E}} \ell(h_\theta, (x+\delta, y)),$$

where $\mathcal{D}$ is the unknown distribution of the dataset, $h_\theta$ is the model parameterized by $\theta$, and $\ell$ is the model's loss function. They claim that their method, known as Gaussian Data Augmentation (GDA), instead of crafting adversarial example only in the direction of the gradient, explore much more directions around the examples. GDAoutperforms AT considering adversarial accuracy measure (*i.e.*, accuracy on an attacked dataset) on MNIST (LECUN *et al.*, 1998a) and CIFAR-10 (KRIZHEVSKY, 2009).

A hypothesis that could explain the adversarial phenomenon is the high expressiveness of the models, *i.e.*, the capacity of the models to learn complex behavior. CISSÉ *et al.* (2017) and TSUZUKU *et al.* (2018) explore this lead by constraining the Lipschitz constant of neural networks to enhance their robustness. Considering all possible couple of points of the input space of a function $h$, the Lipschitz constant is the smallest value that upperbounds the absolute value of the slopes given by each pair of points. Intuitively, we would say that constraining the Lipschitz constant of a function $h$ to a small value would smooth the function and reduce its expressiveness. CISSÉ *et al.* (2017) develop specific training called PARSEVAL TRAINING, which ensures that the Lipschitz constant of all the network's layers is smaller than $1$ by having Parseval tight weight's matrices. We refer interested readers to the paper CISSÉ *et al.*, 2017 for the mathematical details. TSUZUKU *et al.* (2018) propose another training method, called Lipschitz Margin Training (LMT), to constrain the Lipschitz constant of a network. They first link the margin of a network to the Lipschitz constant. From that, they derive an algorithm, relying on the relation between the margin and the Lipschitz constant, that enhances the robustness of the network. Besides, LMT provides a "certified" lower bound on the smallest perturbation that can defeat the network. This latter property could be desirable for demonstrating conformity to the regulatory requirements of an embedded system based on ML. Indeed, theoretical results are great assets for the software certification process. For example, GOURDEAU *et al.* (2019) leverage the *PAC theory* (Probably Approximately Correct theory) to provide theoretical proof of the feasibility of robust

Table 4.2: Summary of verification methods: the table shows the papers that provide a verification method. The column "# parameters" reports the size of the Neural Network in terms of the number of parameters used in the papers; it gives an insight into the scalability of the associated method.

| | **Verification** | | | |
|---|---|---|---|---|
| Method Name | Papers | Sound | Complete | # paramaters |
| AI$^2$ | Gehr *et al.* (2018) | ✓ | | > 238,000 |
| DLV | Huang *et al.* (2017) | ✓ | ✓ | > 138,000,000 |
| RELUPLEX | Katz *et al.* (2017) | ✓ | ✓ | ∼ 13,000 |
| MARABOU | Katz *et al.* (2019) | ✓ | ✓ | ∼ 13,000 |
| REFINEZONO | Singh *et al.* (2019b) | ✓ | | > 1,000,000 |
| RELUVAL | Wang *et al.* (2018) | ✓ | | $\sim 670,000$ |
| PRIMA | Müller *et al.* (2022) | ✓ | | > 250,000 |
| CNN-ABS | Ostrovsky *et al.* (2022) | ✓ | ✓ | ∼ 850 |
| DEEPCERT | Paterson *et al.* (2021) | ✓ | ✓ | $> 1,600,000$ |
| $\beta$-CROWN BaB | Wang *et al.* (2021) | ✓ | ✓ | $> 210,000$ |
| LIRPA | Xu *et al.* (2020) | ✓ | | $> 19,000,000$ |
| MIP VERIFY | Tjeng *et al.* (2019) | ✓ | ✓ | $> 4,000,000$ |

learning from the perspective of computational learning theory. The authors focus on the setting where the input space is the boolean hypercube $\mathbb{X} = \{0, 1\}^n$ and show that robust learning is feasible or not for different classes of models.

From a methodological standpoint, Carlini *et al.* (2019) provide advice and good practices for evaluating adversarial robustness. Especially they claim that one that develops a new defense mechanism must think about an "adaptive attack" to evaluate the efficiency of their defense mechanism. In other words, to evaluate the defense efficiency of a new defense mechanism, the worst attack must be tested against it. Gilmer *et al.* (2019) conduct experiments showing that improving adversarial robustness also improves corruption robustness, *i.e.*, the robustness of the model against distributional shift. They want to show that both types of robustness could be the manifestation of the same phenomenon. Nevertheless, it is encouraging that improving adversarial robustness positively impacts corruption robustness.

Robustness verification is part of the objectives stated in the first guidance delivered by the EASA (2021b): the methods seen in this section improve the robustness of ML models and hence, help to complete the objective stated by the EASA. Though those methods might be efficient, they do not provide any proof of robustness, which is the ultimate goal. We will see in the next section how to obtain such proof.

## 4.3 Provability

In the context of ML qualification, provability will help to demonstrate that the model preserves its properties. Existing ML model verification frameworks allow verifying different types of properties (*e.g.*, functional or safety-related property) as long as they are expressed in the logic understood by the verification framework.There are two important criteria for a verifier: *completeness* and *soundness* (see Chapter 2 for more details). As a reminder, *completeness* means that the verifier can decide the validity of all properties[4] that hold, whereas *soundness* means that the verifier cannot prove any wrong property. All the methods presented in Table 4.2 are at least sound; otherwise, we cannot obtain any guarantee from their outputs. Complete methods suffer from scalability issues, while incomplete ones suffer from a lack of precision. Despite these drawbacks, formal methods are worth some attention since they provide proof and may save testing time.

KATZ *et al.* (2017) develop a method that uses the Simplex algorithm — a method to solve linear programming problems — and Satisfiability Modulo Theories (SMT) solvers — solvers for formulas of first-order logic with respect to some background theories such as arithmetic or arrays. (see Section 2.3 for more details) — which the authors adapt to work with neural network using the Rectified Linear Unit (ReLU) activation function defined as $y = max(0, x)$. Their method is called RELUPLEX and the authors assess its performance on the ACAS XU case study (MANFREDI and JESTIN, 2016). The framework uses quantifier-free formulas to formalize properties, consisting of constrained domains for inputs and outputs. The paper presents ten properties for which verification with RELUPLEX is attempted, two of them terminating with a timeout, *i.e.*, RELUPLEX was unable to end the verification within the given time. The longest verification took almost five days, while the fastest took less than eight minutes. MARABOU(KATZ *et al.*, 2019), the successor of RELUPLEX, is based on the same principles. The framework now supports piecewise linear layers and activations and has a "divide-and-conquer mode" which improves the computation time of the verification.

Many verifiers exist in the literature, coming with their own specificity (URBAN and MINÉ, 2021). SHRIVER *et al.* (2021) highlight that this may be a burden for users who want to compare verifiers and propose a framework for Deep Neural Network Verification (DNNV) aiming to ease the use of verifiers and benchmarks.

Now, we focus on the works that used the robustness property as a benchmark to test their verifier, although most are not restricted to the robustness property. Robustness verification is an emerging field (CISSÉ *et al.*, 2017; GEHR *et al.*, 2018; HUANG *et al.*, 2017; MÜLLER *et al.*, 2022; OSTROVSKY *et al.*, 2022; PATERSON *et al.*, 2021; SHRIVER *et al.*, 2021; SINGH *et al.*, 2019b; TSUZUKU *et al.*, 2018; WANG *et al.*, 2021; XU *et al.*, 2020). Verifying robustness means ensuring that the algorithm outputs the same label $y$ for an example $x$ and its neighborhood whose computation usually relies on metric distance (*e.g.*, $L_p$-norm). This property could be stated as follows:

$$\forall x' \in \mathbb{X} \text{ such that } \|x - x'\|_p \leq b, \quad h(x) = h(x'), \tag{4.8}$$

where $h$ is the ML model and $\|x - x'\|_p \leq b$ represents the neighborhood around $x$ such that the distance between $x$ and any neighbor $x'$ does not exceed $b$. Indeed, the existence of adversarial examples could be seen as the negation of Equation (4.8). Thus, verifying that Equation (4.8) holds boils down to checking the adversarial robustness of an algorithm.

TJENG *et al.* (2019) propose MIP VERIFY, a tool that uses a MILP solver to check the adversarial robustness of ML model (see Section 2.3 for more details). The principle

[4]that is, all properties expressible in the logical formalism supported by the verifier

is to express ML models as linear programs comprising the encoding of the robustness property, and solving the linear program indicates whether the property holds. The encoding of the activation function and the bounds on each neuron determine the efficiency of this method. MIP VERIFY uses an asymmetric formulation for the activation function and includes a procedure named progressive bound tightening, which seems to improve the running time.

SINGH *et al.* (2019b) develop REFINEZONO, a verification framework mixing optimization techniques (MILP, LP/MILP relaxation) and abstract interpretation — a method mainly used in the static analysis that leverages over-approximation to analyze the behavior of computer programs (see Section 2.3 for more details). Mixing those techniques allows better scalability for complete verifiers and improves the precision of incomplete ones. The principle of REFINEZONO is to compute the boundaries of the neurons using optimization techniques and abstract interpretation. As in KATZ *et al.* (2017), a property is expressed as domains for inputs and outputs. A robustness property expressed as *same label predicted whatever the point within a region around $x$* can be easily transformed into domains for input and output: the region around $x$ is the domain for the input and the domain for the output is given to have the correct class. For example, Figure 4.3 shows a robustness property where we verify that in the neighborhood of the point $x = (0.9, 0.7, 0.4)$, the model outputs only the desired class (the output layer always predicts the same class). OSTROVSKY *et al.* (2022) propose to improve complete verifiers scalability by using an abstraction-refinement approach and propose a framework named CNN-ABS. The principle is to find sound over-approximation of the original network (abstraction) to perform the verification on: it corresponds to a smaller network created by removing neurons from the original network so that if the property holds for the smaller network, it also holds for the original one. However, due to the abstraction, the verifier may output a spurious counter-example, *i.e.*, a counter-example for the smaller network but not for the original one. In this case, the smaller network is refined by adding a subset of the removed neurons before rerunning the verification. At a higher level, CNN-ABS alternates between abstraction and refinement steps until reaching the level of abstraction that outputs a correct answer.

Instead of improving complete verifiers scalability, such as MILP with abstract interpretation (SINGH *et al.*, 2019b), it is possible to make abstract interpretation as precise as possible (GEHR *et al.*, 2018). The benefit of applying only abstract interpretation is to verify properties on large neural networks (*e.g.*, CNN). GEHR *et al.* (2018) develop the AI$^2$ framework, which implements abstract interpretation techniques to verify neural networks. Their framework also focuses on verifying the robustness property (Equation (4.8)). In AI$^2$, the input is replaced by an abstract domain, then it is propagated through the network thanks to abstract transformers until the output layer. Property verification is applied to the abstract domain obtained at the output layer. The method is sound but incomplete, meaning that the result can either be true or inconclusive for a property. Hence, the choice of the abstract domain for the input is important because it influences the precision of the verifier.

Most of incomplete techniques approximate the activation function of the neurons wisely (GEHR *et al.*, 2018; SINGH *et al.*, 2019b; WANG *et al.*, 2018, 2021; XU *et al.*,

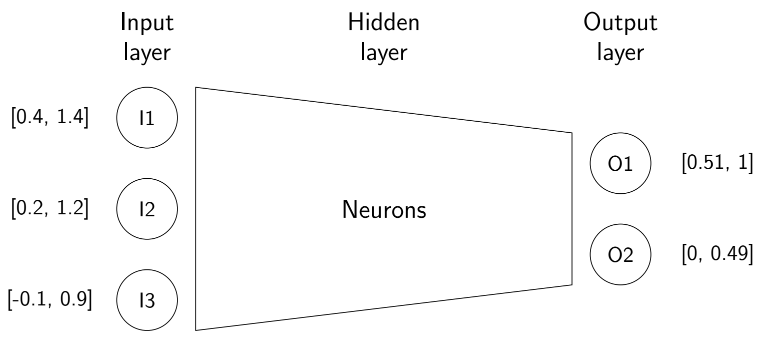

Figure 4.3: Robustness property example. We consider a hypercube (of size 1) as a neighborhood around $x = (0.9, 0.7, 0.4)$ and the output must always be the first class, *i.e.*, the lower bound of the output neuron O1 must be greater than the upper bound of the neuron O2.

2020). Nevertheless, several works state that involving multiple neurons simultaneously for the over-approximation may improve the precision of the techniques (Müller *et al.*, 2022; Palma *et al.*, 2021; Salman *et al.*, 2019b; Singh *et al.*, 2019a; Tjandraatmadja *et al.*, 2020). Particularly, Müller *et al.* (2022) propose prima, a framework that leverages convex hull approximation of convex polyhedra to compute the abstraction of several neurons. The method used in prima allows to get tighter abstractions and thus outperforms the other incomplete method regarding the number of inconclusive answers.

Huang *et al.* (2017) reduce the proof of robustness to a search of adversarial examples. They focus on the robustness of image classification, which is quite a challenging problem considering the large input domain of an image and all perturbation it can have. The first assumption is that *images are discrete*. Therefore, Huang *et al.* (2017) explore entirely the region of an image $x$, searching for an adversarial example, by defining a set of *minimum* manipulation. To develop a verification procedure that ensures the safety of a point, the authors first formally define a *safe point*. Based on their assumption of *discrete images*, the authors develop an algorithm of *safety verification* using SMT solvers; their tool is called Deep Learning Verification (dlv). Since the method of Huang *et al.*, 2017 is sound and complete, if the algorithm does not find any adversarial example, it means that there are none for the region and the manipulation set considered. Moreover, Huang *et al.* (2017) show that their algorithm is scalable to big neural networks by doing experiments on neural networks with more than one hundred million parameters.

Xu *et al.* (2020) provide a framework, named lirpa, that unifies the methods of linear relaxation whose goal is to find affine functions that upper-bound and lower-bound the network's output within a sub-domain of the input. For example, to prove a robustness property (as stated in Equation (4.8)) with lirpa, one must show that the lower bound of the true label is greater than all upper bounds of all other classes; this would mean that in the worst case, the model will always output the correct class. More recently, Wang *et al.* (2021) propose a method called $\beta$-crown, also based on linear relaxation that gives more tight bounds than other existing linear relaxation-based methods. The novelty of Wang *et al.* (2021) stands in the bound formulation, which relies on Lagrange functions and ReLU splits encoding. As is, $\beta$-CROWN is an incomplete method. However, WANG *et al.* (2021) add a branch-and-bound mechanism to $\beta$-CROWN, namely $\beta$-CROWN BaB, to make it complete and show through experiments that $\beta$-CROWN BaB is time efficient.

The adversarial robustness verifiers mentioned above are promising for ensuring the absence of unintended behavior in ML models, but these guarantees hold for small perturbations which are not necessarily meaningful. In the aeronautical context, robustness against meaningful perturbation, such as weather conditions, might be relevant for use cases dealing with images. PATERSON *et al.*, 2021 take a step towards it and propose DEEPCERT, a tool verifying meaningful perturbation, namely "contextually relevant perturbations". DEEPCERT encodes three perturbations: blur, haze, and contrast variation.

Another side of the *provability* domain is about generalization bounds (see Section 1.4 for an introduction on the concept of generalization). The theory relies on metrics that measure the model complexity such as the VC-dimension (VAPNIK and CHERVONENKIS, 1971), the Rademacher complexity (BARTLETT and MENDELSON, 2003) or the KL-divergence with the Probably Approximately Correct-Bayesian (PAC-Bayesian) framework (MCALLESTER, 1999; SHAWE-TAYLOR and WILLIAMSON, 1997) to obtain these generalization bounds. Notably, it has been shown that the PAC-Bayesian theory is applicable to neural networks (DZIUGAITE and ROY, 2017) and can also provide tight bounds (PARRADO-HERNÁNDEZ *et al.*, 2012).

Generalization bounds provide guarantees on the performance of ML models on unseen data. Recent works empowered the idea of giving guarantees on the performance of the ML model under adversarial attacks, *i.e.*, generalization bounds on the so-called adversarial risk (KHIM and LOH, 2018; MONTASSER *et al.*, 2020; VIALLARD *et al.*, 2021; YIN *et al.*, 2019). As for the classical bounds, some of them rely on the Rademacher complexity (KHIM and LOH, 2018; YIN *et al.*, 2019), the VC-dimension (MONTASSER *et al.*, 2020) or the KL-divergence using the PAC-bayesian theory (VIALLARD *et al.*, 2021).

Robustness verification and generalization guarantees are objectives stated in the first guidance of the EASA (EASA, 2021b). We saw in this section some methods that could be the means to reach both of these objectives.

## 4.4 Explainability

As already stated, explainability is a new constraint for embedding ML-based systems. Implicitly contained in the traditionally programmed components, it has become challenging to demonstrate that the ML system's outcomes are trustworthy. Improving this confidence level seems inescapable to meet the acceptance criteria of the ML application users (*e.g.*, designers, authorities, investigators, or pilots). It has been clearly identified as a means of acceptance in the EASA AI roadmap (EASA, 2020) and their first usable guidance (EASA, 2021b).

Table 4.3: Summary of explanation method: the column "method name" report the technique used when the proposed method has no name

| | Method name | Papers | Model targeted |
|---|---|---|---|
| Local Explanation | MASKING MODEL | DABKOWSKI and GAL (2017) | ALL |
| | MPSM | FONG and VEDALDI (2017) | ALL |
| | LORE | GUIDOTTI *et al.* (2018) | ALL |
| | GVE | HENDRICKS *et al.* (2016) | 🕸 |
| | INFLUENCE FUNCTION | KOH and LIANG (2017) | 🕸 |
| | LIME | RIBEIRO *et al.* (2016) | ALL |
| | ANCHORS | RIBEIRO *et al.* (2018) | ALL |
| | DECONVNET | ZEILER and FERGUS (2014) | 🕸 |
| Global Explanation | SHAP | LUNDBERG and LEE (2017) | ALL |
| | DEEPLIFT | SHRIKUMAR *et al.* (2017) | 🕸 |
| | NPSEM | ZHAO and HASTIE (2019) | ALL |

where ALL = model agnostic and 🕸 = DNN specific.

In this section, we have reviewed methods that may help demonstrate ML model compliance. We will see methods that work on images, text, or numerical data.

We identify two types of explanations: local explanations and global explanations. Both subdomains suit different needs: *(i)* explanation at the prediction level (local) and *(ii)* explanation of the behavior of the model regarding inputs evolution (global). In each subdomain, we face two types of explanation techniques: *model agnostic* and *model specific*. Model agnostic refers to the fact that whatever the ML algorithm, we can provide an explanation (DABKOWSKI and GAL, 2017; FONG and VEDALDI, 2017; GUIDOTTI *et al.*, 2018; LUNDBERG and LEE, 2017; RIBEIRO *et al.*, 2016, 2018; ZHAO and HASTIE, 2019). Model-specific explanations will only work with a specific type of ML algorithm; most papers in the literature focus on DNN (HENDRICKS *et al.*, 2016; KENDALL and GAL, 2017; KOH and LIANG, 2017; SHRIKUMAR *et al.*, 2017; ZEILER and FERGUS, 2014).

## Local Explanation

Computer Vision is a complex domain because of the nature of the data we deal with: images or videos. Since the development of DNN, it has become easier to perform well on image classification or object detection, for example. However, we still do not fully understand why methods based on DNN work so well. Hence, researchers develop new methods to explain, assess, and increase the confidence we can have in the decision taken by DNN for computer vision tasks.

**Model agnostic local explanation.** Let us consider three systems: LIME — Local Interpretable Model-agnostic Explanations — (RIBEIRO *et al.*, 2016), ANCHORS

- **LORE**

$$\begin{aligned} r = \quad & (\{\textit{credit\_amount} > 836,\ \textit{housing} = \textit{own},\ \textit{other\_debtors} = \\ & \quad \textit{none},\ \textit{credit\_history} = \textit{critical account}\} \to \textit{decision} = 0) \\ \Phi = \quad & \{\ (\{\textit{credit\_amount} \leq 836,\ \textit{housing} = \textit{own},\ \textit{other\_debtors} = \\ & \quad \textit{none},\ \textit{credit\_history} = \textit{critical account}\} \to \textit{decision} = 1), \\ & (\{\textit{credit\_amount} > 836,\ \textit{housing} = \textit{own},\ \textit{other\_debtors} = \\ & \quad \textit{none},\ \textit{credit\_history} = \textit{all paid back}\} \to \textit{decision} = 1)\ \} \end{aligned}$$

Figure 4.4: Explanations of LORE. (taken from GUIDOTTI *et al.* (2018). we only use the part of Figure 9 that concerns their method, LORE.)

(RIBEIRO *et al.*, 2018), and LORE — LOcal Ruled-based Explanation — (GUIDOTTI *et al.*, 2018). These techniques use inputs and outputs and try to approximate the behavior of the ML algorithm locally. We can note that LORE is not applicable to images. These three systems are based on the same principle: explore a neighborhood around a sample $x$ and provide an explanation for the sample $x$ thanks to the neighborhood. If the reader wants to go further, GARREAU and LUXBURG (2020) provide theoretical explanations of LIME.

The generation of the neighborhood differs according to the system. LIME and ANCHORS use an interpretable representation and apply some perturbations to it. An interpretable representation could be a binary vector where the ones are the relevant information, and a perturbation could be the modification on the vector (0 switches to 1), but this should be defined according to the problem that needs to be solved. RIBEIRO *et al.* (2016) define metrics to measure the distance between the neighbors and the original sample. ANCHORS and LIME are using the same method of neighborhood generation. LORE uses a genetic algorithm to generate a balanced neighborhood. The genetic algorithm is tuned to create the best possible neighbors around the desired sample.

Since the neighborhood is generated, it remains to provide the explanation. In LIME, linear models are used, while in ANCHORS, decision trees are used. For the ANCHORS' explanation, "anchors" are used and is defined as the minimum number of features that lead to the right prediction. Note that RIBEIRO *et al.* (2018) define several bandit algorithms to find the best anchors. An explanation given by LIME corresponds to the image area that helps to make the decision. ANCHORS and LORE provide explanations in the IF-THEN form. GUIDOTTI *et al.* (2018) — LORE — extract explanations from the decision tree while RIBEIRO *et al.* (2018) — ANCHORS — only check the presence or absence of anchors. Figure 4.4 shows an explanation given by LORE: $r$ corresponds to the explanation of the decision, and $\Phi$ corresponds to the minimum changes that should occur to flip the decision. The original decision comes from an algorithm which was trained on the German credit dataset[5] to recognize good ("0") or bad ("1") creditor according to a set of attributes (age, sex, job, credit amount, duration, etc.).

A popular tool to explain decision on images is the *saliency map* (ADEBAYO *et al.*, 2018; DABKOWSKI and GAL, 2017; FONG and VEDALDI, 2017). Intuitively, a saliency map highlights the features, *i.e.*, the pixels on images, that the model considers to make

[5]https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

its decision. Fong and Vedaldi (2017) propose an agnostic gradient-based method that considers explanations as meta-predictors. A meta-predictor is a rule used to explain the model's prediction. One advantage of using meta-predictors as explanations is that one can measure the *faithfulness* of the explanation by computing its prediction error, *i.e.*, it represents the number of times the model and the rule disagree on the prediction. Then, Fong and Vedaldi (2017) claim that a good explanatory rule (*i.e.*, meta-predictor) must rely on the local explanation principle to produce a saliency map. Indeed, the authors of Fong and Vedaldi, 2017 want to study the model's behavior using the neighborhood of the original example. This neighborhood is given by perturbing the original example. They introduce the notion of "meaningful perturbations". They argue that the explanation would be better if the perturbations applied to build the neighborhood of the original example mimic *natural image effect*. Based on meaningful perturbations[6], the authors define an optimization problem, which tries to find the smallest area of the original image that must be perturbed to reduce the prediction probability of the true class of the original image; Finally, this method outputs the saliency map that explains the model's prediction. In Table 4.3, we named their method Meaningful Perturbation Saliency Map (mpsm).

On the other side, Dabkowski and Gal (2017) develop a masking model, which learns how to generate interpretable and accurate saliency maps for ML model. The authors craft a new objective function, which ensures the quality of the saliency map: it ensures that the region is necessary to the good classification, while its absence leads to a low probability of picking the good class and, at the same time, it penalizes large and non-smooth regions. The masking model comprises features filter (input) given by a trained network and upsampler layer, which upscale the input into the original image dimension to provide a mask. Dabkowski and Gal (2017) train the masking model by directly minimizing its new objective function; their resulting saliency maps provide accurate salient areas (Figure 4.5).

**DNN specific local explanation.** Adebayo *et al.* (2018) develop a methodology to test the usefulness of a saliency method used for explanations. It is based on two tests: model parameters randomization test and data randomization test. A saliency method will fail the tests if it shows the same result for the randomized and trained cases. A failure points out the independency of the saliency method regarding the architecture parameters or labeled data. With the methodology proposed by Adebayo *et al.* (2018), we can test model-agnostic and model-specific methods.

A few years before the works presented above, Zeiler and Fergus (2014) introduced a method that allows the visualization of the internal workings of convolutional neural networks (CNN). Their motivation was to understand how and why CNNs work well by studying the hidden layers of CNNs. The principle is to sample the features learned by a neural network back to the original image dimension. However, contrary to the methods discussed above, the goal was not to provide a saliency map that explains the model's predictions but only to visualize the features that a neural network learns for

[6]the authors used blur and noise as meaningful perturbation

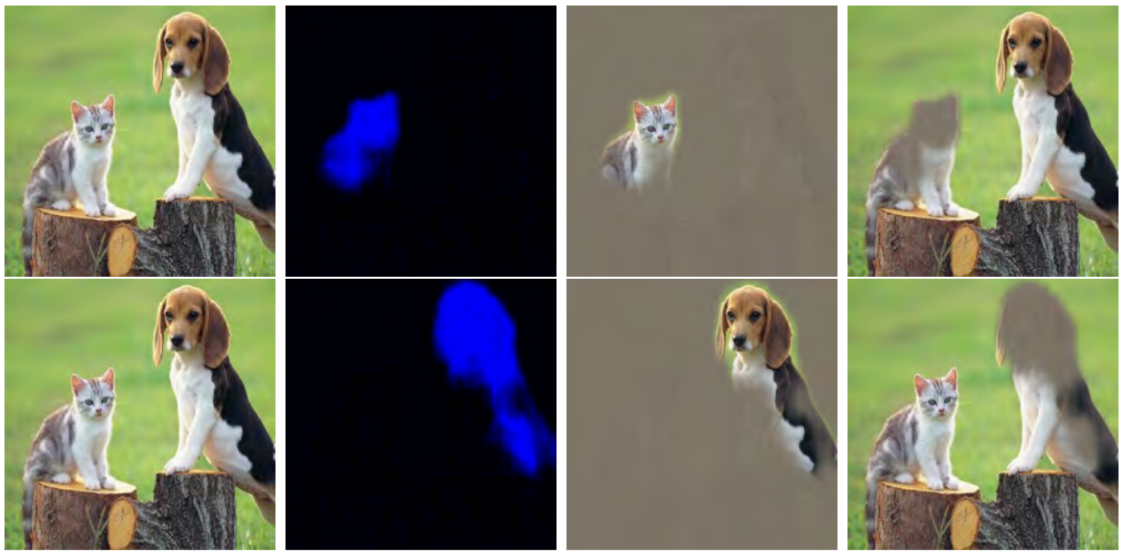
(a) Input Image (b) Generated saliency map (c) Image multiplied by the mask (d) Image multiplied by inverted mask

Figure 4.5: An example of explanations produced by DABKOWSKI and GAL (2017). The top row shows the explanation for the "Egyptian cat" while the bottom row shows the explanation for the "Beagle". Note that produced explanations can precisely highlight or remove the selected object from the image.

classifying. The system developed by ZEILER and FERGUS (2014) is known as "multi-layered Deconvolutional Network" (DECONVNET). One advantage of this technique is the possibility to start from any layer to see the features learned by the neural network at a specific layer.

Another way of explaining an image is literally to provide written explanations: HENDRICKS *et al.* (2016) propose a method known as GVE — Generating Visual Explanation — which provides *visual explanation* for images. Visual explanations are defined as *class discriminative* and *accurately described*, *i.e.*, the textual explanation highlights elements of the image specific to the class. The authors use VGG (SIMONYAN and ZISSERMAN, 2015), a CNN for image classification, and add to it an LSTM (Long Short Term Memory) (HOCHREITER and SCHMIDHUBER, 1997) that generates visual explanations; the efficiency of this system relies on the loss function. More precisely, HENDRICKS *et al.* (2016) define two loss functions for their network: one for the accurate description (relevance loss) that handles the probability of word occurrence in the sentence and the other for the class discrimination (discriminative loss), which is based on a reward function. The authors reach good results in an experiment that deals with bird classification. Figure 4.6, taken from the paper HENDRICKS *et al.* (2016), shows the difference between visual explanation, image description, and class definition.

KOH and LIANG (2017) increase the explainability of DNNs through the use of INFLUENCE FUNCTIONS. Unlike the other methods, where explainability is increased by providing evidence directly on the examples, KOH and LIANG (2017) provide the training data that leads to a prediction in order to explain that prediction. The principle of their method is to compute the impact on the predictions while modifying the training dataset. For explainability purposes, they study the effect of removing training data. The

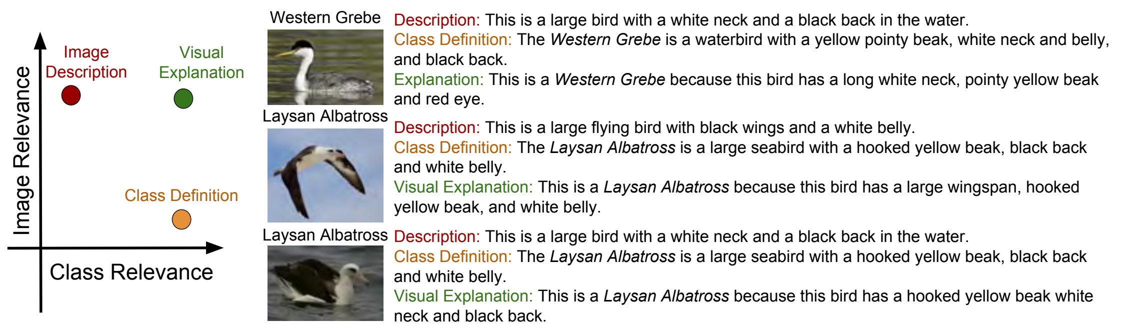
Image Relevance
Class Relevance
Image Description
Visual Explanation
Class Definition
Western Grebe
Description: This is a large bird with a white neck and a black back in the water.
Class Definition: The Western Grebe is a waterbird with a yellow pointy beak, white neck and belly, and black back.
Explanation: This is a Western Grebe because this bird has a long white neck, pointy yellow beak and red eye.
Laysan Albatross
Description: This is a large flying bird with black wings and a white belly.
Class Definition: The Laysan Albatross is a large seabird with a hooked yellow beak, black back and white belly.
Visual Explanation: This is a Laysan Albatross because this bird has a large wingspan, hooked yellow beak, and white belly.
Laysan Albatross
Description: This is a large bird with a white neck and a black back in the water.
Class Definition: The Laysan Albatross is a large seabird with a hooked yellow beak, black back and white belly.
Visual Explanation: This is a Laysan Albatross because this bird has a hooked yellow beak white neck and black back.

Figure 4.6: Visual explanations are both image relevant and class relevant. In contrast, image descriptions are image relevant, but not necessarily class relevant, and class definitions are class relevant but not necessarily image relevant (taken from HENDRICKS *et al.* (2016)).

removed training data that strongly decreases the probability of getting the right class of a given example are actually the data that leads to the prediction. The training data, given as an explanation of a prediction, provide insight into the model's behavior. Indeed, by looking at the data that influence the prediction, it is possible to detect abnormal behavior of the model.

## Global Explanation

Global explanation techniques manage to explain the evolution of the model outputs according to the trends of the inputs. A popular method is known as *feature importance*. The principle of this method is to find out which input features play a significant role in the decision of the ML algorithm (LUNDBERG and LEE, 2017; SHRIKUMAR *et al.*, 2017; ZHAO and HASTIE, 2019).

**Model agnostic global explanation.** One way to exploit the feature importance principle is to use the Shapley value from cooperative game theory. The idea is to find a fair share of the gain between players in a cooperative game regarding the involvement of each player in the coalition. LUNDBERG and LEE (2017) develop SHAP (SHapley Additive exPlanations), which uses the Shapley value to explain ML algorithm predictions. In this context, the features are the players, and the prediction is the payout. Then, SHAP computes the contribution of each feature to the prediction. Since the computation of the Shapley value requires all the possible permutations of the players (features), the problem becomes intractable for inputs with a large number of features. However, the authors provide an efficient way to compute an approximation of the Shapley value, which makes its computation tractable.

ZHAO and HASTIE (2019) study the importance of the features through a causal model, called Non-Parametric Structural Equation Model (NPSEM), and use Partial Dependence Plot to visualize the results. Thanks to this approach, they go further than only computing the impact of features on the prediction and manage to catch the tendency of a prediction according to the evolution of features.

**DNN specific global explanation.** Shrikumar *et al.* (2017) bring a new way of handling features importance with deeplift (Deep Learning Important FeaTures). The principle is to compare inputs to an input reference and outputs to an output reference. These references represent a kind of *neutral* behavior of the neural network. Since these references are considered as a baseline, the authors backpropagate the differences between the references and the actual sample through the network by comparing the activation values. Then, they define a contribution score system that derives the features importance from the differences. Hence, at the end of the backpropagation, we end up with the importance of each feature for a given prediction. Intuitively, one could say that the greater the difference, the greater its impact on the importance of the feature. The efficiency of the algorithm depends on the choice of references that relies on domain-specific knowledge.

In EASA (2021b) several objectives are dedicated to explainability. EASA highlights that the given explanation should match the audience; global explainability is related to the ML-based item itself and hence will be helpful for engineers during development and post-operation, while local explainability is related to the output of the ML-based item and hence will be helpful for any stakeholders (engineers, end users, authorities).

## 4.5 Conclusion

This chapter reviewed many methods in three domains: explainability, adversarial robustness, and provability. On the one hand, explainability will help increase confidence in such systems. On the other hand, adversarial robustness and provability tackle the verification of the safety requirements. According to the essential role of safety in the demonstration of conformity and the outcomes of our literature review, we focus our remaining research on adversarial robustness and provability. Part III presents our contributions on both domains.

# Part III

# Safety and Robustness

# 5 Adversarial Robustness Enhancement

**This chapter is based on the following paper**

Paul VIALLARD, Eric Guillaume VIDOT, Amaury HABRARD, and Emilie MORVANT,
Advances in Neural Information Processing Systems (NeurIPS 2021), pp. 14421–14433
A PAC-Bayes Analysis of Adversarial Robustness

## Contents

**Abstract**

We propose the first general PAC-Bayesian generalization bounds for adversarial robustness, that estimate, at test time, how much a model will be invariant to imperceptible perturbations in the input. Instead of deriving a worst-case analysis of the risk of a hypothesis over all the possible perturbations, we leverage the PAC-Bayesian framework to bound the averaged risk on the perturbations for majority votes (over the whole class of hypotheses). Our theoretically founded analysis has the advantage of providing general bounds *(i)* that are valid for any kind of attacks (*i.e.*, the adversarial attacks), *(ii)* that are tight thanks to the PAC-Bayesian framework, *(iii)* that can be directly minimized during the learning phase to obtain a robust model on different attacks at test time.

## 5.1 Introduction

**Context.** While ML algorithms are able to solve a huge variety of tasks, SZEGEDY *et al.* (2014) pointed out a crucial *weakness*: the adversarial example (see Figure 4.1). As already mentioned, adversarial examples contribute to the impossibility of ensuring the safety of machine learning algorithms for safety-critical applications such as aeronautic functions (*e.g.*, vision-based navigation), autonomous driving, or medical diagnosis (see, *e.g.*, HUANG *et al.* (2020)). Adversarial robustness is thus a critical issue in machine learning that studies the ability of a model to be robust or invariant to perturbations of its input. One line of research is referred to as adversarial robustness verification (*e.g.*, GEHR *et al.*, 2018; HUANG *et al.*, 2017; SINGH *et al.*, 2019b; TSUZUKU *et al.*, 2018), where the objective is to formally check whether the neighborhood of each sample does not contain any adversarial examples (see Section 4.3). This kind of method comes with some limitations, such as scalability or overapproximation (GEHR *et al.*, 2018; KATZ *et al.*, 2017; SINGH *et al.*, 2019b). In this chapter, we stand in another setting called adversarial attack/defense (*e.g.*, CARLINI and WAGNER, 2017; GOODFELLOW *et al.*, 2015; KURAKIN *et al.*, 2017b; MADRY *et al.*, 2018; PAPERNOT *et al.*, 2016; ZANTEDESCHI *et al.*, 2017). We recall that an adversarial attack consists in finding perturbed examples that defeat ML algorithms while the adversarial defense techniques enhance their adversarial robustness to make the attacks useless (see Section 4.2). While many methods exist, adversarial robustness suffers from a lack of general theoretical understandings (see Section 5.2).

**Motivation for the aeronautical domain.** Lately, the aeronautic industry considers lots of new applications using ML algorithms without the possibility of using them since the certification of ML-based systems is not yet achievable (see Chapter 3). Indeed, some of these applications would be embedded within safety-critical systems. We expressed our concern about the safety issue that this specific phenomenon may introduce when embedded in safety-critical ML-based systems. Actually, the robustness of the ML algorithms when operating in its usage domain is already identified as an objective by the EASA (see objective LM-12 from EASA, 2021b). This work is then relevant with respect to two points: *(i)* improving the robustness of ML algorithms, which increase the safety of the ML-based systems, and *(ii)* fulfilling some objective stated by the EASA regarding the certification of ML-based systems. Another objective stated in the same document regarding generalization guarantee (see objective LM-04 from EASA, 2021b) will be tackled at the same time with our contribution (see the *Contributions* paragraph).

**Contributions.** We propose to formulate the adversarial robustness through the lens of a well-founded statistical machine learning theory called PAC-Bayes and introduced by MCALLESTER (1999) and SHAWE-TAYLOR and WILLIAMSON (1997) (see Section 1.4 for detail on PAC-Bayes). This theory has the advantage of providing tight generalization bounds on average over the set of hypotheses considered (leading to bounds for a weighted majority vote over this set), in contrast to other theories, such as VC-dimension (VAPNIK and CHERVONENKIS, 1971) or Rademacher-based approaches

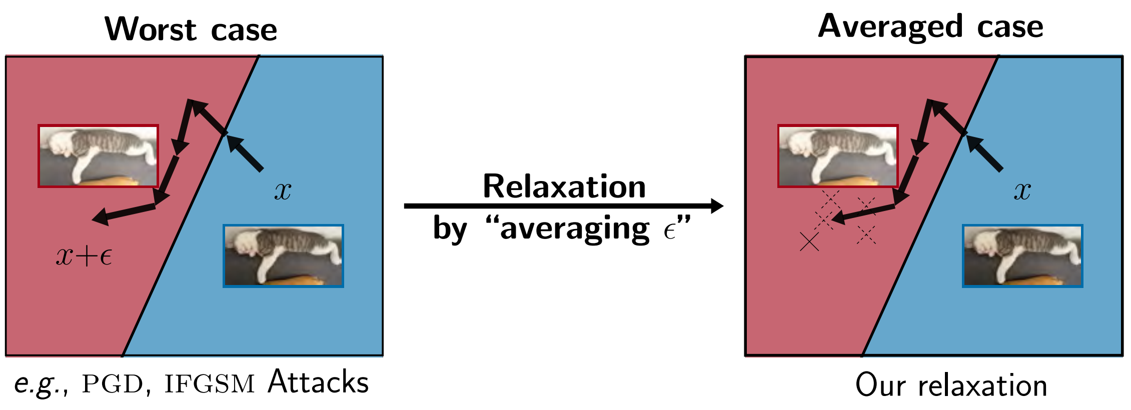

Figure 5.1: The left plot illustrates an example of a gradient-based method searching for an adversarial example (worst case) while the right plot shows our relaxation, which consists in considering the sampled (one or more) perturbations $\epsilon$ in average.

(Bartlett and Mendelson, 2003) that give worst-case analysis, *i.e.*, for all the hypotheses. We start by defining our setting called *adversarially robust PAC-Bayes*.

The idea consists in relaxing the worst-case adversarial risk as formulated in Definition 4.2.1. Figure 5.1 gives an intuition on what we call *relaxation*: The left plot shows what happens when using classical techniques, such as PGD or IFGSM, which approximate the perturbation $\epsilon$ that maximizes the loss Equation (4.3) (worst case), while the right plot explains how we relax the worst case. Instead of taking the $\epsilon$ maximizing the loss, we consider the perturbations $\epsilon$ in "average" using sampling. From this relaxed attack, we first derive the *averaged adversarial robustness risk* that corresponds to the probability that the model misclassifies a perturbed example (this can be seen as an averaged risk over the perturbations). This measure can be too optimistic and not informative enough since we sample only one perturbation for each example. Thus we also define an *averaged-max adversarial risk* as the probability that there exists at least one perturbation (taken in a set of sampled perturbations) that leads to misclassification.

Then, instead of applying the classical adversarial training as stated in Equation (4.6) we will minimize one of the relaxed risks, which is basically the average probability of being fooled by an adversarial example.

Finally, these definitions, based on averaged quantities, have the advantage *(i)* of still being suitable for the PAC-Bayesian framework and majority vote classifiers and *(ii)* to be related to the classical adversarial robustness risk (Definition 4.2.1). Then, for each of our adversarial risks, we derive a PAC-Bayesian generalization bound that are valid to any kind of attack. From an algorithmic point of view, these bounds can be directly minimized in order to learn a majority vote robust in average to attacks. We empirically illustrate that our framework is able to provide generalization guarantees with non-vacuous bounds for the adversarial risk while ensuring efficient protection against adversarial attacks.

**Organization of the chapter.** We start with the related work on generalization bound for adversarial robustness in Section 5.2. Then, we state our new adversarial robustness

PAC-Bayesian setting along with our theoretical results in Section 5.3 and we empirically show its soundness in Section 5.4. All the proofs of the results are deferred in Appendix.

## 5.2 Related works

We discuss in this section about the generalization bounds used for adversarial robustness. We refer the reader to Section 4.2 for details on adversarial robustness. However, note that contrary to the most popular techniques that look for a model with a low adversarial robust risk (Equation (4.5)), our work stands in another line of research where the idea is to relax this worst-case risk measure by considering an *averaged* adversarial robust risk over the noises instead of a $\max$-based formulation (see, *e.g.*, HENDRYCKS and DIETTERICH, 2019; ZANTEDESCHI *et al.*, 2017). Our averaged formulation is introduced in Section 5.3.

**Generalization Bounds.** Recently, few generalization bounds for adversarial robustness have been introduced (*e.g.* COHEN *et al.*, 2019; KHIM and LOH, 2018; MONTASSER *et al.*, 2019, 2020; SALMAN *et al.*, 2019a; YIN *et al.*, 2019). KHIM and LOH, and YIN *et al.*'s results are Rademacher complexity-based bounds. The former makes use of a surrogate of the adversarial risk; The latter provides bounds in the specific case of neural networks and linear classifiers and involves an unavoidable polynomial dependence on the input dimension. MONTASSER *et al.* study robust PAC-learning for PAC-learnable classes with finite VC-dimension for unweighted majority votes that have been "robustified" with a boosting algorithm. However, their algorithm requires to consider all possible adversarial perturbations for each example, which is intractable in practice, and their bound also suffers from a large constant as indicated at the end of the MONTASSER *et al.* (Theorem 3.1 2019)'s proof. COHEN *et al.* provide bounds that estimate the minimum noise to get an adversarial example (in the case of perturbations expressed as Gaussian noise) while our results give the probability of being fooled by an adversarial example. SALMAN *et al.* leverage COHEN *et al.*'s method and adversarial training in order to get tighter bounds. Moreover, FARNIA *et al.* present margin-based bounds on the adversarial robust risk for specific neural networks and attacks (such as FGSM or PGD). While they made use of a classical PAC-Bayes bound, their result is not a PAC-Bayesian analysis and stands in the family of uniform-convergence bounds, *i.e.*, when all the empirical risk of each hypothesis converge towards the true risk (see NAGARAJAN and KOLTER, 2019, Ap. J for details). In this chapter, we provide PAC-Bayes bounds for general models expressed as majority votes, their bounds are thus not directly comparable to ours.

## 5.3 Adversarially robust PAC-Bayes

### Setting

We tackle binary classification tasks with the input space $\mathbb{X} = \mathbb{R}^d$ and the output/label space $\mathbb{Y} = \{-1, +1\}$. We assume that $\mathcal{D}$ is a fixed but unknown distribution on $\mathbb{X}\times\mathbb{Y}$. An example is denoted by $(x,y)\in\mathbb{X}\times\mathbb{Y}$. Let $S=\{(x_i,y_i)\}_{i=1}^m$ be the learning sample consisted of $m$ examples *i.i.d.* from $\mathcal{D}$; We denote the distribution of such $m$-sample by $\mathcal{D}^m$. Let $\mathcal{H}$ be a set of real-valued functions from $\mathbb{X}$ to $[-1,+1]$ called voters or hypotheses. Usually, one follows the ERM principle as defined in Definition 1.3.3 to find the best model, which consists in minimizing the empirical risk (see Definition 1.3.2) of a model $h\in\mathcal{H}$.

On the one hand, considering the PAC-Bayesian framework, our goal is not anymore to learn one classifier from $\mathcal{H}$ but to learn a well-performing $\mathcal{Q}$-weighted majority vote $\mathbf{B}_{\mathcal{Q}}$ — or Bayes classifier — as defined in Definition 1.4.1 where the posterior distribution $\mathcal{Q}$ is learned from $S$ given a prior distribution $\mathcal{P}$ on $\mathcal{H}$. We consider the 0-1 loss function classically used for majority votes in PAC-Bayes defined in Equation (1.5) and its surrogate linear loss defined in Equation (1.6).

On the other hand, we are tackling adversarial robustness (see Section 4.2): an imperceptible perturbation of the input (*e.g.*, due to a malicious attack or a noise) can have a bad influence on the classification performance on unseen data (Szegedy *et al.*, 2014) and the usual guarantees do not stand anymore. Such imperceptible perturbation can be modeled by a (relatively small) noise in the input. We define the set of possible noises as

**Definition 5.3.1.** *Set of possible noises*
Let $b>0$ and $\|\cdot\|$ be an arbitrary norm (the most used norms are the $L_1$, $L_2$ and $L_\infty$-norms), the set of possible noises $\mathbb{B}$ is defined as

$$\mathbb{B}=\left\{\epsilon\in\mathbb{R}^d \,\middle|\, \|\epsilon\|\le b\right\}.$$

With this new formulation of the noises, we can adapt Definition 4.2.1, we have

$$\mathcal{R}_{\mathcal{D}}^{\mathtt{ROB}}(h)=\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\ \max_{\epsilon\in\mathbb{B}}\ \ell\left(h,(x+\epsilon,y)\right). \tag{5.1}$$

## Adversarially robust majority vote

First, when studying the adversarial robustness of the weighted majority vote, the adversarial perturbation related to Equation (4.3) becomes

$$\epsilon^*(x,y)\ \in\ \mathop{\mathrm{argmax}}_{\epsilon\in\mathbb{B}}\ \mathbf{I}(\mathbf{B}_{\mathcal{Q}}(x+\epsilon)\neq y). \tag{5.2}$$

Optimizing this problem is intractable due to the non-convexity of $\mathbf{B}_{\mathcal{Q}}$ induced by the sign function. The adversarial attacks of the literature (like PGD or IFGSM) aim at finding the optimal perturbation $\epsilon^*(x,y)$, but, in practice, one considers an approximation of this perturbation. If necessary, the optimal perturbation can be found by using formal methods but at the price of a longer running time.

Hence, instead of searching for the noise that maximizes the chance of fooling the algorithm, we propose to model the perturbation according to an example-dependent distribution. First, let us define $\omega_{(x,y)}$, a distribution on the set of possible noises $\mathbb{B}$, that is dependent on an example $(x, y) \in \mathbb{X}{\times}\mathbb{Y}$. Then we denote as $\mathfrak{D}$ the distribution on $(\mathbb{X}{\times}\mathbb{Y}) \times \mathbb{B}$ defined as

$$\mathfrak{D}((x, y), \epsilon) = \mathcal{D}(x, y) \cdot \omega_{(x,y)}(\epsilon),$$

which further permits to generate *perturbed examples*. To estimate our risks (defined below) for a given example $(x_i, y_i){\sim}\mathcal{D}$, we consider a set of $n$ perturbations sampled from $\omega_{(x_i,y_i)}$ denoted by $\mathbb{e}_i = \{\epsilon^i_j\}^n_{j=1}$. Then we consider as a learning set the $m{\times}n$-sample $\mathbf{S} = \{((x_i, y_i), \mathbb{e}_i)\}^m_{i=1} \in (\mathbb{X}{\times}\mathbb{Y}{\times}\mathbb{B}^n)^m$. In other words, each $((x_i, y_i), \mathbb{e}_i) \in \mathbf{S}$ is sampled from a distribution that we denote by $\mathfrak{D}^n$ such that

$$\mathfrak{D}^n((x_i, y_i), \mathbb{e}_i) = \mathcal{D}(x_i, y_i) \cdot \prod_{j=1}^{n} \omega_{(x_i,y_i)}(\epsilon^i_j).$$

Then, inspired by the works of Hendrycks and Dietterich (2019) and Zantedeschi *et al.* (2017), we define our *robustness averaged adversarial risk* as follows.

**Definition 5.3.2.** *Averaged Adversarial Risk*
For any distribution $\mathfrak{D}$ on $(\mathbb{X}{\times}\mathbb{Y}) \times \mathbb{B}$, for any distribution $\mathcal{Q}$ on $\mathcal{H}$, the averaged adversarial risk of $\mathbf{B}_\mathcal{Q}$ is defined as

$$\begin{aligned}\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q}) &= \Pr_{((x,y),\epsilon)\sim\mathfrak{D}} (\mathbf{B}_\mathcal{Q}(x + \epsilon) \neq y) \\ &= \mathop{\mathbb{E}}_{((x,y),\epsilon)\sim\mathfrak{D}} \mathbf{I}(\mathbf{B}_\mathcal{Q}(x + \epsilon) \neq y).\end{aligned}$$

The empirical averaged adversarial risk computed on a $m{\times}n$-sample $\mathbf{S} = \{((x_i, y_i), \mathbb{e}_i)\}^m_{i=1}$ is

$$\mathcal{R}_{\mathbf{S}}(\mathbf{B}_\mathcal{Q}) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} \mathbf{I}(\mathbf{B}_\mathcal{Q}(x_i{+}\epsilon^i_j) \neq y_i).$$

As we will show in Proposition 5.3.1, the risk $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q})$ can be considered optimistic regarding $\epsilon^*(x, y)$ of Equation (5.2). Indeed, instead of taking the $\epsilon$ maximizing the loss, a unique $\epsilon$ is drawn from a distribution. Hence, it can lead to a non-informative risk regarding the occurrence of adversarial examples. To overcome this, we propose an extension that we refer as *averaged-max adversarial risk*.

**Definition 5.3.3.** *Averaged-Max Adversarial Risk*
For any distribution $\mathfrak{D}$ on $(\mathbb{X}{\times}\mathbb{Y}) \times \mathbb{B}$, for any distribution $\mathcal{Q}$ on $\mathcal{H}$, the averaged-max adversarial risk of $\mathbf{B}_\mathcal{Q}$ is defined as

$$\mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_\mathcal{Q}) = \Pr_{((x,y),\mathbb{e})\sim\mathfrak{D}^n} \big(\exists \epsilon \in \mathbb{e}, \mathbf{B}_\mathcal{Q}(x + \epsilon) \neq y\big).$$

The empirical averaged-max adversarial risk computed on a $m{\times}n$-sample $\mathbf{S} = \{((x_i, y_i), \mathbb{e}_i)\}^m_{i=1}$ is

$$\mathcal{A}_{\mathbf{S}}(\mathbf{B}_\mathcal{Q}) = \frac{1}{m}\sum_{i=1}^{m} \max_{\epsilon\in\mathbb{e}_i} \mathbf{I}(\mathbf{B}_\mathcal{Q}(x_i + \epsilon) \neq y_i).$$

For an example $(x, y) \sim \mathcal{D}$, instead of checking if one perturbed example $x + \epsilon$ is adversarial, we sample $n$ perturbed examples $x + \epsilon_1, \ldots, x + \epsilon_n$ and we check if at least one example is adversarial.

## Relations between the adversarial risks

Proposition 5.3.1 below shows the intrinsic relationships between the classical adversarial risk $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}})$ and our two relaxations $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$ and $\mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_{\mathcal{Q}})$. In particular, Proposition 5.3.1 shows that the larger $n$, the number of perturbed examples, the higher is the chance to get an adversarial example and then to be close to the adversarial risk $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}})$.

**Proposition 5.3.1.**
For any distribution $\mathfrak{D}$ on $(\mathbb{X}{\times}\mathbb{Y}) \times \mathbb{B}$, for any distribution $\mathcal{Q}$ on $\mathcal{H}$, for any $(n, n') \in \mathbb{N}^2$, with $n \geq n' \geq 1$, we have

$$\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}) \;\leq\; \mathcal{A}_{\mathfrak{D}^{n'}}(\mathbf{B}_{\mathcal{Q}}) \;\leq\; \mathcal{A}_{\mathfrak{D}^{n}}(\mathbf{B}_{\mathcal{Q}}) \;\leq\; \mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}}). \tag{5.3}$$

The left-hand side of Equation (5.3) confirms that the averaged adversarial risk $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$ is optimistic regarding the classical $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}})$. Proposition 5.3.2 estimates how close $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$ can be to $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}})$.

**Proposition 5.3.2.** For any distribution $\mathfrak{D}$ on $(\mathbb{X}{\times}\mathbb{Y}) \times \mathbb{B}$, for any distribution $\mathcal{Q}$ on $\mathcal{H}$, we have

$$\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}}) - \mathrm{TV}(\Pi \| \Delta) \;\leq\; \mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}),$$

where $\Delta$ and $\Pi$ are distributions on $\mathbb{X}{\times}\mathbb{Y}$. $\Delta(x', y')$ corresponds to the probability of drawing a perturbed example $(x + \epsilon)$ with $((x, y), \epsilon) \sim \mathfrak{D}$ and is defined as

$$\Delta(x', y') = \Pr_{((x,y),\epsilon)\sim\mathfrak{D}} \left[x + \epsilon = x', y = y'\right] \tag{5.4}$$

$\Pi(x', y')$ corresponds to the probability of drawing an adversarial example $(x{+}\epsilon^*(x{,}y), y)$ with $(x, y) \sim \mathcal{D}$ and is defined as

$$\Pi(x', y') = \Pr_{(x,y)\sim\mathcal{D}} \left[x + \epsilon^*(x, y) = x', y = y'\right], \tag{5.5}$$

and $\mathrm{TV}(\Pi\|\Delta) = \mathop{\mathbb{E}}_{(x',y')\sim\Delta} \frac{1}{2}\left|\frac{\Pi(x', y')}{\Delta(x', y')} - 1\right|$, is the Total Variation (TV) distance between $\Pi$ and $\Delta$.

Note that $\epsilon^*(x{,}y)$ depends on $\mathcal{Q}$, and hence $\Pi$ depends on $\mathcal{Q}$. From Equations (5.4) and (5.5), $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}})$ and $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$ can be rewritten (see Lemmas A.2.1 and A.2.2 in Appendix A.2) respectively with $\Pi$ and $\Delta$ as

$$\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}) = \Pr_{(x',y')\sim\Delta} \left[\mathbf{B}_{\mathcal{Q}}(x') \neq y'\right], \quad \text{and} \quad \mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}}) = \Pr_{(x',y')\sim\Pi} \left[\mathbf{B}_{\mathcal{Q}}(x') \neq y'\right].$$

Finally, Propositions 5.3.1 and 5.3.2 relate the adversarial risk $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$ to the "standard" adversarial risk $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}})$. Indeed, by merging the two propositions, we obtain

$$\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}}) - \mathrm{TV}(\Pi\|\Delta) \;\leq\; \mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}) \;\leq\; \mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_{\mathcal{Q}}) \;\leq\; \mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}}). \tag{5.6}$$

Hence, the smaller the TV distance $\mathrm{TV}(\Pi\|\Delta)$, the closer the averaged adversarial risk $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$ is from $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}})$ and the more probable an example $((x,y),\epsilon)$ sampled from $\mathfrak{D}$ would be adversarial, *i.e.*, when our "averaged" adversarial example looks like a "specific" adversarial example. Moreover, Equation (5.6) justifies that the PAC-Bayesian point of view makes sense for adversarial learning with theoretical guarantees: the PAC-Bayesian guarantees we derive in the next section for our adversarial risks also give some guarantees on the "standard risk" $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}})$.

## PAC-Bayesian bounds on the adversarially robust majority vote

First of all, since $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$ and $\mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_{\mathcal{Q}})$ risks are not differentiable due to the indicator function, we propose to use a common surrogate in PAC-Bayes (known as the Gibbs risk defined in Definition 1.4.3): instead of considering the risk of the $\mathcal{Q}$-weighted majority vote, we consider the expectation over $\mathcal{Q}$ of the individual risks of the voters involved in $\mathcal{H}$. In our case, we define the surrogates with the linear loss as

$$\begin{aligned} \overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{G}_{\mathcal{Q}}) &= \mathop{\mathbb{E}}_{((x,y),\epsilon)\sim\mathfrak{D}} \frac{1}{2}\left[1 - y \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x+\epsilon)\right], \\ \text{and} \quad \overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{G}_{\mathcal{Q}}) &= \mathop{\mathbb{E}}_{((x,y),\mathbb{e})\sim\mathfrak{D}^n} \frac{1}{2}\left[1 - \min_{\epsilon\in\mathbb{e}}\left(y \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x+\epsilon)\right)\right]. \end{aligned}$$

The next theorem relates these surrogates to our risks, implying that a generalization bound for $\overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{G}_{\mathcal{Q}})$, respectively for $\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{G}_{\mathcal{Q}})$, leads to a generalization bound for $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$, respectively $\mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_{\mathcal{Q}})$.

**Theorem 5.3.1.**
For any distributions $\mathfrak{D}$ on $(\mathbb{X}\times\mathbb{Y})\times\mathbb{B}$ and $\mathcal{Q}$ on $\mathcal{H}$, for any $n>1$, we have

$$\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}) \leq 2\overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{G}_{\mathcal{Q}}), \qquad \text{and} \qquad \mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_{\mathcal{Q}}) \leq 2\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{G}_{\mathcal{Q}}).$$

Theorem 5.3.2 below presents our PAC-Bayesian generalization bounds for $\overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{G}_{\mathcal{Q}})$. Before that, it is important to mention that the empirical counterpart of $\overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{G}_{\mathcal{Q}})$ is computed on $\mathbf{S}$, which is composed of non identically independently distributed samples, meaning that a "classical" proof technique is not applicable. The trick here is to make use of a result of Ralaivola *et al.* (2010) that provides a *chromatic PAC-Bayes bound*, *i.e.*, a bound which supports non-independent data.

**Theorem 5.3.2.**
For any distribution $\mathfrak{D}$ on $(\mathbb{X}\times\mathbb{Y})\times\mathbb{B}$, for any set of voters $\mathcal{H}$, for any prior $\mathcal{P}$ on

$\mathcal{H}$, for any $n$, with probability at least $1-\delta$ over $\mathbf{S}$, for all posteriors $\mathcal{Q}$ on $\mathcal{H}$, we have

$$\mathrm{kl}(\overline{\mathcal{R}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}})\|\overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{G}_{\mathcal{Q}})) \le \frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) + \ln\frac{m+1}{\delta}\right], \tag{5.7}$$

$$\text{and}\quad \overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{G}_{\mathcal{Q}}) \le \overline{\mathcal{R}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}}) + \sqrt{\frac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) + \ln\frac{m+1}{\delta}\right]}, \tag{5.8}$$

$$\text{where}\quad \overline{\mathcal{R}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}}) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{1}{2}\left[1 - y_i \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x_i+\epsilon_j^i)\right],$$

$\mathrm{kl}(a\|b) = a\ln\frac{a}{b} + (1-a)\ln\frac{1-a}{1-b}$, and $\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) = \mathop{\mathbb{E}}_{h\sim\mathcal{P}}\ln\frac{\mathcal{P}(h)}{\mathcal{Q}(h)}$ the KL-divergence between $\mathcal{P}$ and $\mathcal{Q}$.

Surprisingly, this theorem states bounds that do not depend on the number of perturbed examples $n$ but only on the number of original examples $m$. The reason is that the $n$ perturbed examples are inter-dependent (see the proof in Appendix A.4). Note that Equation (5.7) is expressed as a Seeger (2002)'s bound and is tighter but less interpretable than Equation (5.8) expressed as a McAllester (1999)'s bound; These bounds involve the usual trade-off between the empirical risk $\overline{\mathcal{R}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}})$ and $\mathrm{KL}(\mathcal{Q}\|\mathcal{P})$.

We now state a generalization bound for $\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{G}_{\mathcal{Q}})$. Since this value involves a minimum term, we cannot use the same trick as for Theorem 5.3.2. To bypass this issue, we use the TV distance between two "artificial" distributions on $\mathbb{e}_i$. Given $((x_i,y_i),\mathbb{e}_i)\in\mathbf{S}$, let $\pi_i$ be an arbitrary distribution on $\mathbb{e}_i$, and given $h\in\mathcal{H}$, let $\rho_i^h$ be a Dirac distribution on $\mathbb{e}_i$ such that $\rho_i^h(\epsilon)=1$ if $\epsilon = \mathrm{argmax}_{\epsilon\in\mathbb{e}_i}\frac{1}{2}\big[1-y_ih(x_i+\epsilon)\big]$ (*i.e.*, if $\epsilon$ is maximizing the linear loss), and $0$ otherwise.

**Theorem 5.3.3.**
For any distribution $\mathfrak{D}$ on $(\mathbb{X}\times\mathbb{Y})\times\mathbb{B}$, for any set of voters $\mathcal{H}$, for any prior $\mathcal{P}$ on $\mathcal{H}$, for any $n$, with probability at least $1-\delta$ over $\mathbf{S}$, for all posteriors $\mathcal{Q}$ on $\mathcal{H}$, for all $i\in\{1,\ldots,m\}$, for all distributions $\pi_i$ on $\mathbb{e}_i$ independent from a voter $h\in\mathcal{H}$, we have

$$\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{G}_{\mathcal{Q}}) \le \frac{1}{m}\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\sum_{i=1}^{m}\max_{\epsilon\in\mathbb{e}_i}\frac{1}{2}\left(1-y_ih(x_i+\epsilon)\right) + \sqrt{\tfrac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) + \ln\tfrac{2\sqrt{m}}{\delta}\right]} \tag{5.9}$$

$$\le \overline{\mathcal{A}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}}) + \frac{1}{m}\sum_{i=1}^{m}\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\mathrm{TV}(\rho_i^h\|\pi_i) + \sqrt{\tfrac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) + \ln\tfrac{2\sqrt{m}}{\delta}\right]}, \tag{5.10}$$

$$\text{where}\ \overline{\mathcal{A}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}}) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\Big[1-\min_{\epsilon\in\mathbb{e}_i}\Big(y_i\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}h(x_i+\epsilon)\Big)\Big],$$

$$\text{and}\ \ \mathrm{TV}(\rho\|\pi) = \mathop{\mathbb{E}}_{\epsilon\sim\pi}\frac{1}{2}\left|\left[\frac{\rho(\epsilon)}{\pi(\epsilon)}\right]-1\right|.$$

To minimize the true average-max risk $\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{G}_{\mathcal{Q}})$ from Equation (5.9), we have to minimize a trade-off between $\mathrm{KL}(\mathcal{Q}\|\mathcal{P})$ (*i.e.*, how much the posterior weights are close to the prior ones) and the empirical risk $\frac{1}{m}\mathbb{E}_{h\sim\mathcal{Q}}\sum_{i=1}^{m}\max_{\epsilon\in\mathbb{e}_i}\frac{1}{2}\left(1-y_i h(x_i+\epsilon)\right)$. However, to compute the empirical risk, the loss for each voter and each perturbation has to be calculated and can be time-consuming. With Equation (5.10), we propose an alternative, which can be efficiently optimized using $\frac{1}{m}\sum_{i=1}^{m}\mathbb{E}_{h\sim\mathcal{Q}}\mathrm{TV}(\rho_i^h\|\pi_i)$ and the empirical average-max risk $\overline{\mathcal{A}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}})$. Intuitively, Equation (5.10) can be seen as a trade-off between the empirical risk, which reflects the robustness of the majority vote, and two penalization terms: the KL term and the TV term. The KL-divergence $\mathrm{KL}(\mathcal{Q}\|\mathcal{P})$ controls how much the posterior $\mathcal{Q}$ can differ from the prior ones $\mathcal{P}$. While the TV term $\mathbb{E}_{h\sim\mathcal{Q}}\mathrm{TV}(\rho_i^h\|\pi_i)$ controls the diversity of the voters, *i.e.*, the ability of the voters to be fooled on the same adversarial example. From an algorithmic view, an interesting behavior is that the bound of Equation (5.10) stands for all distributions $\pi_i$ on $\mathbb{e}_i$. This suggests that given $(x_i, y_i)$, we want to find $\pi_i$ minimizing $\mathbb{E}_{h\sim\mathcal{Q}}\mathrm{TV}(\rho_i^h\|\pi_i)$. Ideally, this term tends to $0$ when $\pi_i$ is close to $\rho_i^h$ and all voters have their loss maximized by the same perturbation $\epsilon\in\mathbb{e}_i$. Note that, since $\rho_i^h$ is a Dirac distribution, we have $\mathbb{E}_{h\sim\mathcal{Q}}\mathrm{TV}(\rho_i^h\|\pi_i)=\frac{1}{2}\Big[1-\mathbb{E}_{h\sim\mathcal{Q}}\pi_i(\epsilon_h^*)+\mathbb{E}_{h\sim\mathcal{Q}}\sum_{\epsilon\neq\epsilon_h^*}\pi_i(\epsilon)\Big]$, with $\epsilon_h^*=\mathrm{argmax}_{\epsilon\in\mathbb{e}_i}\frac{1}{2}\big[1-y_i h(x_i+\epsilon)\big]$.

To learn a well-performing majority vote, one solution is to minimize the right-hand side of the bounds, meaning that we would like to find a good trade-off between a low empirical risk $\overline{\mathcal{R}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}})$ or $\overline{\mathcal{A}_{\mathbf{S}}}(\mathbf{G}_{\mathcal{Q}})$ and a low divergence between the prior weights and the learned posterior ones $\mathrm{KL}(\mathcal{Q}\|\mathcal{P})$.

## 5.4 Experimental evaluation

In this section, we illustrate the soundness of our framework in the context of differentiable decision trees learning. First of all, we describe our learning procedure designed from our theoretical results.

### Differentiable decision trees

In this section, we introduce the differentiable decision trees, *i.e.*, the voters of our majority vote. Note that we adapt the model of KONTSCHIEDER *et al.* (2016) in order to fit with our framework: a voter must output a real between $-1$ and $+1$. An example of such a tree is represented in Figure 5.2.

This differentiable decision tree is stochastic by nature: at each node $i$ of the tree, we continue recursively to the left sub-tree with a probability of $p_i(x)$ and to the right sub-tree with a probability of $1-p_i(x)$; When we attain a leaf $j$, the tree predicts the label $s_j$. Precisely, the probability $p_i(x)$ is constructed by *(i)* selecting randomly 50% of the input features $x$ and applying a random mask $M_i\in\mathbb{R}^d$ on $x$ (where the $k$-th entry of the mask is $1$ if the $k$-th feature is selected and $0$ otherwise), by *(ii)* multiplying this quantity by a learned weight vector $v_i\in\mathbb{R}^d$, and by *(iii)* applying a sigmoid function to output a probability. Indeed, we have

$$p_i(x)=\sigma\Big(\langle v_i, M_i\odot x\rangle\Big),$$

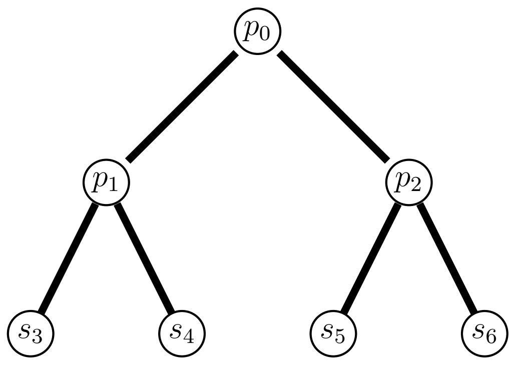

Figure 5.2: Representation of a (differentiable) decision tree of depth $l = 2$; The root is the node 0, and the leaves are 3; 4; 5 and 6. The probability $p_i(x)$ (respectively $1-p_i(x)$) to go left (respectively right) at the node $i$ is represented by $p_i$ (we omitted the dependence on $x$ for simplicity). Similarly, the predicted label (a "score" between $-1$ and $+1$) at the leaf $i$ is represented by $s_i$.

where $\sigma(a) = [1+e^{-a}]^{-1}$ is the sigmoid function; $\langle a, b\rangle$ is the dot product between the vector $a$ and $b$ and $a \odot b$ is the elementwise product between the vector $a$ and $b$. Moreover, $s_i$ is obtained by learning a parameter $u_i \in \mathbb{R}$ and applying a $\tanh$ function, *i.e.*, we have

$$s_i = \tanh\Big(u_i\Big).$$

Finally, instead of having a stochastic voter, $h$ will output the expected label predicted by the tree (see Kontschieder *et al.* (2016) for more details). It can be computed by $h(x) = f(x, 0, 0)$ with

$$f(x,i,l') = \begin{cases} s_i & \text{if } l' = l \\ p_i(x)f(x, 2i+1, l'+1) + (1-p_i(x))f(x, 2i+2, l'+1) & \text{otherwise} \end{cases}.$$

## From the bounds to an algorithm

We consider a finite voters set $\mathcal{H}$ consisting of differentiable decision trees (Kontschieder *et al.*, 2016) where each $h \in \mathcal{H}$ is parametrized by a weight vector $\boldsymbol{w}^h$. Inspired by Masegosa *et al.*, 2020c, we learn the decision trees of $\mathcal{H}$ and a data-dependent prior distribution $\mathcal{P}$ from a first learning set $\mathcal{S}'$ (independent from $\mathcal{S}$); This is a common approach in PAC-Bayes (Dziugaite *et al.*, 2021; Dziugaite and Roy, 2018; Lever *et al.*, 2013; Parrado-Hernández *et al.*, 2012). Then, the posterior distribution is learned from the second learning set $\mathcal{S}$ by minimizing the bounds. This means we need to minimize the risk and the KL-divergence term. Our two-step learning procedure is summarized in Algorithm 2.

**Step 1.** Starting from an initial prior $\mathcal{P}_0$ and an initial set of voters $\mathcal{H}_0$, where each voter $h$ is parametrized by a weight vector $\boldsymbol{w}_0^h$, the objective of this step is to construct the hypothesis set $\mathcal{H}$ and the prior distribution $\mathcal{P}$ to give as input to Step 2 for minimizing the bound. To do so, at each epoch $t$ of the Step 1, we learn from $\mathcal{S}'$ an "intermediate" prior $\mathcal{P}_t$ on an "intermediate" hypothesis set $\mathcal{H}_t$ consisting of voters $h$

parametrized by the weights $\boldsymbol{w}_t^h$; Note that the optimization in Line 9 is done with respect to $\boldsymbol{w}_t{=}\{\boldsymbol{w}_t^h\}_{h\in\mathcal{H}_t}$. At each iteration of the optimizer, from Lines 4 to 7, for each $(x,y)$ of the current batch $\mathbb{S}'$, we attack the majority vote $\mathbf{B}_{\mathcal{P}_t}$ to obtain a perturbed example $x{+}\epsilon$. Then, in Lines 8 and 9, we perform a forward pass in the majority vote with the perturbed examples and update the weights $\boldsymbol{w}_t$ and the prior $\mathcal{P}_t$ according to the linear loss. To sum up, from Lines 11 to 20 at the end of Step 1, the prior $\mathcal{P}$ and the hypothesis set $\mathcal{H}$ constructed for Step 2 are the ones associated to the best epoch $t^* \in \{1,\ldots,T'\}$ that permits to minimize $\overline{\mathcal{R}_{\mathcal{S}_t}}(\mathbf{G}_{\mathcal{P}_t})$, where $\mathcal{S}_t{=}\{\texttt{attack}(x,y) \,|\, (x,y)\in\mathcal{S}\}$ is the perturbed set obtained by attacking the majority vote $\mathbf{B}_{\mathcal{P}_t}$.

**Step 2.** Starting from the prior $\mathcal{P}$ on $\mathcal{H}$ and the learning set $\mathcal{S}$, we perform the same process as in Step 1 except that the considered objective function corresponds to the desired bound to optimize (Line 30, denoted $\mathtt{B}(\cdot)$). Unlike Equation (5.9) or Equation (5.10), Equation (5.7) is not directly optimizable since we upper-bound a deviation (the $\mathrm{kl}$) between the empirical and true risk. Hopefully, we can compute the bound when it is expressed with the inverse binary $\mathrm{kl}$ divergence $\mathrm{kl}^{-1}$ defined as $\mathrm{kl}^{-1}(a|\epsilon) = \max_{b\in[0,1]}\{\mathrm{kl}(a\|b)\le\epsilon\}$. Equation (5.7) can be rewritten as

$$\overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{B}_{\mathcal{Q}}) \le \mathrm{kl}^{-1}\!\left(\overline{\mathcal{R}_{\mathbf{S}}}(\mathbf{B}_{\mathcal{Q}}) \,\middle|\, \frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}){+}\ln\frac{T(m{+}1)}{\delta}\right]\right).$$

The objective function associated to Equation (5.7) of Theorem 5.3.2 is

$$\mathtt{B}_{\mathbb{S}}(\mathbf{B}_{\mathcal{Q}_t}) = \mathrm{kl}^{-1}\left(\overline{\mathcal{R}_{\mathbb{S}}}(\mathbf{B}_{\mathcal{Q}_t}) \,\middle|\, \frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}_t\|\mathcal{P}) + \ln\frac{T(m+1)}{\delta}\right]\right).$$

Note that the derivative of $\mathrm{kl}^{-1}$ and its computation can be found in Reeb *et al.* (2018, Appendix A). On the other hand, the objective function to optimize Equation (5.10) of Theorem 5.3.3 is defined as

$$\mathtt{B}_{\mathbb{S}}(\mathbf{B}_{\mathcal{Q}_t}) = \overline{\mathcal{A}_{\mathbb{S}}}(\mathbf{B}_{\mathcal{Q}_t}) + \sqrt{\frac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}_t\|\mathcal{P}) + \ln\tfrac{2T\sqrt{m}}{\delta}\right]}.$$

Note that the $\mathrm{TV}$ distance is $0$ when we sample one noise for each example, *i.e.*, when $n=1$. Consequently, the distance is not optimized in Algorithm 2. However, if we had $n>1$, we would have to minimize it. Note that the "intermediate" priors do not depend on $\mathcal{S}$, since they are learned from $\mathcal{S}'$: the bounds are then valid.

**Attacking the examples.** The `attack` function used in Algorithm 2 differs from the attack that generates the perturbed set $\mathbf{S}$ (for the bound). Indeed, at each iteration (in both steps), the function attacks an example with the current model while $\mathbf{S}$ is generated with the prior majority vote $\mathbf{B}_{\mathcal{P}}$ (the output of Step 1). Note that for all attacks, in order to be differentiable with respect to the input, we remove the $\mathrm{sign}$ function on the voters' outputs.

## Experiments

In this section, we empirically illustrate that our PAC-Bayesian framework for adversarial robustness is able to provide generalization guarantees with non-vacuous bounds for

**Algorithm 2** Average Adversarial Training with Guarantee

**Require:** $\mathcal{S}, \mathcal{S}'$: disjoint learning sets – $T, T'$: number of epochs – $\mathcal{P}_0$: initial prior – $\mathcal{H}_0$ (with $\boldsymbol{w}_0$): initial hypothesis set – `attack()`: the attack function – $\mathtt{B}(\cdot)$: the objective function associated to a bound

**Step 1** – prior and voters' set construction

```
 1: for $t$ from $1$ to $T'$ do
 2:   $\mathcal{P}_t \leftarrow \mathcal{P}_{t-1}$ and $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1}$ $(\boldsymbol{w}_t \leftarrow \boldsymbol{w}_{t-1})$
 3:   for all batches $\mathbb{S}'$ (from $\mathcal{S}'$) do
 4:     for all $(x,y) \in \mathbb{S}'$ do
 5:       $(x+\epsilon, y) \leftarrow \mathtt{attack}(x,y)$
 6:       $\mathbb{S}' \leftarrow (\mathbb{S}' \setminus \{(x,y)\}) \cup \{(x+\epsilon, y)\}$
 7:     end for
 8:     Update $\mathcal{P}_t$ with $\nabla_{\mathcal{P}_t} \overline{\mathcal{R}_{\mathbb{S}'}}(\mathbf{G}_{\mathcal{P}_t})$
 9:     Update $\boldsymbol{w}_t$ with $\nabla_{\boldsymbol{w}_t} \overline{\mathcal{R}_{\mathbb{S}'}}(\mathbf{G}_{\mathcal{P}_t})$
10:   end for
11:   $\mathcal{S}_t \leftarrow \emptyset$
12:   for all $(x,y) \in \mathcal{S}$ do
13:     $(x+\epsilon, y) \leftarrow \mathtt{attack}(x,y)$
14:     $\mathcal{S}_t \leftarrow \mathcal{S}_t \cup \{(x+\epsilon, y)\}$
15:   end for
16:   $t^* \leftarrow \operatorname{argmin}_{t' \in \{1,\dots,t\}} \overline{\mathcal{R}_{\mathcal{S}_{t'}}}(\mathbf{G}_{\mathcal{P}_{t'}})$
17:   $\mathcal{P} \leftarrow \mathcal{P}_{t^*}$
18:   $\mathcal{H} \leftarrow \mathcal{H}_{t^*}$
19: end for
20: return $(\mathcal{P}, \mathcal{H})$
```

**Step 2** – bound minimization

```
21: $(\mathcal{P}, \mathcal{H}) \leftarrow$ Output of Step 1
22: $\mathcal{Q}_0 \leftarrow \mathcal{P}$
23: for $t$ from $1$ to $T$ do
24:   for all batches $\mathbb{S}$ (from $\mathcal{S}$) do
25:     $\mathcal{Q}_t \leftarrow \mathcal{Q}_{t-1}$
26:     for all $(x,y) \in \mathbb{S}$ do
27:       $(x+\epsilon, y) \leftarrow \mathtt{attack}(x,y)$
28:       $\mathbb{S} \leftarrow (\mathbb{S} \setminus \{(x,y)\}) \cup \{(x+\epsilon, y)\}$
29:     end for
30:     Update $\mathcal{Q}_t$ with $\nabla_{\mathcal{Q}_t} \mathtt{B}_{\mathbb{S}}(\mathbf{G}_{\mathcal{Q}_t})$
31:   end for
32:   $\mathcal{S}_t \leftarrow \emptyset$
33:   for all $(x,y) \in \mathcal{S}$ do
34:     $(x+\epsilon, y) \leftarrow \mathtt{attack}(x,y)$
35:     $\mathcal{S}_t \leftarrow \mathcal{S}_t \cup \{(x+\epsilon, y)\}$
36:   end for
37:   $t^* \leftarrow \operatorname{argmin}_{t' \in \{1,\dots,t\}} \mathtt{B}_{\mathcal{S}_{t'}}(\mathbf{G}_{\mathcal{Q}_{t'}})$
38:   $\mathcal{Q} \leftarrow \mathcal{Q}_{t^*}$
39: end for
40: return $(\mathcal{Q}, \mathcal{H})$
```

the adversarial risk. The source code of the experiments is available at `https://github.com/paulviallard/NeurIPS21-PB-Robustness`.

**Setting.** We stand in a white-box setting, meaning that the attacker knows the voters set $\mathcal{H}$, the prior distribution $\mathcal{P}$, and the posterior one $\mathcal{Q}$. We empirically study $2$ attacks with the $L_2$-norm and $L_\infty$-norm: the Projected Gradient Descent (PGD, MADRY *et al.* (2018)) and the iterative version of FGSM (IFGSM, KURAKIN *et al.* (2017b)). We fix the number of iterations at $k = 20$ and the step size at $\frac{b}{k}$ for PGD and IFGSM (where $b = 1$ for $L_2$-norm and $b = 0.1$ for $L_\infty$-norm). One specificity of our setting is that we deal with the perturbation distribution $\omega_{(x,y)}$. We propose PGD$_\text{U}$ and IFGSM$_\text{U}$, two variants of PGD and IFGSM. To attack an example with PGD$_\text{U}$ or IFGSM$_\text{U}$, we proceed with the following steps: *(1)* We attack the prior majority vote $\mathbf{B}_{\mathcal{P}}$ with the attack PGD or IFGSM: we will obtain a first perturbation $\epsilon'$ ; *(2)* We sample $n$ uniform noises $\eta_1, \dots, \eta_n$ between $-10^{-2}$ and $+10^{-2}$ ; *(3)* We set the $i$-th perturbation as $\epsilon_i = \epsilon' + \eta_i$. Note that, for PGD$_\text{U}$ and IFGSM$_\text{U}$, after one attack, we end up with $n = 100$ perturbed examples. We set $n = 1$ when these attacks are used as a defense mechanism in Algorithm 2. Indeed since the adversarial training is iterative, we do not need to sample numerous perturbations for each example: we sample a new perturbation each time the example is forwarded through the decision trees. We also consider a naive defense referred to

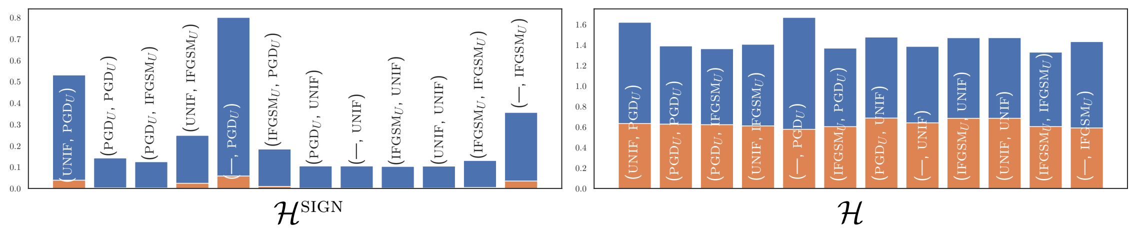

Figure 5.3: Visualization of the impact of the TV term in Equation (5.10). The left, respectively the right, bar plot show the bounds for the set of voters $\mathcal{H}^{\text{SIGN}}$, respectively $\mathcal{H}$. We plot the bounds for all the scenarios of Table 5.1 that use the TV distance, *i.e.*, all except the pairs $(\cdot, —)$. In orange, we represent the value of the TV term, while in blue, we represent all the remaining terms of the bound.

as UNIF that only adds a noise uniformly such that the $L_p$-norm of the added noise is lower than $b$.
We study the following scenarios of defense/attack. These scenarios correspond to all the pairs (Defense, Attack) belonging to the set $\{—, \text{UNIF}, \text{PGD}, \text{IFGSM}\} \times \{—, \text{PGD}, \text{IFGSM}\}$ for the baseline, and $\{—, \text{UNIF}, \text{PGD}_\text{U}, \text{IFGSM}_\text{U}\} \times \{—, \text{PGD}_\text{U}, \text{IFGSM}_\text{U}\}$, where "—" means that we do not defend, *i.e.*, the attack returns the original example (note that $\text{PGD}_\text{U}$ and $\text{IFGSM}_\text{U}$ when "Attack without U" refers to PGD and IFGSM for computing the classical adversarial risk $\mathcal{R}^{\text{ROB}}()$).

**Datasets and algorithm description.** We perform our experiment on six binary classification tasks from MNIST (LeCun *et al.*, 1998b) (1vs7, 4vs9, 5vs6) and Fashion MNIST (Xiao *et al.*, 2017) (Coat vs Shirt, Sandal vs Ankle Boot, Top vs Pullover). We decompose the learning set into two disjoint subsets $\mathcal{S}'$ of around $7,000$ examples (to learn the prior and the voters) and $\mathcal{S}$ of exactly $5,000$ examples (to learn the posterior). We keep as test set $\mathcal{T}$ the original test set that contains around $2,000$ examples. Moreover, we need a perturbed test set, denoted by $\mathbf{T}$, to compute our averaged(-max) adversarial risks. Depending on the scenario, $\mathbf{T}$ is constructed from $\mathcal{T}$ by attacking the prior model $\mathbf{B}_\mathcal{P}$ with $\text{PGD}_\text{U}$ or $\text{IFGSM}_\text{U}$ with $n = 100$. We run our Algorithm 2 for Equation (5.7) (Theorem 5.3.2), respectively Equation (5.10) (Theorem 5.3.3), and we compute our risk $\mathcal{R}_\mathbf{T}(\mathbf{B}_\mathcal{Q})$, respectively $\mathcal{A}_\mathbf{T}(\mathbf{B}_\mathcal{Q})$, the bound value and the usual adversarial risk associated to the model learned $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_\mathcal{Q})$. Note that, during the evaluation of the bounds, we have to compute our relaxed adversarial risks $\mathcal{R}_\mathbf{S}(\mathbf{B}_\mathcal{Q})$ and $\mathcal{A}_\mathbf{S}(\mathbf{B}_\mathcal{Q})$ on $\mathcal{S}$. For Step 1, the initial prior $P_0$ is fixed to the uniform distribution, the initial set of voters $\mathcal{H}_0$ is constructed with weights initialized with Xavier Initializer (Glorot and Bengio, 2010) and bias initialized at $0$ (more details are given in Appendix). During Step 2, to optimize the bound, we fix the confidence parameter $\delta = 0.05$, and we consider as the set of voters $\mathcal{H}$ two settings: $\mathcal{H}$ as it is output by Step 1, and the set $\mathcal{H}^{\text{SIGN}} = \{h'(\cdot) = \text{sign}(h(\cdot)) \,|\, h \in \mathcal{H}\}$ for which the theoretical results are still valid (we will see that in this latter situation we are able to better minimize the TV term of Theorem 5.3.3). For the two steps, we use Adam optimizer (Kingma and Ba, 2015) for $T = T' = 20$ epochs with a learning rate at $10^{-2}$ and a batch size at $64$.

Table 5.1: Test risks and bounds for **MNIST 1vs7** with $n = 100$ perturbations for all pairs (Defense,Attack) with the two voters' set $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. The results in **bold** correspond to the best values between results for $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. To quantify the gap between our risks and the classical definition, we put in *italic* the risk of our models against the classical attacks: we replace $\text{PGD}_\text{U}$ and $\text{IFGSM}_\text{U}$ by PGD or IFGSM (*i.e.*, we did *not* sample from the uniform distribution). Since Equation (5.10) upper-bounds Equation (5.9) thanks to the TV term, we compute the two bound values of Theorem 5.3.3.

| $L_2$-norm |  | Algo.2 with Eq. (5.7) |  |  |  |  |  | Algo.2 with Eq. (5.10) |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b = 1$ |  | Attack without U |  |  |  |  |  | Attack without U |  |  |  |  |  |  |  |
|  |  | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_\mathcal{Q})$ |  | $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_\mathcal{Q})$ |  | Th. 5.3.2 |  | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_\mathcal{Q})$ |  | $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_\mathcal{Q})$ |  | Th. 5.3.3 - Eq. (5.10) |  | Th. 5.3.3 - Eq. (5.9) |  |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.005* | *.005* | .005 | .005 | **.017** | .019 | *.005* | *.005* | .005 | .005 | **.099** | 0.100 | **.099** | .100 |
| — | $\text{PGD}_\text{U}$ | **.245** | *.255* | **.263** | .276 | .577 | **.448** | *.315* | **.313** | **.325** | .326 | **.801** | 1.667 | .684 | **.515** |
| — | $\text{IFGSM}_\text{U}$ | **.084** | *.086* | **.066** | .080 | **.170** | .185 | *.117* | **.113** | **.106** | .110 | **.356** | 1.431 | .286 | **.251** |
| UNIF | — | *.005* | *.005* | .005 | .005 | **.018** | .019 | *.005* | *.005* | .005 | .005 | **.099** | 0.100 | **.099** | .100 |
| UNIF | $\text{PGD}_\text{U}$ | *.151* | **.146** | **.151** | .158 | .355 | **.292** | *.183* | **.178** | .190 | **.189** | **.531** | 1.620 | .454 | **.355** |
| UNIF | $\text{IFGSM}_\text{U}$ | *.063* | **.061** | **.031** | .035 | **.088** | .114 | *.071* | **.070** | .056 | **.054** | **.248** | 1.405 | .200 | **.186** |
| $\text{PGD}_\text{U}$ | — | **.006** | *.007* | **.006** | .007 | **.023** | .024 | **.006** | *.007* | **.006** | .007 | **.102** | 0.103 | **.102** | .103 |
| $\text{PGD}_\text{U}$ | $\text{PGD}_\text{U}$ | **.028** | *.030* | **.021** | .025 | .065 | **.064** | **.028** | *.029* | **.025** | .028 | **.143** | 1.389 | .137 | **.136** |
| $\text{PGD}_\text{U}$ | $\text{IFGSM}_\text{U}$ | **.021** | *.022* | **.013** | .016 | **.043** | .045 | *.022* | *.022* | **.018** | .019 | **.125** | 1.362 | .121 | **.119** |
| $\text{IFGSM}_\text{U}$ | — | **.006** | *.007* | **.006** | .007 | **.019** | .021 | **.006** | *.007* | **.006** | .007 | **.100** | 0.102 | **.100** | .102 |
| $\text{IFGSM}_\text{U}$ | $\text{PGD}_\text{U}$ | **.040** | *.041* | **.033** | .035 | **.086** | .094 | *.040* | **.039** | .040 | **.038** | **.184** | 1.368 | .166 | **.163** |
| $\text{IFGSM}_\text{U}$ | $\text{IFGSM}_\text{U}$ | **.021** | *.022* | **.013** | .014 | **.039** | .049 | **.021** | *.022* | **.018** | .021 | **.131** | 1.329 | **.122** | .123 |

**Analysis of the results.** For the sake of readability, we exhibit the detailed results for one task (MNIST:1vs7) and all the pairs (Defense,Attack) with $L_2$-norm in Table 5.1, and we report in Figure 5.3 the influence of the TV term in the bound of Theorem 5.3.3 (Equation (5.10)). The detailed results on the other tasks are reported in Appendix A.7; We provide in Figure 5.4 an overview of the results we obtained on all the tasks for the pairs (Defense,Attack) where "Defense = Attack" and with $\mathcal{H}^{\text{SIGN}}$.

First of all, from Table 5.1 the bounds of Theorem 5.3.2 are tighter than the ones of Theorem 5.3.3: this is an expected result since we showed that the averaged-max adversarial risk $\mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_\mathcal{Q})$ is more pessimistic than its averaged counterpart $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q})$. Note that the bound values of Equation (5.9) are tighter than the ones of Equation (5.10). This is expected since Equation (5.9) is a lower bound on Equation (5.10).

Second, the bounds with $\mathcal{H}^{\text{SIGN}}$ are all informative (lower than $1$) and give insightful guarantees for our models. For Theorem 5.3.3 (Equation (5.10)) with $\mathcal{H}$, while the risks are comparable to the risks obtained with $\mathcal{H}^{\text{SIGN}}$, the bound values are greater than $1$, meaning that we have no more guarantee on the model learned. As we can observe in Figure 5.3, this is due to the TV term involved in the bound. Considering $\mathcal{H}^{\text{SIGN}}$ when optimizing $\mathcal{A}(\cdot)$ helps to control the TV term. Even if the bounds are non-vacuous for Theorem 5.3.2 with $\mathcal{H}$, the best models with the best guarantees are obtained with $\mathcal{H}^{\text{SIGN}}$. This is confirmed by the columns $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_\mathcal{Q})$ that are always worse than $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_\mathcal{Q})$ and mostly worse than $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_\mathcal{Q})$ with $\mathcal{H}^{\text{SIGN}}$. The performance obtained with $\mathcal{H}^{\text{SIGN}}$ can be explained by the fact that the sign "saturates" the output of the voters, which makes the majority vote more robust to noises. Thus, we focus the rest of the analysis on results obtained with $\mathcal{H}^{\text{SIGN}}$.

Third, we observe that the naive defense UNIF is able to improve the risks $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_\mathcal{Q})$ and $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_\mathcal{Q})$, but the improvement with the defenses based on $\text{PGD}_\text{U}$ and $\text{IFGSM}_\text{U}$ is much more significant specifically against a $\text{PGD}_\text{U}$ attack (up to $13$ times better). We observe the same phenomenon for both bounds (Theorems 5.3.2 and 5.3.3). This is an

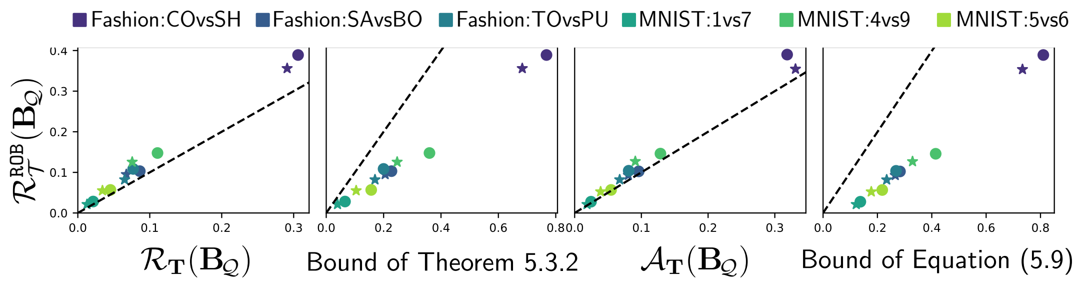

Figure 5.4: Visualization of the risk and bound values when "Defense=Attack" and when the set of voters is $\mathcal{H}^{\text{SIGN}}$. Results obtained with the PGD$_{\text{U}}$, respectively IFGSM$_{\text{U}}$, defense are represented by a star ★, respectively a circle ● *(reminder: $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_Q)$ is computed with a* PGD*, respectively* IFGSM*, attack).* The dashed line corresponds to bisecting line $y{=}x$. For $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_Q)$ and $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_Q)$, the closer the datasets are to the bisecting line, the more accurate our relaxed risk is compared to the classical adversarial risk $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_Q)$. For the bounds, the closer the datasets are to the bisecting line, the tighter the bound.

interesting fact because this behavior confirms that we are able to learn models that are robust against the attacks tested with theoretical guarantees.

Lastly, from Figure 5.4 and Table 5.1, it is important to notice that the gap between the classical risk and our relaxed risks is small, meaning that our relaxations are not too optimistic. Despite the pessimism of the classical risk $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_Q)$, it remains consistent with our bounds, *i.e.*, it is lower than the bounds. In other words, in addition to giving upper bounds for our risks $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_Q)$ and $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_Q)$, our bounds give non-vacuous guarantees on the classical risks $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_Q)$.

## 5.5 Conclusion

To the best of our knowledge, our work is the first one that studies from a general standpoint adversarial robustness through the lens of the PAC-Bayesian framework. We have started by formalizing a new adversarial robustness setting (for binary classification) specialized for models that can be expressed as a weighted majority vote; we referred to this setting as Adversarially Robust PAC-Bayes. This formulation allowed us to derive PAC-Bayesian generalization bounds on the adversarial risk of general majority votes. We illustrated the usefulness of this setting on the training of (differentiable) decision trees.

We believe that our new adversarial robustness setting can offer theoretical guarantees and well-founded algorithms when the model we learn is expressed as a majority vote, whether for ensemble methods with weak voters (*e.g.* Lorenzen *et al.*, 2019; Roy *et al.*, 2011), or for fusion of classifiers (*e.g.* Morvant *et al.*, 2014), or for multimodal/multiview learning (*e.g.* Goyal *et al.*, 2019; Sun *et al.*, 2017).

Regarding the aeronautical domain, this work is a first contribution to the respect of several objectives stated in the first guidance given by the EASA (EASA, 2021b) related to the certification of ML-based systems. We contribute to the safety aspects by providing generalization bounds on the adversarial robustness of ML algorithms. Hence, we increase the safety of ML-based systems but also the confidence one can have in the system. Nevertheless, these guarantees are probabilistic then the verification of the robustness is still required to ensure the good behavior of the system. The verification of ML-based systems' properties is another line of research complementary to this approach, which might be of high interest for the certification of ML-based systems. Thus, the next chapter will focus on this topic.

# 6 Monotony Property Verification

**This chapter is based on the following paper**

Guillaume VIDOT, Mélanie DUCOFFE, Christophe GABREAU, Ileana OBER, and Iulian OBER, Formal Methods for Industrial Critical Systems (FMICS 2022), pp. 15–31
Formal Monotony Analysis of Neural Networks with Mixed Inputs: An Asset for Certification

## Contents

**Abstract**

The use of ML technology to design safety-critical systems requires a complete understanding of the neural network's properties. Among the relevant properties in an industrial context, the verification of partial monotony may become mandatory for certain ML models. This chapter proposes a method to evaluate the monotony property using a Mixed Integer Linear Programming (MILP) solver. Contrary to the existing literature, this monotony analysis provides a lower and upper bound of the space volume where the property does not hold, which we denote "Non-Monotonic Space Coverage". This work has several advantages: *(i)* our formulation of the monotony property works on discrete inputs, *(ii)* the iterative nature of our algorithm allows for refining the analysis as needed, and *(iii)* from an industrial point of view, the results of this evaluation are valuable to the aeronautical domain where it can support the certification demonstration. We applied this method to an avionic case study (braking distance estimation using a neural network) where the verification of the monotony property is of paramount interest from a safety perspective.

## 6.1 Introduction

**Context.** Among all the properties required to certify AI-based systems, robustness is of paramount importance and is a widely studied property in the machine learning and verification communities (Carlini and Wagner, 2017; Katz *et al.*, 2019; Madry *et al.*, 2018; Müller *et al.*, 2022; Raghunathan *et al.*, 2018; Tjeng *et al.*, 2019; Tsuzuku *et al.*, 2018; Wang *et al.*, 2018, 2021; Weng *et al.*, 2018; Xu *et al.*, 2020; Zhang *et al.*, 2018). Although robustness is a critical property for classification tasks, we see the emergence of safety-related properties for regression tasks in many industries. For example, numerical models have been developed to approximate physical phenomena inherent to their systems (Biannic *et al.*, 2016). As these models are based on physical equations, whose relevancy is asserted by scientific experts, their qualification is carried out without any issue. However, since their computational costs and running time prevent us from embedding them on board, the use of these numerical models in the aeronautical field remains mainly limited to the development and design phase of the aircraft. Thanks to the current success of deep neural networks, previous works have already investigated neural network-based surrogates for approximating numerical models (Jian *et al.*, 2017; Sudakov *et al.*, 2019). Those surrogates come with additional safety properties linked to induced physics. One of them is the monotony which is motivated by the fact that monotonic functions are ubiquitous in most physics phenomena. For instance, one should expect that the estimate of a braking distance should be a monotonic function with respect to specific parameters such as the state of the brakes (nominal or degraded) or the state of the runway (dry or wet). Another case where monotony is relevant is in DNNs used for control.

**Motivation for the aeronautical domain.** As explained in Chapter 3, the development of systems is requirements-based. Indeed, the verification of requirements is still needed even for the development of ML-based items (see ML development process described in Figure 3.6). Hence, the interest in property verification is straightforward, considering the aeronautical domain. As a first step towards complete property verification, we focus on the monotony property in this work. Moreover, from a certification perspective, we highlight how the provided method could be useful to mitigate the risk of unintended behavior; we elaborate on this specific topic in Section 6.2.

**Contributions.** Today, state-of-the-art methods for enforcing partial monotony assume that if the property is not respected on the whole operational domain of the ML-based item, this puts the certification at risk (*i.e.*, the conformance to certification requirements) and therefore, the use of such function in an embedded system. We believe that this risk can be covered, especially for models that, in the future, would also be penalized for being too loose given the reference function. We propose an iterative method that *(i)* measures the part of the domain for which the monotony property is violated by providing a lower and upper bound of this "volume" and *(ii)* identifies their location in the space. Both pieces of evidence can be used to demonstrate the conformity.

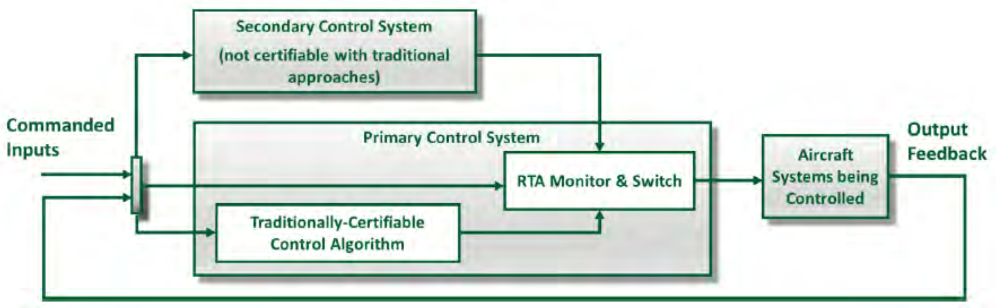
Secondary Control System
(not certifiable with traditional approaches)
Commanded Inputs
Primary Control System
RTA Monitor & Switch
Traditionally-Certifiable Control Algorithm
Aircraft Systems being Controlled
Output Feedback

Figure 6.1: Typical runtime assurance architecture proposed by PETERSON *et al.* (NASA, 2020)

**Organization of the chapter.** We start by pointing out the few certification principles tackled by our work (Section 6.2). We continue by reviewing the related work on monotony verification. Then, we dive into the details of our proposed method to analyze the monotony property of n ML model. We finish the chapter by applying our method to an industrial use case, which consists in estimating the braking distance of an aircraft using a neural network.

## 6.2 Certification preamble

As stated before, to certify a product, the following principle shall apply to the systems composing that product: each system performs its intended function and safely operates in foreseeable operating and environmental conditions. It means that even if the system performs a function with poor performance (this is obviously not desirable from an industrial viewpoint), it can be certifiable if the product's safety is guaranteed in the usage domain. Once that is said, a primordial principle emerges: safety is to be considered at the system level, meaning that even if a specific algorithm is not robust in some areas of its input space, then the system can remain certifiable if one can demonstrate that any unsafe behavior is prevented or mitigated. One possible mitigation of this risk is the use of runtime assurance illustrated in Figure 6.1 extracted from the NASA paper (PETERSON *et al.*, 2020), which ensures the system's safety by the design of redundant control system architecture, where a certifiable function takes over the function not certifiable with traditional approaches when an unsafe context is detected. This mitigation is relevant regardless of the technology used during the system development. Coming back to the specific context of ML-based development, the ability to formally define the areas of the input space where the safety properties of the model are not preserved can be a powerful asset in the conformity demonstration. Using formal methods can save significant testing efforts while preserving the safe behavior of the function.

## 6.3 Related Work

In recent years, assessing the robustness of neural networks has been tackled with formal verification (*i.e.*, sound algorithms demonstrating whether a property holds or not). Verifying properties on a neural network is challenging because neural networks are non-convex functions with many non-linearities, with hundreds to millions of parameters. Even if the type of architecture studied is restricted to piecewise-linear networks, it is known that this problem is already NP-hard (Weng *et al.*, 2018). There has been tremendous progress in the field of verification, from robustness proof of networks trained on MNIST to scaling up to CIFAR 10 and even TinyImagenet. These successes are particularly due to the collaboration of different communities that made it possible to formulate and tackle the robustness problem from different perspectives. Without being exhaustive, we can cite the methods that rely on Lipschitz-based optimization (Zhang *et al.*, 2018, 2019b), input refinement (Wang *et al.*, 2018) and semi-definite relaxations (Raghunathan *et al.*, 2018).

So far, the verification community has mainly tackled robustness verification from adversarial robustness (Urban and Miné, 2021) to computing the reachable set of a network (Xiang *et al.*, 2018) despite a few other properties that are highly relevant for the industry. Among those properties, partial monotony under specific inputs appears to be a key property, especially for regression tasks. Indeed, the need for monotony appeared in various contexts such as surrogate modeling (Hao *et al.*, 2021), economics (Feelders, 2000), fairness (Karpf, 1991), or interpretability (Nguyen and Martínez, 2019) and is thus a highly desirable feature in the industry. Previous works proposed to enforce the monotony property during the design; In Gupta *et al.* (2019), they relied on heuristics regularizers to promote monotony, whose main drawback lies in the absence of guarantees and, therefore, will require verification as a post-processing step. On the other side, Gauffriau *et al.* (2021) and Liu *et al.* (2020) adopted hand-designed monotonic architectures, which may harden the training and not perform well in practice. Lastly, up to our knowledge, previous works mainly considered monotony under continuous inputs, while many industrial use cases have monotony constraints on discrete inputs. One notable exception is the fairness verification in Urban *et al.* (2020) that can be applied on both a discrete or a continuous input and holds similarity with monotony verification.

When it comes to continuous inputs, monotony is equivalent to verifying a property on the gradients on the whole domain. Indeed the sign of the gradient component corresponding to monotonous inputs should always be positive or negative. However, for a neural network with discrete inputs, the gradient sign condition needs not to hold for the monotony to hold, even when the gradient can be computed by extending the input domain to reals. For piece-wise linear neural networks such as ReLU networks, we can base verification on the very definition of monotony (Definition 6.4.1), which can be cast as solving a Mixed Integer Linear Programming (MILP) problem. This method is complementary to the literature using the gradient condition and can verify monotony over discrete inputs.

Verifying the monotony is recognized to be more challenging than robustness since it is a global property on the whole domain rather than a local neighborhood (Liu *et al.*, 2020). However, we argue that applying partial monotony over the whole domain, which may affect the performance and put at risk the product's release, is a very drastic approach. Indeed, in an industrial context, it is necessary to balance quality and safety, especially as the systems will be constrained by other specifications than just monotony, such as accuracy. The solution we propose is a partitioning scheme that splits the operational domain into areas where the monotony property is respected and areas where it is (partially) violated; in the latter, the neural network's behavior could be mitigated. This possibility has been considered on a collision detection use case in Damour *et al.* (2021) and studied at a higher level for the certification of a system before an ML component (Mamalet *et al.*, 2021).

## 6.4 Monotony Analysis

In this section, we define the concept of *partial* monotony with respect to a set of inputs. Let $\mathbb{V}$ be a (finite) set of input features. For each feature $v \in \mathbb{V}$ we denote $\mathrm{D}(v)$ the domain in which $v$ ranges. Hence, let $\mathbb{X} = \times_{v \in \mathbb{V}} \mathrm{D}(v)$ be the input space, $\mathbb{Y}$ be the output space and $h : \mathbb{X} \mapsto \mathbb{Y}$ be the neural network. Note that the features are generally of two types ($\mathbb{V} = \mathbb{V}_d \sqcup \mathbb{V}_c$):

- $v \in \mathbb{V}_d$ are features whose domain $\mathrm{D}(v)$ is discrete (*e.g.*, a finite set of labels or categorical values)
- $v \in \mathbb{V}_c$ are features whose domain $\mathrm{D}(v)$ is a real interval

We are interested in monotony properties, which supposes that the set $\mathbb{Y}$ has an order relation denoted $\preceq$; usually, $\mathbb{Y} \subseteq \mathbb{R}$ and $\preceq$ is one of the usual orders ($\leq$, $\geq$). The monotony property will be relative to a subset of discrete features, $\alpha \subseteq \mathbb{V}_d$ for which a partial order is defined on $\times_{v \in \alpha} \mathrm{D}(v)$, also denoted $\preceq$ without risk of confusion. For $x \in \mathbb{X}$, let us denote $x{\downarrow_\alpha}$ the projection of $x$ onto the dimensions in $\alpha$, and $\bar{\alpha} = \mathbb{V} \setminus \alpha$.

**Definition 6.4.1.** *Monotony Property*
A function $h$ is monotone with respect to an order $\preceq$ on the output domain $\mathbb{Y}$ and to a subset of discrete features $\alpha \subseteq \mathbb{V}_d$ endowed with a partial order defined on $\times_{v \in \alpha} \mathrm{D}(v)$ also denoted $\preceq$ (without risk of confusion) if and only if

$$\forall (x_1, x_2) \in \mathbb{X}^2 : x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}} \ \wedge \ x_1{\downarrow_\alpha} \preceq x_2{\downarrow_\alpha} \quad \Longrightarrow \quad h(x_1) \preceq h(x_2)$$

### Goal of the Analysis

Our analysis aims to identify the sub-spaces where the monotony does not hold using a MILP solver. Example 6.4.1 describes a toy example (a simplified version of the case study in section 6.5) that we will use to explain the main concepts.

**Example 6.4.1.** *Simplified Braking Distance Estimation*
Setup: let $f$ be a neural network estimating the braking distance of an aircraft based on its speed, weight, and the runway's state (dry or wet).
Property: *for the same speed and weight ($x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}}$), the braking distance on a*

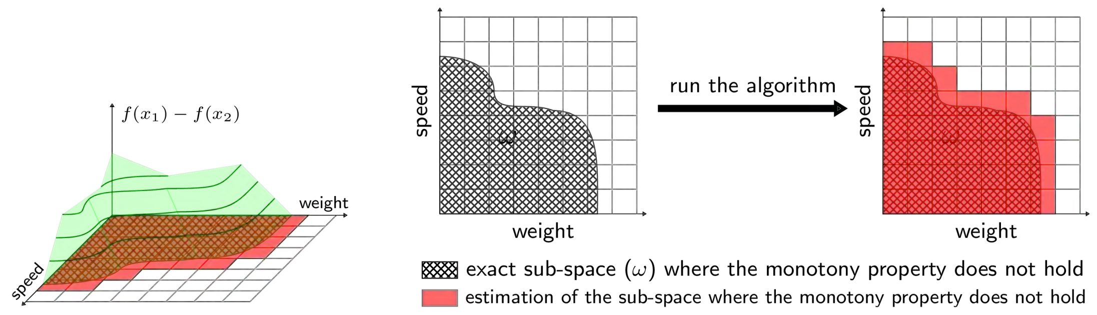
$f(x_1) - f(x_2)$
weight
speed
speed
weight
run the algorithm
speed
weight
exact sub-space ($\omega$) where the monotony property does not hold
estimation of the sub-space where the monotony property does not hold

Figure 6.2: Purpose of the algorithm through Example 6.4.1. In $x_1$ the runway is dry and in $x_2$ the runway is wet. The left plot represents $h(x_1) - h(x_2)$ where only the positive values are displayed (monotony property violated). The two plots on the right are the projection of these points on the plane composed of features 1 and 2.

> *wet runway must be higher than on a dry one ($x_1{\downarrow_\alpha} \preceq x_2{\downarrow_\alpha} \implies h(x_1) \preceq h(x_2)$).*
> Goal: Identify and quantify the input areas where the property does not hold.

If we plot $h(x_1) - h(x_2)$ versus the speed and the weight, the Definition 6.4.1 holds if and only if all the values are negative. The 3D plot in Figure 6.2 shows a sketch of this example when the monotony property partially holds, *i.e.*, $h(x_1) - h(x_2)$ is partially positive. To ease the visualization, we only draw the positive values. The cross-hatched areas in the 2D plots are projections of the positive values of the curve on the plane representing the speed and the weight features and models the area where the monotony property is not respected, namely the Non-Monotonic Space Coverage denoted as $\omega$. The rightmost 2D plot shows what we expect from our analysis on Example 6.4.1: identifying and estimating $\omega$. To estimate $\omega$, we partition the space (grid in Figure 6.2), and then the monotony property is checked on each sub-space. The red area represents the identified sub-space where monotony issues occur, *i.e.*, an over-approximation of $\omega$. In addition, our approach provides a lower and upper bound of the size of $\omega$ relative to the whole input domain, respectively denoted as $\underline{\Omega}$ and $\overline{\Omega}$ (see Figure 6.3).

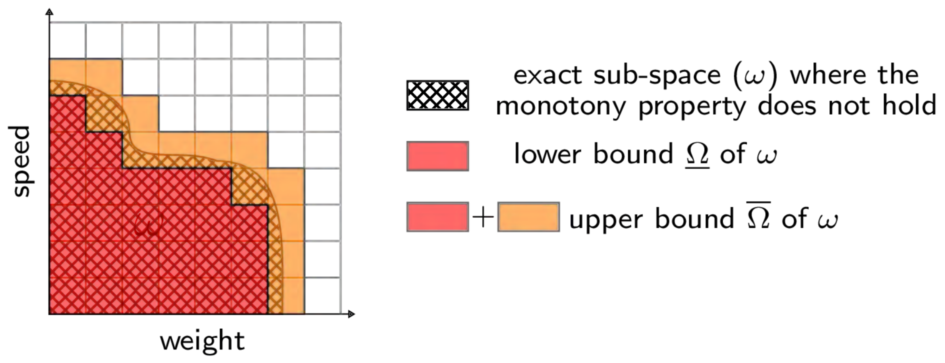
speed
weight
$\omega$
exact sub-space ($\omega$) where the monotony property does not hold
lower bound $\underline{\Omega}$ of $\omega$
+
upper bound $\overline{\Omega}$ of $\omega$

Figure 6.3: Based on Figure 6.2: representation of $\underline{\Omega}$ and $\overline{\Omega}$ considering Example 6.4.1

Our approach can distinguish the sub-spaces where the monotony property does not hold (red area in Figure 6.3) from the ones where it partially holds (orange area in

Figure 6.3). Hence, the lower bound is the red area, while the upper bound is the red and orange areas. The benefit of having a lower and upper bound, instead of just an overestimation, is to be able to assess whether our estimation is precise: a large gap between the upper and lower bound may reveal that our bounds are not representative of $\omega$. The iterative nature of our approach overcomes this problem: we refine our space, which leads to a finer grid for the Figure 6.3 and run again the MILP solver where the property partially holds to have a better estimation of $\omega$.

## MILP formulation

**Neural Network encoding** Let $h : \mathbb{X} \to \mathbb{Y}$ be a neural network composed of $n$ layers with ReLU activations. The layer $0$ corresponds to the input layer, while the layer $n$ to the output one. We use the MILP formulation proposed by Cheng *et al.* (2017), which uses the big-M method (Grossmann, 2002) to encode the ReLU activation. By convention, the notations in bold denote the MILP variables, and those not in bold denote constants. For $1 \leq i \leq (n-1)$, let $C^i$ be the conjunction of constraints for the layer $i$:

$$\begin{aligned} C^i \triangleq \quad & \hat{\boldsymbol{x}}^{\boldsymbol{i}} = W^i \boldsymbol{x}^{\boldsymbol{i-1}} + b^i && && (6.1) \\ & \wedge \boldsymbol{x}^{\boldsymbol{i}} \leq \hat{\boldsymbol{x}}^{\boldsymbol{i}} + M^i(1 - \boldsymbol{a}^{\boldsymbol{i}}) && \wedge && \boldsymbol{x}^{\boldsymbol{i}} \geq \hat{\boldsymbol{x}}^{\boldsymbol{i}} \quad (6.2) \\ & \wedge \boldsymbol{x}^{\boldsymbol{i}} \leq M^i \cdot \boldsymbol{a}^{\boldsymbol{i}} && \wedge && \boldsymbol{x}^{\boldsymbol{i}} \geq 0 \quad (6.3) \\ & \wedge \boldsymbol{a}^{\boldsymbol{i}} \in \{0,1\}^{|\boldsymbol{x}^{\boldsymbol{i}}|} && && (6.4) \end{aligned}$$

where $\hat{\boldsymbol{x}}^{\boldsymbol{i}}$ and $\boldsymbol{x}^{\boldsymbol{i}}$ are the vector of neuron values at the layer $i$, respectively, before and after the ReLU activation. $M^i$ is a large valid upper bound s.t. $-M^i \leq \hat{\boldsymbol{x}}^{\boldsymbol{i}}$ and $\boldsymbol{x}^{\boldsymbol{i}} \leq M^i$ (Cheng *et al.*, 2017). $W_i$ and $b_i$ are, respectively, the weights and bias at the layer $i$, and $\boldsymbol{a}^{\boldsymbol{i}}$ is a binary vector representing whether the neurons are activated or not. The Equation (6.1) is the constraint for the affine transformation and the Equations (6.2) to (6.4) are the constraints encoding the ReLU activation. For the output layer $n$, there is no ReLU activation; then we have the following:

$$C^n \triangleq \quad \hat{\boldsymbol{x}}^{\boldsymbol{n}} = W^n \boldsymbol{x}^{\boldsymbol{n-1}} + b^n \tag{6.5}$$

It remains to encode the constraints of the input layer, which enforce the lower and upper bounds of the domain of the input features. Our analysis relies on a partition of the input space $\mathbb{X}$, thus the encoding of the input layer will depend on it: let $\mathcal{P}$ be a partition of $\mathbb{X}$, $p \in \mathcal{P}$ be a subset of $\mathbb{X}$ represented by a set of linear constraints (also denoted $p$). Hence, the neural network $h$ is encoded as the conjunction of the constraints defined for each layer and $p$, which constrains the input layer:

$$C^h(p) \triangleq p \wedge \left( \bigwedge_{i=1}^{n} C^i \right) \wedge C^n \tag{6.6}$$

**Monotony property encoding** Following Definition 6.4.1, we must encode $h$ twice in MILP: $C_1^h$ and $C_2^h$. Similarly to the encoding of the input space's constraints, we encode the monotony property regarding the partition $\mathcal{P}$. So, let $p_i, p_j \in \mathcal{P}^2$ be two sub-spaces of $\mathbb{X}$ such that $\exists x_1, x_2 \in p_i \times p_j,\ x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}} \wedge\ x_1{\downarrow_{\alpha}} \prec x_2{\downarrow_{\alpha}}$. Then, we have:

$$C^{mon}(p_i,p_j) \triangleq \Big(\boldsymbol{x_1^0}{\downarrow_{\bar{\alpha}}} = \boldsymbol{x_2^0}{\downarrow_{\bar{\alpha}}} \wedge\ \boldsymbol{x_1^0}{\downarrow_{\alpha}} \preceq \boldsymbol{x_2^0}{\downarrow_{\alpha}}\Big) \wedge \Big(C_1^h(p_i) \wedge C_2^h(p_j)\Big) \wedge \Big(\boldsymbol{\hat{x}_1^n} \leq \boldsymbol{\hat{x}_2^n}\Big) \tag{6.7}$$

$$C^{\neg mon}(p_i,p_j) \triangleq \Big(\boldsymbol{x_1^0}{\downarrow_{\bar{\alpha}}} = \boldsymbol{x_2^0}{\downarrow_{\bar{\alpha}}} \wedge\ \boldsymbol{x_1^0}{\downarrow_{\alpha}} \preceq \boldsymbol{x_2^0}{\downarrow_{\alpha}}\Big) \wedge \Big(C_1^h(p_i) \wedge C_2^h(p_j)\Big) \wedge \Big(\boldsymbol{\hat{x}_1^n} \geq \boldsymbol{\hat{x}_2^n} + \epsilon\Big) \tag{6.8}$$

The MILP solver may output either `SAT`, `UNSAT` or `TIMEOUT`. For Equations (6.7) and (6.8), `TIMEOUT` means that the time limit is reached. $C^{mon}$ checks whether the neural network $h$ is monotonic:

- `SAT`: there is an assignment for $\boldsymbol{x_1^0}, \boldsymbol{x_2^0} \in p_i \times p_j$ which respects the monotony.
- `UNSAT`: the monotony is not respected on the complete sub-space $p_i \times p_j$.

$C^{\neg mon}$ checks whether the neural network is not monotonic:

- `SAT`: there is an assignment for $\boldsymbol{x_1^0}, \boldsymbol{x_2^0} \in p_i \times p_j$ which violates the monotony.
- `UNSAT`: the monotony is respected on the complete sub-space $p_i \times p_j$.

To avoid having `SAT` for $C^{\neg mon}$ when $\boldsymbol{\hat{x}_1^n} = \boldsymbol{\hat{x}_2^n}$, we introduce the $\epsilon$ term (Equation (6.8)).

To determine for each sub-space $p_i \times p_j$ whether the monotony property holds, partially holds, or does not hold (see Figure 6.3), we must solve successively $C^{\neg \text{mon}}$ and $C^{\text{mon}}$ (see Section 6.4 for more detail).

## Verification Procedure

As explained in Section 6.4, our verification procedure implies the partition of the space and the verification of each sub-space. In Algorithm 3, the monotony property is iteratively analyzed regarding a partition while refining this partition in the zone of interest to sharpen the analysis. Algorithm 4 details the verification run at each iteration.

**Algorithm 3** The monotony is defined on the space $\mathbb{X}^2$, but for the analysis, we deal with the partition $\mathcal{P}$ of $\mathbb{X}$ as defined earlier. Hence to verify the monotony on the complete space $\mathbb{X}^2$, we need to go through all the sub-spaces $p_i \times p_j$, $\forall (p_i, p_j) \in \mathcal{P}^2$. However, it may happen that the monotony does not apply to the sub-space $(p_i, p_j)$ because there are no comparable elements within the sub-space: $P_1$, in Line 1, contains all and only the $(p_i, p_j)$ including comparable elements. We denote the elements of $P_1$ and more generally, $P_t$, "monotony scenario".

We propose an iterative procedure where at each iteration we use, in Line 6, $\mathbf{F}(\cdot)$ (see Algorithm 4) to retrieve $P_t^{\neg\text{mon}}$ and $P_t^{\text{partially mon}}$. Then, we compute in Line 7 the metrics $\underline{\Omega_t}$ and $\overline{\Omega_t}$ for the iteration $t$, which respectively lower-bounds and upper-bounds $\omega$; $\omega$ is the exact ratio of the space where $h$ is not monotonic, which corresponds to the ratio of monotony scenarios where $h$ is not monotonic. In Line 11, we refine the partition of the space for the next iteration: "`partition`" is the function that takes the current

**Algorithm 3** Monotony analysis refinement

```
Require: $T$: the number of iterations of the procedure
 1: $P_1 \leftarrow \{(p_i, p_j) \in \mathcal{P}^2 \mid \exists (x_1, x_2) \in p_i \times p_j,\ x_1{\downarrow_\alpha} \prec x_2{\downarrow_\alpha}$ and $x_1{\downarrow_{\bar{\alpha}}} = x_2{\downarrow_{\bar{\alpha}}}\}$
 2: $\underline{\Omega_0} \leftarrow 0$ and $\widehat{P_0} \leftarrow P_1$
 3: $\Omega \leftarrow [\,]$, $\mathbb{P}^{\neg\text{mon}} \leftarrow \emptyset$, and $\mathbb{P}^{\text{partially mon}} \leftarrow \emptyset$
 4: for $t$ from $1$ to $T$ do
 5:     $\widehat{P_t} \leftarrow \widehat{P}_{t-1} \wedge P_t$
 6:     $(P_t^{\neg\text{mon}}, P_t^{\text{partially mon}}) \leftarrow \mathbf{F}(\widehat{P_t})$
 7:     $\underline{\Omega_t} \leftarrow \underline{\Omega_{t-1}} + \frac{|P_t^{\neg\text{mon}}|}{|P_t|}$   and   $\overline{\Omega_t} \leftarrow \underline{\Omega_t} + \frac{|P_t^{\text{partially mon}}|}{|P_t|}$        ▷ See Figure 6.4
 8:     $\Omega \leftarrow \Omega + (\underline{\Omega_t}, \overline{\Omega_t})$
 9:     $\mathbb{P}^{\neg\text{mon}} \leftarrow \mathbb{P}^{\neg\text{mon}} \cup P_t^{\neg\text{mon}}$   and   $\mathbb{P}^{\text{partially mon}} \leftarrow P_t^{\text{partially mon}}$
10:     $\widehat{P_t} \leftarrow P_t^{\text{partially mon}}$
11:     $P_{t+1} \leftarrow \mathtt{partition}(P_t)$
12: end for
13: return $\Omega$, $\mathbb{P}_{\neg\text{mon}}$ and $\mathbb{P}_{\text{partially mon}}$
```

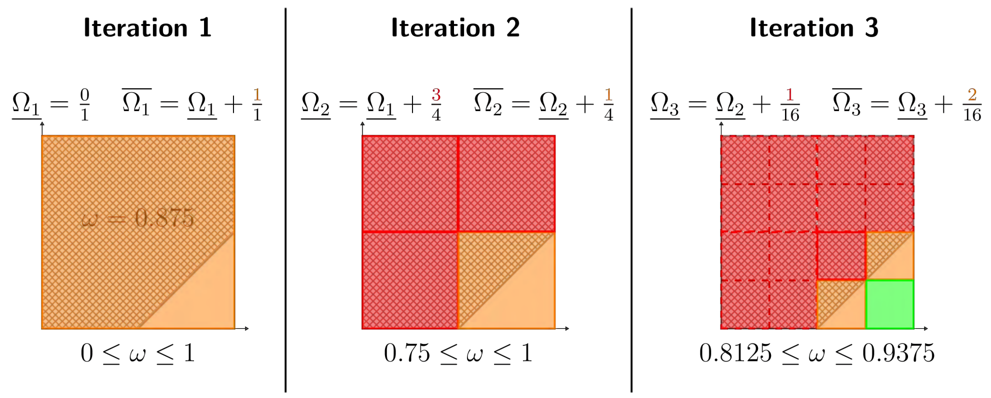
Iteration 1
$\underline{\Omega_1} = \frac{0}{1}$ $\overline{\Omega_1} = \underline{\Omega_1} + \frac{1}{1}$
$\omega = 0.875$
$0 \leq \omega \leq 1$
Iteration 2
$\underline{\Omega_2} = \underline{\Omega_1} + \frac{3}{4}$ $\overline{\Omega_2} = \underline{\Omega_2} + \frac{1}{4}$
$0.75 \leq \omega \leq 1$
Iteration 3
$\underline{\Omega_3} = \underline{\Omega_2} + \frac{1}{16}$ $\overline{\Omega_3} = \underline{\Omega_3} + \frac{2}{16}$
$0.8125 \leq \omega \leq 0.9375$

Figure 6.4: Run of Algorithm 3 on Example 6.4.1 with the detailed computation of $\underline{\Omega_t}$ and $\overline{\Omega_t}$. The cross-hatched area represents the sub-space the algorithm strives to estimate.

partition of the space and returns a finer partition; we suppose that all elements in the partition have the same size. Note that $P_t$ gets finer and finer through the iterations: the more we refine, the more elements $P_t$ will have. We highlight that in Line 5, the operator $\wedge$ applies the intersection between each subset of $\widehat{P}_{t-1}$ and $P_t$ where $P_t$ is a finer partition of the space than $\widehat{P}_{t-1}$. It allows to get the elements of interest $(\widehat{P}_{t-1})$ in the right level of details $(P_t)$. For the first iteration, we run $\mathbf{F}(\cdot)$ on all the elements (initialization of $\widehat{P_0}$ to $P_1$). However, we only need to refine the sub-spaces where the monotony property is partially respected for the other iterations. Finally, the algorithm returns the lower and upper bounds of each iteration and all the sub-spaces where the monotony property does not hold or partially holds.

In Figure 6.4, we run Algorithm 3 on Example 6.4.1[1]: $\alpha$ contains the runway's state, we partition $\mathbb{X}$ on $\alpha$ and we have a unique $(p_i, p_j)$ in $P_1$; in $p_i$ the runway is dry and in

[1]Note that we simplify the cross-hatched area's shape in order to know the omega value for the explanation.

**Algorithm 4** $\mathbf{F}(P) \longrightarrow$ Monotony analysis of $P \subseteq \mathcal{P}^2$

```
Require: $P \subseteq \mathcal{P}^2$ gathers the sub-spaces that need to be verified.
 1: $P^{\neg\text{mon}} \leftarrow \emptyset$
 2: $P^{\text{partially mon}} \leftarrow \emptyset$
 3: for all $(p_i, p_j) \in P$ do
 4:   if solve($C^{\neg mon}(p_i, p_j)$) is SAT then
 5:     if solve($C^{mon}(p_i, p_j)$) is SAT then
 6:       $P^{\text{partially mon}} \leftarrow P^{\text{partially mon}} \cup \{(p_i, p_j)\}$
 7:     else
 8:       $P^{\neg\text{mon}} \leftarrow P^{\neg\text{mon}} \cup \{(p_i, p_j)\}$
 9:     end if
10:   else
11:     Continue to the next $(p_i, p_j)$          ▷ Monotonic on the whole domain $p_i \times p_j$
12:   end if
13: end for
14: return $P^{\neg\text{mon}}$ and $P^{\text{partially mon}}$          ▷ $\{P^{\neg\text{mon}} \cup P^{\text{partially mon}}\} \subseteq P$
```

$p_j$ wet. Then, the two axes represent features of $\bar{\alpha}$ (speed and weight) and the squares, the partition of the space. The cross-hatched surface is the exact sub-space where the monotony property does not hold. The orange squares means that the monotony property partially holds, the red squares means it does not hold, and the green squares means it holds. Through the iteration, we refine the partition (smaller squares) while running the verification only for the smaller squares (in solid lines) coming from a bigger orange square (in Line 5; $\widehat{P}_{t-1}$ is the orange square of iteration 2 and $P_t$ is the small squares of iteration 3).

**Algorithm 4** The verification function $\mathbf{F}(P)$ aims to analyze the monotony of $h$ regarding $P$ a subset of $\mathcal{P}^2$, which gathers the sub-spaces where the monotony property must be checked. Intuitively, the partition $\mathcal{P}$ and thus $P$ can be seen as the level of details of the monotony analysis. Indeed, a finer partition $\mathcal{P}$ results in smaller sub-spaces in $P$; hence a more detailed analysis.

Then, from Lines 4 to 12, we identify in which sub-space $p_i \times p_j$ the neural network $h$ partially respects or does not respect the monotony property and sort them in $P^{\text{partially mon}}$ and $P^{\neg\text{mon}}$. In Lines 4 and 5, `solve(·)` refers to any off-the-shelf MILP solver taking as input a MILP problem. Table 6.1 shows the interpretation of the monotony of $h$ within the sub-space regarding every truth values of the conditions of Lines 4 and 5. Note that we arrive in Line 11 when the condition of Line 4 is False, and we jump to the next sub-space (or monotony scenario) because the monotony property holds for the current sub-space $p_i \times p_j$. Finally, we return the two sets gathering the sub-spaces where the monotony property does not hold and where it partially holds.

**Non-Monotonic Space Coverage** $\underline{\Omega_t}$ and $\overline{\Omega_t}$ are defined as the ratio of sub-spaces (monotony scenarios) where $h$ has monotony issue over the total number sub-spaces in $P_t$ (contains all the monotony scenarios):

Table 6.1: State of the monotony property regarding the condition of Lines 4 and 5.

| Case | Line 4 | $C^{\neg mon}$ | Line 5 | $C^{mon}$ | Monotony property on $p_i \times p_j$ |
|---|---|---|---|---|---|
| 1 | True | `SAT` | True | `SAT` | partially holds |
| 2 | True | `SAT` | False | `UNSAT` | does not hold |
| 3 | False | `UNSAT` | - | - | holds |

**Definition 6.4.2.** *Lower bound and upper bound of $\omega$*

$$\underline{\Omega_t} = \underline{\Omega_{t-1}} + \frac{|P_t^{\neg\text{mon}}|}{|P_t|} \tag{6.9}$$

$$\overline{\Omega_t} = \underline{\Omega_t} + \frac{\left|P_t^{\text{partially mon}}\right|}{|P_t|} \tag{6.10}$$

On the one hand, $\underline{\Omega_t}$ takes into account only the sub-spaces where the monotony property holds not; hence, it lower-bounds $\omega$. On the other hand, $\overline{\Omega_t}$ considers the sub-spaces where the monotony property holds not and partially holds; hence, it upper-bounds $\omega$. Figure 6.4 details the computation of these metrics along with the iteration: at each iteration, the lower bound $\underline{\Omega_t}$ is represented by all the red squares and the upper bound $\overline{\Omega_t}$ by all the red and orange squares.

**Example 6.4.2.** *Computation of $\underline{\Omega_t}$ and $\overline{\Omega_t}$ considering Example 6.4.1.*

**Iteration 1** We consider the entire space. Hence, we only have one sub-space where we assess the monotony property ($|P_t| = 1$). There is no red square, *i.e.*, sub-space where the monotony property does not hold, which means that $|P_1^{\neg\text{mon}}| = 0$, then $\underline{\Omega_1} = 0$. We have one orange square: in this sub-space, the monotony property partially holds, then $|P_1^{\text{partially mon}}| = 1$ and $\overline{\Omega_1} = 1$.

**Iteration 2** We partition the space in 4 smaller sub-spaces ($|P_t| = 4$) and run again the verification on each sub-space. We proceed similarly to the previous iteration for the computation of $\underline{\Omega_2}$ and $\overline{\Omega_2}$. We have 3 red squares ($|P_2^{\neg\text{mon}}| = 3$) and 1 orange square ($|P_2^{\text{partially mon}}|$=1): $\underline{\Omega_2}$=$\underline{\Omega_1} + \frac{3}{4}$=0.75 and $\overline{\Omega_2} = \underline{\Omega_2} + \frac{1}{4} = 1$.

**Iteration 3** We refine the partition of the previous step and end up with 16 sub-spaces. However, we only run the verification on the sub-spaces coming from an orange square (Lines 5 of Algorithm 3), *i.e.*, sub-spaces where $h$ is partially monotonic. We have $\underline{\Omega_3} = \frac{3}{4} + \frac{1}{16} = 0.8125$ and $\overline{\Omega_3} = \frac{3}{4} + \frac{1}{16} + \frac{2}{16} = 0.9375$.

The Proposition 6.4.1 shows that the lower and upper bounds are tighter over the iterations: the more iterations we run, the closer to $\omega$ we are.

**Proposition 6.4.1.** For any $t \geq 1$, we have

$$\underline{\Omega_{t-1}} \leq \underline{\Omega_t} \tag{6.11}$$

and

$$\overline{\Omega_t} \leq \overline{\Omega_{t-1}} \tag{6.12}$$

*Proof.* For Equation (6.11), considering the facts that

$$\underline{\Omega_0} = 0 \qquad \text{and} \qquad \frac{|P_t^{\neg\text{mon}}|}{|P_t|} \geq 0 \qquad \text{and} \qquad \underline{\Omega_t} = \underline{\Omega_{t-1}} + \frac{|P_t^{\neg\text{mon}}|}{|P_t|},$$

we can deduce $\underline{\Omega_{t-1}} \leq \underline{\Omega_t}$.

Then, to prove Equation (6.12), we need first to state some invariant: for the computation of $\overline{\Omega_t}$ (Algorithm 3, Line 7) we have,

$$\left(P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right) \subseteq \widehat{P}_t \tag{6.13}$$

Based on Equation (6.13), we have:

$$\begin{aligned}
\left|P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right| &\leq \left|\widehat{P}_t\right| \\
&\leq \left|P_{t-1}^{\text{partially mon}}\right| * \frac{|P_t|}{|P_{t-1}|} && \text{By construction of } \widehat{P}_t \text{ which is a finer partition of } P_{t-1}^{\text{partially mon}} \\
\underline{\Omega_{t-1}} + \frac{\left|P_t^{\text{partially mon}} \cup P_t^{\neg\text{mon}}\right|}{|P_t|} &\leq \underline{\Omega_{t-1}} + \frac{\left|P_{t-1}^{\text{partially mon}}\right|}{|P_{t-1}|} \\
\underline{\Omega_{t-1}} + \frac{|P_t^{\neg\text{mon}}|}{|P_t|} + \frac{\left|P_t^{\text{partially mon}}\right|}{|P_t|} &\leq \overline{\Omega_{t-1}} && P_t^{\text{partially mon}} \cap P_t^{\neg\text{mon}} = \emptyset \\
\underline{\Omega_t} + \frac{\left|P_t^{\text{partially mon}}\right|}{|P_t|} &\leq \overline{\Omega_{t-1}} \\
\overline{\Omega_t} &\leq \overline{\Omega_{t-1}}.
\end{aligned}$$

■

## 6.5 Case Study: Braking Distance Estimation

### Description of the case study

Our case study comes from the aeronautical industry. It is an R&D project consisting in training a neural network to estimate the braking distance of an aircraft based on physical information. The R&D team provides us with a trained feedforward neural network composed of two layers (30 and 29 neurons on the first and second layers, respectively) and ReLU activation functions. There are 15 input features, including 13 discrete and 2 continuous. Among the discrete features, ten describe the state of the brakes. The aircraft has four brakes, and each brake has five possible modes: Normal (N), Altered (A), Emergency (E), Burst (B), and Released (R). Then, the network has two features for each mode: *(i)* the total number of brakes in a given mode (referred to as "symmetric") and *(ii)* the difference between the number of brakes on the left and right side of a given mode (referred to as "asymmetric"). From this information, we can find back the states of the pairs of brakes on the left and right sides of the aircraft (see Figure 6.5), although the state of each individual brake cannot be retrieved. For clarity

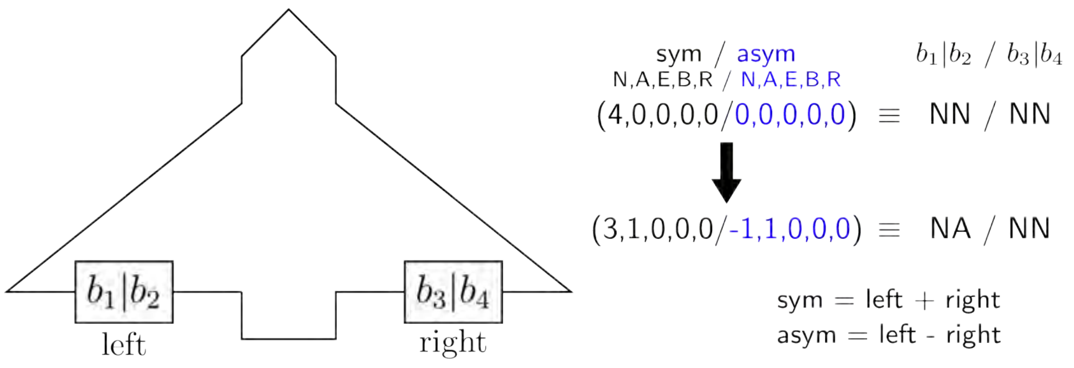

Figure 6.5: Representation of the position of the four brakes on an aircraft denoted by $b_i \in \{N, A, E, B, R\}$. For example, we have NN-NN when all the brakes are in the normal state. Then if the state of one of the left brakes becomes *Altered*, we have NA-NN. Note that NA-NN≡AN-NN due to the choice of the representation of the brakes.

and since we have the equivalence between both notations, we will describe the state of the brakes using the form "$b_1b_2$-$b_3b_4$".

To show how to handle simultaneously several input features within the monotony property, we focus on the one involving the state of the brakes. We can textually express the monotony property as follows:

*When the brakes' state deteriorates, the braking distance should increase.*

To perform the monotony analysis, we need to define what *deteriorates* means formally. Relying on the system expert's knowledge, the following partial order applies to the different modes of the brake:

$$N \prec_b A \prec_b E \prec_b B \prec_b R \tag{6.14}$$

where $b_i \prec_b b_j$ means that the state $b_j$ is more deteriorated than the state $b_i$.
We can easily extend the partial order $\preceq_b$ on one brake to the state of an aircraft's brakes composed of 4 brakes. Let $S_1 = (b_1, b_2, b_3, b_4)$ and $S_2 = (b'_1, b'_2, b'_3, b'_4)$ be two states of an aircraft's brakes, we have

$$S_1 \prec S_2 \iff \forall b_i, b'_i \in S_1 \times S_2,\ b_i \preceq_b b'_i \text{ and } \exists b_i, b'_i \in S_1 \times S_2\ b_i \prec_b b'_i \tag{6.15}$$

It means that $S_2$ is deteriorated compared to $S_1$ if and only if for all brakes in $S_2$, the brake's mode in $S_2$ is at most as good as its counterpart in $S_1$ and there exists a brake in $S_2$ whose mode is strictly worse than its counterpart in $S_1$.

## Experimentation

**Setup** Let $\mathbb{V}$ be the set of 15 input features described above, $\mathbb{X} = \times_{v\in\mathbb{V}} \mathrm{D}(v)$ be the input space, $\mathbb{Y} = \mathbb{R}^+$ be the output space, and $h : \mathbb{X} \mapsto \mathbb{Y}$ be the neural network estimating the braking distance of an aircraft. We consider the monotony property formulated in Definition 6.4.1. As stated earlier, we are dealing with the monotony with respect to the brakes' space. Hence, $\alpha \subseteq \mathbb{V}$ is made up of the ten features describing the state of the brakes and the partial order $\preceq$ on $\times_{v\in\alpha} \mathrm{D}(v)$ is as defined in Equation (6.15). We take advantage of the discrete nature of the brakes features: a natural partition $\mathcal{P}$ is to enumerate all the possible values for the ten brakes features. We have $|\mathcal{P}| = 225$.

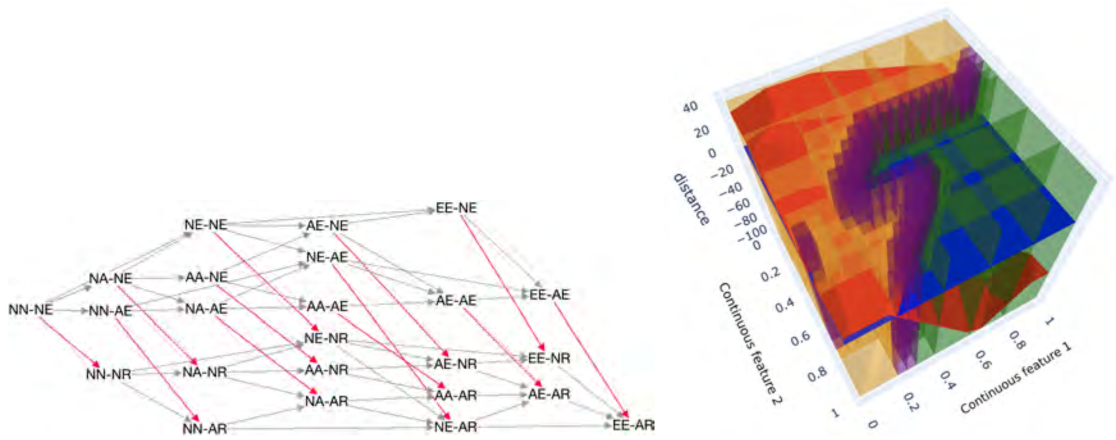

Figure 6.6: Example of visualization of features in $\alpha$ (left) and $\bar{\alpha}$ (right).

**Monotony Analysis** We run Algorithm 3 for five iterations with the setup described above. The algorithm is explained in Section 6.4. Here we only focus on the partitions used for the analysis and the refinement strategy, which are specific to the case study. Additionally, we will see how to capitalize on the data available at each iteration to perform some space exploration. $P_1$ is setup using $\mathcal{P}$, and $\preceq$; it represents the brakes sub-space. Then our partition's refining strategy is to start with the remaining discrete features (second iteration) and then consider the continuous features (the last three iterations). For the discrete features, the partitioning consists in enumerating the possible values, while for the continuous features, it consists of a uniform partition (finer through the iterations). To illustrate the impact of the refinement on the level of details of the analysis, we detail the total number of sub-spaces in each partition $P_t$: $|P_1| = 10800$, $|P_2| = 259200$, $|P_3| = 6480000$, $|P_4| = 25920000$ and $|P_5| = 103680000$.

Based on the partition and the outcomes of $\mathbf{F}(\cdot)$, the algorithm yields at each iteration the metrics $\underline{\Omega_t}$ and $\overline{\Omega_t}$. Nonetheless, for our case study, we put in place visualization means (see Figure 6.6). However, the relevant visualizations helping space exploration are case-dependent, so we do not propose any generic way to do it. Firstly, it might be relevant to visualize the sub-space composed of the features on which the partial order $\preceq$ is defined, *i.e.*, $\alpha$ corresponding to the brakes' space. It is modeled as a graph where the nodes are the brakes states (the elements of $\mathcal{P}$). The edges are the transitions between the states models by the partial order $\prec$ (the elements of $P_1$). With the outcomes of the first iteration, we can highlight the transitions which violate the monotony property. Then, to include the information of the formal verification of the following iteration in the space visualization, we plot some features versus the difference of distances $(h(x_1) - h(x_2))$ and visualize in which sub-spaces monotony issue occurs. These visualizations are helpful for exploration purposes after the analysis for the expert of the system (*e.g.*, if the expert can identify some place of interest within the space and wants to know what happens there).

Table 6.2: Values of $\underline{\Omega}$ and $\overline{\Omega}$ bounding the percentage of the space where the monotony property is violated.

| Metrics | It.1 | It.2 | It.3 | It.4 | It.5 |
|---|---|---|---|---|---|
| $\overline{\Omega_t}$ | 11.57% | 4.11% | 1.95% | 1.72% | 1.61% |
| $\underline{\Omega_t}$ | 0.03% | 0.45% | 1.12% | 1.29% | 1.39% |

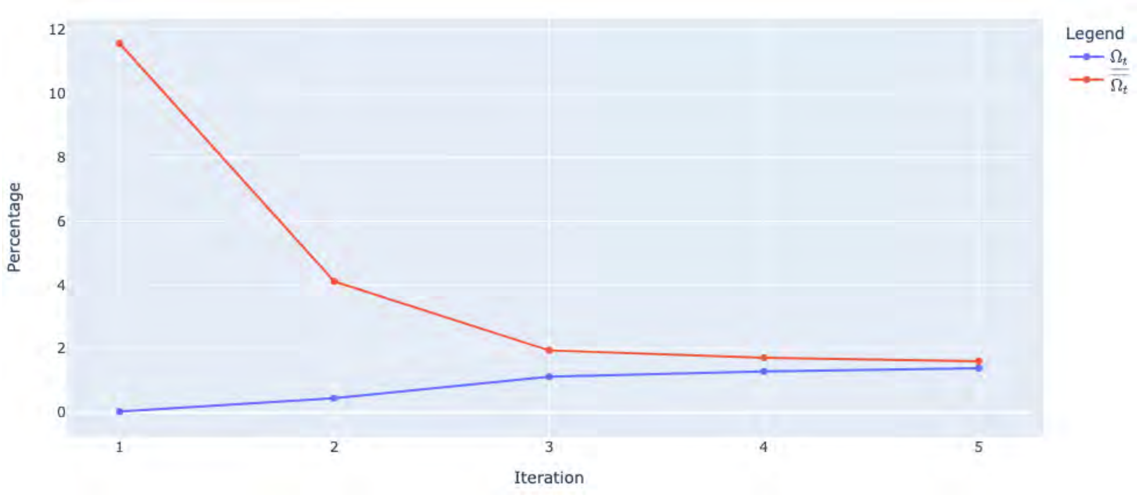

Figure 6.7: Evolution of $\underline{\Omega_t}$ and $\overline{\Omega_t}$

**Metrics: Non-Monotonic Space Coverage** At each iteration, we compute $\underline{\Omega_t}$ and $\overline{\Omega_t}$, which bound the ratio of the space where $h$ violates the monotony property (*i.e.*, $\omega$). The results are summarized in Table 6.2. In Figure 6.7, we can clearly observe the convergence of $\underline{\Omega_t}$ and $\overline{\Omega_t}$. At the first iteration, we can explain the notable gap between $\underline{\Omega_1}$ and $\overline{\Omega_1}$ by the rough partitioning of the space. Indeed, if the neural network is not monotonous, the larger the sub-space, the greater the chance to find a few couples $(x_1, x_2)$ for which the monotony property is violated or respected. That is why $\overline{\Omega_1}$ is large (numerous sub-spaces where the monotony property is partially respected) and $\underline{\Omega_1}$ small (only a few sub-spaces where the monotony property does not hold). We can notice a significant drop of $\overline{\Omega_t}$ compared to the rise of $\underline{\Omega_t}$: there are more sub-spaces where the monotony property holds than not. Eventually, the algorithm yields tight bounds; we obtain at the fifth iteration: $1.39\% \leq \omega \leq 1.61\%$. The stopping criterion of the algorithm may depend on various things such that the requirements of the system (*e.g.*, bounds tightness, max value to not cross for $\underline{\Omega}$ or min value to reach for $\overline{\Omega}$).

Through these five steps, we analyze the monotony of $h$ considering finer and finer partition of the space; we obtain: *(i)* metrics bounding the percentage of the space where the neural network is non-monotonic and *(ii)* the identification of the sub-spaces where the monotony issue occurs thanks to the formal verification of each element of the partitions.

We run our experiments on MacBook Pro 8 core 2,3 GHz Intel Core i9 with 32 Go of RAM. The MILP solver used is Gurobi 9 (Gurobi Optimization, LLC, 2022) and our monotony analysis took less than 10 hours.

## 6.6 Conclusion

In this chapter, we develop an iterative method to assess the monotony of a neural network using a MILP solver. The monotony property defined is suited for discrete features. This iterative method allows for lower and upper bounding the space where the neural network does not hold the property and formally identifies these areas. This is a step further in the demonstration that neural networks can preserve important function properties and, therefore, in the capability to embed ML technology in an aeronautical safety-critical system.

We applied this method to an aeronautical case study that consists in estimating the braking distance of an aircraft using a neural network mixing discrete and continuous inputs. We managed to quantify the percentage of the space where the neural network does not preserve the monotony property and to identify formally each sub-space where it occurs. In addition, we showed that we can capitalize on the available data to visualize the sub-spaces for helping the braking function's experts in processing the results of the algorithm.

From a higher perspective, this chapter proposes a new brick to be laid in the demonstration of conformity of ML-based systems. We partially address the objective LM-10 of EASA (2021b), which states that requirements-based verification should be performed on the ML-based system: the requirements are sorted into different "types" (*e.g.*, safety, functional, operational, or related to the learning process; see objective DA-02 and LM-02 of EASA (2021b)). To continue towards the completion of this verification objective, yet another perspective would be to generalize or adapt this analysis to verify other (types of) requirements.

# Conclusion

## Conclusion

In this thesis, we were interested in the certification of ML-based systems. We were evolving in a rich environment where several stakeholders were tackling the same subject (*e.g.*, EASA, the DEEL project[2], or Daedalean[3]). Despite the challenging context addressed by this research, we have been able to obtain several theoretical innovations that turn out to have a genuine industrial interest. We approached this demanding task by analyzing and defining the scope of our core research (objectives O1 and O2 from the introduction chapter) and then focusing on the issue of robustness and provability in the specific area of the compliance demonstration (objective O3).

Indeed, in Chapter 3, we perform a gap analysis between the demonstration of conformity of a classical system and that of an ML-based system, which raises several challenges. In light of our literature review on the different challenges and the expectations of authorities, we decided to focus on the safety aspects.

We addressed the safety issue in two complementary ways: from a design perspective — pre-learning (robustness challenge), and from a verification perspective — post-learning (provability challenge). In Chapter 5, we provide generalization bounds on the relaxed adversarial risks of ML models, which can be optimized during a robust learning phase. In other words, we provide a method to obtain ML models robust to adversarial attacks with probabilistic guarantees. It is consistent with the robustness and generalization requirements specified in EASA (2021b). In Chapter 6, we cope with the verification of the monotony property of a surrogate neural network. We propose a new way of analyzing the monotony property with mixed inputs (*i.e.*, integer and real). This new method leverages a MILP solver and provides lower and upper bounds on the proportion of the input space that violates the monotony property. With this technique, we partially answer to the objective concerning the verification of the ML model stated in EASA (2021b).

In a nutshell, each of our contributions was driven by the need to demonstrate compliance. In particular, we tackle trustworthiness requirements (such as generalization, robustness, or safety property verification), which reflect the essence of the certification, *i.e.*ensuring safety. For each of them, we have proposed innovative approaches that tackle the safety issue while answering to some requirements of the first guidance released by the EASA (EASA, 2021b).

[2]https://www.deel.ai/

[3]https://daedalean.ai/

## Perspectives

The work on the generalization bounds on the relaxed adversarial robust risks (Chapter 5) gives rise to many interesting questions and lines of future research. One idea would be to focus on extending our results to other classification settings, such as multiclass or multilabel.

Another line of research would be to focus on taking advantage of other tools of the PAC-Bayesian literature. Among them, one could make use of other bounds on the risk of the majority vote that take into consideration the diversity between the individual voters. For example, the C-bound (Lacasse *et al.*, 2006), or more recently the tandem loss (Masegosa *et al.*, 2020c). Another very recent PAC-Bayesian bound for majority votes that needs investigation in the case of adversarial robustness is the one proposed by Zantedeschi *et al.* (2021) that has the advantage to be directly optimizable with the 0-1 loss. In real-life applications, one often wants to combine different input sources (from different sensors, cameras, etc). Being able to combine these sources in an effective way is then a crucial issue.

In the aeronautical domain, the training distribution may differ from the Operational Design Domain. Indeed, the engineers may augment the occurrence of the corner cases during the training phase to reach sufficient performances and hence keep a high level of safety for these corner cases. Thus, an interesting perspective would be to consider domain adaptation in our robust PAC-Bayes setting to be able to handle different distributions and still have guarantees on the robustness of the model.

Concerning the work on the formal verification (Chapter 6), the proposed method leaves room for some improvements, such as tighter big-M values using incomplete methods providing bounds for the intermediate layers of the neural network (Xu *et al.*, 2020) or the use of asymmetric bounds (Tjeng *et al.*, 2019). As an extension of this monotony analysis, we plan to estimate the integral under the curve of $h(x_1) - h(x_2)$ in the sub-spaces where the monotony is violated by leveraging our definition of the monotony property. This would give a key indicator of the level of violation of the monotony property that could support the performance of the training phase. Another natural continuation of our work would be to extend the analysis we performed in a discrete context, to continuous features by using the formulation of the monotony based on the gradient. Our approach proved the suitability of monotony analysis to real-life industrial problems. Hence, a perspective would be first to develop an industrial tool with the current method that will, in the future, be completed with the new feature mentioned above.

To the best of our knowledge, the scalability of complete methods remains a challenge in the verification community and is mainly used with "shallow" neural networks. Therefore a lot remains to be done to keep up the verification to deep model. A perspective to accelerate MILP solver for neural network verification is to consider the dependency between the ReLU activations to prune the search tree of the MILP solver as proposed in Botoeva *et al.* (2020).

It is clear that a lot remains to be done in order to accommodate the presence of ML-based items in the certified systems. This issues thought provoking research questions, triggers innovating industrial practices, and opens the way to new practices in the context of avionics systems and not only. The current contribution is a humble step in that way and aims at showing the realism of such a vision. Finally, this thesis is a living proof that one of the keys to moving towards the demonstration of compliance of ML-based systems is the close collaboration between academia and industry.

# Bibliography

Julius ADEBAYO, Justin GILMER, Michael MUELLY, Ian J. GOODFELLOW, Moritz HARDT, and Been KIM (2018). Sanity Checks for Saliency Maps. *NeurIPS*
⤳ **Cited on pages 79, 80.**

Saleema AMERSHI, Andrew BEGEL, Christian BIRD, Robert DELINE, Harald C. GALL, Ece KAMAR, Nachiappan NAGAPPAN, Besmira NUSHI, and Thomas ZIMMERMANN (2019). Software Engineering for Machine Learning: A Case Study. *ICSE (SEIP)*, pp. 291–300
⤳ **Cited on page 61.**

Rob ASHMORE, Radu CALINESCU, and Colin PATERSON (2019). Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. *CoRR* vol. abs/1905.04223
⤳ **Cited on page 61.**

Dzmitry BAHDANAU, Kyunghyun CHO, and Yoshua BENGIO (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR*
⤳ **Cited on pages 1, 11.**

Peter L. BARTLETT and Shahar MENDELSON (2003). Rademacher and Gaussian Complexities: Risk Bounds and Structural Results. Vol. 3 no. null, pp. 463–482
⤳ **Cited on pages 31, 60, 77, 89.**

Denis BAYLOR, Eric BRECK, Heng-Tze CHENG, Noah FIEDEL, Chuan Yu FOO, Zakaria HAQUE, Salem HAYKAL, Mustafa ISPIR, Vihan JAIN, Levent KOC, Chiu Yuen KOO, Lukasz LEW, Clemens MEWALD, Akshay Naresh MODI, Neoklis POLYZOTIS, Sukriti RAMESH, Sudip ROY, Steven Euijong WHANG, Martin WICKE, Jarek WILKIEWICZ, Xin ZHANG, and Martin ZINKEVICH (2017). TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. *ACM SIGKDD*, pp. 1387–1395
⤳ **Cited on page 61.**

JM BIANNIC, G HARDIER, C ROOS, C SEREN, and L VERDIER (2016). Surrogate models for aircraft flight control: some off-line and embedded applications. *Aerospace Lab* no. 12
⤳ **Cited on page 106.**

Armin BIERE, Marijn HEULE, and Hans van MAAREN, eds. (2021). Handbook of satisfiability. Second edition. Frontiers in Artificial Intelligence and Applications volume

336. Amsterdam ; Washington, DC: IOS Press
⤳ **Cited on page 42.**

Christopher M. Bishop (2006). Pattern recognition and machine learning. Springer
⤳ **Cited on page 25.**

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener (2020). Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis. *Proceedings of the AAAI Conference on Artificial Intelligence* vol. 34 no. 04, pp. 3291–3299. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5729
⤳ **Cited on pages 8, 124.**

Stephen P. Boyd and Lieven Vandenberghe (2004). Convex optimization. Cambridge, UK ; New York: Cambridge University Press. ISBN: 9780521833783
⤳ **Cited on page 44.**

Abderrahmane Brahmi, David Delmas, Mohamed Habib Essoussi, Famantanantsoa Randimbivololona, Abdellatif Atki, and Thomas Marie (2018). Formalise to automate: deployment of a safe and cost-efficient process for avionics software. *Embedded Real Time Systems (ERTS)*. URL: https://hal.archives-ouvertes.fr/hal-01708332
⤳ **Cited on page 38.**

Li Cai and Yangyong Zhu (2015). The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. *Data Science Journal* vol. 14
⤳ **Cited on page 61.**

Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian J. Goodfellow, Aleksander Madry, and Alexey Kurakin (2019). On Evaluating Adversarial Robustness. *CoRR* vol. abs/1902.06705
⤳ **Cited on pages 60, 73.**

Nicholas Carlini and David A. Wagner (2017). Towards Evaluating the Robustness of Neural Networks. *IEEE SP*, pp. 39–57
⤳ **Cited on pages 67, 69, 72, 88, 106.**

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh (2017). ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models. *AISec@CCS 2017*, pp. 15–26
⤳ **Cited on page 70.**

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess (2017). Maximum Resilience of Artificial Neural Networks. *Automated Technology for Verification and Anal-*

*ysis*, pp. 251–268
⤳ **Cited on page 111.**

Moustapha CISSÉ, Piotr BOJANOWSKI, Edouard GRAVE, Yann N. DAUPHIN, and Nicolas USUNIER (2017). Parseval Networks: Improving Robustness to Adversarial Examples. *ICML*. Vol. 70, pp. 854–863
⤳ **Cited on pages 60, 67, 72, 74.**

Jeremy COHEN, Elan ROSENFELD, and Zico KOLTER (2019). Certified Adversarial Robustness via Randomized Smoothing. *ICML*
⤳ **Cited on page 90.**

Michele CONFORTI, Gérard CORNUÉJOLS, and Giacomo ZAMBELLI (2014). Integer Programming. 1st ed. 2014. Graduate Texts in Mathematics 271. Cham: Springer International Publishing : Imprint: Springer
⤳ **Cited on page 44.**

Stephen A. COOK (1971). The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. ACM Press, pp. 151–158
⤳ **Cited on page 41.**

Patrick COUSOT (2021). Principles of abstract interpretation. The MIT Press
⤳ **Cited on page 45.**

Patrick COUSOT and Radhia COUSOT (2010). A gentle introduction to formal verification of computer systems by abstract interpretation. *Logics and Languages for Reliability and Security*. Vol. 25, pp. 1–29. URL: https://doi.org/10.3233/978-1-60750-100-8-1
⤳ **Cited on page 45.**

Piotr DABKOWSKI and Yarin GAL (2017). Real Time Image Saliency for Black Box Classifiers. *NeurIPS*, pp. 9525–9536
⤳ **Cited on pages 59, 78–81.**

Mathieu DAMOUR, Florence De GRANCEY, Christophe GABREAU, Adrien GAUFFRIAU, Jean-Brice GINESTET, Alexandre HERVIEU, Thomas HURAUX, Claire PAGETTI, Ludovic PONSOLLE, and Arthur CLAVIÈRE (2021). Towards Certification of a Reduced Footprint ACAS-Xu System: A Hybrid ML-Based Solution. *Computer Safety, Reliability, and Security - 40th International Conference, SAFECOMP 2021, York, UK, September 8-10, 2021, Proceedings*. Vol. 12852, pp. 34–48. URL: https://doi.org/10.1007/978-3-030-83903-1%5C_3
⤳ **Cited on pages 63, 109.**

Martin DAVIS, George LOGEMANN, and Donald LOVELAND (1962). A Machine Program for Theorem-Proving. *Commun. ACM* vol. 5 no. 7, pp. 394–397. DOI: `10.1145/368273.368557`. URL: `https://doi.org/10.1145/368273.368557`
⤳ **Cited on page 41**.

Jacob DEVLIN, Ming-Wei CHANG, Kenton LEE, and Kristina TOUTANOVA (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACL-HLT*, pp. 4171–4186
⤳ **Cited on pages 1, 11**.

Gintare Karolina DZIUGAITE, Kyle HSU, Waseem GHARBIEH, Gabriel ARPINO, and Daniel ROY (2021). On the role of data in PAC-Bayes. *AISTATS*
⤳ **Cited on page 97**.

Gintare Karolina DZIUGAITE and Daniel ROY (2018). Data-dependent PAC-Bayes priors via differential privacy. *NeurIPS*
⤳ **Cited on page 97**.

Gintare Karolina DZIUGAITE and Daniel M. ROY (2017). Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data. *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. URL: `http://auai.org/uai2017/proceedings/papers/173.pdf`
⤳ **Cited on pages 31, 60, 77**.

EASA (2020). EASA Artificial Intelligence Roadmap 1.0. URL: `https://www.easa.europa.eu/sites/default/files/dfu/EASA-AI-Roadmap-v1.0.pdf`
⤳ **Cited on pages 2, 12, 77**.

EASA (2021a). CS-25 Amendment 27. URL: `https://www.easa.europa.eu/downloads/136622/en`
⤳ **Cited on pages 52–54, 56**.

EASA (2021b). EASA Concept Paper: First usable guidance for Level 1 machine learning applications. URL: `https://www.easa.europa.eu/downloads/134357/en`
⤳ **Cited on pages 2, 7, 12, 22, 56, 59, 63, 66, 73, 77, 83, 88, 102, 120, 123**.

Farzan FARNIA, Jesse ZHANG, and David TSE (2019). Generalizable Adversarial Training via Spectral Normalization. *ICLR*
⤳ **Cited on page 90**.

Ad J FEELDERS (2000). Prior knowledge in economic applications of data mining. *European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 395–400
⤳ **Cited on page 108**.

Ruth C. Fong and Andrea Vedaldi (2017). Interpretable Explanations of Black Boxes by Meaningful Perturbation. *ICCV*, pp. 3449–3457
⤳ **Cited on pages 59, 78–80.**

Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli (2004). DPLL(T): Fast Decision Procedures. *Computer Aided Verification*. Springer Berlin Heidelberg, pp. 175–188
⤳ **Cited on page 42.**

Damien Garreau and Ulrike von Luxburg (2020). Explaining the Explainer: A First Theoretical Analysis of LIME. *AISTATS*. Vol. 108, pp. 1287–1296
⤳ **Cited on page 79.**

Adrien Gauffriau, François Malgouyres, and Mélanie Ducoffe (2021). Overestimation learning with guarantees. *arXiv preprint arXiv:2101.11717*
⤳ **Cited on page 108.**

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev (2018). AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. *IEEE SP*, pp. 3–18
⤳ **Cited on pages 60, 73–75, 88.**

Pascal Germain, Francis Bach, Alexandre Lacoste, and Simon Lacoste-Julien (2016). PAC-Bayesian Theory Meets Bayesian Inference. *Advances in Neural Information Processing Systems*. Vol. 29
⤳ **Cited on page 31.**

Pascal Germain, Alexandre Lacasse, François Laviolette, and Mario Marchand (2009). PAC-Bayesian learning of linear classifiers. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*. ACM Press, pp. 1–8
⤳ **Cited on page 34.**

Pascal Germain, Alexandre Lacasse, François Laviolette, Mario Marchand, and Jean-Francis Roy (2015). Risk Bounds for the Majority Vote: From a PAC-Bayesian Analysis to a Learning Algorithm. *J. Mach. Learn. Res.* vol. 16, pp. 787–860
⤳ **Cited on pages 31, 33, 60, 150.**

Justin Gilmer, Nicolas Ford, Nicholas Carlini, and Ekin D. Cubuk (2019). Adversarial Examples Are a Natural Consequence of Test Error in Noise. *ICML*. Vol. 97, pp. 2280–2289
⤳ **Cited on pages 60, 66, 73.**

Xavier Glorot and Yoshua Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. *AISTATS*
⤳ **Cited on page 100.**

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy (2015). Explaining and Harnessing Adversarial Examples. *ICLR*
⤳ **Cited on pages 60, 66–68, 70, 88.**

Pascale Gourdeau, Varun Kanade, Marta Kwiatkowska, and James Worrell (2019). On the Hardness of Robust Classification. *NeurIPS*, pp. 7444–7453
⤳ **Cited on pages 66, 72.**

Anil Goyal, Emilie Morvant, Pascal Germain, and Massih-Reza Amini (2019). Multiview Boosting by Controlling the Diversity and the Accuracy of View-specific Voters. *Neurocomputing*
⤳ **Cited on page 102.**

Ignacio E. Grossmann (2002). Review of Nonlinear Mixed-Integer and Disjunctive Programming Techniques. *Optimization and Engineering*
⤳ **Cited on page 111.**

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti (2018). Local Rule-Based Explanations of Black Box Decision Systems. *CoRR* vol. abs/1806.09936
⤳ **Cited on pages 59, 78, 79.**

Akhil Gupta, Naman Shukla, Lavanya Marla, Arinbjörn Kolbeinsson, and Kartik Yellepeddi (2019). How to Incorporate Monotonicity in Deep Networks While Preserving Flexibility? URL: https://arxiv.org/abs/1909.10662
⤳ **Cited on page 108.**

Gurobi Optimization, LLC (2022). Gurobi Optimizer Reference Manual. URL: https://www.gurobi.com
⤳ **Cited on pages 44, 119.**

Jia Hao, Wenbin Ye, Liangyue Jia, Guoxin Wang, and Janet Allen (2021). Building surrogate models for engineering problems by integrating limited simulation data and monotonic engineering knowledge. *Advanced Engineering Informatics* vol. 49, p. 101342
⤳ **Cited on page 108.**

Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell (2016). Generating Visual Explanations. *ECCV*. Vol. 9908, pp. 3–19
⤳ **Cited on pages 59, 78, 81, 82.**

Dan Hendrycks and Thomas Dietterich (2019). Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. *ICLR*
⤳ **Cited on pages 90, 92.**

Sepp HOCHREITER and Jürgen SCHMIDHUBER (1997). Long Short-term Memory. *Neural computation* vol. 9, pp. 1735–80
⤳ **Cited on page 81.**

Qian HUANG, Isay KATSMAN, Zeqi GU, Horace HE, Serge J. BELONGIE, and Ser-Nam LIM (2019). Enhancing Adversarial Example Transferability With an Intermediate Level Attack. *ICCV*, pp. 4732–4741
⤳ **Cited on pages 67, 70.**

Xiaowei HUANG, Daniel KROENING, Wenjie RUAN, James SHARP, Youcheng SUN, Emese THAMO, Min WU, and Xinping YI (2020). A Survey of Safety and Trustworthiness of Deep Neural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability. *Computer Science Review* vol. 37, pp. 100–270
⤳ **Cited on page 88.**

Xiaowei HUANG, Marta KWIATKOWSKA, Sen WANG, and Min WU (2017). Safety Verification of Deep Neural Networks. *Computer Aided Verification CAV*. Vol. 10426, pp. 3–29
⤳ **Cited on pages 60, 73, 74, 76, 88.**

Zong-De JIAN, Hung-Jui CHANG, Tsan-sheng HSU, and Da-Wei WANG (2017). Learning from Simulated World - Surrogates Construction with Deep Neural Network. *Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*
⤳ **Cited on page 106.**

Jørgen KARPF (1991). Inductive modelling in law: example based expert systems in administrative law. *Proceedings of the 3rd international conference on Artificial intelligence and law*, pp. 297–306
⤳ **Cited on page 108.**

Guy KATZ, Clark W. BARRETT, David L. DILL, Kyle JULIAN, and Mykel J. KOCHENDERFER (2017). Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *Computer Aided Verification CAV*. Vol. 10426, pp. 97–117
⤳ **Cited on pages 60, 73–75, 88.**

Guy KATZ, Derek A. HUANG, Duligur IBELING, Kyle JULIAN, Christopher LAZARUS, Rachel LIM, Parth SHAH, Shantanu THAKOOR, Haoze WU, Aleksandar ZELJIC, David L. DILL, Mykel J. KOCHENDERFER, and Clark W. BARRETT (2019). The Marabou Framework for Verification and Analysis of Deep Neural Networks. *Computer Aided Verification (CAV)*. Vol. 11561, pp. 443–452
⤳ **Cited on pages 60, 73, 74, 106.**

Alex KENDALL and Yarin GAL (2017). What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? *NeurIPS*, pp. 5574–5584
⤳ **Cited on pages 59, 78.**

Justin Khim and Po-Ling Loh (2018). Adversarial Risk Bounds for Binary Classification via Function Transformation. *CoRR*
⤳ **Cited on pages 77, 90.**

Diederik Kingma and Jimmy Ba (2015). Adam: A Method for Stochastic Optimization. *ICLR*
⤳ **Cited on page 100.**

Pang Wei Koh and Percy Liang (2017). Understanding Black-box Predictions via Influence Functions. *ICML*. Vol. 70, pp. 1885–1894
⤳ **Cited on pages 60, 66, 78, 81.**

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulò (2016). Deep Neural Decision Forests. *IJCAI*
⤳ **Cited on pages 96, 97.**

A. Krizhevsky (2009). Learning Multiple Layers of Features from Tiny Images
⤳ **Cited on page 72.**

Jim Krodel (2008). Technology Changes In Aeronautical Systems. *Embedded Real Time Software and Systems (ERTS)*. URL: `https://hal.archives-ouvertes.fr/hal-02269834`
⤳ **Cited on pages 1, 11.**

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio (2017a). Adversarial examples in the physical world. *ICLR*
⤳ **Cited on pages 66, 70.**

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio (2017b). Adversarial Machine Learning at Scale. *ICLR*
⤳ **Cited on pages 60, 66–68, 70, 88, 99.**

Alexandre Lacasse, François Laviolette, Mario Marchand, Pascal Germain, and Nicolas Usunier (2006). PAC-Bayes Bounds for the Risk of the Majority Vote and the Variance of the Gibbs Classifier. *NIPS*
⤳ **Cited on pages 8, 124.**

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998a). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* vol. 86 no. 11, pp. 2278–2323
⤳ **Cited on page 72.**

Yann LeCun, Corinna Cortes, and Christopher Burges (1998b). THE MNIST DATASET of handwritten digits. URL: `http://yann.lecun.com/exdb/mnist/`
⤳ **Cited on page 100.**

Guy Lever, François Laviolette, and John Shawe-Taylor (2013). Tighter PAC-Bayes bounds through distribution-dependent priors. *Theoretical Computer Science*
⤳ **Cited on page 97.**

Qizhang Li, Yiwen Guo, and Hao Chen (2020). Practical No-box Adversarial Attacks against DNNs. *NeurIPS 2020*
⤳ **Cited on pages 67, 70.**

Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu (2020). Certified monotonic neural networks. *Advances in Neural Information Processing Systems* vol. 33, pp. 15427–15438
⤳ **Cited on page 108.**

Stephan Sloth Lorenzen, Christian Igel, and Yevgeny Seldin (2019). On PAC-Bayesian bounds for random forests. *Machine Learning*
⤳ **Cited on page 102.**

Scott M. Lundberg and Su-In Lee (2017). A Unified Approach to Interpreting Model Predictions. *NeurIPS*, pp. 4765–4774
⤳ **Cited on pages 59, 78, 82.**

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu (2018). Towards Deep Learning Models Resistant to Adversarial Attacks. *ICLR*
⤳ **Cited on pages 60, 66–68, 70, 71, 88, 99, 106.**

Franck Mamalet, Eric Jenn, Gregory FLANDIN, Hervé Delseny, Christophe Gabreau, Adrien Gauffriau, Bernard Beaudouin, Ludovic Ponsolle, Lucian Alecu, Hugues Bonnin, Brice Beltran, Didier Duchel, Jean-Brice Ginestet, Alexandre Hervieu, Sylvain Pasquet, Kevin Delmas, Claire Pagetti, Jean-Marc Gabriel, Camille Chapdelaine, Sylvaine Picard, Mathieu Damour, Cyril Cappi, Laurent Gardès, Florence De Grancey, Baptiste Lefevre, Sébastien Gerchinovitz, and Alexandre Albore (2021). White Paper Machine Learning in Certified Systems. URL: `https://hal.archives-ouvertes.fr/hal-03176080/file/White_Paper_Machine_Learning_in_Certified_System_DEEL_project_v2.0.pdf`
⤳ **Cited on pages 56, 109.**

Guido Manfredi and Yannick Jestin (2016). An introduction to ACAS Xu and the challenges ahead. *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pp. 1–9
⤳ **Cited on page 74.**

Silvano Martello and Paolo Toth (1990). Knapsack problems: algorithms and computer implementations. Wiley-Interscience series in discrete mathematics and optimiza-

tion. Chichester ; New York: J. Wiley & Sons. ISBN: 9780471924203
↝ **Cited on page 44.**

Robert MARTIN (2017). Assured Software - A journey and discussion. URL: https://www.his-2019.co.uk/session/cwe-cve-its-history-and-future
↝ **Cited on page 62.**

Andres MASEGOSA, Stephan LORENZEN, Christian IGEL, and Yevgeny SELDIN (2020a). Second Order PAC-Bayesian Bounds for the Weighted Majority Vote. *Advances in Neural Information Processing Systems*. Vol. 33, pp. 5263–5273. URL: https://proceedings.neurips.cc/paper/2020/file/386854131f58a556343e056f03626e00-Paper.pdf
↝ **Cited on pages 31, 60.**

Andres MASEGOSA, Stephan LORENZEN, Christian IGEL, and Yevgeny SELDIN (2020b). Second Order PAC-Bayesian Bounds for the Weighted Majority Vote. *Advances in Neural Information Processing Systems*. Ed. by H. LAROCHELLE, M. RANZATO, R. HADSELL, M.F. BALCAN, and H. LIN. Vol. 33. Curran Associates, Inc., pp. 5263–5273
↝ **Cited on page 31.**

Andrés MASEGOSA, Stephan Sloth LORENZEN, Christian IGEL, and Yevgeny SELDIN (2020c). Second Order PAC-Bayesian Bounds for the Weighted Majority Vote. *NeurIPS*
↝ **Cited on pages 8, 97, 124.**

David A. MCALLESTER (1999). Some PAC-Bayesian Theorems. *Mach. Learn.* vol. 37 no. 3, pp. 355–363
↝ **Cited on pages 31, 34, 60, 77, 88, 95.**

Antoine MINÉ (2017). Tutorial on static inference of numeric invariants by abstract interpretation. eng. OCLC: 1053837496. Hanover, Massachusetts: Now Publishers. ISBN: 9781680833874
↝ **Cited on page 45.**

Matthew MIRMAN, Timon GEHR, and Martin T. VECHEV (2018). Differentiable Abstract Interpretation for Provably Robust Neural Networks. *ICML*. Vol. 80, pp. 3575–3583
↝ **Cited on page 60.**

Jean François MONIN and Michael G. HINCHEY (2003). Understanding formal methods. Springer
↝ **Cited on page 40.**

Omar MONTASSER, Steve HANNEKE, and Nathan SREBRO (2019). VC Classes are Adversarially Robustly Learnable, but Only Improperly. *COLT*
↝ **Cited on page 90.**

Omar Montasser, Steve Hanneke, and Nathan Srebro (2020). Reducing Adversarially Robust Learning to Non-Robust PAC Learning. *NeurIPS*
⤳ **Cited on pages 77, 90.**

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard (2016). DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. *Computer Vision and Pattern Recognition CVPR*, pp. 2574–2582
⤳ **Cited on pages 67, 69.**

Emilie Morvant, Amaury Habrard, and Stéphane Ayache (2014). Majority Vote of Diverse Classifiers for Late Fusion. *S+SSPR*
⤳ **Cited on page 102.**

Yannick Moy, Emmanuel Ledinot, Hervé Delseny, Virginie Wiels, and Benjamin Monate (2013). Testing or Formal Verification: DO-178C Alternatives and Industrial Experience. *IEEE Software* vol. 30, pp. 50–57
⤳ **Cited on pages 38, 45.**

Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev (2022). PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. *Proc. ACM Program. Lang.* vol. 6 no. POPL. URL: https://doi.org/10.1145/3498704
⤳ **Cited on pages 73, 74, 76, 106.**

Vaishnavh Nagarajan and J. Zico Kolter (2019). Uniform convergence may be unable to explain generalization in deep learning. *NeurIPS*
⤳ **Cited on page 90.**

An-phi Nguyen and María Rodríguez Martínez (2019). Mononet: towards interpretable models by learning monotonic features. *arXiv preprint arXiv:1909.13611*
⤳ **Cited on page 108.**

Yuki Ohnishi and Jean Honorio (2021). Novel Change of Measure Inequalities with Applications to PAC-Bayesian Bounds and Monte Carlo Estimation. *AISTATS*
⤳ **Cited on pages 148, 150.**

Matan Ostrovsky, Clark W. Barrett, and Guy Katz (2022). An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks. *CoRR*. URL: https://arxiv.org/abs/2201.01978
⤳ **Cited on pages 73–75.**

Alessandro De Palma, Harkirat Behl, Rudy R Bunel, Philip Torr, and M. Pawan Kumar (2021). Scaling the Convex Barrier with Active Sets. *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=

uQfOy7Lr1TR
⟿ **Cited on page 76**.

Nicolas PAPERNOT and Patrick D. MCDANIEL (2018). Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning. *CoRR* vol. abs/1803.04765
⟿ **Cited on pages 60, 66, 67, 71, 72**.

Nicolas PAPERNOT, Patrick D. MCDANIEL, Somesh JHA, Matt FREDRIKSON, Z. Berkay CELIK, and Ananthram SWAMI (2016). The Limitations of Deep Learning in Adversarial Settings. *IEEE EuroS&P*, pp. 372–387
⟿ **Cited on pages 60, 66, 67, 69, 88**.

Emilio PARRADO-HERNÁNDEZ, Amiran AMBROLADZE, John SHAWE-TAYLOR, and Shiliang SUN (2012). PAC-Bayes Bounds with Data Dependent Priors. *Journal of Machine Learning Research* vol. 13 no. 112, pp. 3507–3531
⟿ **Cited on pages 31, 60, 77, 97**.

Colin PATERSON, Haoze WU, John GRESE, Radu CALINESCU, Corina S. PĂSĂREANU, and Clark BARRETT (2021). DeepCert: Verification of Contextually Relevant Robustness for Neural Network Image Classifiers. *Computer Safety, Reliability, and Security: 40th International Conference, SAFECOMP 2021, York, UK, September 8–10, 2021, Proceedings*, pp. 3–17. URL: `https://doi.org/10.1007/978-3-030-83903-1_5`
⟿ **Cited on pages 73, 74, 77**.

E.M. PETERSON, M. DEVORE, J. COOPER, and G. CARR (2020). Run Time Assurance as an Alternate Concept to Contemporary Development Assurance Processes. NASA/CR-2020-220586
⟿ **Cited on page 107**.

P PHILLIPS, Amanda HAHN, Peter FONTANA, David BRONIATOWSKI, and Mark PRZYBOCKI (2020). Four Principles of Explainable Artificial Intelligence (Draft)
⟿ **Cited on page 59**.

Sylvaine PICARD, Camille CHAPDELAINE, Cyril CAPPI, Laurent GARDES, Eric JENN, Baptiste LEFÈVRE, and Thomas SOUMARMON (2020). Ensuring Dataset Quality for Machine Learning Certification. *ISSRE*, pp. 275–282
⟿ **Cited on page 61**.

Nicolai PISARUK (2019). Mixed Integer Programming: Models and Methods
⟿ **Cited on page 44**.

Aditi RAGHUNATHAN, Jacob STEINHARDT, and Percy S LIANG (2018). Semidefinite relaxations for certifying robustness to adversarial examples. *Advances in Neural Information Processing Systems*, pp. 10877–10887
⟿ **Cited on pages 106, 108**.

Liva Ralaivola, Marie Szafranski, and Guillaume Stempfel (2010). Chromatic PAC-Bayes Bounds for Non-IID Data: Applications to Ranking and Stationary β-Mixing Processes. *JMLR*
⤳ **Cited on pages 94, 149.**

Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi (2016). You Only Look Once: Unified, Real-Time Object Detection. *CVPR*, pp. 779–788
⤳ **Cited on pages 1, 11.**

David Reeb, Andreas Doerr, Sebastian Gerwinn, and Barbara Rakitsch (2018). Learning Gaussian Processes by Minimizing PAC-Bayesian Generalization Bounds. *NeurIPS*
⤳ **Cited on page 98.**

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *SIGKDD*, pp. 1135–1144
⤳ **Cited on pages 59, 78, 79.**

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin (2018). Anchors: High-Precision Model-Agnostic Explanations. *AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*, pp. 1527–1535
⤳ **Cited on pages 59, 78, 79.**

Jean-Francis Roy, François Laviolette, and Mario Marchand (2011). From PAC-Bayes Bounds to Quadratic Programs for Majority Votes. *ICML*
⤳ **Cited on page 102.**

John Rushby (2015). The Interpretation and Evaluation of Assurance Cases. Tech. rep. URL: `http://www.csl.sri.com/users/rushby/papers/sri-csl-15-1-assurance-cases.pdf`
⤳ **Cited on page 62.**

Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sébastien Bubeck, and Greg Yang (2019a). Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers. *NeurIPS*
⤳ **Cited on page 90.**

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang (2019b). A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks. *NeurIPS*, pp. 9832–9842
⤳ **Cited on pages 60, 76.**

Edward Scheinerman and Daniel Ullman (2011). Fractional Graph Theory: A Rational Approach to the Theory of Graphs. Courier Corporation
⤳ **Cited on page 149.**

Matthias SEEGER (2002). PAC-Bayesian Generalisation Error Bounds for Gaussian Process Classification. *JMLR*
⤳ **Cited on page 95.**

Vera SHALAEVA, Alireza FAKHRIZADEH ESFAHANI, Pascal GERMAIN, and Mihaly PETRECZKY (2020). Improved PAC-Bayesian Bounds for Linear Regression. *Proceedings of the AAAI Conference on Artificial Intelligence* vol. 34 no. 04, pp. 5660–5667
⤳ **Cited on page 31.**

John SHAWE-TAYLOR and Robert C. WILLIAMSON (1997). A PAC Analysis of a Bayesian Estimator. *COLT*. Ed. by Yoav FREUND and Robert E. SCHAPIRE, pp. 2–9
⤳ **Cited on pages 31, 60, 77, 88.**

Avanti SHRIKUMAR, Peyton GREENSIDE, and Anshul KUNDAJE (2017). Learning Important Features Through Propagating Activation Differences. *ICML*. Vol. 70, pp. 3145–3153
⤳ **Cited on pages 59, 78, 82, 83.**

David SHRIVER, Sebastian ELBAUM, and Matthew B. DWYER (2021). DNNV: A Framework for Deep Neural Network Verification. *Computer Aided Verification, CAV*, pp. 137–150
⤳ **Cited on page 74.**

Karen SIMONYAN and Andrew ZISSERMAN (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. URL: `http://arxiv.org/abs/1409.1556`
⤳ **Cited on page 81.**

Gagandeep SINGH, Rupanshu GANVIR, Markus PÜSCHEL, and Martin T. VECHEV (2019a). Beyond the Single Neuron Convex Barrier for Neural Network Certification. *NeurIPS*
⤳ **Cited on page 76.**

Gagandeep SINGH, Timon GEHR, Markus PÜSCHEL, and Martin VECHEV (2019b). Boosting Robustness Certification of Neural Networks. *ICLR*
⤳ **Cited on pages 60, 73–75, 88.**

Michael SIPSER (2013). Introduction to the theory of computation. eng. 3. ed. Andover: Cengage Learning. ISBN: 9781133187790
⤳ **Cited on page 44.**

Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny (2009). Formal Verification of Avionics Software Products. *Formal Methods (FM)*, pp. 532–546
⤳ **Cited on pages 38, 45.**

Stefan Studer, Thanh Binh Bui, Christian Drescher, Alexander Hanuschkin, Ludwig Winkler, Steven Peters, and Klaus-Robert Müller (2020). Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology. *CoRR* vol. abs/2003.05155
⤳ **Cited on page 61.**

Oleg Sudakov, Dmitri Koroteev, Boris Belozerov, and Evgeny Burnaev (2019). Artificial neural network surrogate modeling of oil reservoir: a case study. *International Symposium on Neural Networks (ISNN)*, pp. 232–241
⤳ **Cited on page 106.**

Shiliang Sun, John Shawe-Taylor, and Liang Mao (2017). PAC-Bayes analysis of multi-view learning. *Inf. Fusion*
⤳ **Cited on page 102.**

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus (2014). Intriguing properties of neural networks. *ICLR*
⤳ **Cited on pages 60, 66, 67, 69, 88, 91.**

Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, KRUNAL KISHOR PATEL, and Juan Pablo Vielma (2020). The Convex Relaxation Barrier, Revisited: Tightened Single-Neuron Relaxations for Neural Network Verification. *NeurIPS*, pp. 21675–21686. URL: `https://proceedings.neurips.cc/paper/2020/file/f6c2a0c4b566bc99d596e58638e342b0-Paper.pdf`
⤳ **Cited on page 76.**

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake (2019). Evaluating Robustness of Neural Networks with Mixed Integer Programming. *ICLR*
⤳ **Cited on pages 8, 60, 73, 74, 106, 124.**

G. S. Tseitin (1983). On the Complexity of Derivation in Propositional Calculus. *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Springer Berlin Heidelberg, pp. 466–483
⤳ **Cited on page 42.**

Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama (2018). Lipschitz-Margin Training: Scalable Certification of Perturbation Invariance for Deep Neural Networks. *NeurIPS*, pp. 6542–6551
⤳ **Cited on pages 60, 67, 72, 74, 88, 106.**

Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang (2020). Perfectly parallel fairness certification of neural networks. *Proceedings of the ACM on Programming Languages* vol. 4 no. OOPSLA, pp. 1–30
⤳ **Cited on page 108.**

Caterina Urban and Antoine Miné (2021). A Review of Formal Methods applied to Machine Learning. *CoRR*. URL: `https://arxiv.org/abs/2104.02466`
⤳ **Cited on pages 74, 108.**

L. G. Valiant (1984). A Theory of the Learnable. *Commun. ACM* vol. 27 no. 11, pp. 1134–1142
⤳ **Cited on pages 31, 60.**

V. N. Vapnik and A. Ya. Chervonenkis (1971). On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability & Its Applications* vol. 16 no. 2, pp. 264–280. URL: `https://doi.org/10.1137/1116025`
⤳ **Cited on pages 31, 60, 77, 88.**

Paul Viallard, Eric Guillaume Vidot, Amaury Habrard, and Emilie Morvant (2021). A PAC-Bayes Analysis of Adversarial Robustness. *Advances in Neural Information Processing Systems (NeurIPS 2021)*, pp. 14421–14433. URL: `https://proceedings.neurips.cc/paper/2021/file/78e8dffe65a2898eef68a33b8db35b78-Paper.pdf`
⤳ **Cited on pages 77, 87.**

Guillaume Vidot, Mélanie Ducoffe, Christophe Gabreau, Ileana Ober, and Iulian Ober (2022). Formal Monotony Analysis of Neural Networks with Mixed Inputs: An Asset for Certification. *Formal Methods for Industrial Critical Systems (FMICS 2022)*. Ed. by Jan Friso Groote and Marieke Huisman. Springer International Publishing, pp. 15–31
⤳ **Cited on page 105.**

Guillaume Vidot, Christophe Gabreau, Ileana Ober, and Iulian Ober (2021). Certification of embedded systems based on Machine Learning: A survey. URL: `https://arxiv.org/abs/2106.07221`
⤳ **Cited on pages 51, 65.**

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana (2018). Formal Security Analysis of Neural Networks using Symbolic Intervals. *USENIX Security*, pp. 1599–1614
⤳ **Cited on pages 60, 73, 75, 106, 108.**

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter (2021). Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. *NeurIPS*. URL: `https://proceedings.neurips.cc/paper/2021/hash/fac7fead96dafceaf80c1daffeae82a4-`

`Abstract.html`
⤳ **Cited on pages 73–77, 106.**

Tsui-Wei WENG, Huan ZHANG, Hongge CHEN, Zhao SONG, Cho-Jui HSIEH, Duane BONING, Inderjit S DHILLON, and Luca DANIEL (2018). Towards fast computation of certified robustness for ReLU networks. *arXiv preprint arXiv:1804.09699*
⤳ **Cited on pages 106, 108.**

Weiming XIANG, Hoang-Dung TRAN, and Taylor T JOHNSON (2018). Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems* vol. 29 no. 11, pp. 5777–5783
⤳ **Cited on page 108.**

Han XIAO, Kashif RASUL, and Roland VOLLGRAF (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*
⤳ **Cited on page 100.**

Kaidi XU, Zhouxing SHI, Huan ZHANG, Yihan WANG, Kai-Wei CHANG, Minlie HUANG, Bhavya KAILKHURA, Xue LIN, and Cho-Jui HSIEH (2020). Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. *NeurIPS*
⤳ **Cited on pages 8, 73–76, 106, 124.**

Dong YIN, Kannan RAMCHANDRAN, and Peter BARTLETT (2019). Rademacher Complexity for Adversarially Robust Generalization. *ICML*
⤳ **Cited on pages 77, 90.**

Valentina ZANTEDESCHI, Maria-Irina NICOLAE, and Ambrish RAWAT (2017). Efficient Defenses Against Adversarial Attacks. *ACM Workshop on Artificial Intelligence and Security, AISec@CCS*
⤳ **Cited on pages 67, 72, 88, 90, 92.**

Valentina ZANTEDESCHI, Paul VIALLARD, Emilie MORVANT, Rémi EMONET, Amaury HABRARD, Pascal GERMAIN, and Benjamin GUEDJ (2021). Learning Stochastic Majority Votes by Minimizing a PAC-Bayes Generalization Bound. *NeurIPS*
⤳ **Cited on pages 8, 31, 124.**

Matthew D. ZEILER and Rob FERGUS (2014). Visualizing and Understanding Convolutional Networks. *ECCV*. Vol. 8689, pp. 818–833
⤳ **Cited on pages 59, 78, 80, 81.**

Dinghuai ZHANG, Tianyuan ZHANG, Yiping LU, Zhanxing ZHU, and Bin DONG (2019a). You Only Propagate Once: Accelerating Adversarial Training via Maximal Principle. *NeurIPS*, pp. 227–238
⤳ **Cited on pages 60, 66, 67, 71.**

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel (2018). Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems (NeurIPS*, pp. 4939–4948
⇝ **Cited on pages 106, 108.**

Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh (2019b). Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 5757–5764
⇝ **Cited on page 108.**

Qingyuan Zhao and Trevor Hastie (2019). Causal Interpretations of Black-Box Models. *JBES* vol. 0 no. 0, pp. 1–10
⇝ **Cited on pages 59, 78, 82.**

# A Appendix of Chapter 5

The appendix of Chapter 5 is structured as follows. Appendix A.1 to Appendix A.5 are devoted to our proofs. We discuss, in Section A.6, the validity of the bound when we select a prior with $\mathcal{S}$ and have a distribution on perturbations depending on this selected prior. Finally, we present additional experiments in Section A.7.

## A.1 Proof of Proposition 5.3.1

**Proposition 5.3.1.**
For any distribution $\mathfrak{D}$ on $(\mathbb{X}{\times}\mathbb{Y}) \times \mathbb{B}$, for any distribution $\mathcal{Q}$ on $\mathcal{H}$, for any $(n, n') \in \mathbb{N}^2$, with $n \geq n' \geq 1$, we have

$$\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}) \;\leq\; \mathcal{A}_{\mathfrak{D}^{n'}}(\mathbf{B}_{\mathcal{Q}}) \;\leq\; \mathcal{A}_{\mathfrak{D}^{n}}(\mathbf{B}_{\mathcal{Q}}) \;\leq\; \mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}}). \tag{5.3}$$

*Proof.* First, we prove $\mathcal{A}_{\mathfrak{D}^1}(\mathbf{B}_{\mathcal{Q}}){=}\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}})$. We have

$$\begin{aligned}
\mathcal{A}_{\mathfrak{D}^1}(\mathbf{B}_{\mathcal{Q}}) &= 1 - \Pr_{((x,y),\mathbb{e})\sim\mathfrak{D}^1} \left(\forall\epsilon \in \mathbb{e}, \mathbf{B}_{\mathcal{Q}}(x+\epsilon) = y\right)\\
&= 1 - \Pr_{((x,y),\mathbb{e})\sim\mathfrak{D}^1} \left(\forall\epsilon \in \{\epsilon_1\}, \mathbf{B}_{\mathcal{Q}}(x+\epsilon) = y\right)\\
&= 1 - \Pr_{((x,y),\mathbb{e})\sim\mathfrak{D}^1} \left(\mathbf{B}_{\mathcal{Q}}(x+\epsilon_1) = y\right) = \mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}).
\end{aligned}$$

Then, we prove the inequality $\mathcal{A}_{\mathfrak{D}^{n'}}(\mathbf{B}_{\mathcal{Q}}) \leq \mathcal{A}_{\mathfrak{D}^{n}}(\mathbf{B}_{\mathcal{Q}})$ from the fact that the indicator function $\mathbf{I}(\cdot)$ is upper-bounded by $1$. Indeed, from Definition 5.3.3 we have

$$\begin{aligned}
1 - \mathcal{A}_{\mathfrak{D}^{n}}(\mathbf{B}_{\mathcal{Q}}) &= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathop{\mathbb{E}}_{\mathbb{e}\sim\omega^n_{(x,y)}} \mathbf{I}\left(\forall\epsilon\in\mathbb{e}, \mathbf{B}_{\mathcal{Q}}(x+\epsilon)=y\right)\\
&= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\prod_{i=1}^{n} \mathop{\mathbb{E}}_{\epsilon_i\sim\omega_{(x,y)}} \mathbf{I}\left(\mathbf{B}_{\mathcal{Q}}(x+\epsilon_i)=y\right)\right]\\
&\leq \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\prod_{i=1}^{n'} \mathop{\mathbb{E}}_{\epsilon_i\sim\omega_{(x,y)}} \mathbf{I}\left(\mathbf{B}_{\mathcal{Q}}(x+\epsilon_i)=y\right)\right]\\
&= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathop{\mathbb{E}}_{\mathbb{e}'\sim\omega^{n'}_{(x,y)}} \mathbf{I}\left(\forall\epsilon\in\mathbb{e}', \mathbf{B}_{\mathcal{Q}}(x+\epsilon)=y\right)\\
&= 1 - \mathcal{A}_{\mathfrak{D}^{n'}}(\mathbf{B}_{\mathcal{Q}}).
\end{aligned}$$

Lastly, to prove the rightmost inequality, we have to use the fact that the expectation over the set $\mathbb{B}$ is bounded by the maximum over the set $\mathbb{B}$. We have

$$\begin{aligned}
\mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_\mathcal{Q}) &= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathop{\mathbb{E}}_{\epsilon_1\sim\omega_{(x,y)}} \dots \mathop{\mathbb{E}}_{\epsilon_n\sim\omega_{(x,y)}} \mathbf{I}\left(\exists\epsilon{\in}\{\epsilon_1,\dots,\epsilon_n\}, \mathbf{B}_\mathcal{Q}(x+\epsilon)\neq y\right)\\
&\le \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \max_{\epsilon_1\in\mathbb{B}}\dots\max_{\epsilon_n\in\mathbb{B}} \mathbf{I}\left(\exists\epsilon\in\{\epsilon_1,\dots,\epsilon_n\}, \mathbf{B}_\mathcal{Q}(x+\epsilon)\neq y\right)\\
&= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \max_{\epsilon_1\in\mathbb{B}}\dots\max_{\epsilon_{n-1}\in\mathbb{B}} \mathbf{I}\left(\exists\epsilon\in\{\epsilon_1,\dots,\epsilon^*\}, \mathbf{B}_\mathcal{Q}(x+\epsilon)\neq y\right)\\
&= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathbf{I}\left(\mathbf{B}_\mathcal{Q}(x+\epsilon^*)\neq y\right)\\
&= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \max_{\epsilon\in\mathbb{B}} \mathbf{I}\left(\mathbf{B}_\mathcal{Q}(x+\epsilon)\neq y\right) \;=\; \mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_\mathcal{Q}).
\end{aligned}$$

Merging the three equations proves the claim. ■

## A.2 Proof of Proposition 5.3.2

In this section, we provide the proof of Proposition 5.3.2 that relies on Lemma A.2.1 and Lemma A.2.2 which are also described and proved. Lemma A.2.1 shows that $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q})$ is equivalent to $\mathcal{R}_{\Delta}(\mathbf{B}_\mathcal{Q})$.

**Lemma A.2.1.**
For any distribution $\mathfrak{D}$ on $(\mathbb{X}{\times}\mathbb{Y}){\times}\mathbb{B}$ and its associated distribution $\Delta$, for any posterior $\mathcal{Q}$ on $\mathcal{H}$, we have

$$\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q}) = \Pr_{(x+\epsilon,y)\sim\Delta}\left[\mathbf{B}_\mathcal{Q}(x{+}\epsilon){\neq}y\right] = \mathcal{R}_{\Delta}(\mathbf{B}_\mathcal{Q}).$$

*Proof.* Starting from the averaged adversarial risk $\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q}){=}\mathop{\mathbb{E}}_{((x,y),\epsilon)\sim\mathfrak{D}} \mathbf{I}\left[\mathbf{B}_\mathcal{Q}(x{+}\epsilon){\neq}y\right]$, we have

$$\begin{aligned}
\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q}) &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Delta} \frac{1}{\Delta(x'{+}\epsilon',y')}\left[\Pr_{((x,y),\epsilon)\sim\mathfrak{D}}\left[\mathbf{B}_\mathcal{Q}(x{+}\epsilon){\neq}y, x'{+}\epsilon'{=}x{+}\epsilon, y'{=}y\right]\right]\\
&= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Delta} \frac{1}{\Delta(x'{+}\epsilon',y')}\left[\mathop{\mathbb{E}}_{((x,y),\epsilon)\sim\mathfrak{D}} \mathbf{I}[\mathbf{B}_\mathcal{Q}(x{+}\epsilon){\neq}y]\,\mathbf{I}[x'{+}\epsilon'{=}x{+}\epsilon, y'{=}y]\right].
\end{aligned}$$

In other words, the double expectation only rearranges the terms of the original expectation: given an example $(x'{+}\epsilon',y')$, we gather probabilities such that $\mathbf{B}_\mathcal{Q}(x{+}\epsilon){\neq}y$ with $(x{+}\epsilon,y){=}(x'{+}\epsilon',y')$ in the inner expectation, while integrating over all couple $(x'{+}\epsilon', y') \in \mathbb{X}{\times}\mathbb{Y}$ in the outer expectation. Then, from the fact that when $x'{+}\epsilon'{=}x{+}\epsilon$ and $y'{=}y$, $\mathbf{I}[\mathbf{B}_\mathcal{Q}(x{+}\epsilon){\neq}y] = \mathbf{I}[\mathbf{B}_\mathcal{Q}(x'{+}\epsilon'){\neq}y']$, we have

$$\begin{aligned}\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q}) &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Delta} \frac{1}{\Delta(x'{+}\epsilon',y')}\left[\mathop{\mathbb{E}}_{((x,y),\epsilon)\sim\mathfrak{D}} \mathbf{I}[\mathbf{B}_\mathcal{Q}(x'{+}\epsilon'){\neq}y']\mathbf{I}[x'{+}\epsilon'{=}x{+}\epsilon, y'{=}y]\right]\\ &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Delta} \frac{1}{\Delta(x'{+}\epsilon',y')}\left[\mathbf{I}[\mathbf{B}_\mathcal{Q}(x'{+}\epsilon'){\neq}y'] \mathop{\mathbb{E}}_{((x,y),\epsilon)\sim\mathfrak{D}} \mathbf{I}[x'{+}\epsilon'{=}x{+}\epsilon, y'{=}y]\right].\end{aligned}$$

Finally, by definition of $\Delta(x'{+}\epsilon',y')$, we can deduce that

$$\begin{aligned}\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q}) &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Delta} \frac{1}{\Delta(x'{+}\epsilon',y')}\left[\mathbf{I}[\mathbf{B}_\mathcal{Q}(x'{+}\epsilon'){\neq}y']\,\Delta(x'{+}\epsilon',y')\right]\\ &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Delta} \mathbf{I}[\mathbf{B}_\mathcal{Q}(x'{+}\epsilon'){\neq}y'] = \mathcal{R}_\Delta(\mathbf{B}_\mathcal{Q}).\end{aligned}$$

■

Similarly, Lemma A.2.2 shows that $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_\mathcal{Q})$ is equivalent to $\mathcal{R}_\Pi(\mathbf{B}_\mathcal{Q})$.

**Lemma A.2.2.**
For any distribution $\mathcal{D}$ on $\mathbb{X}\times\mathbb{Y}$ and its associated distribution $\Pi$, for any posterior $\mathcal{Q}$ on $\mathcal{H}$, we have

$$\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_\mathcal{Q}) = \Pr_{(x+\epsilon,y)\sim\Pi}[\mathbf{B}_\mathcal{Q}(x{+}\epsilon){\neq}y] = \mathcal{R}_\Pi(\mathbf{B}_\mathcal{Q}).$$

*Proof.* The proof is similar to the one of Lemma A.2.1. Indeed, starting from the definition of $\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_\mathcal{Q}) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\,\mathbf{I}[\mathbf{B}_\mathcal{Q}(x{+}\epsilon^*(x,y)) \neq y]$, we have

$$\begin{aligned}\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_\mathcal{Q}) &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Pi} \frac{1}{\Pi(x'{+}\epsilon',y')}\left[\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathbf{I}\left[\mathbf{B}_\mathcal{Q}(x{+}\epsilon^*(x,y)) \neq y\right]\mathbf{I}[x'{+}\epsilon'{=}x{+}\epsilon^*(x,y), y'{=}y]\right]\\ &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Pi} \frac{1}{\Pi(x'{+}\epsilon',y')}\left[\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \mathbf{I}\left[\mathbf{B}_\mathcal{Q}(x'{+}\epsilon') \neq y'\right]\mathbf{I}[x'{+}\epsilon'{=}x{+}\epsilon^*(x,y), y'{=}y]\right].\end{aligned}$$

Finally, by definition of $\Pi(x'{+}\epsilon', y')$, we can deduce that

$$\begin{aligned}\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_\mathcal{Q}) &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Pi} \frac{1}{\Pi(x'{+}\epsilon',y')}\left[\mathbf{I}\left[\mathbf{B}_\mathcal{Q}(x'{+}\epsilon') \neq y'\right]\Pi(x'{+}\epsilon',y')\right]\\ &= \mathop{\mathbb{E}}_{(x'+\epsilon',y')\sim\Pi} \mathbf{I}\left[\mathbf{B}_\mathcal{Q}(x'{+}\epsilon'){\neq}y'\right] = \mathcal{R}_\Pi(\mathbf{B}_\mathcal{Q}).\end{aligned}$$

■

We can now prove Proposition 5.3.2.

**Proposition 5.3.2.** For any distribution $\mathfrak{D}$ on $(\mathbb{X}{\times}\mathbb{Y})\times\mathbb{B}$, for any distribution $\mathcal{Q}$ on $\mathcal{H}$, we have

$$\mathcal{R}^{\texttt{ROB}}_{\mathcal{D}}(\mathbf{B}_\mathcal{Q}) - \mathrm{TV}(\Pi\|\Delta) \ \leq\ \mathcal{R}_{\mathfrak{D}}(\mathbf{B}_\mathcal{Q}),$$

where $\Delta$ and $\Pi$ are distributions on $\mathbb{X}\times\mathbb{Y}$. $\Delta(x',y')$ corresponds to the probability of drawing a perturbed example $(x+\epsilon)$ with $((x,y),\epsilon)\sim\mathfrak{D}$ and is defined as

$$\Delta(x',y') = \Pr_{((x,y),\epsilon)\sim\mathfrak{D}}\left[x+\epsilon=x', y=y'\right] \tag{5.4}$$

$\Pi(x',y')$ corresponds to the probability of drawing an adversarial example $(x{+}\epsilon^*(x{,}y), y)$ with $(x,y)\sim\mathcal{D}$ and is defined as

$$\Pi(x',y') = \Pr_{(x,y)\sim\mathcal{D}}\left[x+\epsilon^*(x,y)=x', y=y'\right], \tag{5.5}$$

and $\mathrm{TV}(\Pi\|\Delta) = \mathbb{E}_{(x',y')\sim\Delta}\frac{1}{2}\left|\frac{\Pi(x',y')}{\Delta(x',y')}-1\right|$, is the Total Variation (TV) distance between $\Pi$ and $\Delta$.

*Proof.* From Lemmas A.2.1 and A.2.2, we have

$$\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}) = \mathcal{R}_{\Delta}(\mathbf{B}_{\mathcal{Q}}), \quad \text{and} \quad \mathcal{R}^{\mathtt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}}) = \mathcal{R}_{\Pi}(\mathbf{B}_{\mathcal{Q}}).$$

Then, we apply Lemma 4 of Ohnishi and Honorio (2021), we have

$$\mathcal{R}_{\Pi}(\mathbf{B}_{\mathcal{Q}}) \le \mathrm{TV}(\Pi\|\Delta) + \mathcal{R}_{\Delta}(\mathbf{B}_{\mathcal{Q}}) \quad\iff\quad \mathcal{R}^{\mathtt{ROB}}_{\mathcal{D}}(\mathbf{B}_{\mathcal{Q}}) \le \mathrm{TV}(\Pi\|\Delta) + \mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}).$$

■

## A.3 Proof of Theorem 5.3.1

**Theorem 5.3.1.**
For any distributions $\mathfrak{D}$ on $(\mathbb{X}\times\mathbb{Y})\times\mathbb{B}$ and $\mathcal{Q}$ on $\mathcal{H}$, for any $n>1$, we have

$$\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}) \le 2\overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{G}_{\mathcal{Q}}), \qquad \text{and} \qquad \mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_{\mathcal{Q}}) \le 2\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{G}_{\mathcal{Q}}).$$

*Proof.* By the definition of the majority vote, we have

$$\begin{aligned}
\frac{1}{2}\mathcal{R}_{\mathfrak{D}}(\mathbf{B}_{\mathcal{Q}}) &= \frac{1}{2}\Pr_{((x,y),\epsilon)\sim\mathfrak{D}}\left(y\,\mathbb{E}_{h\sim\mathcal{Q}}h(x{+}\epsilon)\le 0\right)\\
&= \frac{1}{2}\Pr_{((x,y),\epsilon)\sim\mathfrak{D}}\left(1-y\,\mathbb{E}_{h\sim\mathcal{Q}}h(x+\epsilon)\ge 1\right)\\
&\le \mathbb{E}_{((x,y),\epsilon)\sim\mathfrak{D}}\frac{1}{2}\left[1-y\,\mathbb{E}_{h\sim\mathcal{Q}}h(x+\epsilon)\right] \qquad \text{(Markov's ineq. on } y\,\mathbb{E}\,h(x+\epsilon)\text{)}.
\end{aligned}$$

Similarly we have

$$\begin{aligned}
\frac{1}{2}\mathcal{A}_{\mathfrak{D}^n}(\mathbf{B}_\mathcal{Q}) &= \frac{1}{2}\Pr_{((x,y),\mathfrak{e})\sim\mathfrak{D}^n}\left(\exists\epsilon\in\mathfrak{e}, y\mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x+\epsilon)\le 0\right)\\
&= \frac{1}{2}\Pr_{((x,y),\mathfrak{e})\sim\mathfrak{D}^n}\left(\min_{\epsilon\in\mathfrak{e}}\left(y\mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x+\epsilon)\right)\le 0\right)\\
&= \frac{1}{2}\Pr_{((x,y),\epsilon)\sim\mathfrak{D}}\left(1-\min_{\epsilon\in\mathfrak{e}}\left(y\mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x+\epsilon)\right)\ge 1\right)\\
&\le \mathop{\mathbb{E}}_{((x,y),\epsilon)\sim\mathfrak{D}}\frac{1}{2}\left[1-\min_{\epsilon\in\mathfrak{e}}\left(y\mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x+\epsilon)\right)\right] \quad \text{(Markov's ineq. on } \min y\,\mathbb{E}\,h(x+\epsilon)\text{)}.
\end{aligned}$$

■

## A.4 Proof of Theorem 5.3.2

**Theorem 5.3.2.**
For any distribution $\mathfrak{D}$ on $(\mathbb{X}\times\mathbb{Y})\times\mathbb{B}$, for any set of voters $\mathcal{H}$, for any prior $\mathcal{P}$ on $\mathcal{H}$, for any $n$, with probability at least $1-\delta$ over $\mathbf{S}$, for all posteriors $\mathcal{Q}$ on $\mathcal{H}$, we have

$$\mathrm{kl}(\overline{\mathcal{R}_\mathbf{S}}(\mathbf{G}_\mathcal{Q})\|\overline{\mathcal{R}_\mathfrak{D}}(\mathbf{G}_\mathcal{Q})) \le \frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{m+1}{\delta}\right], \tag{5.7}$$

$$\text{and}\quad \overline{\mathcal{R}_\mathfrak{D}}(\mathbf{G}_\mathcal{Q}) \le \overline{\mathcal{R}_\mathbf{S}}(\mathbf{G}_\mathcal{Q}) + \sqrt{\frac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{m+1}{\delta}\right]}, \tag{5.8}$$

$$\text{where}\quad \overline{\mathcal{R}_\mathbf{S}}(\mathbf{G}_\mathcal{Q}) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{1}{2}\left[1-y_i\mathop{\mathbb{E}}_{h\sim\mathcal{Q}} h(x_i+\epsilon_j^i)\right],$$

$\mathrm{kl}(a\|b) = a\ln\frac{a}{b}+(1-a)\ln\frac{1-a}{1-b}$, and $\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) = \mathop{\mathbb{E}}_{h\sim\mathcal{P}}\ln\frac{\mathcal{P}(h)}{\mathcal{Q}(h)}$ the KL-divergence between $\mathcal{P}$ and $\mathcal{Q}$.

*Proof.* Let $\Gamma{=}(V,E)$ be the graph representing the dependencies between the random variables where *(i)* the set of vertices is $V{=}\mathbf{S}$, *(ii)* the set of edges $E$ is defined such that $(((x,y),\epsilon),((x',y'),\epsilon'))\notin E \Leftrightarrow x\neq x'$. Then, applying Th. 8 of Ralaivola *et al.* (2010) with our notations gives

$$\mathrm{kl}(\overline{\mathcal{R}_\mathbf{S}}(\mathbf{B}_\mathcal{Q})\|\overline{\mathcal{R}_\mathfrak{D}}(\mathbf{B}_\mathcal{Q})) \le \frac{\chi(\Gamma)}{mn}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{mn+\chi(\Gamma)}{\delta\chi(\Gamma)}\right],$$

where $\chi(\Gamma)$ is the fractional chromatic number of $\Gamma$. From a property of Scheinerman and Ullman (2011), we have

$$c(\Gamma)\le\chi(\Gamma)\le\Delta(\Gamma)+1,$$

where $c(\Gamma)$ is the order of the largest clique in $\Gamma$ and $\Delta(\Gamma)$ is the maximum degree of a vertex in $\Gamma$. By construction of $\Gamma$, $c(\Gamma){=}n$ and $\Delta(\Gamma){=}n{-}1$. Thus, $\chi(\Gamma){=}n$ and rearranging the terms proves Equation (5.7). Finally, by applying Pinsker's inequality (*i.e.*, $|a{-}b|{\le}\sqrt{\frac{1}{2}\mathrm{kl}(a\|b)}$), we obtain Equation (5.8). ■

## A.5 Proof of Theorem 5.3.3

**Theorem 5.3.3.**
For any distribution $\mathfrak{D}$ on $(\mathbb{X}{\times}\mathbb{Y}){\times}\mathbb{B}$, for any set of voters $\mathcal{H}$, for any prior $\mathcal{P}$ on $\mathcal{H}$, for any $n$, with probability at least $1{-}\delta$ over $\mathbf{S}$, for all posteriors $\mathcal{Q}$ on $\mathcal{H}$, for all $i \in \{1,\ldots,m\}$, for all distributions $\pi_i$ on $\mathbb{e}_i$ independent from a voter $h \in \mathcal{H}$, we have

$$\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{G}_\mathcal{Q}) \le \frac{1}{m}\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\sum_{i=1}^{m}\max_{\epsilon\in\mathbb{e}_i}\frac{1}{2}\left(1{-}y_ih(x_i{+}\epsilon)\right) + \sqrt{\tfrac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\tfrac{2\sqrt{m}}{\delta}\right]} \quad (5.9)$$

$$\le \overline{\mathcal{A}_{\mathbf{S}}}(\mathbf{G}_\mathcal{Q}) + \frac{1}{m}\sum_{i=1}^{m}\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\mathrm{TV}(\rho_i^h\|\pi_i) + \sqrt{\tfrac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\tfrac{2\sqrt{m}}{\delta}\right]}, \quad (5.10)$$

where $\overline{\mathcal{A}_{\mathbf{S}}}(\mathbf{G}_\mathcal{Q}) = \dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\frac{1}{2}\left[1-\min_{\epsilon\in\mathbb{e}_i}\left(y_i\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}h(x_i+\epsilon)\right)\right]$,

and $\mathrm{TV}(\rho\|\pi) = \displaystyle\mathop{\mathbb{E}}_{\epsilon\sim\pi}\frac{1}{2}\left|\left[\frac{\rho(\epsilon)}{\pi(\epsilon)}\right]-1\right|$.

*Proof.* Let $L_{h,(x,y),\epsilon}{=}\frac{1}{2}\left[1{-}yh(x{+}\epsilon)\right]$ for the sake of readability. The losses $\max_{\epsilon\in\mathbb{e}_1}L_{h,(x_1,y_1),\epsilon},\ldots,\max_{\epsilon\in\mathbb{e}_1}L_{h,(x_m,y_m),\epsilon}$ are *i.i.d.* for any $h\in\mathcal{H}$. Hence, we can apply Theorem 20 of Germain *et al.* (2015) and Pinsker's inequality (*i.e.*, $|q{-}p|{\le}\sqrt{\frac{1}{2}\mathrm{kl}(q\|p)}$) to obtain

$$\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\ \mathop{\mathbb{E}}_{(x,y),\mathbb{e})\sim\mathfrak{D}^n}\ \max_{\epsilon\in\mathbb{e}}L_{h,(x,y),\epsilon} \le \mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\frac{1}{m}\sum_{i=1}^{m}\max_{\epsilon\in\mathbb{e}_i}L_{h,(x_i,y_i),\epsilon} + \sqrt{\frac{\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{2\sqrt{m}}{\delta}}{2m}}.$$

Then, we lower-bound the left-hand side of the inequality with $\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{B}_\mathcal{Q})$, we have

$$\overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{B}_\mathcal{Q}) \le \mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\ \mathop{\mathbb{E}}_{((x,y),\mathbb{e})\sim\mathfrak{D}^n}\ \max_{\epsilon\in\mathbb{e}}L_{h,(x,y),\epsilon}.$$

Finally, from the definition of $\rho_i^h$, and from Lemma 4 of Ohnishi and Honorio

(2021), we have

$$\begin{aligned}
\mathop{\mathbb{E}}_{h\sim\mathcal{Q}} \frac{1}{m}\sum_{i=1}^{m} \max_{\epsilon\in\mathfrak{E}_i} L_{h,(x_i,y_i),\epsilon} &= \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} \frac{1}{m}\sum_{i=1}^{m} \mathop{\mathbb{E}}_{\epsilon\sim\rho_i^h} L_{h,(x_i,y_i),\epsilon}\\
&\le \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} \frac{1}{m}\sum_{i=1}^{m} \mathrm{TV}(\rho_i^h\|\pi_i) + \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} \frac{1}{m}\sum_{i=1}^{m} \mathop{\mathbb{E}}_{\epsilon\sim\pi_i} L_{h,(x_i,y_i),\epsilon}\\
&= \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} \frac{1}{m}\sum_{i=1}^{m} \mathrm{TV}(\rho_i^h\|\pi_i) + \frac{1}{m}\sum_{i=1}^{m} \mathop{\mathbb{E}}_{\epsilon\sim\pi_i} \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} L_{h,(x_i,y_i),\epsilon}\\
&\le \mathop{\mathbb{E}}_{h\sim\mathcal{Q}} \frac{1}{m}\sum_{i=1}^{m} \mathrm{TV}(\rho_i^h\|\pi_i) + \overline{\mathcal{A}_{\mathbf{S}}}(\mathbf{B}_{\mathcal{Q}}).
\end{aligned}$$

■

## A.6 Details on the Validity of the Bounds

In this section, we discuss about the validity of the bound when *(i)* generating perturbed sets such as $\mathbf{S}$ from a distribution $\mathfrak{D}$ dependent on the prior $\mathcal{P}$ *(ii)* selecting the prior $\mathcal{P}$ with $\mathcal{S}_t$.

Actually, computing the bounds implies perturbing examples, *i.e.*, generating examples from $\mathfrak{D}$ that is defined as $\mathfrak{D}((x,y),\epsilon) = \mathcal{D}(x,y)\cdot\omega_{(x,y)}(\epsilon)$. However, in order to obtain valid bounds, $\omega_{(x,y)}$ must be defined *a priori*. Since the prior $\mathcal{P}$ is defined *a priori* as well, $\omega_{(x,y)}$ can be dependent on $\mathcal{P}$. Hence, $\omega_{(x,y)}$ boils down to generating perturbed example $(x{+}\epsilon, y)$ by attacking the prior majority vote $\mathbf{B}_{\mathcal{P}}$ with $\textsc{pgd}_{\mathrm{U}}$ or $\textsc{ifgsm}_{\mathrm{U}}$. Nevertheless, our selection of the prior $\mathcal{P}$ with $\mathcal{S}$ may seem like "cheating", but this remains a valid strategy when we perform a union bound.

We explain the union bound for Theorem 5.3.2, and the same technique can be applied for Theorem 5.3.3.
Let $\mathfrak{D}_1,\ldots,\mathfrak{D}_T$ be $T$ distributions defined as $\mathfrak{D}_1 = \mathcal{D}(x,y)\cdot\omega^1_{(x,y)}(\epsilon),\ldots,\mathfrak{D}_T = \mathcal{D}(x,y)\cdot\omega^T_{(x,y)}(\epsilon)$ on $(\mathbb{X}{\times}\mathbb{Y}){\times}\mathbb{B}$ where each distribution $\omega^t_{(x,y)}$ depends on the example $(x,y)$ and possibly on the fixed prior $\mathcal{P}_t$. Furthermore, we denote as $(\mathfrak{D}_t^n)^m$ the distribution on the perturbed learning sample consisted of $m$ examples and $n$ perturbations for each example. Then, for all distributions $\mathfrak{D}_t$, we can derive a bound on the risk $\overline{\mathcal{R}_{\mathfrak{D}_t}}(\mathbf{B}_{\mathcal{Q}})$ which holds with probability at least $1-\frac{\delta}{T}$, we have

$$\begin{aligned}
&\Pr_{\mathbf{S}_t\sim(\mathfrak{D}_t^n)^m}\left[\forall\mathcal{Q},\ \mathrm{kl}(\overline{\mathcal{R}_{\mathbf{S}_t}}(\mathbf{B}_{\mathcal{Q}})\|\overline{\mathcal{R}_{\mathfrak{D}_t}}(\mathbf{B}_{\mathcal{Q}}))\le\frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}_t)+\ln\frac{T(m{+}1)}{\delta}\right]\right]\\
=&\Pr_{\mathbf{S}_1\sim(\mathfrak{D}_1^n)^m,\ldots,\mathbf{S}_T\sim(\mathfrak{D}_T^n)^m}\left[\forall\mathcal{Q},\ \mathrm{kl}(\overline{\mathcal{R}_{\mathbf{S}_t}}(\mathbf{B}_{\mathcal{Q}})\|\overline{\mathcal{R}_{\mathfrak{D}_t}}(\mathbf{B}_{\mathcal{Q}}))\le\frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}_t)+\ln\frac{T(m{+}1)}{\delta}\right]\right]\ge1-\tfrac{\delta}{T}.
\end{aligned}$$

Then, from a union bound argument, we have

$$\Pr_{\mathbf{S}_1\sim(\mathfrak{D}_1^n)^m,\ldots,\mathbf{S}_T\sim(\mathfrak{D}_T^n)^m}\Bigg[\forall\mathcal{Q},\ \mathrm{kl}(\overline{\mathcal{R}_{\mathbf{S}_1}}(\mathbf{B}_\mathcal{Q})\|\overline{\mathcal{R}_{\mathfrak{D}_1}}(\mathbf{B}_\mathcal{Q}))\le\frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}_t)+\ln\frac{T(m+1)}{\delta}\right],$$

$$\text{and } \ldots,$$

$$\text{and } \mathrm{kl}(\overline{\mathcal{R}_{\mathbf{S}_T}}(\mathbf{B}_\mathcal{Q})\|\overline{\mathcal{R}_{\mathfrak{D}_T}}(\mathbf{B}_\mathcal{Q}))\le\frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P}_T)+\ln\frac{T(m+1)}{\delta}\right]\Bigg]\ge 1-\delta.$$

Hence, we can select $\mathcal{P}\in\{\mathcal{P}_1,\ldots,\mathcal{P}_T\}$ with $\mathcal{S}$, and let $\mathfrak{D}((x,y),\epsilon)=\mathcal{D}(x,y)\cdot\omega_{(x,y)}(\epsilon)$ be the distributions on $(\mathbb{X}\times\mathbb{Y})\times\mathbb{B}$ where $\omega_{(x,y)}(\epsilon)$ is dependent on $\mathcal{P}$ and on the example $(x,y)$, we can say that

$$\Pr_{\mathbf{S}\sim(\mathfrak{D}^n)^m}\left[\forall\mathcal{Q},\ \mathrm{kl}(\overline{\mathcal{R}_{\mathbf{S}}}(\mathbf{B}_\mathcal{Q})\|\overline{\mathcal{R}_{\mathfrak{D}}}(\mathbf{B}_\mathcal{Q}))\le\frac{1}{m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{T(m+1)}{\delta}\right]\right]\ge 1-\delta. \tag{A.1}$$

Additionally, when applying the same process for Equations (5.9) and (5.10) in Theorem 5.3.3, we have

$$\Pr_{\mathbf{S}\sim(\mathfrak{D}^n)^m}\Bigg[\forall\mathcal{Q},\ \overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{B}_\mathcal{Q})\le\frac{1}{m}\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\sum_{i=1}^{m}\max_{\epsilon\in\mathfrak{e}_i}\frac{1}{2}\left(1-y_ih(x_i+\epsilon)\right)$$
$$+\sqrt{\frac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{2T\sqrt{m}}{\delta}\right]}\Bigg]\ge 1-\delta, \tag{A.2}$$

and

$$\Pr_{\mathbf{S}\sim(\mathfrak{D}^n)^m}\Bigg[\forall\mathcal{Q},\ \overline{\mathcal{A}_{\mathfrak{D}^n}}(\mathbf{B}_\mathcal{Q})\le\overline{\mathcal{A}_{\mathbf{S}}}(\mathbf{B}_\mathcal{Q})+\frac{1}{m}\sum_{i=1}^{m}\mathop{\mathbb{E}}_{h\sim\mathcal{Q}}\mathrm{TV}(\rho_i^h\|\pi_i)$$
$$+\sqrt{\frac{1}{2m}\left[\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{2T\sqrt{m}}{\delta}\right]}\Bigg]\ge 1-\delta. \tag{A.3}$$

## A.7 Additional experimental results

In this section, we present the detailed results for the 6 tasks (3 on MNIST and 3 on Fashion MNIST) on which we perform experiments that show the test risks and the bounds for the different scenarios of (Defense, Attack). We train all the models using the same parameters as described in Section 5.4. Table A.1 and Table A.2 complement Table 5.1 to present the results for all the tasks when using the $L_2$-norm with $b=1$ (the maximum noise allowed by the norm). Then, we run again the same experiment but we use the $L_\infty$-norm with $b=0.1$ and exhibit the results in Table A.3 and Table A.4. For the experiments on the 5 other tasks using the $L_2$-norm, we have a similar behavior than **MNIST 1vs7** (presented in the paper). Indeed, using the attacks $\text{PGD}_\text{U}$ and $\text{IFGSM}_\text{U}$ as defense mechanism allows to obtain better risks and also tighter bounds compared to the bounds obtained with a defense based on UNIF (which is a naive defense). For the experiments on the 6 tasks using the $L_\infty$-norm, the trend is the same as with the $L_2$-norm, *i.e.*, the appropriate defense leads to better risks and bounds.

We also run experiments that do not rely on the PAC-Bayesian framework. In other words, we train the models following only Step 1 of our adversarial training procedure (*i.e.*, Algorithm 2) using classical attacks (PGD or IFGSM): we refer to this experiment as a baseline. In our cases, it means learning a majority vote $\mathbf{B}_{\mathcal{P}'}$ that follows a distribution $\mathcal{P}'$. As a reminder, the studied scenarios for the baseline are all the pairs (Defense, Attack) belonging to the set $\{\text{—}, \textsc{unif}, \textsc{pgd}, \textsc{ifgsm}\}\times\{\text{—}, \textsc{pgd}, \textsc{ifgsm}\}$. We report the results in Table A.5 and Table A.6. With this experiment, we are now able to compare our defense based on PGD$_\text{U}$ or IFGSM$_\text{U}$ and a classical defense based on PGD and IFGSM. Hence, considering the test risks $\mathcal{R}_{\mathcal{T}}^{\text{ROB}}(\mathbf{B}_{\mathcal{Q}})$ (columns "Attack without U" of Tables 5.1 to A.4) and $\mathcal{R}_{\mathcal{T}}^{\text{ROB}}(\mathbf{B}_{\mathcal{P}'})$ (in Tables A.5 and A.6) , we observe similar results between the baseline and our framework.

Table A.1: Test risks and bounds for 2 tasks of **MNIST** with $n=100$ perturbations for all pairs (Defense,Attack) with the two voters' set $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. The results in **bold** correspond to the best values between results for $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. To quantify the gap between our risks and the classical definition we put in *italic* the risk of our models against the classical attacks: we replace $\text{PGD}_\text{U}$ and $\text{IFGSM}_\text{U}$ by PGD or IFGSM (*i.e.*, we did *not* sample from the uniform distribution). Since Equation (5.10) upperbounds Equation (5.9) thanks to the TV term, we compute the two bound values of Theorem 5.3.3.

| $L_2$-norm $b=1$ | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Attack without U | | | | | | Attack without U | | | | | | | |
| | | $\mathcal{R}^{\text{ROB}}_{T}(\mathrm{B}_Q)$ | | $\mathcal{R}_{\mathbf{T}}(\mathrm{B}_Q)$ | | Theorem 5.3.2 | | $\mathcal{R}^{\text{ROB}}_{T}(\mathrm{B}_Q)$ | | $\mathcal{A}_{\mathbf{T}}(\mathrm{B}_Q)$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.015* | *.015* | .015 | .015 | **0.060** | .067 | *.015* | *.015* | .015 | .015 | **0.129** | 0.135 | **0.129** | .135 |
| — | $\text{PGD}_\text{U}$ | *.632* | **.628** | **.520** | .526 | 1.059 | **.847** | *.672* | **.641** | **.683** | .684 | **1.718** | 2.405 | 1.392 | **.962** |
| — | $\text{IFGSM}_\text{U}$ | *.447* | **.443** | **.157** | .166 | **0.387** | .572 | *.461* | **.451** | **.337** | .345 | **1.137** | 2.090 | 0.776 | **.669** |
| UNIF | — | *.024* | *.024* | .024 | .024 | **0.073** | .083 | *.024* | *.024* | .024 | .024 | **0.140** | 0.148 | **0.140** | .148 |
| UNIF | $\text{PGD}_\text{U}$ | *.646* | **.619** | **.486** | .500 | 1.016 | **.809** | *.649* | **.626** | **.648** | .650 | **1.646** | 2.417 | 1.338 | **.915** |
| UNIF | $\text{IFGSM}_\text{U}$ | *.442* | *.442* | **.128** | .139 | **0.316** | .528 | *.442* | *.442* | **.281** | .293 | **0.907** | 2.118 | 0.633 | **.617** |
| $\text{PGD}_\text{U}$ | — | **.024** | *.025* | **.024** | .025 | **0.094** | .101 | **.024** | *.025* | **.024** | .025 | **0.158** | 0.163 | **0.158** | .163 |
| $\text{PGD}_\text{U}$ | $\text{PGD}_\text{U}$ | *.148* | **.135** | .111 | **.103** | 0.360 | **.355** | *.146* | **.136** | .129 | **.120** | **0.442** | 2.062 | 0.414 | **.403** |
| $\text{PGD}_\text{U}$ | $\text{IFGSM}_\text{U}$ | *.104* | **.103** | .072 | .072 | 0.277 | .277 | *.102* | *.102* | .090 | **.084** | **0.358** | 1.954 | 0.335 | **.328** |
| $\text{IFGSM}_\text{U}$ | — | *.027* | **.025** | .027 | **.025** | **0.080** | .091 | *.027* | **.025** | .027 | **.025** | **0.146** | 0.154 | **0.146** | .154 |
| $\text{IFGSM}_\text{U}$ | $\text{PGD}_\text{U}$ | *.188* | **.178** | **.111** | .119 | **0.383** | .405 | *.190* | **.178** | **.126** | .134 | **0.501** | 2.063 | 0.454 | .454 |
| $\text{IFGSM}_\text{U}$ | $\text{IFGSM}_\text{U}$ | *.126* | **.115** | .076 | **.070** | **0.248** | .290 | *.127* | **.115** | .091 | **.085** | **0.371** | 1.918 | **0.329** | .342 |

(a) MNIST 4vs9

| $L_2$-norm $b=1$ | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Attack without U | | | | | | Attack without U | | | | | | | |
| | | $\mathcal{R}^{\text{ROB}}_{T}(\mathrm{B}_Q)$ | | $\mathcal{R}_{\mathbf{T}}(\mathrm{B}_Q)$ | | Theorem 5.3.2 | | $\mathcal{R}^{\text{ROB}}_{T}(\mathrm{B}_Q)$ | | $\mathcal{A}_{\mathbf{T}}(\mathrm{B}_Q)$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.015* | *.015* | .015 | .015 | **.043** | .045 | *.015* | *.015* | .015 | .015 | **.117** | 0.118 | **.117** | .118 |
| — | $\text{PGD}_\text{U}$ | *.279* | **.271** | **.232** | .234 | .600 | **.453** | *.284* | **.274** | .284 | .284 | **.829** | 1.929 | .724 | **.524** |
| — | $\text{IFGSM}_\text{U}$ | *.143* | **.137** | **.089** | .090 | **.204** | .227 | *.144* | **.139** | **.125** | .127 | **.422** | 1.662 | .337 | **.293** |
| UNIF | — | *.017* | *.017* | .017 | .017 | **.054** | .055 | *.017* | *.017* | .017 | .017 | **.124** | 0.125 | **.124** | .125 |
| UNIF | $\text{PGD}_\text{U}$ | *.219* | **.201** | **.172** | .177 | .433 | **.350** | *.219* | **.209** | **.217** | .218 | **.671** | 1.810 | .565 | **.419** |
| UNIF | $\text{IFGSM}_\text{U}$ | *.122* | *.122* | **.052** | .055 | **.119** | .181 | **.122** | *.123* | **.077** | .082 | **.307** | 1.554 | **.242** | .248 |
| $\text{PGD}_\text{U}$ | — | **.013** | *.015* | **.013** | .015 | .061 | .061 | **.013** | *.015* | **.013** | .015 | **.131** | 0.130 | .131 | **.130** |
| $\text{PGD}_\text{U}$ | $\text{PGD}_\text{U}$ | *.057* | *.057* | .045 | **.041** | **.157** | .160 | *.057* | *.057* | .055 | **.045** | **.227** | 1.536 | .218 | .218 |
| $\text{PGD}_\text{U}$ | $\text{IFGSM}_\text{U}$ | *.043* | *.043* | **.027** | .031 | **.114** | .119 | **.042** | *.043* | .037 | **.035** | **.187** | 1.433 | **.179** | .181 |
| $\text{IFGSM}_\text{U}$ | — | *.014* | **.012** | .014 | **.012** | .057 | .057 | *.014* | **.013** | .014 | **.013** | .128 | **0.127** | .128 | **.127** |
| $\text{IFGSM}_\text{U}$ | $\text{PGD}_\text{U}$ | *.077* | **.072** | .054 | **.043** | **.170** | .174 | *.076* | **.075** | .055 | **.052** | **.252** | 1.510 | **.233** | .236 |
| $\text{IFGSM}_\text{U}$ | $\text{IFGSM}_\text{U}$ | *.055* | **.048** | .034 | **.030** | **.105** | .121 | *.052* | **.051** | .039 | **.032** | **.191** | 1.379 | **.177** | .185 |

(b) MNIST 5vs6

Table A.2: Test risks and bounds for 3 tasks **Fashion MNIST** with $n = 100$ perturbations for all pairs (Defense,Attack) with the two voters' set $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. The results in **bold** correspond to the best values between results for $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. To quantify the gap between our risks and the classical definition we put in *italic* the risk of our models against the classical attacks: we replace PGD$_{\text{U}}$ and IFGSM$_{\text{U}}$ by PGD or IFGSM (*i.e.*, we did *not* sample from the uniform distribution). Since Equation (5.10) upperbounds Equation (5.9) thanks to the TV term, we compute the two bound values of Theorem 5.3.3.

| $L_2$-norm | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b=1$ | | Attack without U | | | | | | Attack without U | | | | | | | |
| | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathrm{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathrm{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathrm{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathrm{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.021* | **.020** | .021 | **.020** | **0.060** | 0.070 | *.019* | *.019* | **.019** | **.019** | **0.130** | 0.139 | **0.130** | 0.139 |
| — | PGD$_{\text{U}}$ | *.695* | **.650** | **.494** | .568 | **1.042** | 1.090 | **.677** | *.686* | **.588** | .674 | **1.326** | 2.307 | 1.152 | **1.082** |
| — | IFGSM$_{\text{U}}$ | *.451* | *.451* | **.269** | .328 | **0.585** | 0.731 | **.405** | *.438* | **.295** | .381 | **0.878** | 1.971 | **0.730** | 0.746 |
| UNIF | — | *.071* | *.071* | .071 | .071 | **0.185** | 0.191 | *.071* | *.071* | .071 | .071 | **0.236** | 0.241 | **0.236** | 0.241 |
| UNIF | PGD$_{\text{U}}$ | **.423** | *.477* | **.418** | .425 | 0.957 | **0.755** | *.486* | *.486* | .513 | .513 | **1.372** | 2.173 | 1.151 | **0.869** |
| UNIF | IFGSM$_{\text{U}}$ | **.326** | *.331* | .105 | .105 | **0.273** | 0.422 | *.333* | **.331** | .144 | **.142** | **0.496** | 1.642 | **0.397** | 0.504 |
| PGD$_{\text{U}}$ | — | *.034* | **.032** | .034 | **.032** | **0.094** | 0.114 | *.034* | **.032** | .034 | **.032** | **0.158** | 0.174 | **0.158** | 0.174 |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | **.103** | *.115* | **.086** | .091 | **0.227** | 0.289 | **.102** | *.115* | **.096** | .101 | **0.299** | 1.985 | **0.283** | 0.338 |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.092** | *.099* | **.073** | .076 | **0.195** | 0.248 | **.092** | *.099* | .082 | .082 | **0.266** | 1.914 | **0.253** | 0.299 |
| IFGSM$_{\text{U}}$ | — | **.028** | *.030* | **.028** | .030 | **0.091** | 0.105 | **.027** | *.030* | **.027** | .030 | **0.155** | 0.166 | **0.155** | 0.166 |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | *.115* | **.114** | .085 | .085 | **0.254** | 0.287 | **.112** | *.114* | **.096** | .101 | **0.331** | 2.026 | **0.313** | 0.337 |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.095** | *.097* | **.067** | .068 | **0.206** | 0.232 | **.093** | *.097* | **.080** | .081 | **0.282** | 1.927 | **0.266** | 0.285 |

(a) Fashion MNIST Sandall vs Ankle Boot

| $L_2$-norm | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b=1$ | | Attack without U | | | | | | Attack without U | | | | | | | |
| | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathrm{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathrm{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathrm{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathrm{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.038* | **.037** | .038 | **.037** | **.088** | .091 | *.038* | **.037** | .038 | **.037** | **.153** | 0.155 | **.153** | .155 |
| — | PGD$_{\text{U}}$ | *.292* | **.248** | .233 | **.112** | .452 | **.363** | *.289* | **.272** | .287 | **.246** | **.578** | 1.314 | .525 | **.479** |
| — | IFGSM$_{\text{U}}$ | *.194* | **.154** | .132 | **.075** | .300 | **.262** | *.193* | **.181** | .176 | **.148** | **.423** | 1.103 | .376 | **.359** |
| UNIF | — | *.039* | *.039* | **.039** | **.039** | **.091** | .093 | *.041* | **.039** | .041 | **.039** | **.155** | 0.157 | **.155** | .157 |
| UNIF | PGD$_{\text{U}}$ | *.240* | **.220** | **.099** | .117 | .346 | **.332** | *.250* | **.231** | .250 | **.245** | **.553** | 1.228 | .490 | **.443** |
| UNIF | IFGSM$_{\text{U}}$ | *.177* | **.171** | **.070** | .078 | **.228** | .247 | *.197* | **.185** | .186 | **.164** | **.445** | 1.046 | .371 | **.346** |
| PGD$_{\text{U}}$ | — | *.045* | **.044** | .045 | **.044** | .108 | **.105** | *.046* | **.045** | .046 | **.045** | .172 | **0.167** | .172 | **.167** |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | *.108* | **.100** | **.077** | .082 | **.203** | .211 | *.104* | **.100** | **.081** | .087 | **.279** | 1.118 | .269 | **.264** |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | *.094* | **.086** | .071 | **.069** | **.184** | .186 | *.090* | **.086** | .076 | **.073** | **.257** | 1.015 | .248 | **.241** |
| IFGSM$_{\text{U}}$ | — | **.041** | *.043* | **.041** | .043 | **.094** | .101 | **.039** | *.042* | **.039** | .042 | **.158** | 0.163 | **.158** | .163 |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | **.106** | *.114* | **.078** | .092 | **.220** | .226 | **.109** | *.113* | **.084** | .095 | **.293** | 1.052 | .279 | **.275** |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.082** | *.087* | **.065** | .072 | **.171** | .176 | **.082** | *.089* | **.068** | .078 | **.247** | 0.927 | .234 | **.232** |

(b) Fashion MNIST Top vs Pullover

| $L_2$-norm | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b=1$ | | Attack without U | | | | | | Attack without U | | | | | | | |
| | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathrm{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathrm{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathrm{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathrm{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.122* | *.122* | .122 | .122 | **0.276** | 0.286 | *.122* | *.122* | .122 | .122 | **0.318** | 0.328 | **0.318** | 0.328 |
| — | PGD$_{\text{U}}$ | *.744* | **.738** | **.674** | .689 | 1.386 | **1.066** | *.745* | **.740** | **.767** | .768 | **1.773** | 2.386 | 1.576 | **1.180** |
| — | IFGSM$_{\text{U}}$ | *.652* | **.646** | **.454** | .474 | 0.947 | **0.887** | *.659* | **.648** | **.618** | .632 | **1.597** | 2.214 | 1.276 | **0.992** |
| UNIF | — | *.204* | *.204* | .204 | .204 | 0.444 | 0.444 | *.204* | *.204* | .204 | .204 | **0.475** | 0.476 | **0.475** | 0.476 |
| UNIF | PGD$_{\text{U}}$ | *.750* | **.714** | .682 | **.671** | 1.350 | **1.069** | *.750* | **.719** | .752 | **.749** | **1.732** | 2.063 | 1.524 | **1.189** |
| UNIF | IFGSM$_{\text{U}}$ | *.605* | **.575** | **.423** | .431 | 0.871 | **0.866** | *.605* | **.578** | .530 | **.526** | **1.304** | 1.860 | 1.091 | **0.956** |
| PGD$_{\text{U}}$ | — | *.168* | **.165** | .168 | **.165** | **0.423** | 0.428 | *.167* | **.165** | .167 | **.165** | 0.463 | **0.461** | 0.463 | **0.460** |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | **.389** | *.402* | **.306** | .369 | 0.768 | **0.719** | **.390** | *.402* | **.319** | .403 | **0.847** | 2.354 | 0.810 | **0.755** |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.361** | *.368* | **.298** | .324 | 0.693 | **0.672** | **.362** | *.368* | **.320** | .361 | **0.799** | 2.258 | 0.754 | **0.707** |
| IFGSM$_{\text{U}}$ | — | **.150** | *.163* | **.150** | .163 | **0.424** | 0.428 | **.149** | *.163* | **.149** | .163 | **0.458** | 0.461 | **0.458** | 0.461 |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | **.391** | *.428* | .347 | **.292** | 0.778 | **0.757** | **.390** | *.426* | .371 | **.298** | **0.856** | 2.327 | 0.820 | **0.791** |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.356** | *.382* | .291 | **.273** | **0.685** | 0.689 | **.354** | *.382* | .331 | **.278** | **0.772** | 2.218 | 0.734 | **0.723** |

(c) Fashion MNIST Coat vs Shirt

Table A.3: Test risks and bounds for 3 tasks of **MNIST** with $n{=}100$ perturbations for all pairs (Defense,Attack) with the two voters' set $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. The results in **bold** correspond to the best values between results for $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. To quantify the gap between our risks and the classical definition we put in *italic* the risk of our models against the classical attacks: we replace PGD$_{\text{U}}$ and IFGSM$_{\text{U}}$ by PGD or IFGSM (*i.e.*, we did *not* sample from the uniform distribution). Since Equation (5.10) upperbounds Equation (5.9) thanks to the TV term, we compute the two bound values of Theorem 5.3.3.

| $L_\infty$-norm | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b = 0.1$ | | Attack without U | | | | | | Attack without U | | | | | | | |
| | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.005* | *.005* | .005 | .005 | **.017** | .019 | *.005* | *.005* | .005 | .005 | **0.099** | 0.100 | **.099** | .100 |
| — | PGD$_{\text{U}}$ | *.454* | *.454* | **.375** | .384 | .770 | **.638** | *.492* | **.484** | .480 | **.476** | **1.127** | 2.031 | .946 | **.716** |
| — | IFGSM$_{\text{U}}$ | *.428* | **.423** | **.350** | .361 | .727 | **.610** | *.474* | **.465** | .448 | **.443** | **1.061** | 2.008 | .886 | **.686** |
| UNIF | — | *.004* | *.004* | .004 | .004 | **.018** | .019 | *.004* | *.004* | .004 | .004 | **0.099** | 0.100 | **.099** | .100 |
| UNIF | PGD$_{\text{U}}$ | **.487** | *.491* | **.369** | .392 | .779 | **.667** | *.512* | **.507** | **.484** | .487 | **1.179** | 2.083 | .972 | **.739** |
| UNIF | IFGSM$_{\text{U}}$ | **.436** | *.442* | **.325** | .337 | .664 | **.598** | *.466* | **.459** | .417 | .417 | **1.023** | 1.959 | .841 | **.671** |
| PGD$_{\text{U}}$ | — | *.006* | *.006* | .006 | .006 | .024 | .024 | **.005** | *.006* | **.005** | .006 | 0.103 | 0.103 | .103 | .103 |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | **.018** | *.020* | **.013** | .016 | **.046** | .050 | **.018** | *.020* | **.015** | .020 | **0.127** | 1.461 | **.122** | .123 |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.020** | *.021* | **.012** | .016 | **.048** | .054 | **.019** | *.021* | **.015** | .020 | **0.130** | 1.455 | **.125** | .127 |
| IFGSM$_{\text{U}}$ | — | **.006** | *.007* | **.006** | .007 | **.023** | .024 | **.006** | *.007* | **.006** | .007 | **0.102** | 0.103 | **.102** | .103 |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | **.018** | *.019* | .016 | .016 | **.046** | .051 | **.018** | *.019* | **.018** | .019 | **0.126** | 1.489 | **.122** | .124 |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | *.020* | *.020* | **.015** | .016 | **.050** | .055 | *.020* | *.020* | .020 | **.019** | **0.131** | 1.481 | **.126** | .127 |

(a) MNIST 1 vs 7

| $L_\infty$-norm | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b = 0.1$ | | Attack without U | | | | | | Attack without U | | | | | | | |
| | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.015* | *.015* | .015 | .015 | **0.060** | 0.067 | *.015* | *.015* | .015 | .015 | **0.129** | 0.135 | **0.129** | 0.135 |
| — | PGD$_{\text{U}}$ | **.929** | *.930* | **.651** | .662 | 1.367 | **1.125** | **.920** | *.925* | **.874** | .880 | **2.213** | 2.661 | 1.792 | **1.266** |
| — | IFGSM$_{\text{U}}$ | *.935* | *.935* | **.601** | .609 | 1.243 | **1.088** | **.926** | *.928* | **.800** | .806 | **2.047** | 2.615 | 1.649 | **1.224** |
| UNIF | — | *.017* | *.017* | .017 | .017 | **0.062** | 0.072 | *.017* | *.017* | .017 | .017 | **0.131** | 0.139 | **0.131** | 0.139 |
| UNIF | PGD$_{\text{U}}$ | *.895* | *.895* | **.615** | .623 | 1.302 | **1.078** | **.884** | *.888* | **.815** | .818 | **2.035** | 2.722 | 1.670 | **1.208** |
| UNIF | IFGSM$_{\text{U}}$ | *.898* | *.898* | **.516** | .528 | 1.112 | **1.027** | **.884** | *.890* | **.697** | .706 | **1.875** | 2.658 | 1.497 | **1.153** |
| PGD$_{\text{U}}$ | — | *.039* | **.037** | .039 | **.037** | **0.093** | 0.094 | *.039* | **.037** | .039 | **.037** | **0.156** | 0.157 | **0.156** | 0.157 |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | **.108** | *.109* | .090 | .090 | **0.200** | 0.209 | **.108** | *.109* | **.110** | .112 | **0.337** | 1.874 | 0.290 | **0.271** |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.121** | *.124* | **.101** | .103 | **0.229** | 0.235 | **.121** | *.124* | .126 | **.125** | **0.378** | 1.890 | 0.326 | **0.297** |
| IFGSM$_{\text{U}}$ | — | *.046* | **.044** | .046 | **.044** | **0.102** | 0.119 | *.046* | **.044** | .046 | **.044** | **0.164** | 0.178 | **0.164** | 0.178 |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | *.105* | **.093** | .091 | **.078** | **0.203** | 0.214 | *.105* | **.093** | .108 | **.089** | **0.321** | 1.810 | 0.286 | **0.269** |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | *.119* | **.095** | .102 | **.080** | **0.220** | 0.229 | *.119* | **.095** | .122 | **.090** | **0.357** | 1.821 | 0.309 | **0.283** |

(b) MNIST 4 vs 9

| $L_\infty$-norm | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b = 0.1$ | | Attack without U | | | | | | Attack without U | | | | | | | |
| | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.015* | *.015* | .015 | .015 | **.043** | .045 | *.015* | *.015* | .015 | .015 | **0.117** | 0.118 | **0.117** | .118 |
| — | PGD$_{\text{U}}$ | *.500* | **.499** | **.387** | .390 | .923 | **.744** | *.502* | **.500** | **.474** | .475 | **1.361** | 2.275 | 1.146 | **.830** |
| — | IFGSM$_{\text{U}}$ | *.519* | **.505** | **.395** | .398 | .915 | **.762** | **.514** | *.516* | .481 | .481 | **1.335** | 2.283 | 1.129 | **.847** |
| UNIF | — | *.015* | *.015* | .015 | .015 | **.052** | .053 | *.015* | *.015* | .015 | .015 | **0.123** | 0.124 | **0.123** | .124 |
| UNIF | PGD$_{\text{U}}$ | **.529** | *.544* | **.388** | .393 | .925 | **.761** | **.517** | *.532* | **.481** | .482 | **1.342** | 2.349 | 1.137 | **.848** |
| UNIF | IFGSM$_{\text{U}}$ | **.536** | *.544* | **.372** | .379 | .881 | **.774** | **.523** | *.544* | **.451** | .456 | **1.268** | 2.348 | 1.077 | **.857** |
| PGD$_{\text{U}}$ | — | *.015* | **.014** | .015 | **.014** | **.060** | .064 | *.015* | **.014** | .015 | **.014** | **0.130** | 0.133 | **0.130** | .133 |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | **.055** | *.058* | **.037** | .039 | **.131** | .143 | **.056** | *.057* | .046 | .046 | **0.219** | 1.619 | **0.202** | .204 |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.061** | *.065* | **.040** | .043 | **.146** | .154 | **.059** | *.062* | .050 | **.046** | **0.232** | 1.626 | 0.216 | **.214** |
| IFGSM$_{\text{U}}$ | — | *.019* | **.014** | .019 | **.014** | .069 | **.064** | *.018* | **.014** | .018 | **.014** | 0.136 | **0.132** | 0.136 | **.132** |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | *.061* | *.061* | **.040** | .050 | .143 | **.142** | *.061* | *.061* | **.045** | .061 | **0.218** | 1.694 | 0.208 | **.205** |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.066** | *.069* | **.044** | .054 | .154 | **.152** | **.065** | *.069* | **.048** | .068 | **0.228** | 1.708 | 0.216 | **.214** |

(c) MNIST 5 vs 6

Table A.4: Test risks and bounds for 3 tasks of **Fashion MNIST** with $n$=100 perturbations for all pairs (Defense,Attack) with the two voters' set $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. The results in **bold** correspond to the best values between results for $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$. To quantify the gap between our risks and the classical definition we put in *italic* the risk of our models against the classical attacks: we replace PGD$_{\text{U}}$ and IFGSM$_{\text{U}}$ by PGD or IFGSM (*i.e.*, we did *not* sample from the uniform distribution). Since Equation (5.10) upperbounds Equation (5.9) thanks to the TV term, we compute the two bound values of Theorem 5.3.3.

| $L_\infty$-norm $b=0.1$ | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Attack without U $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | Attack without U $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.021* | **.020** | .021 | **.020** | **0.060** | 0.070 | *.019* | *.019* | .019 | .019 | **0.130** | 0.139 | **0.130** | 0.139 |
| — | PGD$_{\text{U}}$ | *.951* | **.944** | **.606** | .719 | **1.275** | 1.333 | *.935* | **.920** | **.762** | .864 | **1.617** | 2.503 | 1.421 | **1.317** |
| — | IFGSM$_{\text{U}}$ | *.957* | **.947** | **.588** | .718 | **1.231** | 1.336 | *.950* | *.950* | **.734** | .851 | **1.587** | 2.495 | 1.395 | **1.316** |
| UNIF | — | **.076** | *.077* | **.076** | .077 | **0.178** | 0.184 | **.076** | *.077* | **.076** | .077 | **0.230** | 0.235 | **0.230** | 0.235 |
| UNIF | PGD$_{\text{U}}$ | *.964* | **.961** | **.714** | .719 | 1.496 | **1.265** | *.966* | **.963** | **.853** | .859 | **2.098** | 2.417 | 1.785 | **1.416** |
| UNIF | IFGSM$_{\text{U}}$ | *.978* | **.976** | **.627** | .632 | 1.306 | **1.259** | *.979* | *.979* | **.758** | .762 | **1.914** | 2.422 | 1.597 | **1.396** |
| PGD$_{\text{U}}$ | — | *.041* | **.040** | .041 | **.040** | 0.114 | **0.111** | *.041* | **.040** | .041 | **.040** | 0.173 | **0.171** | 0.173 | **0.171** |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | *.098* | **.097** | .089 | **.086** | **0.207** | 0.210 | *.099* | **.097** | .101 | **.100** | **0.306** | 1.826 | 0.281 | **0.267** |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | *.113* | **.112** | .105 | **.101** | **0.244** | 0.246 | *.115* | **.112** | .120 | **.113** | **0.353** | 1.853 | 0.321 | **0.302** |
| IFGSM$_{\text{U}}$ | — | **.045** | *.047* | **.045** | .047 | **0.131** | 0.137 | **.045** | *.047* | **.045** | .047 | **0.188** | 0.194 | **0.188** | 0.194 |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | **.100** | *.102* | .089 | **.085** | **0.203** | 0.232 | **.100** | *.102* | **.102** | **.102** | **0.298** | 1.645 | **0.274** | 0.287 |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.112** | *.116* | .099 | **.096** | **0.232** | 0.260 | **.112** | *.116* | .114 | **.112** | **0.328** | 1.687 | **0.301** | 0.313 |

(a) Fashion MNIST Sandall vs Ankle Boot

| $L_\infty$-norm $b=0.1$ | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Attack without U $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | Attack without U $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.038* | **.037** | .038 | **.037** | **.088** | .091 | *.038* | **.037** | .038 | **.037** | **0.153** | 0.155 | **0.153** | .155 |
| — | PGD$_{\text{U}}$ | *.596* | **.515** | .477 | **.218** | .844 | **.662** | *.590* | **.576** | .570 | **.502** | **1.049** | 1.924 | 0.948 | **.857** |
| — | IFGSM$_{\text{U}}$ | *.723* | **.623** | .573 | **.257** | .971 | **.751** | *.716* | **.695** | .678 | **.598** | **1.189** | 2.031 | 1.080 | **.980** |
| UNIF | — | *.032* | *.032* | .032 | .032 | **.083** | .085 | **.032** | *.033* | **.032** | .033 | **0.149** | 0.151 | **0.149** | .151 |
| UNIF | PGD$_{\text{U}}$ | **.438** | *.439* | .356 | **.245** | .813 | **.563** | *.435* | *.435* | .423 | **.312** | **1.082** | 1.867 | 0.959 | **.688** |
| UNIF | IFGSM$_{\text{U}}$ | **.546** | *.547* | .453 | **.325** | .974 | **.690** | **.544** | *.547* | .530 | **.409** | **1.266** | 2.009 | 1.128 | **.823** |
| PGD$_{\text{U}}$ | — | **.048** | *.053* | **.048** | .053 | **.115** | .130 | **.048** | *.053* | **.048** | .053 | **0.177** | 0.188 | **0.177** | .188 |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | **.102** | *.116* | **.089** | .099 | **.205** | .223 | **.102** | *.116* | **.096** | .115 | **0.282** | 1.323 | **0.266** | .278 |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.120** | *.135* | **.102** | .115 | **.237** | .255 | **.120** | *.135* | **.109** | .133 | **0.318** | 1.380 | **0.299** | .309 |
| IFGSM$_{\text{U}}$ | — | *.051* | **.045** | .051 | **.045** | .120 | **.115** | *.051* | **.045** | .051 | **.045** | 0.179 | **0.175** | 0.179 | **.175** |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | *.106* | **.094** | .091 | **.085** | .211 | **.193** | *.106* | **.094** | .102 | **.097** | **0.292** | 1.488 | 0.273 | **.252** |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | *.120* | **.111** | **.101** | .102 | .239 | **.218** | *.119* | **.111** | .113 | .113 | **0.322** | 1.546 | 0.299 | **.277** |

(b) Fashion MNIST Top vs Pullover

| $L_\infty$-norm $b=0.1$ | | Algorithm 2 with Equation (5.7) | | | | | | Algorithm 2 with Equation (5.10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Attack without U $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{R}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Theorem 5.3.2 | | Attack without U $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{Q}})$ | | $\mathcal{A}_{\mathbf{T}}(\mathbf{B}_{\mathcal{Q}})$ | | Th. 5.3.3 - Eq. (5.10) | | Th. 5.3.3 - Eq. (5.9) | |
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | *.122* | *.122* | .122 | .122 | **0.276** | 0.286 | *.122* | *.122* | .122 | .122 | **0.318** | 0.328 | **0.318** | 0.328 |
| — | PGD$_{\text{U}}$ | **.884** | *.887* | **.781** | .795 | 1.579 | **1.268** | **.882** | *.886* | **.864** | .872 | **2.020** | 2.640 | 1.803 | **1.390** |
| — | IFGSM$_{\text{U}}$ | **.901** | *.902* | **.756** | .774 | 1.558 | **1.272** | **.901** | *.902* | **.865** | .876 | **2.032** | 2.651 | 1.795 | **1.393** |
| UNIF | — | *.166* | *.166* | .166 | .166 | **0.352** | 0.357 | .166 | *.166* | *.166* | .166 | **0.389** | 0.394 | **0.389** | 0.394 |
| UNIF | PGD$_{\text{U}}$ | **.911** | *.914* | **.796** | .798 | 1.402 | **1.326** | **.913** | *.914* | .896 | **.888** | **1.934** | 2.325 | 1.713 | **1.447** |
| UNIF | IFGSM$_{\text{U}}$ | **.935** | *.937* | **.787** | .798 | 1.392 | **1.350** | **.934** | *.936* | .887 | **.882** | **1.905** | 2.378 | 1.693 | **1.469** |
| PGD$_{\text{U}}$ | — | *.163* | **.162** | .163 | **.162** | **0.386** | 0.395 | *.163* | **.162** | .163 | **.162** | **0.419** | 0.430 | **0.419** | 0.430 |
| PGD$_{\text{U}}$ | PGD$_{\text{U}}$ | **.394** | *.396* | .359 | **.329** | 0.764 | **0.673** | **.394** | *.396* | .403 | **.394** | **0.954** | 2.321 | 0.865 | **0.726** |
| PGD$_{\text{U}}$ | IFGSM$_{\text{U}}$ | **.475** | *.480* | .442 | **.410** | 0.910 | **0.769** | **.477** | *.480* | .487 | **.472** | **1.121** | 2.411 | 1.020 | **0.826** |
| IFGSM$_{\text{U}}$ | — | **.167** | *.168* | **.167** | .168 | 0.411 | **0.395** | **.167** | *.168* | **.167** | .168 | 0.445 | **0.429** | 0.445 | **0.429** |
| IFGSM$_{\text{U}}$ | PGD$_{\text{U}}$ | *.396* | **.373** | .359 | **.293** | 0.772 | **0.641** | *.396* | **.373** | .405 | **.328** | **0.970** | 2.368 | 0.877 | **0.692** |
| IFGSM$_{\text{U}}$ | IFGSM$_{\text{U}}$ | *.465* | **.428** | .424 | **.334** | 0.891 | **0.705** | *.465* | **.429** | .470 | **.372** | **1.090** | 2.425 | 0.995 | **0.758** |

(c) Fashion MNIST Coat vs Shirt

Table A.5: Test risks for 6 tasks of **MNIST** and **Fashion MNIST** datasets for all pairs (Defense,Attack) with the two voters' set $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$ using $L_2$-norm. The results of these tables are computed considering defenses of the literature, *i.e.*, adversarial training using PGD or IFGSM. We also add an adversarial training using UNIF for the completeness of comparison between this baseline defense and our algorithm. The results in **bold** correspond to the best values between results for $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$.

| $L_2$-norm, $b=1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .005 | .005 |
| — | PGD | **.326** | .327 |
| — | IFGSM | .122 | **.121** |
| UNIF | — | .005 | .005 |
| UNIF | PGD | .191 | **.190** |
| UNIF | IFGSM | **.071** | .072 |
| PGD | — | .007 | .007 |
| PGD | PGD | .027 | **.026** |
| PGD | IFGSM | .022 | **.021** |
| IFGSM | — | **.005** | .006 |
| IFGSM | PGD | .041 | **.035** |
| IFGSM | IFGSM | .021 | .021 |

(a) MNIST 1 vs 7

| $L_2$-norm, $b=1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .015 | .015 |
| — | PGD | .692 | .692 |
| — | IFGSM | .464 | **.462** |
| UNIF | — | .024 | .024 |
| UNIF | PGD | .653 | .653 |
| UNIF | IFGSM | .441 | **.438** |
| PGD | — | **.024** | .027 |
| PGD | PGD | **.136** | .138 |
| PGD | IFGSM | **.097** | .102 |
| IFGSM | — | **.022** | .027 |
| IFGSM | PGD | **.166** | .186 |
| IFGSM | IFGSM | **.113** | .124 |

(b) MNIST 4 vs 9

| $L_2$-norm, $b=1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .015 | .015 |
| — | PGD | .283 | .283 |
| — | IFGSM | .144 | .144 |
| UNIF | — | .017 | .017 |
| UNIF | PGD | .220 | **.219** |
| UNIF | IFGSM | .122 | .122 |
| PGD | — | .014 | **.013** |
| PGD | PGD | .056 | **.055** |
| PGD | IFGSM | .045 | **.041** |
| IFGSM | — | **.013** | .014 |
| IFGSM | PGD | .077 | **.070** |
| IFGSM | IFGSM | .053 | **.047** |

(c) MNIST 5 vs 6

| $L_2$-norm, $b=1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .019 | .019 |
| — | PGD | .709 | **.708** |
| — | IFGSM | .426 | **.414** |
| UNIF | — | **.071** | .072 |
| UNIF | PGD | .531 | .531 |
| UNIF | IFGSM | .331 | **.329** |
| PGD | — | **.034** | .036 |
| PGD | PGD | .107 | **.103** |
| PGD | IFGSM | .091 | **.087** |
| IFGSM | — | .031 | **.029** |
| IFGSM | PGD | .125 | **.108** |
| IFGSM | IFGSM | .104 | **.090** |

(d) Fashion MNIST Sandall vs Ankle Boot

| $L_2$-norm, $b=1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .038 | .038 |
| — | PGD | .286 | **.285** |
| — | IFGSM | .188 | **.186** |
| UNIF | — | .041 | **.039** |
| UNIF | PGD | .249 | **.248** |
| UNIF | IFGSM | .197 | **.192** |
| PGD | — | **.043** | .045 |
| PGD | PGD | **.102** | .117 |
| PGD | IFGSM | **.090** | .094 |
| IFGSM | — | **.038** | .040 |
| IFGSM | PGD | .120 | **.106** |
| IFGSM | IFGSM | .092 | **.080** |

(e) Fashion MNIST Top vs Pullover

| $L_2$-norm, $b=1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .122 | .122 |
| — | PGD | .768 | **.767** |
| — | IFGSM | .683 | **.680** |
| UNIF | — | .204 | .204 |
| UNIF | PGD | **.753** | .754 |
| UNIF | IFGSM | .607 | **.606** |
| PGD | — | .182 | **.178** |
| PGD | PGD | .453 | **.412** |
| PGD | IFGSM | .408 | **.379** |
| IFGSM | — | .148 | **.146** |
| IFGSM | PGD | **.405** | .411 |
| IFGSM | IFGSM | .369 | **.364** |

(f) Fashion MNIST Coat vs Shirt

Table A.6: Test risks for 6 tasks of **MNIST** and **Fashion MNIST** datasets for all pairs (Defense,Attack) with the two voters' set $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$ using $L_\infty$-norm. The results of these tables are computed considering defenses of the literature, *i.e.*, adversarial training using PGD or IFGSM. We also add an adversarial training using UNIF for the completeness of comparison between this baseline defense and our algorithm. The results in **bold** correspond to the best values between results for $\mathcal{H}$ and $\mathcal{H}^{\text{SIGN}}$.

| $L_\infty$-norm, $b = 0.1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .005 | .005 |
| — | PGD | .499 | **.498** |
| — | IFGSM | **.479** | .480 |
| UNIF | — | .004 | .004 |
| UNIF | PGD | .516 | **.515** |
| UNIF | IFGSM | .467 | .467 |
| PGD | — | **.006** | .007 |
| PGD | PGD | .019 | .019 |
| PGD | IFGSM | .021 | .021 |
| IFGSM | — | .007 | .007 |
| IFGSM | PGD | **.017** | .018 |
| IFGSM | IFGSM | **.019** | .020 |

(a) MNIST 1 vs 7

| $L_\infty$-norm, $b = 0.1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .015 | .015 |
| — | PGD | .921 | .921 |
| — | IFGSM | .923 | .923 |
| UNIF | — | .017 | .017 |
| UNIF | PGD | .877 | **.876** |
| UNIF | IFGSM | .877 | .877 |
| PGD | — | .041 | **.040** |
| PGD | PGD | **.108** | .109 |
| PGD | IFGSM | **.122** | .123 |
| IFGSM | — | .057 | **.044** |
| IFGSM | PGD | .109 | **.101** |
| IFGSM | IFGSM | .119 | **.108** |

(b) MNIST 4 vs 9

| $L_\infty$-norm, $b = 0.1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .015 | .015 |
| — | PGD | .498 | .498 |
| — | IFGSM | .511 | **.510** |
| UNIF | — | .015 | .015 |
| UNIF | PGD | .512 | **.511** |
| UNIF | IFGSM | .511 | .511 |
| PGD | — | .014 | .014 |
| PGD | PGD | .065 | **.058** |
| PGD | IFGSM | .068 | **.065** |
| IFGSM | — | .018 | **.017** |
| IFGSM | PGD | **.061** | .063 |
| IFGSM | IFGSM | **.069** | .071 |

(c) MNIST 5 vs 6

| $L_\infty$-norm, $b = 0.1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .019 | .019 |
| — | PGD | .938 | .938 |
| — | IFGSM | **.948** | .949 |
| UNIF | — | **.076** | .077 |
| UNIF | PGD | .970 | **.969** |
| UNIF | IFGSM | .981 | .981 |
| PGD | — | .041 | **.040** |
| PGD | PGD | .098 | **.097** |
| PGD | IFGSM | .115 | **.111** |
| IFGSM | — | .112 | **.047** |
| IFGSM | PGD | **.045** | .100 |
| IFGSM | IFGSM | **.101** | .114 |

(d) Fashion MNIST
Sandall vs Ankell Boot

| $L_\infty$-norm, $b = 0.1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .038 | .038 |
| — | PGD | **.574** | .577 |
| — | IFGSM | .700 | **.696** |
| UNIF | — | **.032** | .033 |
| UNIF | PGD | **.428** | .435 |
| UNIF | IFGSM | **.540** | .550 |
| PGD | — | **.047** | .049 |
| PGD | PGD | .101 | **.097** |
| PGD | IFGSM | .118 | **.112** |
| IFGSM | — | .049 | **.048** |
| IFGSM | PGD | .100 | **.090** |
| IFGSM | IFGSM | .112 | **.108** |

(e) Fashion MNIST
Top vs Pullover

| $L_\infty$-norm, $b = 0.1$ | | $\mathcal{R}^{\text{ROB}}_{\mathcal{T}}(\mathbf{B}_{\mathcal{P}'})$ | |
|---|---|---|---|
| Defense | Attack | $\mathcal{H}^{\text{SIGN}}$ | $\mathcal{H}$ |
| — | — | .122 | .122 |
| — | PGD | .879 | .879 |
| — | IFGSM | .898 | .898 |
| UNIF | — | .166 | .166 |
| UNIF | PGD | .913 | **.911** |
| UNIF | IFGSM | .934 | **.933** |
| PGD | — | **.164** | .167 |
| PGD | PGD | .398 | **.395** |
| PGD | IFGSM | **.479** | .481 |
| IFGSM | — | **.163** | .169 |
| IFGSM | PGD | **.356** | .391 |
| IFGSM | IFGSM | **.422** | .461 |

(f) Fashion MNIST
Coat vs Shirt