Titre: Title:	Systèmes hybrides logiques-continus et auto-antagonistes pour l'apprentissage machine en présence d'un adversaire
Auteur: Author:	François Menet
Date:	2019
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Menet, F. (2019). Systèmes hybrides logiques-continus et auto-antagonistes pour l'apprentissage machine en présence d'un adversaire [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/3808/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/3808/
Directeurs de recherche: Advisors:	Michel Gagnon, & Jose Manuel Fernandez
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

SYSTÈMES HYBRIDES LOGIQUES-CONTINUS ET AUTO-ANTAGONISTES POUR L'APPRENTISSAGE MACHINE EN PRÉSENCE D'UN ADVERSAIRE

FRANÇOIS MENET
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
MARS 2019

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

SYSTÈMES HYBRIDES LOGIQUES-CONTINUS ET AUTO-ANTAGONISTES POUR L'APPRENTISSAGE MACHINE EN PRÉSENCE D'UN ADVERSAIRE

présenté par : MENET François
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. ALOISE Daniel, Ph. D., président
M. FERNANDEZ José M., Ph. D., membre et directeur de recherche
M. GAGNON Michel, Ph. D., membre et codirecteur de recherche
M. PAL Christopher, Ph. D., membre

DÉDICACE

À Adèle et Jules, pour toutes vos bêtises, vos jeux, et vos rires,
À Liliane, pour les souvenirs et les combats que je porte,
Aux Menet, aux Bernard, aux Blachas et aux Bonenfant...
Aux plus belles choses que l'Atlantique m'a données et ravies.

REMERCIEMENTS

J'ai un énorme problème : il faudrait trop de pages pour remercier tout ceux qui m'ont guidé jusqu'au point final de ce mémoire. Je vais en oublier.

Avant toute chose, je tiens à remercier ma famille. Sophie, Baptiste, Elisabeth, Didier, Liliane, Lorette, Michel, Jean-Louis, Dominique, Hervé, Kim, Juliette, Tess, Adèle, Jules... Ce mémoire est aussi un peu le vôtre. Question de génétique...

Je tiens à remercier les professeurs Fernandez et Gagnon, pour leur encadrement et leur support.

Des conseils très pertinents ont été donnés par Nicolas Papernot et Paul Berthier lors de la rédaction de mon article. C'est très apprécié.

Merci à toutes les équipes de Polytechnique Montréal, et en particulier les affaires administratives. L'erreur est humaine, mais votre pardon est divin.

Je remercie Marise Bonenfant d'avoir fait de ma vie une aventure à deux sur un bateau pirate.

Je remercie les équipes extraordinaires de Myelin, et d'ElementAI, qui m'offrent ou m'ont offert de beaux problèmes, de riches échanges, et des raisons de me battre. Une pensée en particulier à Marc-Olivier, Marise, Noémie, Marie-Michèle, Geneviève, Daniel, Abderahmane, Yasmina, Benoît, Frédéric, Mickaël, et Nicolas

Merci à nos conseillers dans l'aventure. Ils sont nombreux, mais j'ai une pensée particulière pour Christelle, Caroline, Geneviève, et Jean-François.

J'ai un immense respect pour les communautés d'Intelligence Artificielle et de Sécurité Informatique de Montréal. Il y a trop de noms pour une page, et, même si un *Buffer Overflow* dans un mémoire en sécurité serait tentant, je compte respecter 'le mémoire alloué'. Une pensée particulière aux MontréHacks, où j'aimerais aller plus souvent.

En parlant de Communauté, je veux remercier tous les membres passés et présents du SecSI Lab : Fanny, François Labrèche, Joan, Paul, Militza, Mikaela, Nedra, Matthieu, Chris,

Étienne, Nader, Marwen, Marielba, Corentin, Benjamin, Simon...

Un merci à mes professeurs marquants : Professeur(e)s Lepez, Marion, Bonfante, ainsi que toute la "Bande à Cohan"... à tout seigneur tout honneur.

Merci à Noémie pour son enthousiasme communicatif.

Merci à tous les amis en dehors du laboratoire, Matthieu, Gaël, Alexandre, Aurélien, Fabien, Raphaël, Rémi, Caroline, Ornella, Geneviève, Iris... de part et d'autre de l'Atlantique, il y a des gens dans cette liste qui doivent kidnapper des bières avec moi.

Nos chats Schmoutch et Peter-Prou ont su me lever tôt le matin quand c'était (pas) nécessaire. Je les en remercierai après avoir coupé quelques griffes.

Enfin merci à tous ceux que j'ai oublié, et à qui je dois un rafraîchissement de plus que les autres.

Aucun citoyen grec n'a été blessé pour l'écriture de ce mémoire.

Merci à Tizòn, un Acer Aspire E15 doté d'une nVidia GeForce 840M

Je dois remercier CALIAN pour son support financier pendant une partie de ce mémoire.

RÉSUMÉ

Certaines tâches cognitives extrêmement simples pour des humains ne peuvent être automatisées avec les principes classiques de programmation. La *vision par ordinateur*, par exemple, ne saurait être programmée avec des règles impératives : personne ne peut donner de règles précises et simples à une machine pour détecter un chat dans une image, car des milliers de conditions différentes pourraient rendre la règle caduque : présence partielle du chat, surexposition de l'image, races de chat diverses...

Pour pallier ce manque, l'Intelligence Artificielle simule les effets les plus basiques de l'intelligence humaine. Il est possible d'automatiser des tâches cognitives de base, comme reconnaissance de la parole, des visages...

Il y a quelques années, les réseaux de neurones profonds ont commencé à se démarquer. Depuis lors, les réseaux de neurones profonds et les algorithmes d'Intelligence Artificielle de nouvelle génération sont en passe de lancer une nouvelle révolution industrielle.

Ces systèmes permettent d'appendre une approximation du comportement désiré à partir d'un ensemble de données d'entraînement et dépassent à l'heure actuelle les performances humaines sur certaines tâches précises. Mais ces systèmes présentent des failles : Les réseaux de neurones peuvent émettre de fausses valeurs quand ils sont soumis à des données empoisonnées par un adversaire malveillant, autant dans leur phase d'entraînement que dans leur phase d'inférence.

Ces vulnérabilités de sécurité sont réelles et leur impact potentiel augmente à chaque jour, car non seulement les applications des systèmes d'Intelligence Artificielle sont de plus en plus critiques (conduite autonome, aide au diagnostic médical...), mais aussi parce qu'elles inspirent de plus en plus confiance et, dans certaines applications, remplacent les travailleurs humains ou deviennent un partenaire de confiance indispensable dans certaines tâches. Les conséquences sont potentiellement catastrophiques en termes de sécurité (en plus des considérations sociétales afférant au problème)

Pour corriger les défauts de ces algorithmes, de nombreux chercheurs ont proposé des solutions innovantes, basés principalement sur des systèmes de régularisation, des entraînements antagonistes, des solutions *statistiques*, consistant à contraindre la ou les topologies

des ensembles de distribution-solution, ou à créer un système de filtrage propre aux données (Compression JPEG dans les datasets d'images)

La présente maîtrise vise à étudier l'influence d'un changement dans ces systèmes d'apprentissage profonds, en ajoutant une couche auto-adversairielle de neurones dans le réseau, dotée de caractéristiques spéciales.

ABSTRACT

Computer programs are traditionnally created with lines of code, that crate a controlled, deterministic way of programming a computer. With time, various amounts of abstractions and ergonomics have been created to help the programmer input more complex programs, in order to execute incredibly complex, real-life tasks.

However, this abstraction-leverage approach does not work for some of the simplest cognitive tasks a human being can do without even thinking about it. *Computer Vision* is a good example : we can not create a simple set of rules to identify a cat in a picture, because of the image blur, the various races of cats, the position of the cat within the frame, the fact that object can hide parts of the cat...

On these kind of tasks, Artificial Intelligence has made great progress, thanks to the era of Big Data, the power of GPUs for massively parallel processing, and the idea of Convolutional Neural Networks. Now, on various cognitive tasks, these systems have achieved suprahuman performances. These tasks range from detecting face recognition to cancer detection on lung radioscopies.

The outstanding performances are bound to bring forth a new Industrial Revolution, but one problem remains : these systems are extremely vulnerable to poisonous data, crafted by a malicious adversary. The issues are numerous, and the threat can grow up to a massive scale : AI algorithms already take the driver seat of our cars, advise banks for moneylending, and get to a position of trusted partner of humans, a key position already attacked by malicious state-sponsored hackers.

Our contribution is a study of the robustness of hybrid systems, made of logical systems as well as statistical systems, in the context of adversarial training. We give intuition and experiments why these systems could be more robust on some datasets, and try an attack with the CleverHans module to see the robustness of both algorithms.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

Afin de respecter les standards de la communauté d'Intelligence Artificielle, les acronymes seront mis en anglais. On évitera cependant les acronymes lorsqu'ils ne sont pas nécessaires.

AI	Artificial Intelligence
AAI	Adversarial AI
DNN	Deep Neural Nertworks
CNN	Convolutional Neural Networks
DL	Deep Learning
FGSM	Fast Gradient Sign Method
IID	Independent and Identically Distributed
ML	Machine Learning
SpartaNet	Spartan Networks
SVM	Support Vector Machines
TI	Technologies de l'Information
XAI	eXplainable AI

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

1.1 Introduction

1.1.1 Contexte

Paradigmes classiques de Développement Au sein du génie logiciel, il existe plusieurs méthodes et paradigmes de développement : On peut considérer l'architecture d'un programme comme des classes de concepts hiérarchiquement imbriquées et liées entre elles par des interfaces ou des méthodes de transfert : c'est la *Programmation Orientée Objet.*[1] On peut également considérer un programme comme un ensemble d'effecteurs tous inter-reliés et imbriqués : c'est la *Programmation Fonctionnelle.*[2] Tous ces types de développement visent à assister un humain dans la création de programmes, afin de livrer le plus de valeur aux clients, par le biais d'un code plus robuste, plus facilement extensible, parfois au détriment de sa vitesse d'exécution.[3] Tous ces paradigmes partent du principe que l'humain possède une vision claire de la sémantique attendue d'un programme et des étapes intermédiaires nécessaires à la réalisation du produit.

Limites des paradigmes classiques Certaines tâches cognitives extrêmement simples pour des humains ne peuvent être automatisées avec les principes classiques de programmation. La *vision par ordinateur*, par exemple, ne saurait être programmée avec des règles impératives : personne ne peut donner de règles précises et simples à une machine pour détecter un chat dans une image, car des milliers de conditions différentes pourraient rendre la règle caduque : présence partielle du chat, surexposition de l'image, races de chat diverses...

l'IA comme paradigme de développement Pour pallier ce manque, l'Intelligence Artificielle simule les effets les plus basiques de l'intelligence humaine, il est possible d'automatiser des tâches cognitives de base, comme la reconnaissance de la parole, ou des visages, en apprenant d'un ensemble de données. De nombreuses applications semblent pouvoir émerger de cette technologie, grâce à la forte augmentation de la puissance de calcul ces dernières années, et grâce à un accès à des quantités massives de données.

1. Langages principaux : C++, Python, Java
2. Langages principaux : Haskell, Lisp, Scala
3. Il suffit pour s'en convaincre de créer deux projets de même sémantique en Java et en C

1.1.2 Historique et Motivations

Historique Construit à la fin des années 1950, le premier système considéré comme étant de l'Intelligence Artificielle, appelé *perceptron*, marque l'histoire comme étant le premier prototype sensé être capable de reconnaître des objets. Il symbolise un premier essai en vue de réaliser des *réseaux de neurones* : des systèmes imitant approximativement le fonctionnement du cerveau humain.

Ces systèmes créaient des modèles très simplifiés d'un réseau de neurones biologiques, mis en réseau. Les premiers modèles ne présentaient qu'une couche. Nous aborderons le point des réseaux de neurones dans la section suivante.

Dans les années 1960, nous assistons à un premier *Hiver de l'IA*. Ces événements correspondent à un retrait soudain du soutien à la recherche en Intelligence Artificielle. Ce premier retrait a été causé par l'incapacité supposée des réseaux de neurones à créer une fonction Ou-Exclusif (XOR). Ce premier hiver dure environ 10 à 15 ans.

Les années 1970 à 1980 marquent l'arrivée des systèmes experts. Ces systèmes ne s'inspirent pas forcément de la biologie pour apprendre, mais utilisent des règles et algorithmes mathématiques pour parcourir un ensemble de connaissances entré manuellement dans la machine. Différents programmes sont créés pour explorer des bases de connaissances. Le langage de requête SQL voit le jour, et permet d'interroger simplement des bases de données relationnelles. Dans les années 1990, les premiers raisonneurs logiques avec preuve par tableau sont intégrés dans des graphes de connaissance, et les premières ontologies voient le jour. Ces systèmes experts déduisent par eux-même ce que la logique de premier ordre leur permet d'inférer.

À la même époque, les réseaux de neurones reviennent sur le devant de la scène, avec plusieurs couches afin d'éviter le problème du XOR précédemment cité. Leur popularité croît à nouveau.

Ces réseaux de neurones nécessitent d'énormes quantités de calculs et de données, à une échelle inaccessible dans les années 1980-1990 : pour ces raisons, un nouvel hiver de l'Intelligence Artificielle eut lieu du milieu des années 1990 à la fin des années 2000.

En 2012, Krizhevsky et al. [38] réutilisent les recherches de LeCun et Bengio [41] sur les réseaux de neurones convolutionnels : ces réseaux spéciaux s'inspirent de la biologie de la vision humaine, et ajoutent une plus forte puissance de calcul et de données. Les réseaux de neurones surclassent leur concurrence. À partir de cette date, la performance des réseaux de neurones profond reste inégalée dans toutes les tâches de vision par ordinateur, critiques pour plusieurs applications comme la reconnaissance de documents[40], la conduite autonome, ou même l'assistance médicale.

Pour toutes ces raisons, la dernière vague de l'Intelligence Artificielle semble être un des prémices d'une "quatrième révolution industrielle" [2, 3, 50, 55] est deviendra sans doute un des futurs "moteurs de l'économie" [42]. En 2030, selon PwC[57], l'Intelligence Artificielle pourrait contribuer à hauteur de 15.7 *trillions* de dollars américains au PIB mondial, c'est-à-dire plus que les PIB actuels réunis de la Chine et l'Inde. Cette technologie ayant été développée principalement à Toronto, à Montréal, et à New York, le Québec est en position stratégique pour obtenir une part notable des potentiels revenus de cette technologie.

Vulnérabilités Manne économique gigantesque, avantage stratégique certain pour le Québec et le Canada, technologie dirsuptive dans les domaines de santé, finance, sécurité, industrie, commerce, et bien d'autres, elle possède des vulnérabilités intrinsèques. Un an après le premier succès de l'apprentissage profond moderne[38], des recherches[60] découvrent des comportements vulnérables dans les systèmes de reconnaissance d'images : des photos légèrement modifiées resteraient parfaitement reconnaissables pour un humain, mais incompréhensibles pour un algorithme d'Intelligence Artificielle. Des algorithmes instables ont déjà généré de graves problèmes économiques[59] et ceux de nouvelle génération, peu interprétables, pourraient créer un nouveau modèle de menace. Il n'existait, jusqu'en 2018, aucune méthode pour interpréter le comportement déviant d'un algorithme d'apprentissage profond.

Menaces Un attaquant pourrait manipuler un système d'inférence sur des données et corrompre les décisions de systèmes automatisés accordant des prêts, conduisant des automobiles[62], analysant les sols agricoles[1], ou aidant des cliniciens[4, 5], avec d'importantes conséquences potentielles.

Intérêts Sociaux L'IA pourrait avoir un impact social réel. L'application sur téléphone intelligent de Microsoft™ Seeing AI[64] permet aux aveugles d'avoir une description sommaire du monde qui les entoure, leur permettant de connaître le contenu d'un menu de restaurant, la valeur d'un billet de banque, d'avoir une description de la scène se produisant devant eux...

il est impossible donc de refuser ce progrès : les récompenses peuvent largement surpasser les risques.

Motivations Dans le cadre du mandat de la cybersécurité, l'étude de tels algorithmes permet aussi de préparer la communauté à l'arrivée de nouveaux modèles de menaces, avec leurs paradigmes de défense et d'attaque.

1.1.3 Définitions

L'Intelligence Artificielle est un domaine transverse à l'informatique, la biologie, et aux sciences cognitives, possédant de nombreuses définitions.

Elle peut être considérée comme tout processus visant à imiter un comportement issu de l'intelligence humaine. Cette définition 'mimétique' peut sembler trop large : peut-on vraiment considérer qu'un automate qui casse des noix est un système d'Intelligence Artificielle ? Cette capacité — et notamment l'utilisation d'outils pour le faire — est pourtant un des traits de notre espèce.

Une autre définition consiste à dire que l'IA est une méthode visant à recréer l'intelligence humaine avec des composants artificiels. Cette définition est acceptée comme celle de l'IAG (Intelligence Artificielle Générale). Les algorithmes d'IA ne sont pas, en l'état actuel de la connaissance, capables de telles performances. Dans certains domaines, au surplus, des algorithmes battent les experts (Jeu de Go, avec AlphaGo Zero). Ainsi, on ne recrée pas l'intelligence humaine si on la dépasse.

Nous choisissons de définir l'Intelligence Artificielle comme l'automatisation de processus cognitifs. Cette définition semble être suffisamment large pour couvrir les besoins de notre recherche, sans la limiter à certaines techniques *(Deep Learning)* ou à certains paradigmes *(Apprentissage Statistique)*. Elle met aussi en avant une vision de l'Intelligence Artificielle en tant qu'outil, au service de l'humanité, et non comme un remplacement.

L'apprentissage est une des capacités cognitives de l'homme les plus intéressantes : fruit de millions d'années d'évolution, elle nous permet d'évoluer dans un monde incertain et menaçant, en nous adaptant à celui-ci. Cet apprentissage nous permet d'inférer à partir de l'existant et de prévoir le futur proche à partir de stimuli extérieurs.

Ainsi, l'*Apprentissage Machine* est un des objectifs principaux des recherches en Intelligence Artificielle. L'Apprentissage Machine permet, avec un ensemble de données, d'apprendre à réaliser une tâche traditionnellement effectuée par un humain. Par exemple, étant donné un ensemble de chiffres écrits à la main [40], il est possible d'automatiser la lecture de montants de chèques de banque, remplis à la main par les clients. Cette lecture permet de traiter d'énormes quantités de chèques, sauvant des employés d'une tâche répétitive et chronophage, et augmentant la performance des employés.

Définitions propres à l'Apprentissage Machine

Les algorithmes d'Apprentissage Machine essayent d'extraire la *sémantique* d'une donnée. Par exemple, une photo, qui est une donnée composée de millions de points de couleur, peut être facilement interprétée par un humain. La description résultante est souvent un mot pour des images simples : "Chien","Chat","Famille","Moi"... on appelle sémantique le ou les sens associés par une ou plusieurs personnes à un stimulus (visuel, auditif, olfactif...)

Un jeu de données standard contient deux parties : une **donnée**, et une **étiquette** : la **donnée** est un ensemble de valeurs encodées dans la mémoire de l'ordinateur. Quand elle est représentée par un système de rendu classique, elle peut être comprise par un humain : un échantillonage pour une musique, des pixels pour une image, ou un ensemble de codes de caractères pour un texte, sont autant d'exemples d'une donnée. L'**étiquette**, remplie par un humain, transcrit la sémantique de l'image en langage informatique (voir figure 1.1) l'Apprentissage Machine se sépare en trois catégories :

1. l'*apprentissage supervisé* : chaque donnée possède son étiquette
2. l'*apprentissage semi-supervisé* : une petite partie des données possède son étiquette
3. l'*apprentissage non-supervisé* : les données ne possèdent pas d'étiquette

Si l'apprentissage supervisé vise à simuler le processus cognitif ayant permis à des humains d'annoter des milliers de données, le non-supervisé, lui, permet d'apprendre à créer des groupements dans ces données. Au lieu d'annoter des milliers de d'exemples, les personnes n'ont qu'à regarder une cinquantaine d'échantillons pour vérifier la cohérence de l'inférence réalisée.

L'apprentissage semi-supervisé entend réussir à simuler un processus par la même méthode que l'apprentissage supervisé, mais avec moins d'étiquettes. La technique veut tirer profit des

exemples sans étiquettes, afin d'extraire des lois générales, avant d'apprendre la sémantique spécifique traduite par les étiquettes disponibles.

1.2 Étapes d'un projet complet de création d'un algorithme d'Apprentissage Machine

Pour tout projet à l'exception de la recherche fondamentale, la partie la plus longue d'un projet constitue la *collecte de données*. Une fois ces données récoltées et nettoyées pour uniformiser leur format, la partie suivante est *l'initialisation puis l'entraînement du système* : l'algorithme essaye de réduire son erreur, grâce à une fonction d'erreur donnée par le programmeur. Les méthodes utilisées pour atteindre l'objectif varient. Nous évoquerons la méthode nous concernant dans la prochaine section. Cette phase est souvent effectuée conjointement avec une phase de *validation du système*. Dans cette partie, l'algorithme est évalué sur une partie préalablement réservée du jeu de données, sur lequel il vérifie sa généralisation. Cela permet d'éviter le surentraînement où l'algorithme mémorise les données d'entraînement plutôt que de définir des règles générales. Enfin, avant tout déploiement en production, on procède à une phase de *test*, sur un troisième jeu de données : en se basant sur les seuls résultats de la validation, la recherche d'hyperparamètres aurait pu créer un algorithme très performant sur les données de l'ensemble de validation uniquement. Un autre essai, sur des données inconnues simulant celles reçues par le système en production, permet de valider le bon fonctionnement de l'algorithme. Un résumé de ces étapes est visible figure 1.1 page 6.

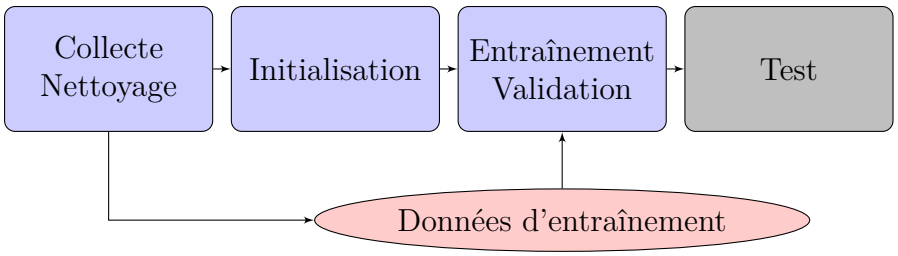


Figure 1.1 Le flot de tâches d'un système d'apprentissage machine

Tableau 1.1 Deux exemples dans deux jeux de données.

Donnée	Etiquette
	(0,0,1)
	(0,0,1)
	(1,0,0)

Données	Étiquettes
string("Ce Film est Formidable")	0.99
string("Nul, les acteurs sont mauvais")	-0.5
string("Superbe !")	0.45
string("Très belle réalisation. Décors somptueux")	0.97
string("J'ai détesté ce Film")	-0.7
string("Ce film fait passer le temps")	-0.04

Entre parenthèses, un nombre encode la sémantique de l'image. Dans la figure 1.1, partie images, nous voyons la notation *one-hot* : plus qu'un nombre, c'est un vecteur de valeurs binaires, valant 1 quand la classe indexée par cette dimension est représentée sur l'image.

1.3 Objectifs de l'Apprentissage Machine

Il existe quatre objectifs atteignables à l'heure actuelle en Apprentissage Machine :

- La **Classification**
- La **Régression**
- La **Représentation**
- La **Génération**

1.3.1 Classification

La **Classification** permet d'apprendre à classer différentes données. L'exemple des oiseaux de la figure 1.1 permet par exemple d'avoir en entrée des photos d'oiseau, et de classer quels oiseaux sont des colombes, des chouettes, ou des aigles par exemple. Pour ces exemples, on préfère utiliser la *notation one-hot* (voir figure 1.1)

Parmi les jeux de données sur lequels sont faits des classifications, citons les notables **MNIST**[40](donnée : matrice 28x28 de pixels en niveau de gris avec un chiffre dessiné à la main, étiquette : entier entre 0 et 9) et **CIFAR10**[37] (donnée : matrice 3x32x32 de pixels RVB, étiquette : valeur entre 0 et 9 correspondant à une des 10 classes d'images représentées) pour l'image, et **20newsgroups**[39] (donnée : occurences d'un corpus de mots choisis dans le corps d'un message de groupe de discussion, étiquette : sujet du groupe) pour le texte.

1.3.2 Régression

La **Régression** consiste à recréer une fonction à partir de données traduisant le comportement d'une fonction sous-jacente, en essayant de minimiser la distance entre les points de données et une loi mathématique apprise par le système. Le deuxième jeu de données de la figure 1.1 montre un exemple de données de régression : le système associe une valeur allant de -1 (sentiment très négatif) à 1 (sentiment très positif) sur des critiques de film. Ainsi quand le système est mis en production il peut associer une appréciation chiffrée de la valeur.

Pour les jeux de données de régression, citons le *Boston Housing Dataset*[30] qui associe à de nombreux critères régionaux [4] le prix de l'immobilier local. Citons aussi le jeu de données Jester[24] associe plusieurs étiquettes, allant de -10.00 à +10.00, sur la base de votes sur 100 blagues produites par des humains facétieux.

4. dont certains éthiquement discutables, comme le taux de noirs dans la ville

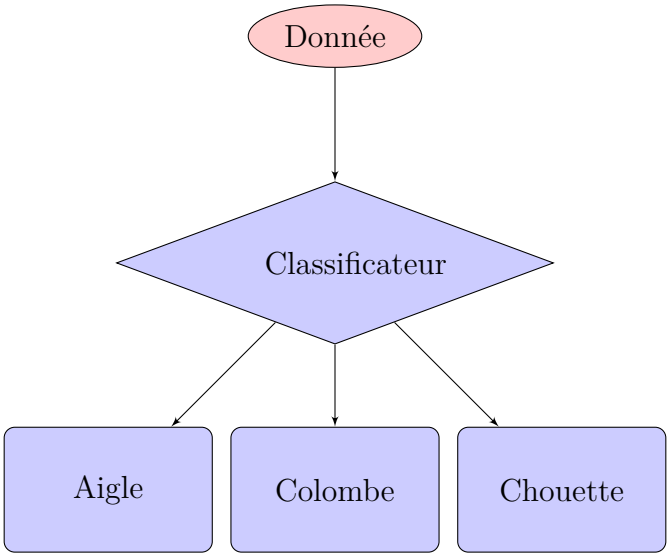


Figure 1.2 Classification

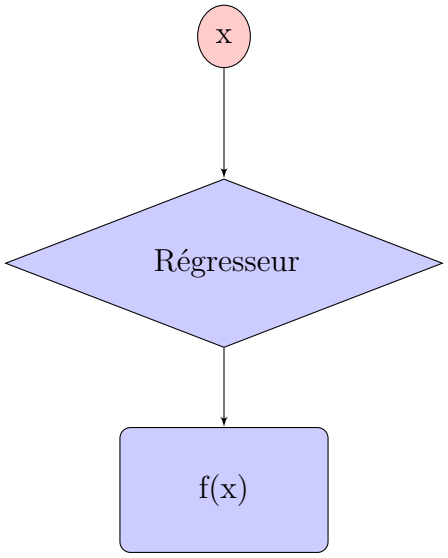


Figure 1.3 Régression

1.3.3 Représentation

La représentation permet de projeter une donnée sur un espace de dimensionnalité réduite. L'Analyse en Composantes Principales, en mathématiques, permet une représentation efficace, et est utilisée en Intelligence Artificielle.

Par exemple, il est possible de représenter un son complexe par son spectre fréquentiel grâce à l'Analyse de Fourier. Il est alors possible, en coupant les fréquences peu utilisées ou inutiles pour la tâche[5], de réduire la quantité d'informations nécessaires pour le reproduire le son : les systèmes d'Apprentissage Machine peuvent apprendre à compresser la donnée afin qu'elle soit plus simple à analyser, ou même être un premier niveau de compression afin qu'un réseau de neurones de régression/classification puisse arriver à inférer plus efficacement.

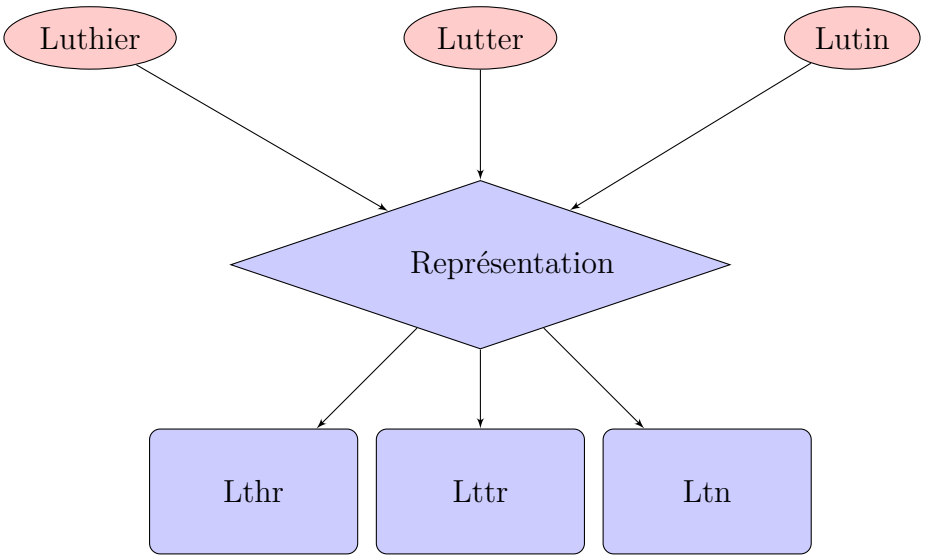


Figure 1.4 Représentation

1.3.4 Génération

Plusieurs modèles d'Apprentissage Machine sont dits *génératifs*, c'est-à-dire qu'ils tentent de capturer la distribution présente dans un jeu de données, et peuvent ainsi créer de nouvelles données, grâce à un générateur aléatoire.

Les Réseaux Antagonistes Génératifs[25] (*Generative Adversarial Networks* en anglais, GAN) ont permis de générer des visages de célébrités plus vrais que nature.[61]

5. comme les fréquences > 20 KHz inaudibles pour l'oreille humaine

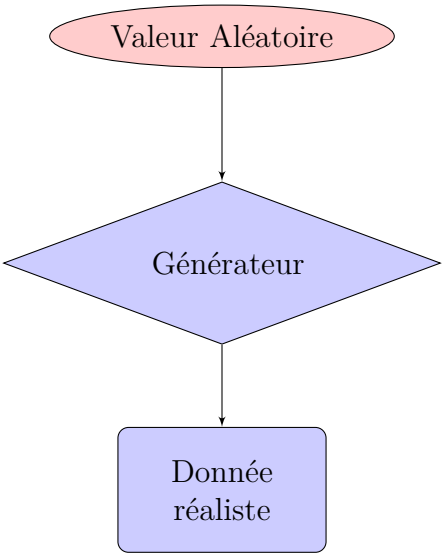


Figure 1.5 Génération

Dans notre contexte, nous allons étudier une méthode permettant sécuriser les *réseaux de neurones profonds*. Avec l'entraînement par rétro-propagation, ils constituent l'algorithme le plus performant à l'heure actuelle pour effectuer des classifications sur des données massives.

1.4 Réseaux de neurones profonds

1.4.1 Structure et principe

Neurone

Un neurone, dans sa forme la plus simple, est une unité de calcul admettant plusieurs entrées et une sortie, ainsi que trois caractéristiques :

- Un ensemble de paramètres, appelés poids, lié aux entrées du neurone
- Un biais, propre au neurone
- Une fonction d'activation, propre au neurone

Le neurone symbolise une fonction de n variables de base, procédant comme suit :

- On collecte les valeurs d'entrée : $(x_i)_{i<n}$
- Chaque dimension d'entrée est multipliée par un poids : $(\alpha_i x_i)_{i<n}$
- Tous ces produits sont sommés : $\sum_{i=1}^{n} \alpha_i x_i$
- À ce produit, on ajoute le biais du neurone : $\sum_{i=1}^{n} \alpha_i x_i + b$
- Cette valeur est entrée dans la fonction d'activation σ : on obtient l'activation neuronale $a = \sigma(\sum_{i=1}^{n} \alpha_i x_i + b)$

Apprentissage Profond

Le neurone, qui est unité de calcul simple, tire sa force du nombre : comme le réseau de neurones peut accepter en entrée plusieurs sorties d'autres neurones, il est possible d'amener dans différentes unités de calcul élémentaires de grandes quantités d'informations, traitées par des couches intermédiaires de neurones.

Ajouter une couche de neurones permet d'augmenter la complexité de la fonction, en créant des combinaisons linéaires des neurones précédents. Prenons le cas d'un seul neurone :

$$f(x) = a_{final} = \sigma(\sum_{i=1}^{n} \alpha_i x_i + b_{final}) \tag{1.1}$$

Les x_i sont eux-mêmes issus de fonctions d'activation. On en déduit que :

$$f(x) = a_{final} = \sigma(\sum_{i=1}^{n} \alpha_i (\sigma(\sum_{j=1}^{m} \beta_{ij} X_j) + b_i) + b_{final}) \tag{1.2}$$

Les X_j étant possiblement issus de neurones, on voit que le raisonnement est récursif et pourrait admettre un nombre illimité de couches de neurones.

Cette force du nombre n'admet aucune limite dans sa capacité à approximer : il est démontré qu'un réseau de neurones possédant au moins une couche cachée peut approximer n'importe quelle fonction à valeurs réelles avec une précision arbitraire, dépendant de sa capacité (taille, architecture...)[34]. Un exemple de structure de réseau de neurones est donné sur la figure 3.3 page 49.

Maintenant que nous pouvons exprimer avec un nombre de paramètres limité n'importe quelle fonction avec un degré de précision arbitraire, nous allons montrer comment approximer la fonction cible.

Entraînement par Rétro-Propagation d'Erreur

Cet algorithme permet le succès des réseaux de neurones. Il est théoriquement réservé aux réseaux de neurones dont toutes les fonctions d'activation sont continues-dérivables. Cet al-

gorithme s'applique dès qu'il est possible de calculer une erreur ou une distance entre le résultat du calcul du réseau de neurones et sa cible.

Son principe est le suivant : étant donné un réseau de neurones initialisé avec des poids aléatoires, le réseau réalise une inférence et regarde l'écart entre la sortie et la cible. Il est possible de voir la variation de l'erreur en fonction des coefficients des neurones les plus proches de la sortie : il suffit de faire des dérivées partielles selon chaque coefficient, ce qui est très simple pusique dans l'équation 1.1, chaque terme de la dérivée partielle est nul à l'exception d'un seul.

Une fois ces coefficients déterminés, on utilise l'astuce de l'équation 1.2 page 12. On utilise la règle de dérivée en chaîne pour les fonctions composées. En répétant ce mécanisme par récurrence il est possible d'obtenir une variation de l'erreur pour tous les poids du réseau, mais aussi les biais, en considérant que le biais est un poids dont l'entrée est toujours à 1.

Une fois la valeur de variation déterminée, on choisit un pas très faible et l'on pousse les coefficients de ce pas multiplié par la variation de l'erreur. Ainsi, on peut approximer une fonction, en se rapprochant toujours plus près de la fonction recherchée.

Si la structure des réseaux de neurones permet d'approximer n'importe quelle fonction sans aucune limite de précision en augmentant le nombre de neurones, l'algorithme de Rétropropagation, lui, possède de nombreuses limitations. La règle de la dérivée en chaîne faisant intervenir un produit de dérivées, une fonction d'activation aux variations faibles fait "disparaître" le gradient, et une aux variations fortes le fait "exploser". À noter que le calcul peut devenir long pour des réseaux très grands.

L'introduction de la fonction d'activation ReLU, de gradient unitaire et l'arrivée de système de calculs massivement parallèles (GPU/TPU) ont permis de gérer ces problèmes.

Il est à noter que cette technique ne permet d'apprendre que les poids et biais des neurones. Les fonctions d'activation, et la forme globale du réseau, n'est pas directement apprise par l'algorithme : on appelle ces variables des *hyperparamètres*.

1.4.2 Applications actuelles de l'Apprentissage Machine

Les applications actuelles de l'Apprentissage Machine sont nombreuses et variées et beaucoup seront sans doute découvertes dans les années à venir.

Reconnaissance d'Image Depuis environ 30 ans, cette technologie est perfectionnée et permet de nombreuses avancées économiques et sociales :

- L'application sur téléphone intelligent de Microsoft™ Seeing AI[64] permet aux aveugles d'avoir une description sommaire du monde qui les entoure, leur permettant de connaître le contenu d'un menu de restaurant, la valeur d'un billet de banque, d'avoir une description de la scène se produisant devant eux...
- Les guichets automatisées utilisent la reconnaissance d'image pour lire automatiquement des montants de chèques, évitant un travail répétitif et long pour des employés de banque.
- De nombreuses applications d'identification de visage permettent d'identifier automatiquement des personnes sur des images, des vidéos.

Reconnaissance de la parole La reconnaissance de la parole a été fortement améliorée ces dernières années par l'utilisation de l'Apprentissage Profond. Les utilisations actuelles sont en développement et nombreuses. Citons notamment :

- La montée des *chatbots*, ou robots conversationnels, permettant un accès à des informations rapidement sans la présence d'un conseiller humain. L'expérience utilisateur, notamment pour les aveugles, est fortement améliorée par ces produits.
- La transcription automatique permet aujourd'hui d'automatiser la prise de compte-rendus dans un cadre professionnel.

Gestion de Données Massives Dans une certaine mesure, la précision des algorithmes d'apprentissage profond augmente avec le nombre de données. Si les algorithmes classiques nécessitent moins de données, les algorithmes d'Apprentissage Profond peuvent créer des règles implicites toujours plus complexes et efficientes, les rendant extrêmement aptes à gérer des quantités massives de données. Par exmple, le FAIR (*Facebook Artificial Intelligence Research*) utilise des algorithmes d'apprentissage profond pour sélectionner les nouvelles les plus pertinentes parmi le contenu visible des utilisateurs du réseau social.

1.5 Conclusion et plan

Nous avons découvert comment les réseaux de neurones profonds peuvent approximer n'importe quelle fonction quand ils ont accès à un grand jeu de données. Cependant, des recherches semblent découvrir des vulnérabilités dans leur comportement, ce qui crée des modèles d'attaques nouveaux, sur lesquels il convient de se pencher.

Pour ce faire, nous allons revoir les différentes recherches au niveau des attaques, et des défenses utilisées pour s'en prémunir dans une revue de littérature. Puis, nous allons évaluer la puissance de nouvelles fonctions d'activations hybrides, nommées *Logiques-Dérivables* (HSID, HSAT, Cos-HSAT, et Cos-HSND dans le papier), dans deux conditions :

1. en conditions saines, afin d'étudier leurs capacités en phase standard,
2. en défense contre une attaque antagoniste, comme la FGSM (voir revue de littérature)

Nous voulons créer une nouvelle méthode d'entraînement, mais est-il possible de le faire ? Voici donc nos **deux questions de recherche** :

— **Est-il possible d'entraîner un réseau de neurones possédant des fonctions logiques, non-dérivables, par rétro-propagation ?**
— **Si oui, pour un gain en précision sur une attaque standard, de combien la précision diminue ? Quelle est l'augmentation du temps d'entraînement ?**

Une analyse des résultats amènera à la conclusion et à la fin de ce mémoire. *Notez que ce mémoire est un mémoire par article.* Le Chapitre 3 établit les expériences préliminaires nécessaires à la compréhension et à la validation de la première question de recherche. Le principal contenu de recherche de ce mémoire est dans l'article inclus.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.0.1 Avant-Propos

Comme tout domaine de sécurité, l'Intelligence Artificielle Antagoniste est un jeu de MinMax entre attaquant et défenseur. Certaines attaques seront présentées en même temps que les défenses. Si certaines défenses semblent parfois plus avancées que d'autres, elles ne sont pas toujours obsolètes. Il convient de noter que certaines attaques — comme certaines défenses — sont plus coûteuses que d'autres.

2.1 Découverte de l'IA Antagoniste

La notion d'adversaire est un sujet connu et exploré depuis longtemps en informatique, et le domaine de la cybersécurité entend comprendre et défendre un système TI contre un adversaire de puissance bornée ou non, selon les hypothèses.

La notion d'adversaire en Machine Learning a été évoquée depuis 1994 [27], dans un cadre d'inspiration : l'adversaire était une notion issue de la théorie des jeux, et permettait d'explorer un nouveau type d'apprentissage : l'apprentissage de la stratégie d'un adversaire.

Le début des années 2000 marque l'arrivée de l'adversaire malveillant en Machine Learning, notamment avec les travaux de Dalvi et al. [15] : dans ce papier, les chercheurs évoquent un concept fondamental : la violation du postulat IID.

2.1.1 Violation du postulat IID

Les systèmes d'Intelligence Artificielle se basent sur le postulat que les données d'entraînement sont représentatives de "la réalisation de variables aléatoires indépendantes et identiquement distribuées" (Darrell et al. [16], traduction personnelle).

Cette hypothèse peut être complètement remise en question par un adversaire, qui peut créer des examples compréhensibles par un humain, mais dont la probabilité d'existence au sein d'une classe donnée est faible.

Par exemple, prenons un système de classification de photos de pandas et de pandas roux

grâce à un algorithme quelconque. Le système apprend à capturer la distribution du jeu de données d'entraînement, en créant une fonction venant séparer l'espace de toutes les images possibles en deux. Le système pourrait capturer la distribution du canal rouge de l'image, sur-représenté (relativement aux autres canaux) dans les images de panda roux (voir 2.1).

Un adversaire malveillant pourrait ainsi créer un example inspiré de 2.2 et exploiter la distribution des images de panda roux : un système d'intelligence artificielle mal entraîné capturerait une distribution naïve (canal rouge prépondérant → panda roux) et pourrait tromper le système en exploitant la distribution des exemples : le mur de brique sur-expose le canal rouge, et, si un humain supprime cette caractéristique de l'image et identifie un panda, le système, lui, identifie la prépondérance du canal rouge et confond le panda pour un panda roux (voir figure 2.2 page 18).

2.1.2 Utilisation de l'IA antagoniste dans le cyberespace : premiers combats

Dans le cyberespace, des algorithmes d'apprentissage machine sont utilisés depuis les années 2000 avec SpamAssassin [6] pour empêcher les auteurs de pourriels d'inonder les boîtes courriel.

Le "Spammeur" est un exemple d'adversaire commun dans le domaine de la cybersécurité, et son étude est donc pertinente dans le cadre de l'apprentissage machine antagoniste, afin d'étudier un mécanisme déjà connu de course à l'armement.

2.1.3 IA Antagoniste en Apprentissage Machine Général

Comme vu précédemment, depuis les années 2000 :

- Sur le terrain, les "spammeurs" ont depuis longtemps implémenté des techniques d'évasion
- La recherche s'intéresse à l'apprentissage antagoniste.

Après l'article fondateur de Dalvi et al. [15], des études portant sur des défenses et attaques possibles commencent à apparaître.

Fogla et al. [22] évoquent des possibilités d'attaques sur les Systèmes de Détection d'Intrusion utilisant une forme d'attaque par mimétisme.

Dans l'attaque par mimétisme, l'attaquant essaye de comprendre quelles sont les données légitimes/normales selon le classificateur, et essaye de capturer la distribution de cette classe.

Figure 2.1 Deux images d'un dataset Panda/Panda Roux

Figure 2.2 Une forme imagée d'exemple antagoniste

L'attaquant modifie, enveloppe ou obfusque la charge utile de son attaque afin que le défenseur classe la donnée comme bénigne.

Des défenses viennent contrer cette attaque, qui se base sur la connaissance de la frontière de séparation entre le comportement normal et déviant.

La randomisation, dont l'efficacité est démontrée par Biggio et al. [9] et auparavant évoquée par Barreno et al. [8], consiste à introduire des variables aléatoires sur l'inférence pour flouter la frontière de décision (la frontière séparant les pandas des pandas roux dans notre exemple). En refusant à l'adversaire la connaissance complète du système, le système prive l'attaquant d'une stratégie optimale. La frontière de décision des classificateurs classiques est souvent linéaire, et donc une légère modification des paramètres de l'hyperplan permet souvent de créer une zone clairement connue et maîtrisable pour le défenseur, qui lui seul connaît le ré-

sultat de son Générateur de Nombres Aléatoires. Cette stratégie n'est applicable que quand la frontière de décision a une relative stabilité par rapport à la modification de ces paramètres.

Un autre type d'attaque connue est l'attaque par empoisonnement. Cette attaque permet, en changeant une partie du jeu de données, de faire dévier le comportement du classificateur. Décrite par Biggio et al. [10] contre les Machines à Vecteur de Support, le concept est investigué depuis de nombreuses années.

2.2 Découverte des Modèles d'Attaques d'IA Antagoniste en Apprentissage Profond

Depuis le début des années 2010, l'Apprentissage Profond a révolutionné le domaine de l'Intelligence Artificielle, en atteignant des niveaux méta-humains dans certains domaines de reconnaissance d'images.

Fin 2013 cependant, Szegedy et al. [60] font une découverte fondatrice : les réseaux de neurones, jusqu'alors peu considérés dans l'Intelligence Artificielle Antagoniste, sont très sensibles à de petites variations de la donnée d'entrée, si elles sont choisies de façon ciblée.

Il est ainsi possible de tromper un tel système avec une donnée extrêmement ressemblante à la donnée originale. Les chercheurs découvrent aussi le phénomène de *transférabilité* des exemples antagonistes : un exemple qui trompe un réseau de neurones profond a de fortes chances de tromper un autre réseau entraîné séparément et avec *différents hyperparamètres*.

Les auteurs indiquent leur incompréhension face à ce phénomène : si les réseaux de neurones sont aussi puissants pour détecter la sémantique d'une image, comment une image sémantiquement équivalente peut passer la frontière de décision aussi facilement ?

Le premier argument, avancé intuitivement, est celui d'un surentraînement du modèle. Les modèles surentraînés ont tendance à apprendre le bruit de leurs données d'entraînement, et de fait sont souvent plus instables que des modèles ayant correctement généralisé.

Cpendant, même les modèles présentant un très bon taux de précision en production (phase de Test) exhibent le comportement vulnérable, alors que ces modèles ne sont pas en surentraînement par définition.

Mis à part la violation du postulat IID (voir précédemment), les causes de cette vulnérabilité restent, en 2013, inconnues, avec uniquement des conjectures.

Les travaux de Goodfellow et al. [26] viennent balayer ces hypothèses et ajoutent une attaque supplémentaire : La Fast Gradient Sign Method, que nous allons expliquer en détail.

2.2.1 Méthode Rapide à Signe de Gradient

Intuition

Étant donné un réseau de neurones auquel il a accès total et un jeu de données que l'attaquant veut voir mal classifié (images illégales analysées comme légales). Un attaquant pourrait simplement procèder donc comme suit :

1. Prendre une donnée saine.
2. Si le système fait une erreur, FIN. Sinon, continuer.
3. Pour chaque dimension d'entrée (pixel, mot-vecteur, échantillon) faire :
   (a) Si je change l'entrée de $\pm\epsilon$, est-ce que l'erreur augmente ?
   (b) Si oui, ajouter $\pm\epsilon$ à l'entrée
   (c) Si non, ne rien ajouter.

Intuitivement, cette méthode va 'pousser', dimension par dimension, le réseau de neurones vers une autre classe. Cette classe peut être choisie si le critère de performance n'est pas d'augmenter l'incertitude sur la classe légitime mais d'augmenter la certitude sur la classe illégitime choisie.

La Méthode Rapide à Signe de gradient se base sur une attaque en boîte blanche, où l'attaquant a donc accès à tout le réseau de neurones.

Justification mathématique

La justification mathématique avancée dans Goodfellow et al. [26] est un argument géométrique. En considérant l'écriture vectorielle d'un neurone complètement connecté à l'entrée, il vient :

$$a = \sigma(w^\top x) \tag{2.1}$$

avec a l'activation de sortie du neurone, σ la fonction d'activation, w le vecteur de poids du neurone ainsi que son biais, et x le vecteur de la donnée auquel on concatène un 1 (voir précédemment).

En ajoutant une perturbation du vecteur d'entrée de faible empleur, ϵ, il vient :

$$\hat{a} = \sigma(w^\top(x + \epsilon)) = \sigma(w^\top x + w^\top \epsilon) \tag{2.2}$$

L'adversaire n'a souvent que l'obligation de présenter une donnée proche de la donnée originale empoisonnée, ce qui signifie que l'attaquant peut jouer, avec son vecteur ϵ, sur l'intégralité des coefficients d'un vecteur de grande dimension (le jeu de données le plus petit utilisé en apprentissage machine, MNIST [40], possède 784 dimensions). En prenant :

$$\epsilon = \pm e \ signe(w)$$

où e est une valeur faible et signe la fonction qui a chaque dimension d'entrée du vecteur, renvoie +1 si la valeur est positive, ou -1 si la valeur est de signe négatif, on obtient :

$$\hat{a} = \sigma(w^\top(x + \epsilon)) = \sigma(w^\top x + e \sum_{i=1}^{dim} |w_i|) \tag{2.3}$$

Ainsi, le vecteur de sortie de 2 peut être très différent de celui de 1, même si le vecteur-donnée est très proche du vecteur original. Cette perturbation peut être amplifiée par ajout de couches successives de neurones, mais l'analyse de ce système complet est complexe. La Méthode Rapide à Signe de Gradient permet rapidement d'évaluer le comportement du système et évite d'expliciter le système d'équations.

Les réseaux de neurones semblent, en 2014, être tellement vulnérables que *chacun* des exemples présents dans les jeux de données d'images les plus standards était vulnérable : on pouvait trouver une attaque par gradient autour de chacun de ceux-ci.

Fort heureusement, des défenses viennent essayer de combler le vide.

2.2.2 Premières défenses

Entraînement antagoniste

La première méthode étudiée par Goodfellow et al. [26], dans le même papier que la Méthode Rapide à Signe de Gradient utilise l'attaque existante pour défendre le système. Pendant l'entraînement, le système reçoit dans sa fonction de perte la partie des données "propres", issues du jeu de données d'entraînement, et la partie "empoisonnée", qui revient à ajouter le terme d'attaque par Méthode Rapide à Signe de Gradient sur le réseau à l'état actuel d'entraînement dans la pénalité. En entraînant le modèle sur une donnée empoisonnée, on ajoute une augmentation de données forçant le modèle à régulariser son comportement. Dans les premières itérations, le modèle va faire des erreurs sur ces pertes générées par l'adversaire, et, par rétropropagation, va adapter ses paramètres de façon à résister à l'attaque.

Cette défense par régularisation est rapidement remise en question : comment ce système pourrait se défendre contre une attaque sur laquelle le réseau ne s'est pas entraîné ?

Distillation

D'autres défenses vont venir essayer de combler le vide : la distillation, découverte par Papernot et al. [52], consiste à entraîner le modèle en plusieurs étapes : tout d'abord, un modèle naïf est entraîné sur les données avec les étiquettes 'brutes'.

Par exemple, un jeu de données de panda vs panda roux aurait une étiquette (1 0) pour un panda et (0 1) pour un panda roux. On appelle cette notation la notation *one-hot*.

Le premier modèle doit avoir une précision satisfaisante mais n'a pas besoin d'être robuste aux attaques.

Le second modèle est entraîné avec les étiquettes issues des prédictions du premier modèle. Ainsi, si le panda est un peu petit et ressemble un peu à un panda roux, l'étiquette pourrait par exemple être (0.74 0.26). Un des effets obtenus par cette méthode est que la réponse du système à de petites variations va être moins violente, car la rétropropagation pour un point proche de la frontière va être beaucoup moins forte qu'avec des étiquettes brutes : ainsi, le système n'aura pas à déterminer une limite très dure séparant deux données très proches.

Toutes ces méthodes sont dites de *Masquage de Gradient*, car elles cherchent à aplatir le gradient pour diminuer la sensibilité du système à de petites variations ainsi que la quantité

d'informations données à un attaquant. Ces défenses vont êtres battues par une méthode d'attaque qui va révolutionner le monde de l'IA adversairielle grâce à son modèle d'attaque.

2.2.3 Attaques en Boîte Noire

Papernot et al. [53] vont créer la première attaque d'un réseau de neurones *en boîte noire*, ce qui signifie que les chercheurs génèrent des exmples antagonistes sans connaître la valeur des paramètres du réseau de neurones du défenseur.

Le coeur de la méthode utilise la transférabilité des exemples antagonistes, évoquée précédemment : comme les exemples antagonistes peuvent être transférés d'un système à un autre, un adversaire pouvant entraîner un modèle similaire mais non équivalent au modèle cible pourrait utiliser *ses propres exemples antagonistes* comme des exemples antagonistes ayant de fortes chances de tromper le modèle cible.

Ainsi, l'attaquant n'a plus qu'à entraîner un réseau de neurones standard, de faible puissance, pour générer des exemples antagonistes.

Il ne reste qu'un problème : l'attaquant n'a pas forcément accès à un grand jeu de données, comme le défenseur ! Dans le même papier, Papernot et al. [53] vont créer une méthode pour approximer le comportement d'un classificateur-cible avec extrêmement peu de données.

Augmentation Jacobienne

Tout d'abord, l'attaquant se crée un petit jeu de données saines, d'un ordre de grandeur très faible (150 données dans l'expérience initiale de Papernot et al. [53] pour un jeu de données de 50000 éléments côté défenseur, c'est-à-dire 3 pour 1000 ou 0,3% du jeu de données total)

Un réseau de neurones de substitution est entraîné sur ce petit jeu de données étiquetté par l'adversaire : les 150 données rassemblées sont étiquettées par l'adversaire par le truchement de son interface de requête.

Une fois ce premier entraînement réalisé, le réseau de neurones approxime très grossièrement la tâche à réaliser. Le système commence donc à exhiber des comportements réalisant la tâche de classification, même si la performance est faible.

Des données synthétiques vont par la suite être générées en se basant sur la Jacobienne du réseau : cette matrice est obtenue en regardant, autour de chaque point du jeu de données original, le comportement du réseau par rapport à une légère modification de l'entrée (c'est une matrice des dérivées). En analysant cete Jacobienne, on peut créer une données en procédant comme suit :

1. On décide d'une cible (une classe, éventuellement la même que l'originale, ou une valeur, pour un modèle de régression)
2. On regarde grâce à la matrice Jacobienne les dimensions (par exemple, les pixels dans une image) à modifier pour se rapprocher de la valeur cible
3. On fait étiquetter à l'adversaire la donnée modifiée

Le principe est simple : en créant cette "augmentation de données" à partir de déformations simples, il est possible de détecter les frontières entre les classes, ou, plus généralement, le comportement du système sur lequel on réalise l'attaque en Boîte Noire.

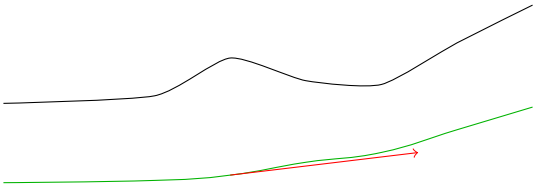
Le classificateur ou l'approximateur obtenu est beaucoup moins précis que l'original, mais la plupart des exemples antagonistes obtenu sur ce systèmes *sont transférables au système original*. Cet effet est démontré empiriquement dans la publication précitée.

Ainsi, en attaquant le système-substitut, sur lequel l'attaquant a tous les droits, on obtient un ensemble d'exemples antagonistes, qui ont de fortes chances de tromper le système original.

Conséquences de l'existence de l'attaque en Boîte Noire

Cette attaque permet d'ignorer complètement toutes les défenses à base de Masquage de Gradient. La raison est simple : lors d'une attaque en boîte noire, l'attaquant n'a jamais accès au gradient, car la seule information auquel l'attaquant a accès est la classe la plus probable, sans même sa probabilité.

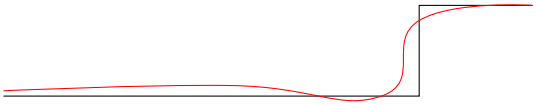
Le Masquage de Gradient base sa défense sur le fait que l'attaquant n'arrive pas à connaître le taux de variation de la prédiction en fonction de l'entrée, car le système diminue au maximum sa dépendance sur ce taux. Privé de ce gradient pour savoir comment modifier la donnée, l'attaquant classique ne peut attaquer le système. Un résumé du Masquage de Gradient est donné sur la figure 2.3 et l'attaque en Boîte Noire sur la figure 2.4



La topologie de réponse d'un réseau de neurones est assez chaotique et de légères variations peuvent créer une méclassification. Le *Masquage de Gradient* rend la topologie de la réponse plus douce, et réduit la valeur du gradient.

Figure 2.3 Masquage de Gradient

.



L'attaque en Boîte Noire n'utilise pas le gradient, car elle n'y a pas accès. En créant un réseau de neurones substitut, elle recrée un gradient là où il n'y en avait pas, et dépasse donc les méthodes de masquage de gradient.

Figure 2.4 Attaque en Boîte Noire.

Ainsi, le Masquage de Gradient coupe l'accès à une donnée... dont cette attaque n'a pas besoin, prouvant ainsi l'inefficacité de cette protection.

2.3 Cybersécurité de l'IA contemporaine

2.3.1 Paradigmes de défense

À partir de cette attaque, les défenses vont évoluer en trois paradigmes :

- Le paradigme de **correction** : Le réseau doit être utilisé sur des exemples proches des exemples sur lequel le système est entraîné. Une "couche de protection" doit rendre le système plus robuste en changeant la donnée d'entrée.
- Le paradigme de **détection** : s'il est possible d'identifier les exemples antagonistes par une méthode intrinsèque ou extrinsèque, il est possible d'exclure l'exemple et de ne pas inférer sur celui-ci (ou d'émettre une alarme)
- Le paradigme de **robustesse** : le réseau doit rester imperturbable face à ces attaques, et être construit et entraîné de façon à ce que sa réponse ne soit pas perturbée par les attaques.

Chaque paradigme a ses avantages : ainsi, les paradigmes de **correction** et de la **détection**

permettent de protéger un réseau déjà existant, mais sont essentiellement séparables du réseau, ce qui rend le système vulnérables à des attaques.

Ainsi, le paradigme de **détection** est vulnérable à une attaque par déni de service, où un attaquant ayant accès à une donnée envoyée légitimement changerait imperceptiblement la donnée de façon à ce que la donnée soit perçue comme adversairielle. Le client ne recevrait pas le traitement de sa donnée légitime.

De même, le paradigme de **correction** est vulnérable à une attaque par évasion fractionnée : s'il est possible de générer une erreur dans la couche de protection, il est possible que la donnée soit rendue légitime... mais dans une mauvaise classe.

Le paradigme de **robustesse** n'est pas vulnérable à ces types d'attaques, mais il nécessite un ré-entraînement du modèle, et est souvent plus coûteux en termes de performances sur un jeu de donéees sain.

2.3.2 Paradigme de correction

Le paradigme de correction se traduit par différentes stratégies, mais l'idée fondamentale peut être exprimée comme suit : le sens de la donnée est exprimable avec beaucoup moins de données que la donnée d'entrée.

Par exemple, il faut 784 pixels de 8 bit (6272 bits, donc) pour identifier un chiffre écrit à la main dans le jeu de données MNIST [40], mais une très grande partie des pixels et valeurs sont inutiles et le sens même de la donnée peut se coder sur 4 bits.

Ainsi, si l'espace au complet est un terrain d'attaque pour l'adversaire, le défenseur n'a qu'à transformer la donnée d'entrée de façon à ce que la donnée considérée soit aussi proche que possible d'une distribution connue : la seule nécessité est que la transformation appliquée garde la sémantique de la donnée. (voir figure 2.5)

Une grande partie des systèmes de correction s'inspirent des algorithmes de compression : ces algorithmes partagent en effet la même contrainte : ils doivent conserver la sémantique de la donnée en réduisant la dimensionalité de la donnée.

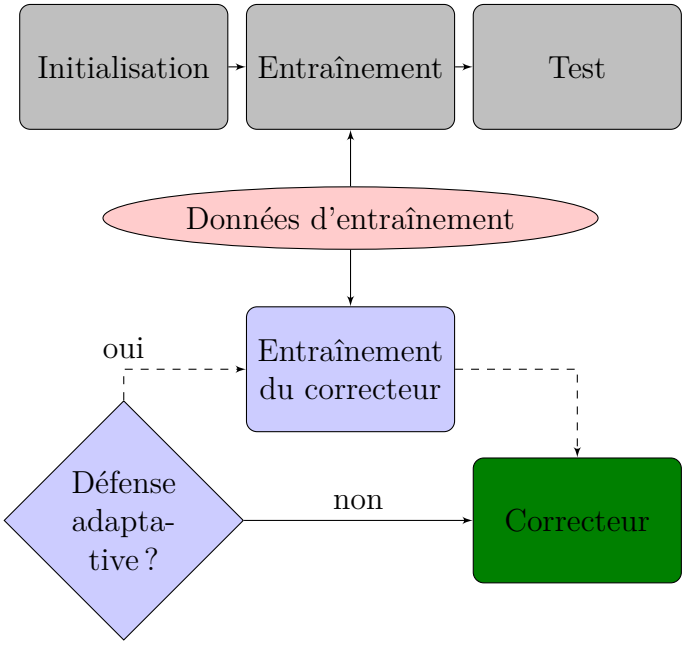


Figure 2.5 Le flot de tâches d'un système protégé avec le paradigme de correction

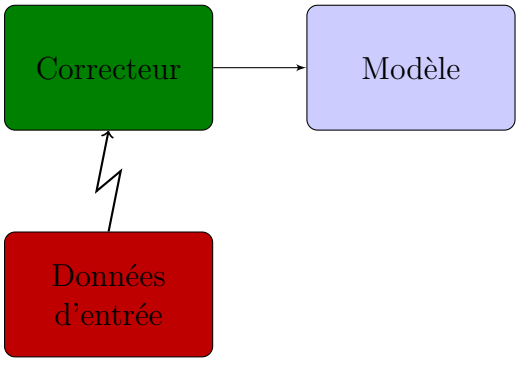


Figure 2.6 Le modèle de protection par correction

Compression

Guo et al. [29] proposent, pour la protection contre les exemples antagonistes dans les images, d'utiliser différentes méthodes de compression d'image, avec quelques résultats. Dans leur rapport, les chercheurs utilisent 5 méthodes pour protéger un ResNet [31], sur le jeu de données ImageNet [18]. Les méthodes de corrections, appliquées sur les données d'entrée sont les suivantes :

- La compression JPEG, l'algorithme de compression d'images.
- La coupe-agrandissement, où une large portion de l'image est extraite puis agrandie à la taille originale de l'image.
- La recomposition d'image, où des images sont recréées à partir de morceaux d'images proches issus d'une base de données.
- La minimisation de la variance totale utilise un processus stochastique pour recréer des images proches
- La réduction de profondeur d'image, où les couleurs sont encodées sur moins de *bits* : une réduction de 8 à 1 bit, par exemple, est l'encodage d'une image en niveaux de gris en une image noir et blanc strict.

Les chercheurs découvrent que les transformations les plus agressives rendent souvent le modèle plus robuste, mais moins performant. La recomposition d'image, par exemple, est quasiment invulnérable aux attaques connues, mais diminue, dans ce papier, la précision de 45%

Différentes méthodes seront par la suite proposées.

Das et al. [17] réutilisent la compression JPEG, en ajoutant un ensemble de modèles en parallèle, utilisant différents niveaux de compression JPEG : la compression JPEG "supprime les composantes de haute fréquence" (traduction libre du papier cité), composantes souvent sur-représentées par les exemples antagonistes.

Encodage et discrétisation

L'encodage consiste à regénérer l'exemple, en utilisant un algorithme créant une nouvelle donnée à partir de l'ancienne.

Aussi cité pour le volet de détection, MagNet, créé par Meng et Chen [48], permet de recréer une donnée en utilisant un autoencodeur, un réseau de neurones fait pour reconstruire son

entrée en perdant le moins d'information utile possible. Le but est de générer une perte d'information (l'autoencodeur possède une représentation interne compressée), tout en s'assurant que l'information gardée soit pertinente.

Le principe de protection par encodage discret — qui a été découvert et exploité séparément dans ce mémoire (la méthode est différente)— a été étudié dans le cadre de l'étude de Buckman et al. [11] :

Au lieu d'avoir des données d'entrée sous forme d'un vecteur de nombres (encodés comme des *int* de 8bits par exemple), les données sont des vecteurs de vecteurs binaires, ces vecteurs encodant un nombre comme un ensemble de seuils à valeur binaire (discrétisation).

Ainsi, si une donnée d'entrée du système est un ensemble de valeurs $i \in [a, b]^n$, la version discrétisée choisit un ensemble $(k_i)_{i \in [\![1,m]\!]}$ tel que :

$$\forall i \in [\![1, m]\!], k_i \in [a, b]$$

. Ainsi, la discrétisation est définie par :

$$f : [a, b] \to \{0, 1\}^m \tag{2.4}$$

$$x \mapsto (\mathbf{1}_{x>k_i})_{i \in [\![1,m]\!]} \tag{2.5}$$

Par exemple, si on choisit les k_i comme étant $(1, 2, 3, 4, 5)$: voici comment pourrait être encodé le vecteur (0.11,1.75,3.25,5.44)

x	f(x)
0.11	(0,0,0,0,0)
1.75	(1,0,0,0,0)
3.25	(1,1,1,0,0)
5.44	(1,1,1,1,1)

Les seuils sont choisis arbitrairement, contrairement à notre recherche qui les apprend. Ainsi, nos seuils deviennent de plus en plus efficaces. La discrétisation est souvent faite d'un encodage vers un autre, pusique toute information numérique est sauvegardée sous forme binaire.

En encodant et discrétisant l'information, une partie de l'information est perdue : cette méthode permet de faire en sorte que les petites variations introduites par l'attaquant soient transformées et modifiées, en gardant la sémantique dans la plupart des cas (voir figure 2.7)

Autre forme pouvant être considérée comme de l'encodage, les mécanismes d'attention [46] permettent, sur des images, de transformer une image en une suite de focalisations sur des points d'intérêt. Deux systèmes interagissent pour ce faire. Le premier définit la prochaine zone d'intérêt à partir de la suite des zones d'intérêts connues, et l'autre constitue le véritable classificateur de ces zones d'intérêt.

Diffusion d'erreur

Prakash et al. [56] utilisent une méthode très simple pour contrer les exemples antagonistes : au lieu de compresser la donnée, ils introduisent volontairement des erreurs, en choisissant aléatoirement des pixels cibles dans les voisinages des pixels source choisis, et en copiant le pixel cible sur la source. Ils appliquent par la suite un code correcteur d'erreur, basée sur une analyse fréquentielle des *ondelettes* de l'image : le spectre cible doit rentrer dans une norme apprise lors de l'entraînement, et l'image est corrigée pour rentrer dans celui-ci.

La perturbation antagoniste est augmentée d'une perturbation contrôlée par le défenseur, et ce mélange entre dans le code correcteur d'erreur. Cette perturbation défensive va intervenir *après* le dernier mouvement de l'attaquant, ce qui en fait une attaque adaptative très intéressante d'un point de vue stratégique.

2.3.3 Paradigme de détection

Le paradigme de détection repose sur le fait que les exemples antagonistes exhibent des caractéristiques et motifs sous-jacents qui leur sont propres. Ainsi, un algorithme pourrait apprendre les caractéristiques des exemples antagonistes, et empêcher l'inférence sur cette donnée "empoisonnée" quand il détecte un exemple dont les caractéristiques sont soit :

— proches de propriétés connues d'exemples antagonistes. [28, 45, 49, 44]
— éloignés de la distribution normale des exemples sains. [32, 48, 21, 65]

Détection par Proximité des Composantes

Dès 2016, Hendrycks et Gimpel [32] créent les première méthodes de détection des exemples antagonistes : ils utilisent notamment l'Analyse en Composantes Principales, traditionnel-


La première image est la version discrétisée de la seconde (passage d'un encodage 3x8 bits, considéré comme la valeur réelle initiale, à 3x3 bits) : la sémantique de l'image ("chat") reste la même.

Figure 2.7 Discrétisation de données

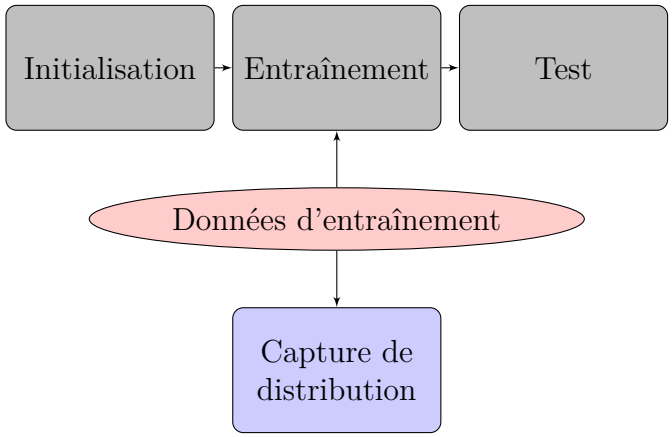


Figure 2.8 Le flot de tâches d'un système protégé avec le paradigme de détection

lement utilisée pour réduire la dimensionalité, et se rendent compte que les directions de covariance les plus élevées sont différentes pour des ensemble exemples antagonistes : les vecteurs propres associés aux valeurs faibles (sur les données saines) sont associés à des valeurs très élevées sur les exemples antagonistes.

Détection par Proximité des Comportements

De leur côté, la même année, Li et Li [44] entraînent algorithme pour détecter, par le biais de statistiques sur les couches convolutionnelles de leur réseau de neurones, des anomalies statistiques : leur algorithme exhibait des propriétés intéressantes, comme la non-différentiabilité : contrairement à plusieurs autres algorithmes, il est impossible pour un attaquant d'obtenir une dérivée, nécessaire à la plupart des attaques. L'algorithme extrait des composantes principales dans les différentes couches convolutionnelles, et utilise des Machines à Vecteur

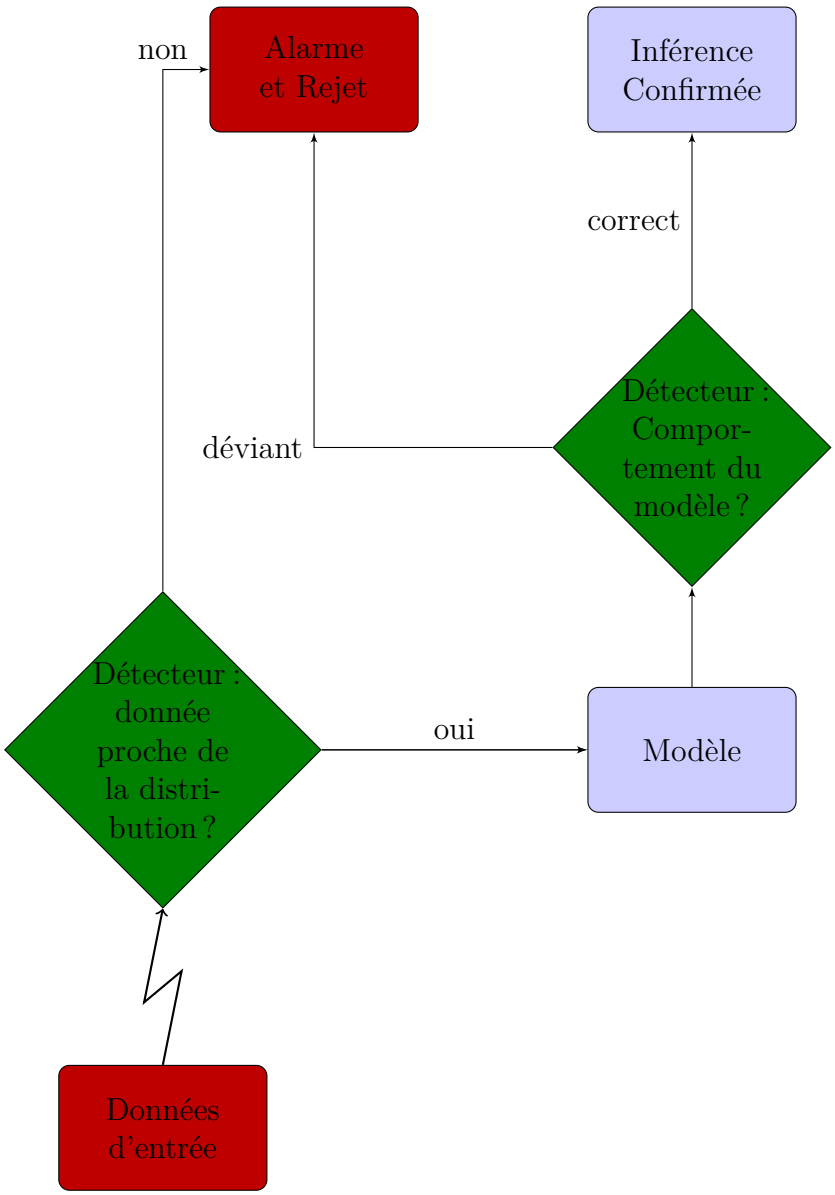


Figure 2.9 Le modèle de protection par détection

de Support pour classifier le comportement, en cascade, c'est-à-dire que chaque couche est analysée afin de voir si son comportement s'écarte du comportement appris lors de l'entraînement. Cette idée est remodelée par Lu et al. [45], en ne regardant que les derniers stages (les couches denses-ReLU), et en utilisant une Machine à Vecteur de Support dotée d'un noyau, une fonction transformant l'espace afin que la séparation devienne plus simple à séparer pour l'algorithme (plus linéaire)

Xu et al. [65], avec l'écrasement de caractéristiques (*feature squeezing*), créent une méthode que nous décrivons en détail ci-après.

Écrasement de Caractéristiques

Les images possèdent beaucoup plus d'informations que nécessaire pour exprimer leur sémantique de base. Par exemple, dans la figure 2.10, il est possible de distinguer un chat, alors même que la photo est mise en noir et blanc, ce qui nécessite 96% de données en moins que l'original.

Un attaquant peut donc jouer dans un espace d'images bien plus grand que l'espace nécessaire pour comprendre la sémantique de base de l'image (chat, plantes).

Xu et al. [65] transforment le désavantage en force pour le défenseur.

Pour créer un détecteur, les auteurs utilisent un classificateur d'image déjà entraîné sur des données saines.

Les auteurs appliquent par la suite un ensemble de filtres d'image : soit l'image voit sa profondeur de couleur diminuée (voir 2.10), soit l'image est floutée de plus en plus fort. À chaque diminution, la donnée est entrée dans le classificateur, qui infère une classe d'image. Ainsi, le détecteur se base sur un ensemble de votes du classificateur

Les images saines n'ont pas de perturbation, et les votes font consensus autour d'une réponse, qu'elle soit bonne ou mauvaise dans le cas d'une erreur du système. Dans certains cas, les images trop écrasées créent une erreur au niveau de l'inférence. Si l'écrasement n'est pas trop puissant, ce vote est minoritaire.

Pour les images perturbées par un attaquant, il y a plusieurs cas :

— Les images peu ou pas écrasées gardent les perturabations et le système les méclassifie

pour une classe cible [1]
- Les images moyennement écrasées restent perturbées, mais il est possible que la méclassification ne touche plus la classe cible
- Les images très écrasées et encore compréhensibles par le système sont correctement classifiées
- Les images trop écrasées créent, comme pour l'image saine, des erreurs.

Il en ressort que les images perturbées par un adversaire génèrent plus de dissensions.

La seule méthode pour l'attaquant serait de créer un exemple gardant la même sémantique *à travers un chemin de transformation dans l'espace d'images* : la taille de l'espace se retourne contre l'attaquant.

Cette méthode, très élégante, ne nécessite même pas de posséder le réseau de neurones : la défense peut être faite en boîte noire. Nous nous inspirerons de cette méthode dans notre proposition de défense.

Détection par régénération

Déjà cités pour leur volet de correction, les auteurs de MagNet, Meng et Chen [48], utilisent aussi un système d'Autoencodeurs pour détecter les exemples antagonistes :

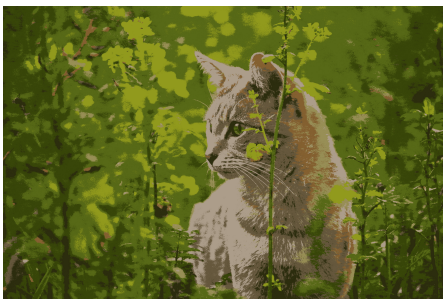
Pour chaque élément du jeu de données d'entraînement, un autoencodeur, qui cherche à reconstruire son entrée en compressant les informations, est entraîné à regénérer la donnée.

Si la donnée est saine, la distribution se rapproche du jeu de données initial, et l'auto-encodeur va reconstruire des exemples très proches de l'exemple initial.

Si la donnée est empoisonnée, les probabilités que l'autoencodeur génère une donnée éloignée de la donnée originale sont grandes.

Le détecteur regénère la donnée présente en entrée et regarde la distance entre l'original et sa version regénérée. Si la distance dépasse un certain seuil, l'exemple est rejeté et le correcteur (voir précédemment) entre en jeu.

1. que la cible soit choisie ou non par l'attaquant

	Pigeon (Exemple antagoniste)
Image encodée sur 24 bits (3×8)	100 %
	Chien (Perturbation modifiée)
Image encodée sur 9 bits (3×3)	33 %
	Chat (Perturbation éliminée)
Image encodée sur 2 bits (4 couleurs indexées)	8,3 %
	Grenouillle (Image trop écrasée)
Image encodée sur 1 bit (Noir & Blanc)	4,2 %

Même encodée sur peu de bits, l'image garde sa sémantique. À droite, un exemple d'inférence du classificateur d'image pour illustrer l'écrasement de caractéristiques, et la taille de l'espace (par rapport à l'espace original)

Figure 2.10 Exemple d'écrasement de caractéristiques

2.3.4 Paradigme de robustesse

Le problème des réseaux de neurones peut se résumer à la métaphore faite par l'équipe du canevas Cleverhans [54] sur le cheval "Clever Hans" : ce cheval, que l'on croyait capable de calculs complexes, tapait du pied autant de fois que nécessaire pour exprimer le résultats de multiplications. Il ne faisait que ressentir l'excitation du public alors qu'il approchait du bon résultat.

En bref, une des thèses peut se résumer comme suit : si les réseaux de neurones profonds sont vulnérables à des exemples antagonistes, c'est qu'ils n'apprennent pas les caractéristiques réelles de la donnée mais juste des artéfacts corrélés avec la présence de la donnée.

En effet, si le système apprenait les mêmes caractéristiques que celles que nous apprenons, le seul moyen de générer un exemple antagoniste serait de tromper un humain. Comme nous avons défini l'Intelligence Artificielle comme "l'automatisation de processus cognitifs", il n'y aurait pas d'exemples antagonistes, puisque les humains à l'origine seraient vulnérables, et que l'IA ferait donc son travail.

Le paradigme de robustesse rentre dans cette logique : en permettant au système d'apprendre les bonnes caractéristiques ou de résister à la présence d'artéfacts, seule une réingénierie des modèles actuels permettrait de rendre le système vraiment résistant aux exemples antagonistes.

Il est intéressant de noter que Elsayed et al. [20] ont découvert que, dans un temps limité, les *humains sont vulnérables aux exemples antagonistes*. Cette hypothèse n'est donc pas forcément vérifiée et une question de recherche reste ouverte actuellement : "les réseaux de neurones exhibent-ils juste plus fort des vulnérabilités présentes dans l'intelligence en général, ou possèdent-ils leur propres vulnérabilités ?"

Variantes de l'entraînement antagoniste

Comme dit précédemment, les chercheurs ont essayé, grâce à l'entraînement antagoniste, de forcer le système à ignorer les artéfacts qu'ils avaient appris. Cet entraînement antagoniste a évolué avec le temps, mais possède un inconvénient majeur : il n'est pas agnostique de l'attaque.

Après les travaux de Papernot et al. [53] et les attaques en boîte noire, ces protections vont se révéler peu efficaces.

Une variante plus intéressante est celle de Tramèr et al. [63], ou les exemples antagonistes ne proviennent pas de données corrompues par une attaque sur le réseau cible, mais de données corrompues par *d'autres réseaux*.
Madry et al. [47] poussent même le raisonnement plus loin en créant la notion d'attaque optimale avec la *descente de gradient projetée*. En partant de plusieurs points dans le voisinage d'un exemple pour vérifier des propriétés d'exemples antagonistes, les chercheurs découvrent qu'il est possible, à cause de la topologie des exemples antagonistes, d'obtenir un *adversaire maximum*. S'entraîner contre cet adversaire maximum permettrait d'être universellement résistant.

Récemment, Li et al. [43] utilise un entraînement antagoniste augmenté d'un objectif intinsèque : la somme des distortions générées par la normalisation du résultat de chaque couche. En forçant chaque couche à posséder une réponse plus plate, les auteurs annoncent un système plus robuste, même à des attaques actuelles, en se basant sur le fait que la défense créée rend plus robuste face à la même attaque *optimale* pour le premier ordre (l'attaquant ne s'adapte pas à la défense)

Ainsi, cette branche de recherche semble relancée par l'existence de ce nouvel attaquant optimal.

Distillation Défensive

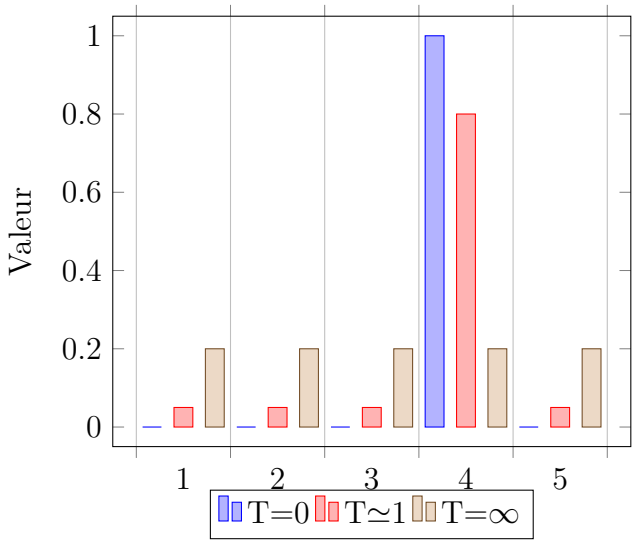
La distillation défensive consiste à adoucir les frontières entre les topologies des classes lors de l'entraînement. Les systèmes ainsi protégés sont moins instables. Les premières distillations, insiprées de Hinton et al. [33] et réalisées par Papernot et al. [52], étaient vulnérables à diverses attaques, pour les même raisons que les attaques en boîte noire ont tué les premiers entraînements antagonistes. (voir précédemment).

Une nouvelle générations d'entraînements par distillation défensive [51] voit le jour : au lieu de nécessiter un entraînement sur des exemples antagonistes, la méthode adoucit la courbe de réponse de l'algorithme avec trois principales caractéristiques :

- la présence de la classe **exception** (*outlier*) [28, 35]
- l'estimation de l'incertitude du réseau. [23]

— la présence d'une **température** : plus celle-ci est élevée, plus les classes ont la même probabilité (voir figure 2.11)

Ces méthodes permettent de défendre efficacement le réseau et sont *agnostiques de l'attaque*, pusiqu'elles ne nécessitent pas d'exemples antagonistes en entraînement.



Encodage d'un exemple de la quatrième classe (0,0,0,1,0) à différentes températures. Une augmentation de la température diminue le rapport entre signal et bruit.

Figure 2.11 Exemple de variation de température de données

Régularisation

La régularisation pénalise des comportements instables d'un réseau de neurones. La recherche de nouveaux types de pénalité permet d'obtenir d'autres comportements à la fin de l'entraînement, quand le modèle est déployé en production.

Cisse et al. [13] proposent une méthode d'entraînement pour régulariser la réponse de réseaux de neurones particuliers créés dans le papier : les Réseaux de Neurones Parseval. Ces réseaux ont des particularités uniques :

1. les fonctions représentées par chaque couche ont une constante de Lipschitz inférieure à 1, les rendant sous-linéaires [2]
2. Pour garantir le point 1, les matrices de poids des couches convolutionnelles et denses sont des matrices *quasi-orthonormales* : la multiplication de cette matrice avec sa transposée est très proche d'une matrice identité.

2. Cela signifie que les perturbations nécessaires pour avoir un impact doivent être plus fortes, car l'adversaire ne peut pas compter sur un comportement aux variations très fortes.

3. Quand deux couches sont aggrégées, elles ne sont pas sommées, mais l'on prend une combinaison convexe de celles-ci.

Toutes ces précautions assurent que le comportement du réseau est extrêmement stable. Un effet très intéressant est rapporté par les auteurs : le réseau de neurones devient *plus précis sur les exemples sains en plus d'être robuste*.

Ross et Doshi-Velez [58] proposent une méthode de régularisation basé sur un principe de double rétropropagation de Drucker et Cun [19]. La double rétropropagation change la fonction de perte : au lieu de de pénaliser uniquement la variation de l'erreur, le mécanisme ajoute une pénalité en fonction de la stabilité de la réponse par rapport à de légère variations de l'entrée : ainsi, le système se stabilise en essayant de réduire l'erreur, mais aussi en réduisant la réactivité à une variation de l'entrée.

Les résultats sont très convaincants : outre une plus grande robustesse aux exemples antagonistes, les chercheurs démontrent que les attaques contre ces réseaux protégés créent des exemples très ressemblants à la classe *cible*, et pas à la classe *source*.

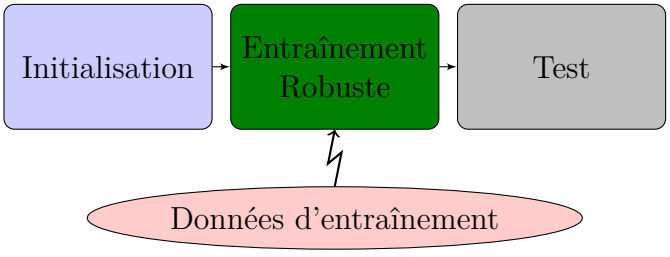


Figure 2.12 Le flot de tâches d'un système protégé avec le paradigme de robustesse

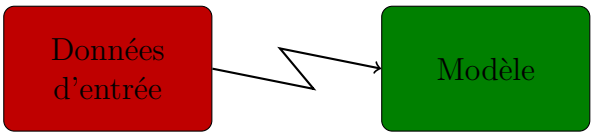


Figure 2.13 Le modèle de protection par robustesse

CHAPITRE 3 PROCESSUS DE RECHERCHE

3.1 Définitions et Termes : une autre modélisation sans statistiques

On définit un espace de problèmes P, un espace solution S(P), et un oracle O(P,S) tel que :

- P est un ensemble admettant un plus grand élément (par exemple, un espace de Banach)
- S est un ensemble de données sans nécessairement de structure
- O est un oracle (dépendant de P et de S) qui associe à tout élément de P une fonction f :

$$\boldsymbol{S \rightarrow [0,1]^n, n \in \mathbb{N}}$$
$$\boldsymbol{e \rightarrow fit}$$

où fit est une mesure de la pertinence de la solution. On pose arbitrairement 1 comme la réponse parfaite, et 0 comme la plus mauvaise réponse.

On définit par système de Machine Learning tout algorithme qui permet d'associer à un ensemble de données dans P (souvent un sous-ensemble très restreint) une fonction ϕ qui a tout point de P associe un point de S. On pose :

$$\forall x \in P, \phi(x) = \max_x(O(x)) - L(x)$$

Pour un point de P, p, et un seuil de validité, v, on définit l'ensemble des solutions $\sigma_{p,v}$ tel que :

$$\sigma_{p,v} = \{x \in S,\ O(p)(x) \geq v\}$$

Le but courant des algorithmes de Machine Learning (ML) est de minimiser L par la norme de leur choix.

De ces quelques définitions, on peut dériver tous les termes existants en Intelligence Artificielle.

Apprentissage Supervise Statistique Ici, P est l'ensemble de toutes les données possibles.

S est l'ensemble des étiquettes possibles.

O est la fonction qui associe pour chaque point p de P une fonction f(p) qui a chaque étiquette associe une pertinence. Pour un jeu de données parfait et sans erreur, et en posant s_p l'étiquette de p, on a :

$$\begin{aligned} f &: S \to 0, 1 \\ f(x) &= \delta_{s_p,x} \end{aligned}$$

Avec δ le symbole de Kroenecker, valant 1 quand $x = s_p$. Par exemple, pour un classificateur binaire sur un ensemble P=$\mathbb{R}^2$, $n = 1, S = a, b$, et on définit précision et rappel pour la classe a comme suit :

$$\begin{aligned} \phi_a &= \{x \in P,\ \phi(x) = a\} \\ O_a &= \{x \in P,\ \sigma_{x,1} \supseteq \{a\}\} \\ B &= \phi_a \cap O_a \\ \textbf{Precision} &= \frac{\int_B 1ds}{\int_{\phi_a} 1ds} \\ \textbf{Rappel} &= \frac{\int_B 1ds}{\int_{O_a} 1ds} \end{aligned} \tag{3.1}$$

Apprentissage Non-Supervise Statistique Ici, S=P, et l'oracle est une fonction qui a $x \in P$ associe une fonction caractérisant la proximité désirée (dans le problème voulu) entre le point de S (qui est un autre point de P ici) et le point de P.

Motivations On montre ici que le paradigme probabiliste de l'Intelligence Artificielle n'est pas nécessaire pour définir les algorithmes. Cette précaution nous permet de parler d'hybridation avec la Logique sans être dans l'erreur.

Note : Les fonctions logiques-dérivables sont la première itération et sont très classiques dans leur méthode d'apprentissage. À terme, l'hybridation entre des règles logiques et un apprentissage statistique pourrait être plus exotique, d'où la présence de cette proposition de cadre, pour formaliser ce qui, possiblement, pourrait s'écarter des statistiques (notamment à cause du postulat iid, souvent remis en question en IA Antagoniste)

3.2 Système Hybride Neuronal : la couche Logique-Dérivable

3.2.1 Contexte

Nous voulons, pour résister aux attaques antagonistes, un système robuste où l'espace de l'attaquant est restreint. Les fonctions d'activation ReLU encode une grande quantité d'information : leur image couvre le domaine $\mathbb{R}_+$, qui est souvent restreint par les encodages informatique à un float 32bits. La ReLU, par sa valeur positive qui restreint le signe, peut encoder 31 bits d'informations. En créant un neurone élémentaire, capable de n'encoder qu'un bit d'information, mais restant dérivable, nous pourrions créer des couches capables d'apprentissage, tout en restreignant fortement l'espace de l'attaquant.

3.2.2 La fonction d'activation Heaviside-Identité

Pour ce faire, nous créons la fonction *Heaviside-Identité*, définie comme suit :

$$HSID(x) = \begin{cases} \alpha HS(x), \ \textit{pour la propagation} \\ x, \textit{pour la rétro-propagation} \end{cases}$$

Avec HS la fonction de heaviside :

$$HS(x) = \begin{cases} 1, \ \textit{pour } x \geq 0 \\ 0, \ \textit{pour } x < 0 \end{cases}$$

Cette fonction d'activation permet l'entraînement du système avec l'algorithme de rétro-propagation, tout en utilisant une fonction dont la dérivée classique, l'impulsion de dirac δ, ne permet pas l'entraînement.

3.2.3 La couche de décalage

Pour écraser au plus fort les caractéristiques, dans l'esprit de Xu et al. [66], nous utilisons une fonction Logique-Dérivable dans une couche de filtrage, inspirée des mécanismes d'attention.

Ainsi, au lieu d'avoir une couche composée de neurones ayant cette règle pour leur activation :

$$a_{dense} = \sigma(\sum_{i=1}^{n} \alpha_i x_i + b_{dense})$$

Chaque neurone de la couche de déclage est liée à un nombre restreint de neurones (entre 1 et 10, dans nos expériences), et le neurone ne possède, dans sa version simplifiée, qu'un biais par neurone. :

$$a_i = HSID(x_i - b_i)$$

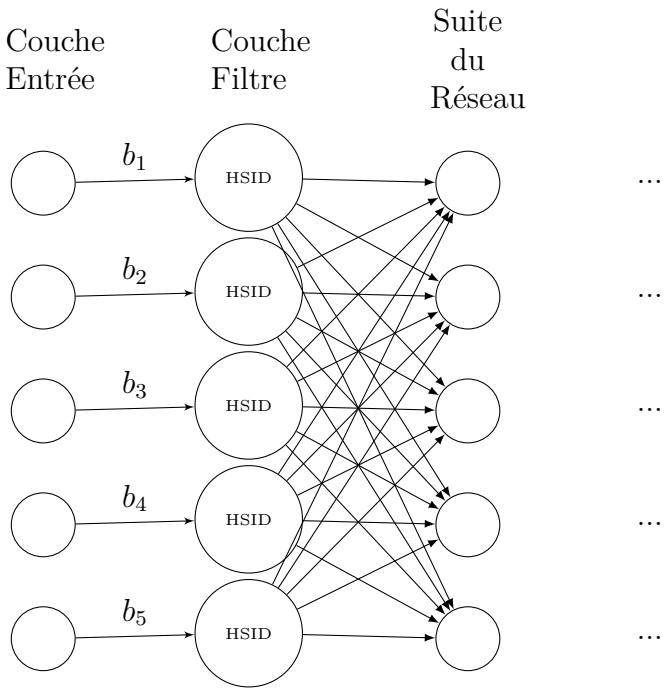


Figure 3.1 Un exemple de couche de décalage, comme celle qui sera utilisée pour protéger le réseau dans la première défense, sur le jeu de données MNIST.

Ces neurones permettent de créer un filtre très simple, sur la valeur de la donnée d'entrée : ainsi, au lieu de donner directement au réseau la valeur brute, on ne donne au réseau qu'un indice : si l'activation neuronale est à 1, alors c'est que $x_i > bi$. Le biais devient alors un outil de filtrage d'information très puissant.

Par exemple, supposons que les x_i sont des valeurs réparties entre 0 et 1, et qu'il n'y ait, dans cette version, qu'une sortie par neurone. Quand les biais sont situés à 0.5, les valeurs vont de -0.5 à 0.5, et la fonction Heaviside-Identité transforme une plage de valeurs d'entrée

en deux valeurs possibles : si la valeur d'entrée est supérieure à 0.5, alors l'activation sera de 1, sinon, elle sera de 0. C'est la plage la plus large que l'on puisse avoir : à ce moment, le réseau filtre le moins possible la donnée.

Si le biais monte à 0.9, alors la donnée devra être dans l'intervalle [0.9,1] afin que le signal passe le filtre et entre dans le réseau. Notons que la situation est, en termes d'information, la même qu'avec un biais à 0.1 : nous avons 90% des données dans une case, et 10% dans l'autre dans les deux cas, seule la valeur d'activation change.

Ceci permet apprendre les paramètres nécessaires pour créer un écrasement de caractéristiques adapté à chaque jeu de données : le biais permet de couper des caractéristiques qui ne sont pas assez prononcées, et, quand il y a plusieurs neurones pour chaque entrée, vient régler un problème essentiel des exemples antagonistes : la plupart des exemples antagonistes sont invisibles pour les humains. Grâce à cette transformation et cette discrétisation, ils sont aussi invisibles pour les algorithmes, car la micro-variation propre aux exemples antagonistes est, la plupart du temps, filtrée.

Si l'Encodage Discret[11], les Réseaux de Neurones Binaires[14], l'Écrasement de Caractéristiques[66], et les Gradients Synthétiques[36] existent déjà, ce mémoire est, à notre connaissance, la seule tentative existante de combiner ces éléments.

Ce réseau de neurones est appelé *"Candidate A"* dans l'article.

Deux autres candidats, inspirés des CNN, sont présents et décrits dans l'article.

3.3 Objectifs

Nous voulons montrer l'existence d'un jeu de données sur lequel, à puissance équivalente, un système hybride, utilisant juste la fonction HSID, montre des performances équivalentes ou en tous cas acceptables par rapport à un système standard. Une fois cet objectif artificiel atteint, nous appliquerons cette fonction d'activation au sein d'une couche de filtrage sur deux jeux de données différents : MNIST[40], et CIFAR10[37].

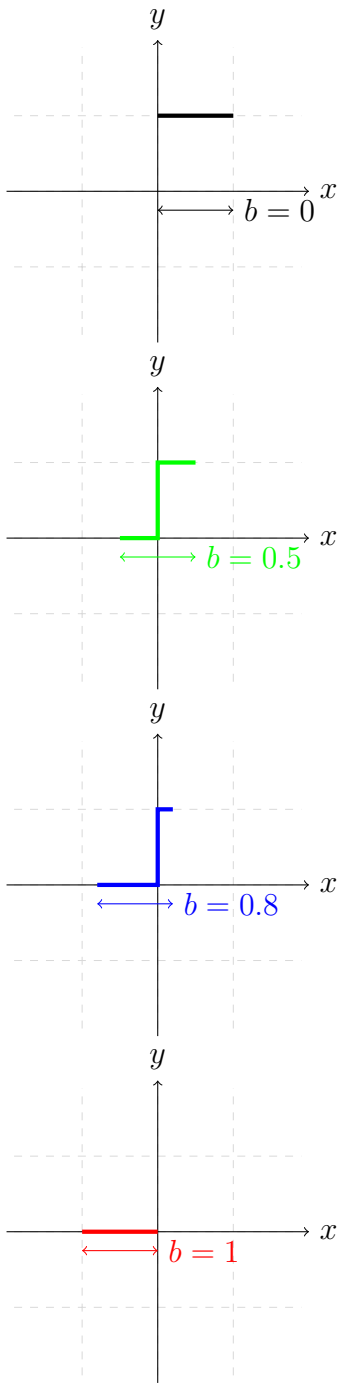


Figure 3.2 Équilibrage du couple détection-robustesse grâce au biais

En changeant le biais, on change la position de la plage de valeurs, et l'on permet de filtrer plus ou moins fort en changeant la barrière à franchir pour émettre le signal.

3.4 Modèle proposé

3.4.1 Jeu sur un N-Damier

Pour vérifier le bien fondé de notre solution dans un environnement sain, on veut tester les capacités du système sur un jeu de données avec un nombre maîtrisé d'informations. Un banc d'essai de réseaux de neurones sera développé, et différentes versions seront comparées.

3.4.2 Attaque antagoniste sur MNIST

Nous attaquerons un réseau convolutionnel classique [1] avec une FGSM[26], puis nous ajouterons la couche de protection et réentraînerons le réseau afin de voir sa robustesse.

3.4.3 Attaque antagoniste sur CIFAR10

Nous tenterons d'étendre notre filtre, très destructeur de caractéristiques, sur un jeu de données où la sémantique est complexe, pour vérifier si notre filtrage actuel est effectivement aggressif. Là encore, nous utiliserons des attaques FGSM.

3.5 Méthodologie expérimentale

3.5.1 Jeu sur N-Damier

Jeu de données synthétique

Nous voulons exposer un jeu de données qui met en œuvre une topologie aux courbures très élevées en certains points. Pour ce faire, nous créons un N-Damier ou *N-Dimensional Checkerboard*.

Ce damier est créé de la façon suivante :

On choisit quatre paramètres :

- un nombre de dimensions N (par défaut, 2)
- un pas inter-case i (par défaut,1)
- une longueur maximum L (par défaut, 3)
- une distribution D (par défaut, une distribution uniforme)

On peu ainsi créer un jeu de données synthétique comme suit :

1. classifiant les chiffres écrits à la main et entraîné sur le jeu de données MNIST[40]

1. On choisit un point de coordonnées $(x_1, ..., x_N)$ dans l'espace délimité par

$$((x_i)_{i\in[\![1,N]\!]}/\forall i \in [\![1, N]\!], x_i \in [0, L])$$

. Chaque coordonnée est tirée aléatoirement selon D.

2. On entre la coordonnée comme donnée d'entrée
3. Pour chaque coordonnée, on calcule :

$$(\sum_{i\in[\![1,N]\!]} x_i \div L\ )\ mod\ 2$$

4. si le résultat est pair (0) on attribue la valeur V=1 ([0,1] en notation *one-hot*) sinon, on attribue la valeur V=0 ([1,0] en notation *one-hot*)
5. on rentre V comme étiquette associée à la coordonnée.

Pour les valeurs par défaut, la construction du dataset est équivalente à lancer des flèchettes au hasard sur un carré de trois cases par trois cases, sur un jeu d'échec. On mesure la position en x,y sur un repère orthonormé par la première case (c'est à dire, l'angle entre (Ox) et (Oy) est l'angle droit de la première case en bas à gauche, et la norme des vecteurs de base est donnée par le côté de la première case)

Structure du banc d'essai neuronal

Nous créons un réseau de neurones classique, possédant une couche d'entrée, une couche à 30 neurones intermédiaires ReLU, une couche variable de 300 unités que l'on nommera V, une couche de neurones de pré-sortie à 20 neurones, et une couche avec deux 'logits', donnant la probabilité d'être respectivement une case noire ou une case blanche (case 0 ou 1). Une figure est présentée ci dessous.

— Dans l'expérience 1 version 1, on entraîne le réseau de neurones avec, pour la couche V, 300 neurones avec activation ReLU.
— Dans l'expérience 1 version 2, on entraîne le réseau de neurones avec, pour la couche V, 300 neurones avec activation Logique-Dérivable *Fake-Heaviside*.
— Dans l'expérience 1 version 3, on entraîne le réseau de neurones avec, pour la couche V, 150 neurones avec activation Logique-Dérivable et 150 avec activation ReLU.

Ainsi, les réseaux ont la même forme et la même complexité en termes de graphe de calcul.

Les résultats de l'entraînement sont disponibles figure 3.4, et faits sur un 2-Damier de 3×3 avec le code python `heaviside_ board.py`

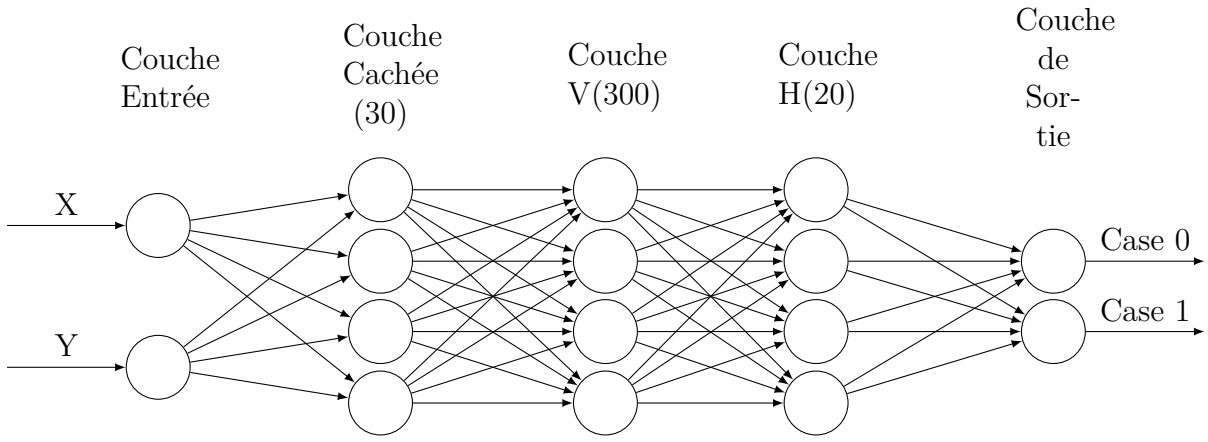


Figure 3.3 l'expérience de base pour vérifier la capacité des réseaux Hybrides

la couche V est variable, la couche H est une couche densément connectée avec des fonctions d'activation ReLU.

Il est donc possible d'entraîner avec une faible perte des réseau de neurones avec la fonction *Logique-Dérivable*.

Pour l'évaluation de la perte, la précision finale moyenne d'entraînement est donnée dans le tableau 3.1 :

Ceci répond à notre première question de recherche numéro 1 : il est possible d'entraîner une fonction logique-dérivable, aux prix d'une augmentation moyenne de l'erreur d'environ 38% si elle est utilisée seule, et d'environ 1% si elle est hybridée avec une fonction d'activation ReLU

Tableau 3.1 Résultat du test de précision Logique-Dérivable contre ReLU

Logique-Dérivable	ReLU	Hybride
0.91852	0.941176	0.9400808

3.5.2 Expériences de la partie réelle : Défense face à des attaques réelles.

Préparation d'un ConvNet (MNIST)

Nous construisons et entraînons un réseau convolutionnel, inspiré de différentes architectures connues [2]. Nous avons choisi ces structures car elles sont déjà utilisées par Xu et al. [66] [3].

L'architecture est résumée comme sur la figure 3.5 page 52.

Le réseau est entraîné sur 15 époques, avec un taux d'apprentissage $l = 10^{-4}$, par la méthode *rmsprop*, dérivée de l'algorithme classique de rétropropagation. Un *dropout* de 25% est appliqué sur les couches pleinement connectées pendant l'entraînement.

Attaque antagoniste FGSM

Nous utilisons le module Python Cleverhans[54] pour réaliser simplement les attaques sur les réseaux de neurones, que nous avions réalisé spécialement dans un mélange de Keras[12] et de TensorFlow[7] pour des raisons de compatibilité.

Les attaques réalisées sont des FGSM avec des forces d'attaque $\epsilon \in [0.1, 0.5]$, 0.1 signifie que la valeur de chaque pixel peut être visuellement modifiée, au maximum, d'un dixième de la distance entre blanc et noir, 0.5 signifie la moitié de cette même distance.

L'attaque est réalisée en *Boîte Noire*, car, la FGSM utilisant des gradients absents de notre système (La fonction Heaviside-Identité a une dérivée nulle en tout point de $\mathbb{R}^*$, en mode inférence), elle ne peut être directement utilisée sur notre système. Ainsi, nous exécutons une attaque en Boîte Noire sur le système sain, en utilisant $\lambda = 0.1$ pour la valeur de la constante d'extension jacobienne du jeu de données (voir la revue de littérature partie Attaque en Boîte Noire / Augmentation Jacobienne), et un jeu de données initial de 150 exemples pour générer les exemples antagonistes, autant que nécessaire pour une attaque fonctionnelle[53].

Nous avons donné les explications nécessaires à la réponse de la première question de recherche, et des éléments de compréhension du protocole expérimental de la deuxième. Le reste du travail est inclus dans l'article de revue joint à ce mémoire.

2. Citons notamment les travaux de Carlini et Wagner

3. Cette figure est générée en adaptant le code de `https://github.com/gwding/draw_convnet` .Merci Weiguang Gavin Ding de laisser son code Libre !

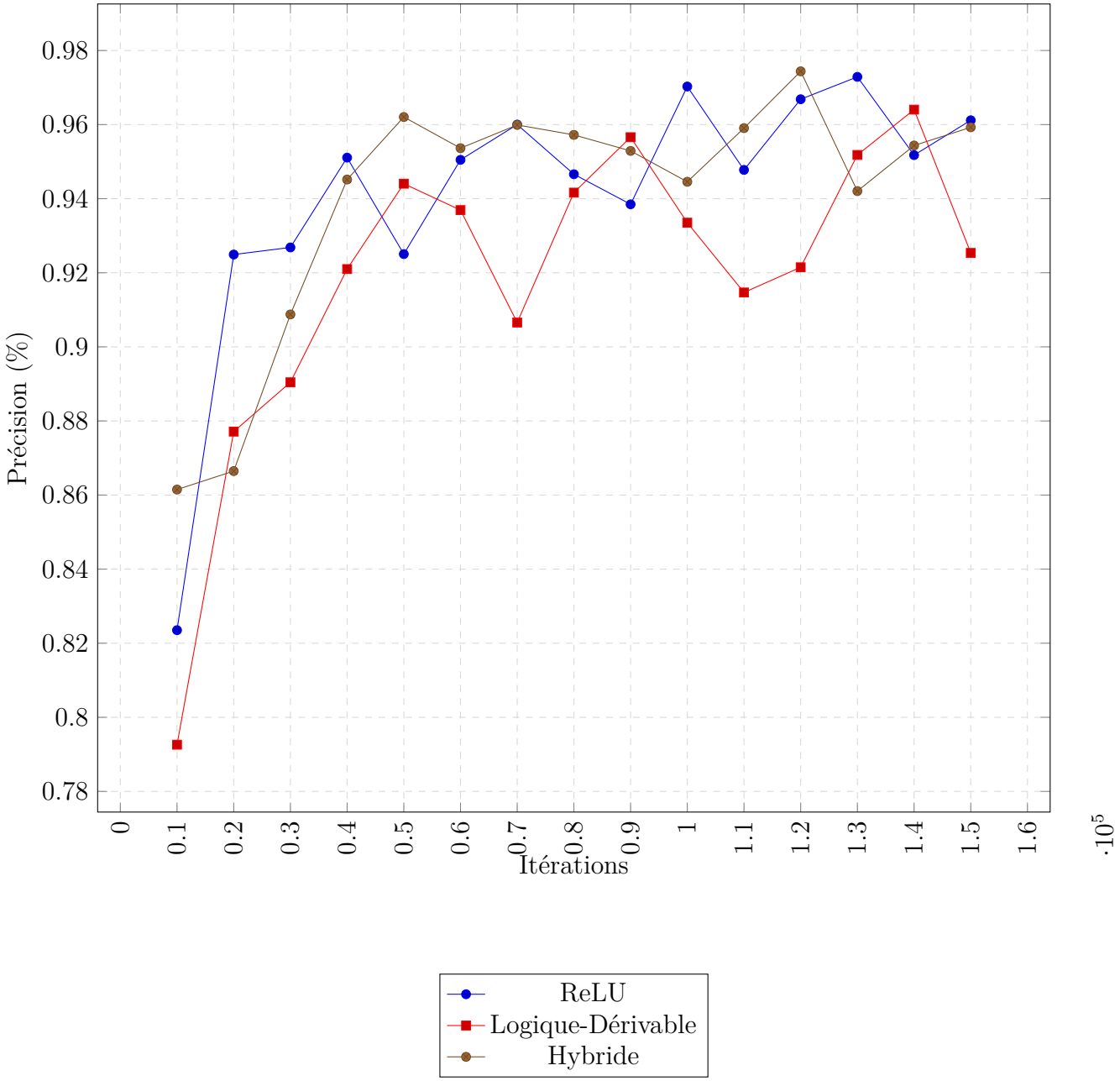


Figure 3.4 Comparaison des différents types d'entraînement

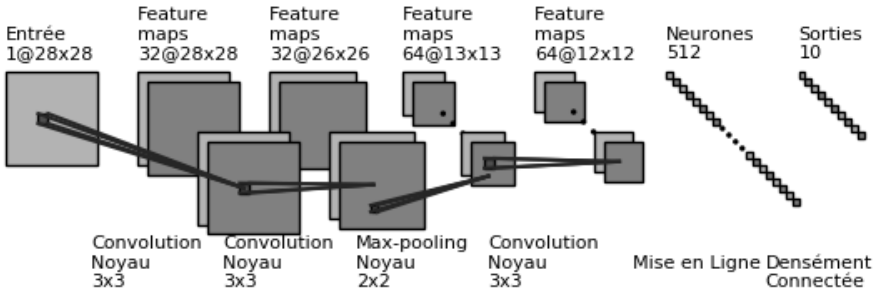


Figure 3.5 La structure du réseau convolutionnel que nous avons entraîné pour l'attaque-défense avec MNIST

CHAPITRE 4 DISCUSSION GÉNÉRALE

4.1 Avant-Propos

L'article de revue est joint à ce mémoire, en annexe.

Anecdote

À titre d'anectode, les Réseaux Spartiates (titre anglais : *Spartan Networks*) ont été nommés comme tels à cause de l'agogée spartiate, qui était l'entraînement militaire de tous les jeunes hommes de Sparte. Il durait plus de dix ans, pendant lesquels les recrues étaient soumises à des conditions qualifiables de traumatiques, afin de s'endurcir. Un fait notable était que ces recrues étaient régulièrement affammées pour leur donner la capacité de résister aux sièges. Cet entraînement a fait de Sparte une des cité grecques les plus victorieuses au combat.

En bref, nos réseaux ont étés nommés ainsi car :

— Ils sont très robustes,
— Leur entraînement est traumatique...
— ... Et il dure 10 ans.

L'article de Journal est joint au mémoire en annexe, et a été déposé pour revue par les pairs dans le journal *Computers and Security*.

4.2 Revue critique de littérature : L'avènement du compromis en IA

Les Réseaux Spartiates démontrent qu'il est possible d'échanger de la précision contre de la robustesse. De tels compromis sont connus dans tous les domaines de l'ingénierie logicielle, et, dans le cadre de la sécurité, la Pyramide CID/CIA ou le Double Tétraèdre Antagoniste sont utilisés pour établir différents types de compromis.

Nous notons que la littérature ne précise pas toujours clairement les modèles d'attaques, les compromis réalisés, ou même les types d'attaques réalisés pour démontrer l'existence d'une résistance.

Ainsi, si nous prenons un système de classification en Apprentissage Machine, les critères seraient les suivants :

— la Performance (en % précision de test)
— la Robustesse (en % précision antagoniste)

— le Coût d'entraînement (en s.GPU)

Nous avons clairement établi le compromis réalisé par les réseaux spartiates dans le papier : le système échange de la Performance et du Coût d'entraînement contre de la Robustesse. Un tableau résumé est disponible en Conclusion, voir tableau 5.1 page 56.

Nous ésperons à l'avenir voir une standardisation des méthodes d'essai en IA Antagoniste, notamment grâce à l'apport de domaines comme la cryptographie, qui a longtemps été étudiée comme un art avant d'être rigoureusement définie comme une science par Claude Shannon dans le milieu du XX$^{\text{ème}}$ siècle.

4.3 Méthodologie

Nous nous sommes basés sur une méthodologie simple de comparaison entre un système reconnu comme standard (pour éviter le biais de sélection ou *cherry-picking*) et des variantes de puissance égale de celui-ci.

Cette méthode permet de démontrer l'existence d'un cas favorable, mais pas l'universalité de l'effet.

CHAPITRE 5 CONCLUSION

5.1 Synthèse des travaux

Les résultats sont donc les suivants : certaines structures de réseau de Neurones profond utilisant les fonctions logiques-dérivables, dotées d'une périodicité, sont robustes à des attaques standard. Notre meilleur système perd 0.5% de précision, mais est jusqu'à 80% plus précise sous le feu d'une attaque FGSM pour des forces d'attaques élevées.

Un résumé est donné dans le tableau ci-dessous : les + et - sont donnés par rapport à tous les autres modèles d'apprentissage machine, pas seulement d'apprentissage profond.

Précision : étant donné un entraînement sur une quantité adéquate de données, quel est le taux d'erreur du système sur des données inconnues ?

⇒ Les Réseaux Spartiates, soumis à de fortes contraintes, sont moins précis que leurs homologues.

Capacité : quelle est la complexité de la règle que je peux apprendre, par rapport au nombre de paramètres évoluant au cours de l'apprentissage ?

⇒ Les Réseaux Spartiates utilisent des fonctions de Heaviside pour détruire une grande partie de l'information qu'ils recoivent. Leur capacité est volontairement réduite. Les résultats obtenus sur le jeu de données CIFAR10 étaient trop faibles pour être publiés, à cause notamment de cela.

Stabilité : quel est le risque de sur-apprentissage de mon algorithme ? (L'algortihme est capable de parfaitement inférer sur des données d'entraînement, mais pas de généraliser à d'autres données)

⇒ Les Réseaux Spartiates n'ont qu'une seule régularisation et nous n'avons pas constaté de surentraînement à aucune étape du processus.

Vitesse : à quelle vitesse mon algortihme apprend à inférer sur de nouvelles données ?

⇒ Les Réseaux Spartiates possèdent une phase d'apprentissage unique pendant laquelle *la précision n'augmente pas*. Cette phase pénalise fortement la vitesse d'apprentissage des réseaux.

Hyperparamètres : combien de paramètres ne peuvent être appris, et restent à l'appréciation de l'expert ?

⇒ Les Réseaux Spartiates n'ajoutent que peu d'hyperparamètres. Même si leur ajustement peut être délicat, les experts en sciences des données sont parfaitement capables de conduire des recherches d'hyperparamètres délicates.

Robustesse : quelle est la robustesse des algorithmes face à des attaques adversairielles

(*connues actuellement*) ?

⇒ C'est sur cette métrique que les Réseaux Spartiates brillent. Ils sont parmi les réseaux de neurones les plus robustes.

Transparence : est-il possible d'avoir une *explication* de l'inférence du réseau ?

⇒ Il est possible de mieux comprendre les premières couches des Réseaux Spartiates, car les valeurs sont des valeurs de filtrage de données. Notons qu'en règle générale, cette caractéristique est un point faible des réseaux de neurones profonds.

5.1.1 Réponse aux questions de recherche

— **Est-il possible d'entraîner un réseau de neurones possédant des fonctions logiques, non-dérivables, par rétro-propagation ?**

⇒ L'expérience présente dans le mémoire déontre qu'un tel entraînement est possible, grâce à la méthode des fonctions d'activations composites nommées *Logiques-Dérivables*

— **Si oui, pour un gain en précision sur une attaque standard, de combien la précision diminue ? Quelle est l'augmentation du temps d'entraînement ?**

⇒ Pour un réseau convolutionnel standard, sur le jeu de données MNIST, une perte de précision de -0.5% en conditions de laboratoire permet d'obtenir un gain de précision de 20% pour une attaque boîte-noire FGSM standard, et de jusqu'à 80% pour une attaque boîte-noire FGSM de puissance élevée. Leurs coefficients de puissance sont respectivement $\epsilon = \{0.3, 0.45\}$.

Tableau 5.1 Comparaison entre Réseaux Spartiates et Réseaux de Neurones Profonds classiques

Réseau	Spartiate	Standard
Précision	++	+++
Capacité	- -	-
Stabilité	++	+
Vitesse	- - -	-
Hyperparamètres	-	-
Robustesse	++	- - -
Transparence	- -	- - -

5.2 Limitations de la solution proposée

5.2.1 Précision

Les Réseaux Spartiates subissent de fortes contraintes, et ces contraintes diminuent la performance, en termes de précision.

5.2.2 Capacité

Les Réseaux Spartiates limitent fortement la capacité du réseau à approximer des règles aussi complexes qu'avec un réseaux de neurones utilisant des activations linéaires rectifiées ($ReLU$), ce qui est le standard à l'époque de la rédaction de cette conclusion. Les Réseaux Spartiates fonctionnent bien sur MNIST car ce jeu de données est relativement simple à classifier. Des tâches plus complexes nécessiteraient sans doute plus de moyens en termes de quantité de neurones.

5.2.3 Vitesse d'entraînement

Comme montré précédemment, les Réseaux Spartiates sont plus lents à entraîner que leurs homologues réguliers, et présentent un moment de latence. Ce phénomène est nouveau. Habituellement, les réseaux voient leur précision augmenter rapidement et se stabiliser. La précision des Réseaux Spartiates reste tout d'abord minimale, avant d'augmenter dans une phase transitoire, puis de se stabiliser et d'augmenter lentement, avec un comportement asymptotique assez similaire aux autres réseaux.

5.2.4 Convergence

Les réseaux de neurones Spartiates sont très sensibles à leur initialisation. Dans certains cas, le système est incapable de sortir de minima locaux, laissant leur précision à des valeurs de l'ordre de 15% sur un problème à 10 classes, où le score minimal est de 10%. L'exploration des hyperparamètres de ces réseaux est potentiellement plus fastidieuses que leurs équivalents linéaires rectifiés.

5.3 Améliorations futures

Ce travail est fondateur en ce sens qu'il amène une nouvelle forme de réseaux de neurones *auto-adversairiels*. Les améliorations futures sont nombreuses :

— Peut-on créer des couches de destruction de données plus progressives, des couches adversarielles moins puissantes mais plus nombreuses ? Nous pourrions sans doute augmenter la précision grâce à ces couches plus 'douces'. Nos pauvres Spartiates ont assez souffert comme ça.
— Est-il possible de compenser la dureté des neurones binaires en multipliant le nombre de neurones par couches, et ce, sans réduire la robustesse ?
— Peut-on détruire l'information dans les couches *supérieures* uniquement ? Cette destruction permettrait de filtrer des informations *complexes*, et de créer un vrai filtrage sémantique. Cette direction nous paraît la plus prometteuse.

RÉFÉRENCES

[1] "ChrysaLabs - Technologie d'analyse de la fertilité des sols". En ligne : http://www.chrysalabs.com/

[2] "Intelligence artificielle : la 4e révolution industrielle est en marche". En ligne : http://www.magazine-decideurs.com/dossiers/intelligence-artificielle-la-4e-revolution-industrielle-est-en-marche

[3] "L'intelligence artificielle, une révolution industrielle ?". En ligne : https://www.franceculture.fr/emissions/entendez-vous-leco/lintelligence-artificielle-une-revolution-industrielle

[4] "L'intelligence artificielle au service de la médecine | UdeMNouvelles". En ligne : https://nouvelles.umontreal.ca/article/2018/04/27/l-intelligence-artificielle-au-service-de-la-medecine/

[5] "Myelin -". En ligne : https://www.myelin.ca/

[6] "SpamAssassin : Welcome to SpamAssassin". En ligne : https://spamassassin.apache.org/

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, et X. Zheng, "TensorFlow : Large-scale machine learning on heterogeneous systems", 2015, software available from tensorflow.org. En ligne : https://www.tensorflow.org/

[8] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, et J. D. Tygar, "Can machine learning be secure ?". ACM Press, 2006, p. 16. DOI : 10.1145/1128817.1128824. En ligne : http://portal.acm.org/citation.cfm?doid=1128817.1128824

[9] B. Biggio, G. Fumera, et F. Roli, "Adversarial pattern classification using multiple classifiers and randomisation", dans *Joint IAPR International Workshops on Statistical Tech-*

niques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR). Springer, 2008, pp. 500–509.

[10] B. Biggio, B. Nelson, et P. Laskov, "Poisoning attacks against support vector machines", *arXiv preprint arXiv :1206.6389*, 2012.

[11] J. Buckman, A. Roy, C. Raffel, et I. Goodfellow, "THERMOMETER ENCODING : ONE HOT WAY TO RESIST ADVERSARIAL EXAMPLES", p. 22, 2018.

[12] F. Chollet *et al.*, "Keras", `https://keras.io`, 2015.

[13] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, et N. Usunier, "Parseval Networks : Improving Robustness to Adversarial Examples", *arXiv :1704.08847 [cs, stat]*, Avr. 2017, arXiv : 1704.08847. En ligne : `http://arxiv.org/abs/1704.08847`

[14] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, et Y. Bengio, "Binarized Neural Networks : Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1", *arXiv :1602.02830 [cs]*, Fév. 2016, arXiv : 1602.02830. En ligne : `http://arxiv.org/abs/1602.02830`

[15] N. Dalvi, P. Domingos, Mausam, S. Sanghai, et D. Verma, "Adversarial classification". ACM Press, 2004, p. 99. DOI : `10.1145/1014052.1014066`. En ligne : `http://portal.acm.org/citation.cfm?doid=1014052.1014066`

[16] T. Darrell, M. Kloft, M. Pontil, G. Rätsch, et E. Rodner, "Machine Learning with Interdependent and Non-identically Distributed Data", p. 38.

[17] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, et D. H. Chau, "Keeping the Bad Guys Out : Protecting and Vaccinating Deep Learning with JPEG Compression", *ArXiv e-prints*, Mai 2017.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, et L. Fei-Fei, "ImageNet : A Large-Scale Hierarchical Image Database", dans *CVPR09*, 2009.

[19] H. Drucker et Y. L. Cun, "Double backpropagation increasing generalization performance", dans *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. ii, July 1991, pp. 145–150 vol.2. DOI : `10.1109/IJCNN.1991.155328`

[20] G. F. Elsayed, S. Shankar, B. Cheung, N. Papernot, A. Kurakin, I. Goodfellow, et J. Sohl-Dickstein, "Adversarial Examples that Fool both Computer Vision and

Time-Limited Humans", *arXiv :1802.08195 [cs, q-bio, stat]*, Fév. 2018, arXiv : 1802.08195. En ligne : http://arxiv.org/abs/1802.08195

[21] R. Feinman, R. R. Curtin, S. Shintre, et A. B. Gardner, "Detecting adversarial samples from artifacts", *arXiv preprint arXiv :1703.00410*, 2017.

[22] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, et W. Lee, "Polymorphic Blending Attacks", Jan. 2006.

[23] Y. Gal et Z. Ghahramani, "Dropout as a Bayesian Approximation : Representing Model Uncertainty in Deep Learning", *arXiv :1506.02142 [cs, stat]*, Juin 2015, arXiv : 1506.02142. En ligne : http://arxiv.org/abs/1506.02142

[24] Goldberg, Roeder, Gupta, et Perkins, "Eigentaste : A Constant Time Collaborative Filtering Algorithm." *Information Retrieval*, 2001.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, et Y. Bengio, "Generative Adversarial Nets", dans *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, et K. Q. Weinberger, éds. Curran Associates, Inc., 2014, pp. 2672–2680. En ligne : http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[26] I. J. Goodfellow, J. Shlens, et C. Szegedy, "Explaining and Harnessing Adversarial Examples", dans *arXiv :1412.6572 [cs, stat]*, Déc. 2014, arXiv : 1412.6572. En ligne : http://arxiv.org/abs/1412.6572

[27] J. Grefenstette et A. Schultz, "An Evolutionary Approach to Learning in Robots." NAVAL RESEARCH LAB WASHINGTON DC, NAVAL RESEARCH LAB WASHINGTON DC, Rapp. tech., 1994. En ligne : http://www.dtic.mil/docs/citations/ADA294080

[28] K. Grosse, P. Manoharan, N. Papernot, M. Backes, et P. McDaniel, "On the (statistical) detection of adversarial examples", *arXiv preprint arXiv :1702.06280*, 2017.

[29] C. Guo, M. Rana, M. Cisse, et L. van der Maaten, "Countering Adversarial Images using Input Transformations", *arXiv :1711.00117 [cs]*, Oct. 2017, arXiv : 1711.00117. En ligne : http://arxiv.org/abs/1711.00117

[30] D. Harrison et D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air", *Journal of Environmental Economics and Management*, vol. 5, no. 1, pp. 81

– 102, 1978. DOI : https://doi.org/10.1016/0095-0696(78)90006-2. En ligne : http://www.sciencedirect.com/science/article/pii/0095069678900062

[31] K. He, X. Zhang, S. Ren, et J. Sun, "Deep residual learning for image recognition", dans *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[32] D. Hendrycks et K. Gimpel, "Early methods for detecting adversarial images", *arXiv preprint arXiv :1608.00530*, 2016.

[33] G. Hinton, O. Vinyals, et J. Dean, "Distilling the Knowledge in a Neural Network", *arXiv :1503.02531 [cs, stat]*, Mars 2015, arXiv : 1503.02531. En ligne : http://arxiv.org/abs/1503.02531

[34] K. Hornik, "Approximation capabilities of multilayer feedforward networks", *Neural Networks*, vol. 4, no. 2, pp. 251 – 257, 1991. DOI : https://doi.org/10.1016/0893-6080(91)90009-T. En ligne : http://www.sciencedirect.com/science/article/pii/089360809190009T

[35] H. Hosseini, Y. Chen, S. Kannan, B. Zhang, et R. Poovendran, "Blocking transferability of adversarial examples in black-box learning systems", 03 2017.

[36] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, et K. Kavukcuoglu, "Decoupled Neural Interfaces using Synthetic Gradients", *arXiv :1608.05343 [cs]*, Août 2016, arXiv : 1608.05343. En ligne : http://arxiv.org/abs/1608.05343

[37] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", p. 60.

[38] A. Krizhevsky, I. Sutskever, et G. E. Hinton, "Imagenet classification with deep convolutional neural networks", dans *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, et K. Q. Weinberger, éds. Curran Associates, Inc., 2012, pp. 1097–1105. En ligne : http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[39] K. Lang, "Newsweeder : Learning to filter netnews", dans *Proceedings of the Twelfth International Conference on Machine Learning*, 1995, pp. 331–339.

[40] Y. Lecun, L. Bottou, Y. Bengio, et P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. DOI : 10.1109/5.726791

[41] Y. LeCun et Y. Bengio, “The handbook of brain theory and neural networks”, M. A. Arbib, éd. Cambridge, MA, USA : MIT Press, 1998, ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. En ligne : http://dl.acm.org/citation.cfm?id=303568.303704

[42] M. Lessard, “La Chine qualifie l’IA de moteur de l’économie de demain | ICI.Radio-Canada.ca”. En ligne : https://ici.radio-canada.ca/nouvelle/801429/triplex-intelligence-artificielle-chine-revolution-industrielle

[43] S. Li, Y. Chen, Y. Peng, et L. Bai, “Learning More Robust Features with Adversarial Training”, *arXiv :1804.07757 [cs, stat]*, Avr. 2018, arXiv : 1804.07757. En ligne : http://arxiv.org/abs/1804.07757

[44] X. Li et F. Li, “Adversarial examples detection in deep networks with convolutional filter statistics”, *CoRR, abs/1612.07767*, vol. 7, 2016.

[45] J. Lu, T. Issaranon, et D. Forsyth, “Safetynet : Detecting and rejecting adversarial examples robustly”, *CoRR, abs/1704.00103*, 2017.

[46] Y. Luo, X. Boix, G. Roig, T. A. Poggio, et Q. Zhao, “Foveation-based mechanisms alleviate adversarial examples”, *CoRR*, vol. abs/1511.06292, 2015.

[47] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, et A. Vladu, “Towards Deep Learning Models Resistant to Adversarial Attacks”, *arXiv :1706.06083 [cs, stat]*, Juin 2017, arXiv : 1706.06083. En ligne : http://arxiv.org/abs/1706.06083

[48] D. Meng et H. Chen, “Magnet : a two-pronged defense against adversarial examples”, dans *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 135–147.

[49] J. H. Metzen, T. Genewein, V. Fischer, et B. Bischoff, “On detecting adversarial perturbations”, *arXiv preprint arXiv :1702.04267*, 2017.

[50] Nouvelle, “[Avis d’expert] La 4ème révolution industrielle ne se fera pas sans l’IT et l’IoT - Intelligence artificielle”, *usinenouvelle.com*, Mars 2018. En ligne : https://www.usinenouvelle.com/editorial/avis-d-expert-la-4eme-revolution-industrielle-ne-se-fera-pas-sans-l-it-et-l-iot.N673404

[51] N. Papernot et P. McDaniel, “Extending Defensive Distillation”, *arXiv :1705.05264 [cs, stat]*, Mai 2017, arXiv : 1705.05264. En ligne : http://arxiv.org/abs/1705.05264

[52] N. Papernot, P. McDaniel, X. Wu, S. Jha, et A. Swami, “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks”, *arXiv :1511.04508 [cs, stat]*, Nov. 2015, arXiv : 1511.04508. En ligne : http://arxiv.org/abs/1511.04508

[53] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, et A. Swami, “Practical Black-Box Attacks against Machine Learning”, *arXiv :1602.02697 [cs]*, Fév. 2016, arXiv : 1602.02697. En ligne : http://arxiv.org/abs/1602.02697

[54] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, et R. Long, “Technical report on the cleverhans v2.1.0 adversarial examples library”, *arXiv preprint arXiv :1610.00768*, 2018.

[55] I. Paré, “Aurez-vous encore un emploi demain ?”. En ligne : https://www.ledevoir.com/societe/513351/jodie-wallis-directrice-de-ai-canada-accenture

[56] A. Prakash, N. Moran, S. Garber, A. DiLillo, et J. A. Storer, “Deflecting adversarial attacks with pixel deflection”, *CoRR*, vol. abs/1801.08926, 2018.

[57] PricewaterhouseCoopers, “Intelligence artificielle, rapport pwc”. En ligne : https://www.pwc.fr/fr/espace-presse/communiques-de-presse/2017/juillet/intelligence-artificielle-un-potentiel-de-15700-milliards-de-dollars.html

[58] A. S. Ross et F. Doshi-Velez, “Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing their Input Gradients”, *arXiv :1711.09404 [cs]*, Nov. 2017, arXiv : 1711.09404. En ligne : http://arxiv.org/abs/1711.09404

[59] D. Sornette et S. von der Becke, “Crashes and High Frequency Trading”, *SSRN Electronic Journal*, 2011. DOI : 10.2139/ssrn.1976249. En ligne : http://www.ssrn.com/abstract=1976249

[60] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, et R. Fergus, “Intriguing properties of neural networks”, *arXiv :1312.6199 [cs]*, Déc. 2013, arXiv : 1312.6199. En ligne : http://arxiv.org/abs/1312.6199

[61] Tero Karras FI, “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. En ligne : https://www.youtube.com/watch?v=XOxxPcy5Gr4

[62] Tesla, “Autopilot”. En ligne : https://www.tesla.com/autopilot

[63] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, et P. McDaniel, "Ensemble Adversarial Training : Attacks and Defenses", *arXiv :1705.07204 [cs, stat]*, Mai 2017, arXiv : 1705.07204. En ligne : `http://arxiv.org/abs/1705.07204`

[64] J. Vincent, "Microsoft's new iPhone app narrates the world for blind people", Juil. 2017. En ligne : `https://www.theverge.com/2017/7/12/15958174/microsoft-ai-seeing-app-blind-ios`

[65] W. Xu, D. Evans, et Y. Qi, "Feature squeezing : Detecting adversarial examples in deep neural networks", *arXiv preprint arXiv :1704.01155*, 2017.

[66] ——, "Feature Squeezing : Detecting Adversarial Examples in Deep Neural Networks", *arXiv :1704.01155 [cs]*, 2018, arXiv : 1704.01155. DOI : `10.14722/ndss.2018.23198`. En ligne : `http://arxiv.org/abs/1704.01155`

ANNEXE A IMPLÉMENTATION DES RÉSEAUX SPARTIATES

Le code complet des Réseaux Spartiates sera disponible en *Open Source* après dépôt de ce mémoire à l'adresse URL suivante : `https://github.com/FMenet/Spartan-Networks`

Voici un morceau de code Python, adapté du fichier `AttackedModel.py` permettant de comprendre l'implémentation des Réseaux Spartiates.

```
import numpy as np
import keras
import tensorflow as tf
from tensorflow.python.platform import flags
from keras import backend as K
from keras.engine.topology import Layer


def Heaviside(x):
    return 0.5 * (tf.sign(x) + 1)

@tf.RegisterGradient("ReplaceHeavisideGradient")
def heaviside_fake_grad(op, grad):
    x = op.inputs[0]

    return 1 * grad
    # ici 1 est la dérivée de la fonction identité :
    # nous implémentons la HSID


def spartan_network(sess=tf.get_default_session(), debug=False, w=4,
train=False, create_surrogate=False):

#Retourne un modèle de Réseau Spartiate 1,2-HSID

    inpt = Input(batch_shape=(None, 28, 28, 1))
    g = tf.get_default_graph()
    with g.gradient_override_map({"Sign": "ReplaceHeavisideGradient"}):
```

```
    x2 = Conv2D(4, (1,1), activation=Heaviside,
     bias_initializer=keras.initializers.random_uniform(
     minval=0.05, maxval=0.5),
     kernel_initializer=keras.initializers.random_uniform(
     minval=1, maxval=1.5))(inpt)
    x2 = Conv2D(32, (3, 3), activation='relu')(x2)
    x2 = Conv2D(32, (3, 3), activation=Heaviside)(x2)
    x2 = MaxPooling2D(pool_size=(2, 2))(x2)
    x2 = Conv2D(32, (3, 3), activation='relu')(x2)
    x2 = Conv2D(32, (3, 3), activation='relu')(x2)
    x2 = MaxPooling2D(pool_size=(2, 2))(x2)
    x2 = Flatten()(x2)
    x2 = Dense(100, activation="relu")(x2)
    x2 = Dense(10)(x2)
predictionsSp = Activation(activation="softmax")(x2)

model = keras.Model(inputs=inpt, outputs=predictionsSp)
model.summary()


hist=LossHistory()
X_train, Y_train, X_test, Y_test = data_mnist(train_start=0,
                                              train_end=60000,
                                              test_start=0,
                                              test_end=10000)
K.set_session(sess)
model.compile(optimizer='adam', loss='categorical_crossentropy',
  metrics=['accuracy'])
model.fit(np.array(X_train), np.array(Y_train), 128, 8,
  verbose=1, callbacks=[hist])
return model
```

ANNEXE B ARTICLE 1 : SPARTAN NETWORKS : SELF-FEATURE-SQUEEZING NEURAL NETWORKS FOR INCREASED ROBUSTNESS IN ADVERSARIAL SETTINGS

Voir pages suivantes. Au cas où l'auteur serait parti de polytechnique, l'addresse de correspondance de celui-ci est `menet.francois@gmail.com`

Spartan Networks: Self-Feature-Squeezing Neural Networks for increased robustness in adversarial settings

François Menet*
Polytechnique Montréal
Montréal, QC
francois.menet@polymtl.ca

Paul Berthier
Polytechnique Montréal
Montréal, QC
paul-2.berthier@polymtl.ca

Michel Gagnon
Polytechnique Montréal
Montréal, QC
michel.gagnon@polymtl.ca

José M. Fernandez
Polytechnique Montréal
Montréal, QC
jose.fernandez@polymtl.ca

ABSTRACT

Deep learning models are vulnerable to adversarial examples which are input samples modified in order to maximize the error on the system. We introduce Spartan Networks, resistant deep neural networks that do not require input preprocessing nor adversarial training. These networks have an adversarial layer designed to discard some information of the network, thus forcing the system to focus on relevant input. This is done using a new activation function to discard data. The added layer trains the neural network to filter-out usually-irrelevant parts of its input. Our performance evaluation shows that Spartan Networks have a slightly lower precision but report a higher robustness under attack when compared to unprotected models. Results of this study in Adversarial AI are based on tests conducted on the MNIST dataset.

KEYWORDS

Artificial Intelligence, Computer Security, Adversarial AI, Deep Learning, Spartan Networks, Non-Differentiable, Robust Optimization

1 ON THE MOTIVATIONS FOR ROBUST DEEP LEARNING

Neural networks and deep learning. Neural networks are machine learning algorithms that are mainly used for supervised learning. They rely on stacked layers of neurons. These stacked layers take a fixed-length input tensor and generate another output fixed-length tensor. The input/output function is differentiable relatively to its weights. By using gradient-based optimisation on large datasets to update the weights —a process also refered to as "training"— the various layers generate output tensors whose values are a pattern-matching-value of their input, allowing these algorithms to detect features. Deep Learning consists of stacking a large amount of layers, allowing a neural network to extract features of features and thus grasp more abstract and complex characteristics from the input. Deep Neural Networks (DNN) have drawn a lot of attention recently.

In the last decade, DNN have revolutionized the automation of perceptive tasks in various domains, especially in computer vision. Deep Learning is increasingly used in autonomous driving software [6], malware analysis [44] (with attacks already implemented on detection systems [20, 25]), and fake news detection [1].

These technological improvements are already being used in safety-critical environments such as vehicles or factories, where automated driving for the former and predictive maintenance for the latter may save millions of dollars and thousands of lives.

New Attack Vector. In late 2013, Szegedy et al.[41] discovered a new kind of vulnerabilities in DNN: given a neural network, the authors propose an algorithm to generate samples that are missclassified while retaining their *meaning* to the human cognitive system. This groundbreaking discovery created an entirely new threat model against machine learning powered applications. Several attacks have then been discovered. As an example, The Fast Gradient Sign Method (FGSM) [18] gives a reliable attack method against DNN. This method exploits the neural network instability to small adversarial input variations. This means that two samples that slightly differ from each other can be classified differently, even if they are indistinguishable for a human observer. The FGSM iteratively adds or substracts a small value ϵ to each element of the input tensor. This simple method yields surprisingly powerful attacks: on FGSM-generated adversarial samples[18], precision of these algorithms can drop from 99% to less than 20%.

While classifier-evasion techniques are widely studied, DNN create an original problem as its lack of explainability and high sensitivity make them vulnerable to undistinguishable evasive samples. The performance of DNN also works against them: users tend to trust these systems because of their comparable performance to humans on some narrow tasks. This new attack vector does not exploit vulnerable code logic,

*Corresponding author

For submission, Dec 2018, C and S
2018.

but abuses users' trust in the classifier's complex decision boundaries.

Potentially any deep learning model can be vulnerable to this kind of attack, which is hard to detect, prevent, and whose impact will only grow in the upcoming years.

1.1 Adversarial Examples and the Clever Hans Effect

The vulnerability of these models to adversarial inputs brings forth another issue. If deep learning algorithms are seeing patterns that a human being can not see *under normal conditions*[14], this means that the models suffer from the so-called *Clever Hans effect*. This term, popularized by Papernot et al. [35], comes from a horse that was deemed capable of complex arithmetical operations, where in fact the animal was guessing the actual answer from the unconscious behaviour of the audience.

The Clever Hans effect is the product of three observations :

(1) Samples that are not generated by adversaries (i.e. *sane samples*) are almost always well classified by deep learning algorithms.
(2) Samples that are generated by adversaries (i.e. *adversarial samples*) are likely to be misclassified by deep learning algorithms.
(3) Human observers can still classify samples generated by adversaries *given enough time* [14].

We can thus deduce that:

- deep learning algorithms focus on features that are not essential to the true class of a sample.
- features captured by deep learning algorithms during training are not the same as those learned by human beings over the course of their lives.

Although the work of [14] shows common vulnerabilites between learning algorithms and a biological learning process, we can still state that the current learning behaviour of these algorithms do not capture the actual *meaning* carried by the sample.

1.2 Overview of our Work

In our work, we consider that the current training dynamics of DNN creates their sensitivity to adversarial input. We will hypothesize on the characteristics of the algorithm causing devious behaviour, and we will create a minimal deep learning algorithm constrained to function without them.

We will study adversarial image classification as this community has created a large number of attacks and defenses. This plethora of research is due to the simplicity of the image space topology. For example, if one changes a pixel, or slighly varies the colors of various pixels in a panda image, the image still resembles a panda. On the other hand, trying to preserve the semantics of a ASM-x86 code while randomly changing a few lines of code would yield very different results.

Thus, we define Spartan Networks, and apply the general framework to Convolutional Neural Networks (CNN), as they are the state-of-the-art for image classification.

Our hypotheses on the current CNN are as follows:

(1) The behaviour of the function F_{nn}, result of the CNN's training, is locally linear, thus allowing an attacker to easily explore the system's state space.
(2) Around sane datapoints, the function F_{nn} is sensitive to features that do not make sense to a human observer.

As we consider perturbations to be unnecessary information taken into account by the network, we try to create a network that learns to ignore parts of the information it is given.

In this paper, we propose a new type of DNN, the Spartan Networks, based on two conflicting elements forced to collaborate during the training phase:

- Firstly, the *filtering layers* severely reduce the amount of information they give to the next layer. They are constrained to output the lowest amount of information through a *filtering loss*.
- Secondly, the other layers connected to the previous ones constitute a standard CNN trying to rely on the information given to minimize its training loss.

These two parts are competing against each other: if the filtering layer destroys all the information, the filtering loss is low, but the network cannot train, and thus the training loss is high. On the opposite, a CNN training given unlimited information allows the network to train efficiently, reaching a low training loss, but increasing the filtering loss.

This construction thus constitutes a *self-adversarial* neural network. The weighted sum of those two losses forces the filtering layer to find the vital pieces of information the rest of the network needs to successfully train. The network thus learns to focus on less information, and selects more relevant features in order to maximise its performance.

This paper is organized as follows. A taxonomy of attacks specific to machine learning-powered applications is given in Section 2. We describe the various test-time adversarial attacks among the aforementioned attacks in section 3. We present the various defense attempts in the litterature in Section 4. We explain the motivations for Spartan Networks in Section 5. We define candidate implementations of our proof-of-concept (PoC) in Section 6. We evaluate and discuss the results in Section 7. In Section 8, we will balance out the performance drop with the robustness gain to evaluate the relevance of Spartan Networks. We will conclude and present future work in Section 9.

2 ATTACKING A NEURAL NETWORK

2.1 Threat Model

There are two main attack vectors available to an adversary to hinder a DNN's performance: Train-Time (or Poisoning), and Test-Time Adversarial Attacks.

Train-Time Attack. This attack aims at modifying a dataset, by either adding patterns into existing samples, or adding new samples. The attacker's intent is to manipulate a deep neural network's training on this dataset in order to:

- create a *backdoor*, which is a range of samples that are missclassified by the target network when it has been trained on the poisoned dataset. As an example, this could allow an attacker to evade malware detection for a specific type of malware, meaning the system would catch other malware with high accuracy, increasing user's trust in the system, but let the attacker's malware through.
- diminish the overall accuracy of the neural network when trained on the poisoned dataset. As an example, an attacker could feed poisonned data that would cause an algorithm to extract the wrong features, causing additional data curation cost.

Plausible attack scenarios within this threat model are given by Gu et al. [22].

Test-Time Attack. In this type of attack, the neural network is already trained, its parameters are frozen and the attacker can only move within the space of all possible inputs. The attacker's objective is to find a sample x_{adv} that is:

- *close* to a sample x correctly classified in class y
- classified in a different class $y_{adv} \neq y$

The main hypothesis here is that when two samples are *close*, their meaning stays the same to a human observer.

If the attacker succeeds most of the time for adversarial samples reasonably *close*, it can reliably output samples indistinguishable from sane samples, that a classifier would fail to categorize well. For example, an attacker trying to bypass an automated content filter could take shocking pictures, slightly modify them, and succesfully bypass the filter. As the samples are close to their original counterpart, their meaning would be preserved.

More formally, we replace the *closeness* by a distance between input and adversarial input, and get a constrained optimization problem:

Given a classifier f, a distance $\mathcal{D}$, a perturbation amplitude budget ϵ, an attacker tries to find the minimal λ on a sample x, with a ground truth label y such that :

$$\begin{aligned} &f(x) = y \\ &f(x+\lambda) = y_{adv}, y_{adv} \neq y \\ &\mathcal{D}(x, x+\lambda) < \epsilon \end{aligned} \tag{1}$$

Note that without the ϵ budget constraint, we could generate misclassifications by using a correctly classified input from another class.

In most cases, this distance is replaced by a standard $\mathcal{L}_n, n \in \{0, 1, 2\}$ or $\mathcal{L}_\infty$ norm for the perturbation λ. We depict below the various standard norms, with $\lambda = (\lambda_i)_{i \in [\![1,k]\!]}$

Norm name	Mathematical expression
$\mathcal{L}_0$	$\mathcal{L}_0(\lambda) = \#\{i \vert \lambda_i \neq 0\}$
$\mathcal{L}_1$	$\mathcal{L}_1(\lambda) = \sum_i \vert\lambda_i\vert$
$\mathcal{L}_2$	$\mathcal{L}_2(\lambda) = \sqrt{\sum_i \lambda_i^2}$
$\mathcal{L}_n$	$\mathcal{L}_n(\lambda) = \sqrt[n]{\sum_i \lambda_i^n}$
$\mathcal{L}_\infty$	$\mathcal{L}_\infty(\lambda) = \max_i(\vert\lambda_i\vert)$

In the rest of this paper we will focus on the Test-Time Attack, as this is where the attack surface is the largest.

3 THREAT MODELS FOR TEST-TIME ATTACKS

There are various ways to attack a deep learning algorithm at test time. We contextualise the attacks by defining the different attack scenarios.

In order to understand the implications of adversarial examples, we will give a security equivalent of our attack scenario through a simple example. This will link the new attack vector to otherwise known attack vectors in the domain of information security.

3.1 Attack scenario

In order to place ourselves in an information security context, we may consider a simple setup:

- The defender runs a check digit identification software powered by DNN on the target computer. It is the only program available on this system.
- The attacker can send samples of checks. They have access to a very small digit database. Their original check is correctly classified with the right amount, but they aim at maliciously changing the input image in order to get a different check value.
- Opportunity: Through two accounts, the attacker can send checks to himself. They can thus check if the system is vulnerable.

3.2 Attacker's access to knowledge

As with any attack, attacks on Deep Learning vary on the knowledge the attacker has on the system prior to the attack.

White-Box Attacks. In White-Box mode, the attacker has access to every parameter of the Neural Network. In this scenario, the attacker has a *read-only* access to the all parameters of the algorithm by the means of an ill-protected file.

Gray-Box Attacks. In Gray-Box mode, the attacker has access to some parameters of the Neural Network, while some remain unkown to them. A common scenario is that the attacker knows the structure of the Deep Neural Network, but not the parameters. In this case, the attacker has managed to get a restricted account on the server, showing only the code used to train, but not the admin-owned weights parameter.

Black-Box Attacks. In Black-Box mode, the attacker has no access to the Neural Network other than through its inferences, like any normal user. In our scenario, this means

that the attacker has no access to the server other than the check submission point.

3.3 Attacker's Objective

Regardless of the operational objectives, we give the two main *technical* objectives attackers can set when they attack deep learning algorithms.

Untargeted Attack. When the attack is untargeted, the attacker's objective is to create *any* misclassification they can, with no control over the y_{adv} in equation 1. In our scenario, the attacker tries to trigger a misclassification from the real digit class.

Targeted Attack. When the attack is targeted, the attacker's objective is to trick the algorithm into classifying the input in a class *chosen by the attacker*, with complete control over the y_{adv}. In our scenario, and for simplicity, the attacker tries to classify all digits as a 9 digit. We will consider that it can only attain this goal through a *targeted attack*.

We summarize the attack characteristics with respect to our scenario below :

Attack	Attacker in the Scenario
White-Box	Already has access to privileged information
Gray-Box	Already has access to restricted information
Black-Box	Has no access to any information
Targeted	Wants to get a 9999 $ check
Untargeted	Wants to modify the check's value

We create an array from the last two sections, and get the following example attackers. U is for Untargeted, T for Targeted :

	White-Box	Gray-Box	Black-Box
U	Disruption	Rogue Employee	DoS extortion
T	Smash-and-Grab	Fraud	Theft

We describe the scenarios:

Disruption. When the attacker has access to every information in the neural network, we consider that they already have access to a privileged account. For the sake of this example, we consider the case of an attacker with root access that wants to minimize its footprint. By allowing themselves to only read the parameters, the attacker can create a copy of the neural network then use it to disrupt business by forcing bank employees to manually review all checks emitted by the attacker.

Rogue Employee. In this context, an IT employee with a restricted account and thus some information on the neural network could trigger a misclassification on the system, and disrupt business by creating subtle modifications on all the checks submitted through its interface.

Denial of Service (DoS) extortion. When the attacker has no access to the system, any misclassification from the original class will allow an attacker to get some form of disruption. If an attacker prints check paper with adversarial patterns on it, they can extort money from the bank by threatening to cause user mistrust and employee time to manually review checks.

Smash-and-Grab. When the attacker is given White-Box access through a visible root access, they can create a base of malicious digit samples allowing them to transform any 4-digit check into a 9999$ check. From this, the attackers could submit some malicious checks and take their money whenever possible.

Fraud. In this case, the attacker could be another rogue employee trying to do more than disruption by targetting all the digits to be 9.

Theft. This is the worst scenario. If successful, the attacker can create a target misclassification with only access to the computer, completely controlling the value of the check.

These attacks could be seen as less risky variants of physically tampering the check's value. If caught, the attacker can blame the system, by arguing that a human reviewer can perfectly see the check's actual value. As these attacks are almost riskless and costless for attacker but disruptive for the defender, these AI attacks constitute an interesting field of study from an information security perspective.

These attacks are not limited to the banking system. As an example, deep learning algorithms are currently used in autonomous driving [6], malware analysis (with already an arms race between attacks and defenses [20, 25, 44]), healthcare[15], and fake news detection[1].

3.4 Main attacks implemented against Deep Learning

For each attack, we will give a formal mathematical definition. We will then write an equivalent attack to understand the inner workings of the algorithm.

First attack discovered. A lot of attacks aiming at generating adversarial samples have been developped in the last five years, since the discovery by Szegedy et al. [41] of a L-BFGS-driven line-search attack. It is the first method known to generate adversarial image samples that are extremely close to their innocuous counterparts.

This attack tries to minimize the distance between the original sample and a modified, misclassified sample, until it is close enough to be indistinguishable from the original. It is a way for an attacker to optimize, for different values of a perturbation norm constraint, a loss-maximization problem until it finds an optimally undistinguishable sample.

3.4.1 FGSM. The Fast Gradient Sign Method or *FGSM* tries to modify a sample by adding or substracting a small value epsilon for each dimension of the input. More formally, the attacker generates a perturbation vector λ such that :

$$\lambda = \epsilon \ \mathrm{sgn}(\nabla_x \mathbf{L}(f(x), y))$$

In untargeted mode, the attacker adds a small noise over the input, adding $\pm\epsilon$ to every dimension of the input in order to increase the error, choosing the sign accordingly.

In targeted mode, the attacker generates the highest output for the desired class, and thus the lowest error relatively to the adversary's target. The vector thus becomes :

$$\lambda = -\epsilon \ \mathrm{sgn}(\nabla_x \mathbf{L}(f(x), y_{adv}))$$

Where $\mathbf{L}$ is the network's loss function.

In both cases, a clipping is made to make sure the values stay in their original range. The sum, along hundreds of dimensions, of small input/output error gradients can result in a large perturbation. Thus, an attacker can easily *push* the sample through a decision boundary and trick the system into a misclassification.

This attack is easy to compute and, for some unprotected classifiers, creates large errors. Computing the gradient values requires a White-Box access to the network.

The FGSM on the field. As the FGSM is a *White-Box* attack, the attacker needs access to privileged information. Using the information, the attacker takes a picture of *any check*. The original sample is likely to be classified as the original check. The attacker then computes, for each color channel of each pixel, whether they add or substract, ϵ to each value, by looking at the sign of the loss' gradient with respect to every dimension of its input.

The ϵ value is chosen according to a tradeoff: the lower the value, the closer the picture will be to the original one, thus making it harder for a system administrator to see whether there was an attack or not. On the other hand, the larger ϵ is, the higher the chances are for the attack to work.

3.4.2 Carlini & Wagner attack. Carlini & Wagner [9] have proposed an attack method based on a custom gradient descent with different candidate losses in order to *learn the perturbation* the same way the network learns.

One cannot use the optimization problem in (1) to find adversarial samples directly. Rather, loss functions point toward the main direction of attack while perturbation norm stays a constraint that can be hard (strict constraint) or loose [9]. In the latter, going further from the constraint adds to the loss.

The loss used by the authors in [9] is a composite based on the original network's loss. They try different losses, linked to different attack behaviours.

These loss functions force the adversary to learn a perturbation that maximizes the confidence of the system on its erroneous decision. When using a loose constraint, they smooth out the clipping process. As the constraints become differentiable, the authors allow gradient-based methods to converge into a local optimum for their problem.

Carlini & Wagner attacks on the field. An attacker needs a **white-box** access to the classifier. They would use a gradient descent method, incorporating one of the losses proposed by the authors. This sample generation algorithm obeys two constraints: the first crushes the perturbation to be zero, but the other pulls it towards a misclassification. Through a weighted sum of the constraints, and given enough computing power, the same kind of optimization that trained the neural network is used to create an adversarial sample.

3.4.3 DeepFool. Moosavi-Dezfooli et al. [33] used an entirely different approach to make their attacks : they considered that any classifier has locally-linear boundaries, allowing the local use of hyperplanes. The decision hyperplane is such that if one moves parallel to it, they will never cross the boundary and thus will never find an adversarial example. By taking the orthgonal direction, one can make sure that they have the *fastest* way to burst out of the decision boundary. In order to compensate for the linearity hypothesis, the author use an iterative method to keep these approximations within a close radius.

DeepFool on the field. A *White-Box* attacker tries to output a wide number of photos triggering the *same* classification confidence for *every* different class. This would yield a local topology of the multi-dimensional geometry of the decision boundary. They could thus identify the closest hyperplane-boundary to their current point. This equivalent naive method would lead to a gigantic amount of computation. The authors use an algorithm based on a per-boundary geometry to quickly find the nearest way out of the current class, thus triggering a misclassification.

We will give a practical explanation of DeepFool. The attacker can compute at every step the set of linear rules that makes them be in the class they want to escape, and make a small step away from the closest combination of those rules. They do so iteratively, to account for the global non-linearity of DNN.

3.4.4 Surrogate Black-Box Models. So far, every attack required a *White-Box* access to the system. But in 2016, Papernot et al. [37] used the fact that Adversarial Examples can transfer from one Deep Learning Model to another[41] to create *Black-Box* attacks that require *no* access to the network's parameters. They train an approximation of the target model in order to create a *White-Box surrogate*. They can then use the transferability property of the adversarial examples: adversarial samples that fool the approximated model have a high probability of fooling the target model. Some White-Box attacks transfer better than others: for example, FGSM attacks transfer well to different classifiers.

Surrogate Black-Box Models on the field. If an attacker has no access to the model other than the *check reading device*, they can first gather a small dataset of handwritten digits, and train a model on them. Through jacobian augmentation[37], the attacker will then distort the digits it has in the direction of the estimated boundaries, and re-submit to the oracle for

evaluation: through this process, the surrogate neural network will get a low-performance approximation of the model. This approximate model will be vulnerable to some adversarial examples. The transferability property make them likely to also fool the target model.

4 ADVERSARIAL AI DEFENSES

For a systematic review of defenses, we refer the reader to [4] for their work in the subset of attacks on images.

4.1 Defenses strategies and our focus

A defender protecting a DNN can use three different strategies in order to increase the model's robustness to adversarial examples.

Detection Strategy. Defenders can use a **Detection** algorithm, above or within the network, in order to detect the adversial nature of a submitted sample [16, 19, 24, 27, 28, 32, 43]. If the sample is detected as adversarial, it is rejected and given to a human reviewer to get its actual meaning.

Reforming Strategy. Defenders can use a **Reforming** system. In this paradigm, the sample is transformed in a way that modifies the sample's numeric values while preserving its semantics. JPEG Compression, Bit-Depth Reduction and Error Diffusion, are valid examples of such transformations. The Reforming system can also be learned through models that re-create their inputs [12, 23, 29, 31, 39]. The defender thus expects that, by adding error or removing information, they can remove some or all the adversarial perturbations. This is equivalent to a form of noise reduction.

Regularization Strategy. The last defense mechanism one can use is based on the **Regularization** strategy. In this paradigm, the behaviour of the original system is regularized during a training in order to modify the way it handles samples that are out of the classical distribution. This is equivalent to a form of patching: by changing the program's behaviour, the defender makes an attack ineffective against it.

We present below a parallel with SQL Injections, a well-known problem in information security.

AI Defenses	SQLI Defenses
Detection	Detect special characters
Reforming	Replace special characters
Regularization	Parametric statements

We will focus on the **Regularization** strategy for the following reasons:

- A **Detection** system creates an opportunity for a Denial-of-Service (DoS): the attacker could slightly transform the input of a user through a Man-In-The-Middle attack and would get a consistent amount of rejection, while keeping the feedback sample undistinguishable from the original one.
- A **Learnable Reforming** system allows an attacker to exploit another neural network. If succesfully attacked, the reformer would regenerate a sample that is close to the target class chosen by the attacker.
- A **Non-Learnable Reforming** system creates a static defender that an adversary can progressively learn to bypass.
- **Regularization** strategies have the advantage of presenting a smaller attack surface, as there would be no other algorithm than the one used for inference. They trade this advantage for the requirement of retraining the whole system, opening themselves to poisoning attacks and creating a high upfront cost for the defender.

As we consider that DNN need to be trained at least once, the **Regularization** strategy will be preferred. We highlight however that other strategies are less ressource-intensive as they can often be implemented without retraining the model.

4.2 Current Regularization Techniques

Adversarial Training. One of the first ideas introduced in the defender's arsenal, the Adversarial Training[18, 41] is based on generating adversarial samples through white-box attacks in order to add them to the existing training set. By triggering the error, and backpropagating it through the network, the system learns to resist an attack the same way it learns the task at hand.

From an information security perspective, this method has the disadvantage of defending the network against *known* attacks only. There is no guarantee that adversarial training protects against attacks other than the attacks the system has been trained on. Moreover, this defense can hinder the network's performance. Madry et al. [30] address this concern by introducing the idea of an optimal first-order adversary that could subsum every attacker with the same order constraints.

With these theoretical guarantees, this approach thus transforms offensive strategies into defensive strategies. This approach searches for optimal attacks to train the network upon. If successfully trained on strong adversarial examples, the system can have security guarantees.

Gradient Masking/Gradient Shattering. As all current techniques employ some form of gradient computation, one of the ideas introduced was to reduce the gradient or otherwise hinder the attacker's access to the gradient. Gu and Rigazio [21] directly penalize the network to forcefully lower its gradients through a regularization, while Papernot et al. [38] use distillation. Distillation is a process where a first network is trained on one-hot labels $(\delta_{i=class})_{i\in[1,k]}$ where k is the number of classes. Then, the first network is discarded, and the predictions of the trained model are used as the new labels. This teacher network smoothes-out the labels. A second network is then trained on these smooth labels, thus considerably lowering the gradient values. The authors claim that the reduction factor is above 1000.

Black-Box surrogate attacks [37] have bypassed this *gradient masking* defense: these attacks do *not* need any gradient. Thus, the only way defensive distillation can impact these gardientless attacks is by regularizing the network's behaviour. Unfortunately, even if a modified and extended defensive distillation has been able to resist some attacks [36], some attacks [9] can still generate adversarial samples bypassing this defense.

Nayebi and Ganguli [34] have used a regularization function rewarding using the saturation values of activations. Network activations that are far from a zero-gradient point are penalized through a regularization.

Various similar methods create a non-differentiable variant of their model *at test time* to deny any access to the gradient: Athalye et al. [5] have systematically reviewed — and bypassed — these so-called "gradient shattering" techniques.

Ross and Doshi-Velez [40] add another regularization term to the loss. Rather than only regularizing the parameters' values during the training, they also regularize the value of the input gradient through *double backpropagation*[13]. This gradient reflects how much each variation of the input can change the class. By diminishing this value, the defender forces any attacker to create larger perturbations. The authors claim that successful adversarial samples against their system have had their meanings modified, and thus cannot be deemed adversarial.

4.3 Other defensive techniques

4.3.1 Feature Squeezing. Xu et al. [43] have shown that, by reducing the attacker's sample space by using semantic-preserving compression / filtering (eg. going from greyscale to Black and White images), one can detect adversarial samples. We do so using the fact that legitimate inputs often create agreeing predictions when inferred on through different compressions. For adversarial samples, these predictions are different.

4.3.2 Thermometer Encoding. Discovered by Buckman et al. [8], this method discretizes the values of the dataset according to arbitrary thresholds. As an example, if we have 4 tresholds at $(80, 60, 40, 20)$ and values between 0 and 100, 42 will be encoded as $(0, 0, 1, 1)$, and 77 as $(0, 1, 1, 1)$

5 MOTIVATIONS FOR SPARTAN NETWORKS

5.1 Discretization

When given labelled data, a classifier is usually given high-dimension inputs. Input dimensionalities can vary from several hundred to a milion. However, label space dimensionality ranges from 2 to a hundred. This means that the input space is orders of magnitude larger than the output space.

For example, the CAL10K music dataset[42] has a compressed size of 2.1GB, but the size of the annotations file is 10MB. For the images dataset, the standard MNIST handwritten digits dataset[26] has 10 different possible annotations and thus can be stored on four bits. The digital image needs 6272 bits to be stored.

DNN can succesfully create a reliable approximator that can classify correctly and reliably given an appropriate amount of labelled data in standard settings. Adversarial examples show that there are pockets of inputs present in the high-dimensional space that are missclassified. Adversarial training provides a way to change the function approximated by neural networks, but it cannot guarantee that it reliably "patched" the system. Even if augmented, the number of training datapoints is indeed negligible in comparison to the number of possible inputs for a given class. High-dimensionality creates a disadvantage for the defender.

Given an adequate ϵ (1) the perturbation is not perceptible to a human observer, but successfully tricks the network into giving a wrong answer. This means that:

- the input space of encoded samples has a local density greater than human distinguishability power.
- the network behaviour is extremely sensitive to small changes.

In neural layers using a ReLU or ReLU-equivalent activation function, stimulus perception is almost always near linear or near multilinear. Thus, any small variation of the stimulus can be propagated through the neural network using the fact that all operations are differentiable. The ReLU is made to be equal to the identity function when recieving a positive input, meaning that the ReLU is a linear function when activated.

As an example, human observers cannot distinguish colors between $\{61, 175, 250\}_{\text{RGB}}$ and $\{61, 170, 250\}_{\text{RGB}}$, but can distinguish cyan from blue. The DNN however can distinguish the former extremely well.

The attacker, by modifying the value of parts of the input by a small amount, can also make the value of activation function vary by a proportional amount. This perturbation will be propagated to all affine combinations including the perturbed inputs. As the propagation goes forward, the perturbation, if cleverly crafted by the attacker, spreads and its amplitude grows within the network, using the weights of the network to trick and force the network to output another value. This exploitation can be done for any network having a succession of locally-linear activation functions, through the Taylor expansion of any differentiable function.

Various authors have stated [3, 8] that excessive linearity seems to be a point of vulnerability of neural networks.

From Xu et al. [43], we learned that squeezing features could sometimes prevent adversarial perturbations from being effective. Buckman et al. [8] extend this idea with the

Thermometer Encoding. We extend on the Thermometer Encoding strategy by learning the tresholds that are useful to correctly do the task while using the minimal number of information.

Hence our hypotheses are as follows:

- Current neural networks exhibit locally-linear properties that can be used to slide the sample classification towards another class.
- The dimensionality is one of the factors allowing attackers to find a good attack vector.

We summarize all the problems discussed above in Fig. 1. A Deep Feedforward Neural Network has a very large input space for a small output space and excessive linearity allows the attacker to explore the space and find adversarial examples.

The human response to color seems to be non-linear, as seen in Fig 2. The little perturbation is not seen, and double this perturbation is. The tresholding does not destroy any semantics, as one can recognize the digit.

When stimuli values are discretized, small changes to any stimulus must go past the next threshold to create any perturbation. The attacker will thus have to push the values further than it normally would in order to put a valid perturbation, that would be visible to an observer. Two samples on the same threshold are indeed considered as the same to the system. The work of Buckman et al. [8] uses this idea.

The non-linearity of discretization as well as its saturation properties could restrain an attacker from propagating its perturbations throughout any DNN.

Previous attemps failed at training with staircase-like functions, that are inherently non-differentiable, as the 'derivative' of these functions is a sequence of impulses.

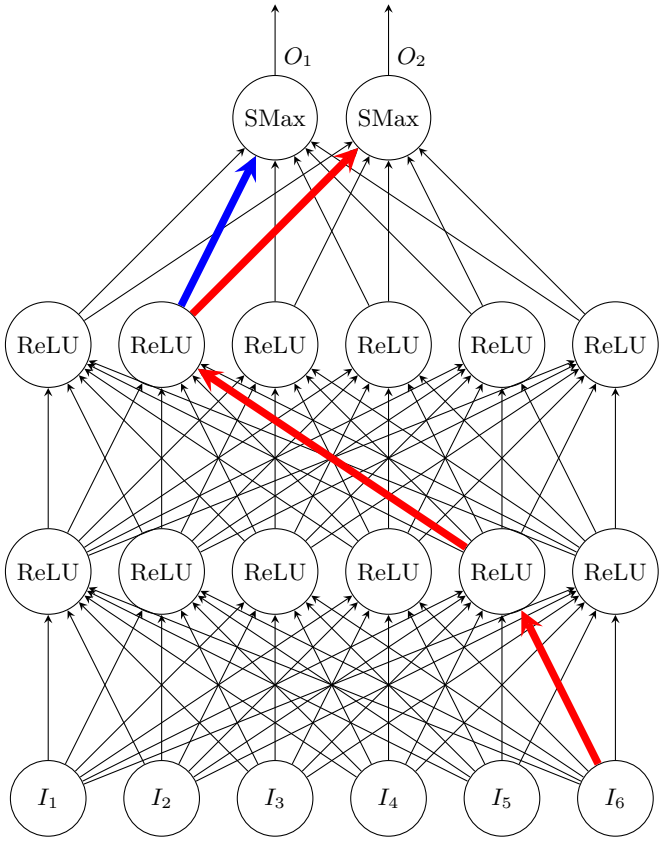


Figure 1: A Deep Feedforward Neural Network. If the ReLU or any locally-linear activation function is used, an attacker can find, out of all the possible weights, the most important weights bending prediction towards misclassification. Red means an increase, blue a decrease. By increasing the value of a part of the input, the attacker can linearly manipulate part of the calculation and decrease the current probability of the current class and increase the probabilty of another

5.2 Binary Encoding

As seen previously in the works of Buckman et al. [8], one can use a vector-of-bits encoding to make a network more robust against adversarial perturbations. Combined with the work of Nayebi and Ganguli [34], we hypothesize that robust learning in DNN can be achieved by:

- making their behaviour as non-linear and saturated as possible;
- reducing the attack space for the attacker by squeezing it;
- creating a binary array instead of float activation values.

We aim at learning an optimal way to achieve this through backpropagation, by creating new layers of neurons. These neurons will learn this behaviour by themselves, however we make sure that the following requirements are met:

- The performance on the test dataset must stay relatively close to an unprotected counterpart.
- We must find a way to learn a highly non-differentiable filtering that can interface with backpropagation.

We thus create a processing layer, that we called filtering layer, whose behaviour is made to be extremely non-linear. This layer has a discrete, low-cardinality value range, and is trainable using backpropagation using a technique we will discuss section 6.3. We modify the DNN of figure 1 into the DNN of figure 3 with these ideas. Note that Courbariaux et al. [11] have already proposed restricted-cardinality activation functions.

The processing would use the Heaviside step function, where the activation function's definition range is orders of

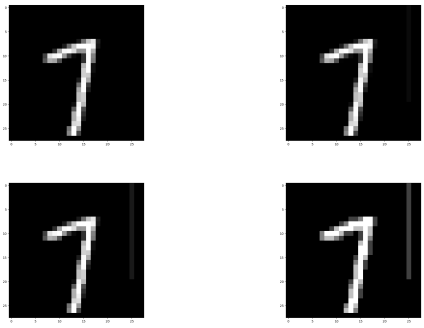


Figure 2: In reading order: a non-perturbed image, an image with a little, 5% white, vertical bar next to the number, the 10% version, and a thresholded version, with 3 equidistant thresholds, effectively putting the perturbation at 25%. The perturbation becomes much easier to see, while the image keeps its meaning

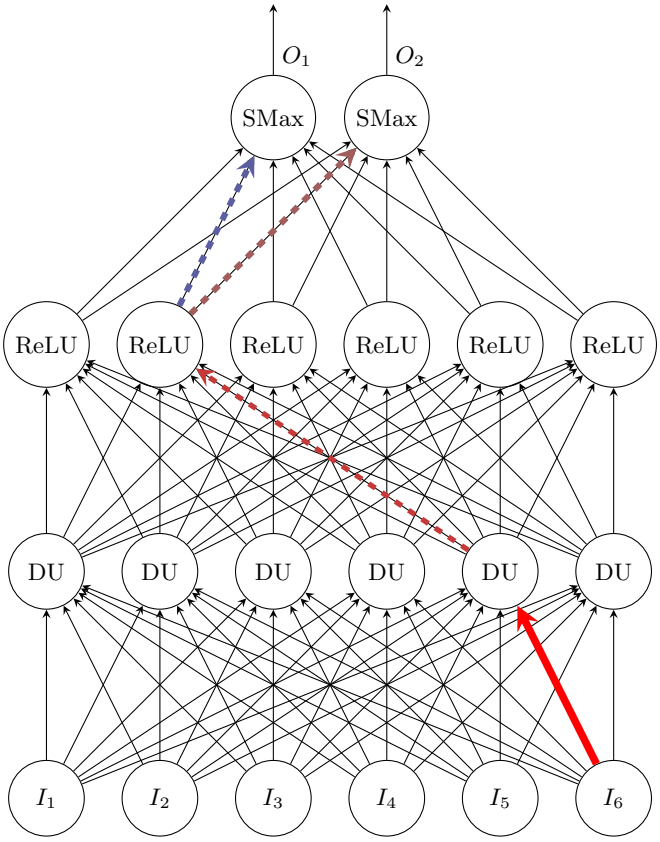


Figure 3: By breaking the linearity and outputting only discrete values, we make sure that an attacker cannot use gradient descent to find an adversarial example. DU stands for Discrete Unit.

magnitude larger than the output range. In this case, 9 orders of magnitude for the Heaviside, instead of less than 1 for the ReLU. As we get less information-bits for each neuron, we destroy more information. This process will be called *data-starving*.

To better understand the interest of data-starving, we emphasize that most activation functions are encoded on from 16 to 64 bits, most commonly 32. 32-bit-float activation functions like the 32-ReLU have 2^{31} possible values, as only positive values are different from one another. While these activation functions show a high number of possible states, training set cardinalities are orders of magitude lower than the possible number of states. Thus, unexplored areas in those activation functions can be exploited by an adversary to find adversarial examples. The defender can thus try to reduce the amount of states in each layer by various methods. These methods include using lower weights, or adding a saturation value. This idea was also exposed in the work of Xu et al. [43].

We decide to reduce the attacker's space as much as possible, by taking an activation function that outputs only *two* possible values. We will thus use the Heaviside step function as an activation function in the filtering Layer.

5.3 Spartan Training

During the training, the filtering layers data-starve the network because there are few possible output values out of them. Moreover, these layers are regularized by the amount of signal they let through. To do this, we use a $\mathcal{L}_1$ loss on the activations of the network. Thus, this added loss is proportional to the amount of signal that the system let through. In addition to this loss, the network uses a training loss based on the error, which is standard for training DNN. The training loss plays directly against the loss of the filtering layer.

To lower the **training loss**, the system needs to get information that increases the **filtering loss**. The only way to reduce the **training loss** is by destroying information that is likely to increase the **filtering loss**. We thus have two competitors within the network fighting to reduce their losses.

The self-adversarial behaviour of the network shall allow the system to harden itself during this seemingly *Spartan Training*[1], by allowing it to reduce the value ranges within the network activations. The attacker will either have to generate a high-amplitude adversarial noise, or stay in the regulated input space, more restricted, requiring more computing power, and thus yielding a higher cost for the attacker.

We define the network composed of a data-starving layer connected to a neural network and trained with a composite loss a *Spartan Network*. To understand the Spartan Training

[1] The ancient Spartan Training was known to be extremely harsh.

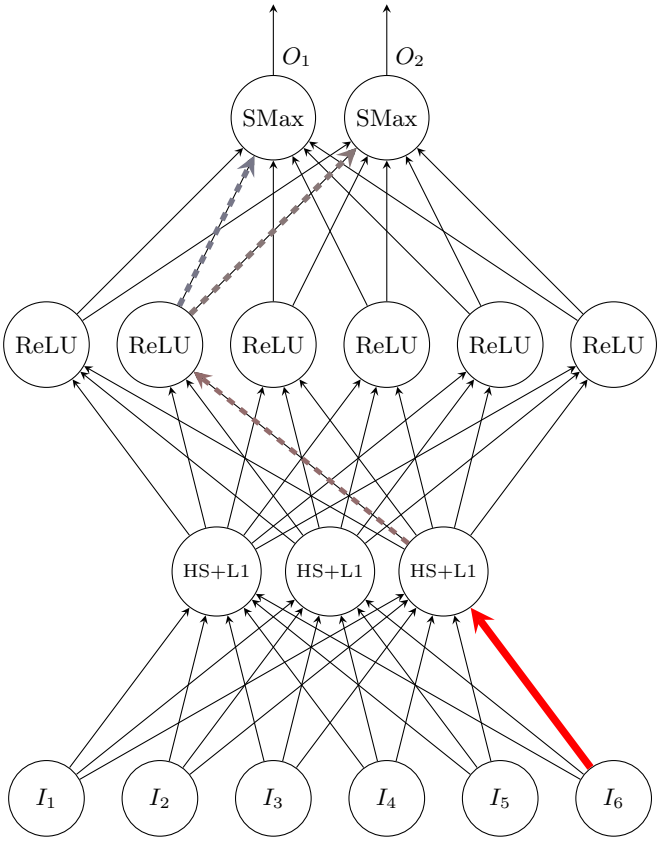


Figure 4: The Spartan version of the DNN: the first layer is severely squeezing the dimensionality of the input, and outputs binary values only. Furthermore, the activation of each neuron is added to the global loss of the neural network (L1), forcing this layer to output as few signal as possible.

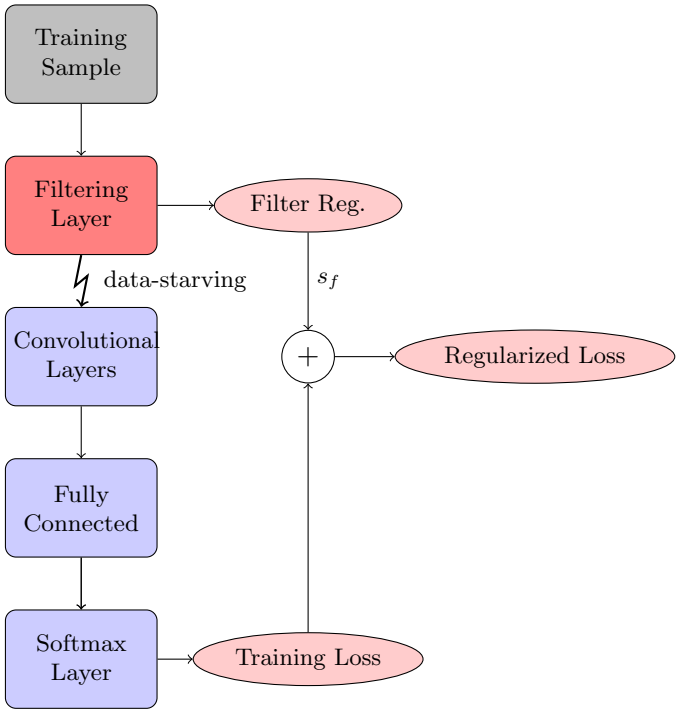


Figure 5: Example structure of a Convolutional Spartan Network for images, s_f is the *scaling factor*, the weight of the filtering regularization relatively to the training loss

and the filtering layer effect, we transform the Discrete-DNN of Fig. 3 into the Spartan DNN of figure 4.

6 SPARTAN NETWORK STRUCTURES

In this section, we create three filtering layers and explore the three different Spartan Networks using them.

6.1 Composite activation function

As stated before, we use the Heaviside step function on the forward propagation. This activation function's gradient is zero where it is defined. The filtering layer thus uses a synthetic gradient in order to use backpropagation through this zero gradient.

The Heaviside step function's definition is:

$$H(x) : \mathbb{R} \to \{0, 1\}$$
$$H(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ else} \end{cases} \tag{2}$$

On the backwards propagation, the function could be, for example, replaced with the arctangent activation function, with the following properties:

$$\text{arctan'}(x) = \frac{1}{1 + x^2} \tag{3}$$

Mixing definitions 2 and 3, we get the HSAT activation function:

$$\text{HSAT}(x) = H(x),$$
$$\text{HSAT'}(x) \longleftarrow \text{arctan'}(x) = \frac{1}{1 + x^2} \tag{4}$$

We use a mnemonic of a non-differentiable (or trivially differentiable) function concatenated with a two-letter mnemonic of the differentiable function whose derivative is used for gradient computing.

HSAT will then be the Heaviside-Arctangent activation function, HSID the Heaviside-Identity, Cos-HSAT for a Cosine-Heaviside-Identity function. To simplify, we consider these activation functions to be one activation function, that we will call *composite activation function*.

We have investigated four such functions. In the following, H is the Heaviside step function, and Φ is the normal distribution used as a function.

Forward	Backward	Mnemonic
H	id : $f(x) = x$	HSID
H	arctan	HSAT
$H \circ \cos$	$\arctan \circ \cos$	Cos-HSAT
$H \circ \cos$	$\Phi \circ \cos$	Cos-HSND

Feedforward Neural Networks are trained through backpropagation, and, while other weight updates exist, this method has demonstrated its power over the past years. If we were to use backpropagation *as-is* on this filtering layer however, we would be unable to update its weights or biases, as the derivative of the Heaviside step function is zero on all values where the function is differentiable. This is due to the fact that the Heaviside step function is the integral of the Dirac impulse function δ. Thus, no gradient can be used to update the biases without the synthetic gradient. Hence the decision to arbitrarily chose to replace the Heaviside step function's derivative by another function's derivative on the backwards pass.

We will test some replacement derivative function candidates stated above in our Experiments section.

With the idea of a Forward-Backwards composite activation function, we decouple the forward propagation from the backwards propagation dynamics on this layer, opening the path to use more complex update functions.

Note that this separation idea has also been explored in a gradient approximation attack context in the work of Athalye et al. [5], under the name *Backward Pass Differentiable Approximation* or BPDA. We differ from this work as the selection of the Backward pass is *arbitrary*, and has no need to be a close approximation of the function. We nonetheless keep the derivative sign to mirror the general behaviour of the original function.

6.2 Candidate Filtering layers

Aiming to learn different thermometer encodings [8] through backpropagation, the filtering layers focus on destroying irrelevant information. We propose and implement candidate layers exhibiting this property.

We will take the number of dimensions of the encoding as an hyperparameter β. $\beta = 4$, for example, will mean that there are four thresholds, and thus five possible values for the encoding.

6.2.1 Convolution-Filtering. The simplest way filter information with convolutions is to use the Heaviside step activation function into standard convolutional layers with a 1×1 kernel.

For every filter, each channel of an input image will be multiplied by a learned parameter and a bias will be added, before going through the Heaviside activation function

This non-differentiable activation function can allow the network to output thermometer encodings, as, with b the bias of the unit and w a weight:

$$\mathrm{HS}(wx + b) = 1 \Leftrightarrow wx + b > 0 \Leftrightarrow x > -\frac{b}{w} \tag{5}$$

By having various values for $-\frac{b}{w}$ we can create a thermometer encoding created by an activation function and learned through backpropagation, as was our goal.

6.2.2 Offset-Filtering. To prove that the robustness of the model is not convolution-dependent, we can also implement the filtering using a locally connected layer. One can vary the amount of neurons in the layer. The filtering layer is as large as the input. The neurons in the layer have only one connection, and their weights are constrained to be one. Only their biases are learned during training. We effectively create a composition between a binary filter and an image mask learned by the system.

6.3 Candidate composite activation functions

6.3.1 HSID. The Heaviside-Identity activation function definition is as follows:

$$\begin{aligned} &\mathrm{HSAT}(x) = H(x), \\ &\mathrm{HSAT'}(x) \longleftarrow \mathrm{id}'_{\mathbb{R}}(x) \\ &\mathrm{HSAT'}(x) \longleftarrow 1 \end{aligned} \tag{6}$$

This composite activation function is interesting because the backward pass is not an approximation of the foward pass.

6.3.2 HSAT. The Heaviside-Arctangent activation function definition is as follows:

$$\begin{aligned} &\mathrm{HSAT}(x) = H(x), \\ &\mathrm{HSAT'}(x) \longleftarrow \arctan'(x) \\ &\mathrm{HSAT'}(x) \longleftarrow \frac{1}{1 + x^2} \end{aligned} \tag{7}$$

6.3.3 HSAT ∘ Cosine. We compose the HSAT function with a cosine in order to get a learnable but square activation function. The Heaviside-Arctangent∘Cosine activation function definition is as follows:

$$\begin{aligned} &\mathrm{HSAT}(\cos(x)) = H(\cos(x)), \\ &\mathrm{HSAT'}(cos(x)) \longleftarrow \arctan'(\cos(x)) \\ &\mathrm{HSAT'}(\cos(x)) \longleftarrow -\frac{\sin(x)}{1 + \cos(x)^2} \end{aligned} \tag{8}$$

6.3.4 HSND ∘ Cosine. We modify the previous activation function using the normal distribution as a function for the gradient part.

$$\begin{aligned} &\mathrm{HSND}(\cos(x)) = H(\cos(x)), \\ &\mathrm{HSND'}(cos(x)) \longleftarrow \Phi'(\cos(x)) \\ &\mathrm{HSND'}(\cos(x)) \longleftarrow -x \sin(x) \Phi(x) \end{aligned} \tag{9}$$

We used $\mu = 0, \sigma = 1$ as the base values for the normal distribution.

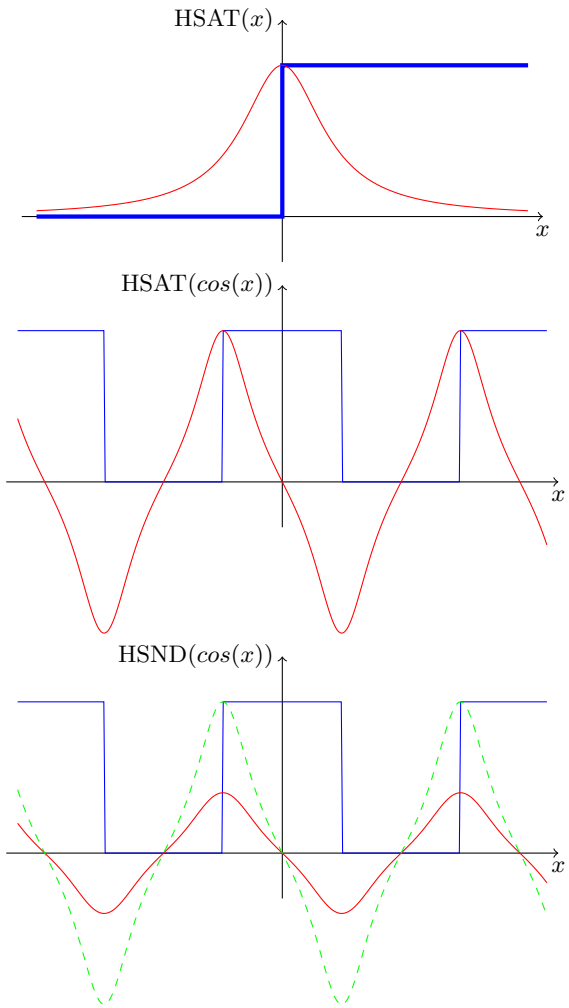


Figure 6: Three candidate composite activation functions: the Heaviside-Arctangent (y = HSAT(x)), the Cosine-Heaviside-Arctangent function y = HSAT$(cos(x))$, and the Cosine-Heaviside-NormalDistribution function y = HSND$(cos(x))$. The blue plot is the forward activation function, the red plot the backwards derivative. The green plot is the comparison between HSID (*green*) and HSND (*red*) derivatives

While the normal distribution requires more computation, it is mathematically more relevant, as the dirac δ impulse is the limit of the normal distribution as $\sigma \Rightarrow 0$. We will test different values for σ to see whether a synthetic gradient that fits the foward activation function's gradient more closely — that is, using a σ closer to 0 — yields better results.

We can see on Fig. 6 that the derivative part of the composite activation functions are "smoothed out" variants of their spiky, non-differentiable counterparts.

6.4 Loss Regularizations of the candidates

The following section shows the regularizations of the various candidate layers we experimented on.

As seen in Fig. 5, Spartan Networks have a scaled loss Regularization that rewards the filtering layer when it data-starves the network if $s_f > 0$.

6.4.1 Convolution-Filtering. A simple $\mathcal{L}_1$ activation regularizer is proposed. As the number of activation is reduced, the number of bright, activated pixels diminishes, data-starving the network.

6.4.2 Offset-Filtering. The filtering layer adds a term to the loss function to penalize the network if it gives away too much information. We need a loss function that attains its maximum at a half of the function value distribution and is 0 at the edges. The entropy function is the best candidate, and we thus define the regularization function as:

$$L((B_i)_{i\in\mathbb{N}}) = \sum_{i=1}^{N} B_i log(B_i)$$
$$B_i = \frac{b_i - x_i^{min}}{x_i^{max} - x_i^{min}} \tag{10}$$

And the complete loss is thus:

$$L_t((B_i), y, \hat{y}) = L((B_{i\in[1,N]})) + \sum_{i}^{C} y_i log(\hat{y}_i) \tag{11}$$

Where N is the input size, (x_i^{max}, x_i^{min}) the maximum and minimum value of the input for the i^{th} input, C the number of output classes, and y and $\hat{y}$ designate the ground-truth and predicted vectors respectively.

B_i is the rescaled bias, ranging from 0 to 1. This rescaled bias puts an *a priori* distribution over the dataset.

Note that this particular B_i holds when we hypothesize that there is a uniform distribution on the values. In an opposite case, one can create a cut-off based on a cumulative distribution that can be used to rescale the biases. A B_i close to $\frac{1}{2}$ will signal that the probability of drawing a sample pixel value above the threshold is the same as below the threshold.

6.5 Data-Starving Behaviours

6.5.1 Offset-Filtering. The filtering layer minimizes its regularization loss if the value of the all rescaled biases of the neurons of the filtering layer are closer to either 0 or 1. Rescaled biases close to 0 or 1 mean neurons change activation close the the minimum or the maximum value. The filtering layer thus forces the network to destroy as much information as it can, due to the behaviour of the entropy function:

- When the rescaled bias is at $\frac{1}{2}$, the ranges where $HS(x - B_i)$ are 0 or 1 are equal and the entropy function is at its maximum: the layer gives more information. We hence penalize the fact that the network thrives on information.

- When the rescaled bias is close to 0, $HS(x - B_i)$ is almost always 1: we get no extra information as this feature is bound to be present. Entropy regularization is close to 0
- When the rescaled bias is close to 1, $HS(x - B_i)$ is almost always 0. The feature will almost never be present. Entropy regularization is close to 0.

6.5.2 Convolutional-Filtering. The more the filtering layer activates, the higher the penalty is because of the activation regularization we put on this Filtering Layer. This means that the filters are biased towards higher thresholds, as a higher threshold value will decrease the number of inputs dimensions that cross the threshold, and will thus reduce the activation regularization penalty.

We thus encourage the network to report on a feature only if this feature is deemed extremely relevant, or if the value is extreme. We thus restrict the attacker to a visible attack.

7 DISCUSSION & RESULTS

7.1 Implementation

We implement our idea in Keras [10], using parts of the TensorFlow backend [2] for the implementation of the synthetic gradients. The general structure of the Spartan Network we implemented is as follow (first layer first):

- A Filtering Layer, whose parameters will vary but that will stay on the first layer. (Filtering Layer 1)
- A Convolutional Layer, 32 filters, 3×3 kernel, no stride, ReLU activation function.
- Another Convolutional Layer, same parameters. ReLU or Composite activation function. (Optional Filtering Layer 2)
- A 2×2 Pooling Layer
- A Convolutional Layer, 32 filters, 3×3 kernel, no stride, ReLU activation function.
- Another Convolutional Layer, same parameters. ReLU or Composite activation function. (Optional Filtering Layer 3)
- A 2×2 Pooling Layer.
- A Densely Connected Layer with 50 Neurons and ReLU activation function.
- A 10-classes Softmax.

The base CNN is close to the CNN used as standard ConvNet in [43].

7.2 Results

We tested a subset of all possible candidates created in the paper. The candidates we have tested are as follow:

7.2.1 Offset-Filtering. We tested only one network with an offset filtering on the filtering layer 1, using a HSAT activation function (Candidate A).

7.2.2 Convolutional-Filtering. We tested the following Spartan Networks using the Convolutional Filtering:

- The standard CNN using a HSND on the layer 1, as well as Cos-HSND layers on layers 2 and 3 (Candidate B)
- The standard CNN using a Cos-HSAT layer on layer 1 (Candidate C)

7.2.3 Robustness. We attack the candidate Spartan Networks with a FGSM attack in surrogate black-box mode with various strengths. We have used the CleverHans module version 2.1.0 [35] to perform the attacks. We show our results on Fig. 7.

The filter-regularization scaling factor is $s_f = 10^{-5}$, for all of the Spartan Networks, and $\mu = 0, \sigma = 1$ if the Normal Distribution is used. The number in parenthesis in the legend shows the test precision (in %) on clean samples.

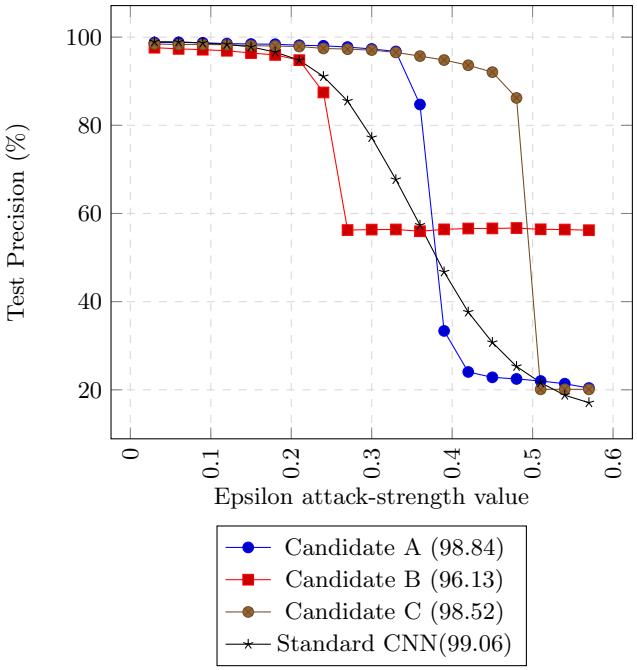


Figure 7: Comparison of a Spartan Network vs its vanilla CNN counterpart, against a FGSM attack with varying epsilon-strength

7.2.4 On the Training of a Network using Composite Activation Functions. We report the variation of loss over training iteration of the Candidate C compared to the standard CNN using only ReLU activation functions with no filtering layers. Results are seen on Fig. 8.

As the synthetic gradient allows the Spartan Network to know the general direction of improvement, it can train even if the Heaviside step function does not yield a progressive, differentiable behaviour. After some time, the networks' weights

are trained enough to go past the threshold when required, and the loss drops as sharply as a standard CNN.

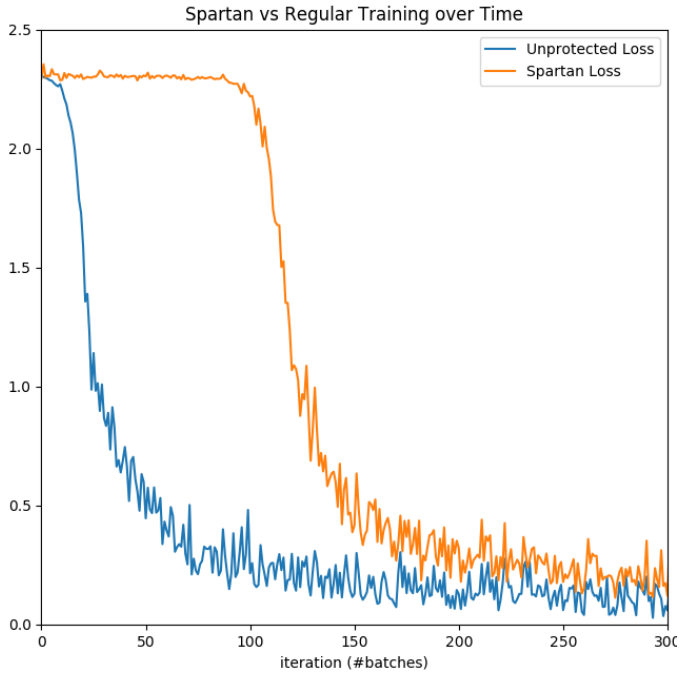


Figure 8: Loss history during the training of a classical CNN and a Spartan Network. We see a latency in the loss drop of the Spartan Network's training.

7.2.5 Resistance to Over-capacity. We note that in Fig. 7 Candidate C's precision begins to collapse at ϵ values close to 0.5. Note that, at this level, the noise value is high enough to create a grey picture. We see however that Candidate B manages to resist even *after* the 0.5 threshold. Upon further inspection, this Network has a 0.9 cutoff threshold in its filters with close to 0 bias. This network focuses on the brightest pixels. The attack did not supress enough bright pixels to entirely corrupt the digits, and the Spartan Networks finds the signal through the perturbation.

We report that out of the four given filters, the Candidate C discarded two filters by making their activation impossible. The network thus learns to ignore unnecessary capacity, by minimizing its activation penalty. This resistance is comforted by the fact that Candidate C *learns* that 1-bit black and white color is enough to classify digits succesfully. This result is aligned with previous results obtained by Xu et al. [43].

Spartan Networks thus have a reduced sensitivity to over-capacity. We argue that this resistance is trading robustness for reduced performance. This resistance assures that the hyperparameters introduced by Spartan Networks do not slow the hyperperameter tuning phase.

8 RISK EVALUATION OF A ROBUSTNESS-PRECISION TRADEOFF

Spartan Networks sacrifice a bit of precision for robustness. Notions of robustness-precision tradeoffs were already mentionned in [17, 23].

To give a risk analysis of a situation, consider a Spartan Network used in the context of a 4-digits check reading system, and consider an adversarial check paper that automatically gets missclassified as a 9999$ check. Let $\Delta_{CN\to SN}$ be the risk delta linked with going from a *"ConvNet"* to a *"SpartaNet"*. As the risk is the probability of occurence times the impact, we get:

$$\Delta_{CN\to SN} = (p_{eSN}I_{eSN}+p_{tSN}I_{tSN})-(p_{eCN}I_{eCN}+p_{tCN}I_{tCN})$$

Where I is an average impact value, p a probability of an event happening, SN and CN stand for Spartan Network and Convolutional Network missclassification and t and e describe a theft scenario, and an error scenario respectively.

As the Network does not change the Impact, only the probability, we have:

$$\Delta_{CN\to SN} = (p_{eSN} - p_{eCN})I_e + (p_{tSN} - p_{tCN})I_t$$

Each of the above probabilities is a joint event of an scenario happening, and a missclassification happening. We consider the two events disjoint.

$$\Delta_{CN\to SN} = (p(e,SN)-p(e,CN))I_e+(p(t,SN)-p(t,CN))I_t$$

We consider only abnormal check reading. α is the amount of malicious bank checks in the overall non-normal checks. Let $p(e,SN) - p(e,CN) = \Delta_{err}, p(t,advSN) - p(t,advCN) = \Delta_{adv}$, we get that:

$$\Delta_{CN\to SN} = (1-\alpha)\Delta_{err}I_e + \alpha\Delta_{adv}I_t$$

Diminishing the risks means that $\Delta_{CN\to SN} < 0$, and in this case, we obtain:

$$\alpha > \frac{\Delta_{err}I_e}{\Delta_{err}I_e - \Delta_{adv}I_t}$$

Spartan Networks are less precise on clean samples. Thus, they are more relevant in adversarial settings. This means that there is a minimum amount of malicious checks that must be in use for the bank to have an interest in them.

For instance, if we hypothesize that the attack is a $\epsilon = 0.3$ FGSM attack, we see in Fig. 7 that a Spartan Network is more robust to this attack by 20%. However, we must also consider that our Network comes with $\sim -0.5\%$ precision on non-adversarial inputs. The risk analysis must take into account the fact that non-malicious checks are being misclassified more often than with a ConvNet.

Risk Management Implications. For example, if a non-malicious error on a 4 digits check costs on average 50$, and a malicious error costs 8999$ (1000$ $\to$ 9999$), Spartan

Networks begin to reduce the risk when around one erroneous check on 7200 is malicious.

We thus recommend that Spartan Networks be used in conjunction with attack detection, that can use **Detection** strategies. In this setup, a classical, high-performance network would do the inferrence until an attack is detected. When such an attack happens, a Spartan Network could be used as a fallback network. In this adversarial setting, their robustness would overcome their performance issues.

9 CONCLUSION

We have presented Spartan Networks, deep neural networks that are given a data-starving layer in order to select relevant features only. The space-lowering effect allows the system to be more resistant to adversarial examples. Filtering layers reduce performance, but can, in specific threat models, be a cost-effective way to reduce an attacker's stealthiness and success rate. This robustness method is made by trying to select relevant features, thus reinforcing deep learning in its promise that feature selection can, in a long-term view, be automated.

Contributions. Our contributions are the following:

- We introduce the concept of *composite activation functions*, by separating the forward propagation and backwards propagation functions in a Deep Neural Network.
- We introduce the idea of a *self-adversarial layer*, putting an attack-agnostic layer *inside* the network to starve the subsequent layers off of information.
- We create the first *Spartan Network* using the ideas above, and test its robustness to black-box attackers.
- We evaluate the performance-robustness tradeoff of Spartan Networks in a simple threat model.

Future Work. This work puts forward a lot of experiments and we encourage our fellow researchers to explore on it. *Our code will be Open-Sourced to support this effort, and will be available at*:
https://github.com/FMenet.

We also see that the possibilities for remplacement gradients are endless: One could replace the gradient of a differentiable function by another to improve the update dynamic while retaining a desireable behaviour, like a gradient close to 1 to avoid exploding or vanishing gradients.

We expect to use more powerful, deeper and wider architectures to train on more complex datasets, but the current implementation of those networks is slower than binary activation would allow. We aim at producing a faster, more efficient way to backpropagate efficiently with our replacement gradients.

Image space topology allows for a smooth exploration of sample space, as a small euclidian distance often means same semantics. We leave the study of Spartan Networks on harder sample space topologies for future work.

ACKNOWLEDGMENTS

We would like to thank all our colleagues from the SecSI lab, and especially Ranwa Al-Mallah for useful comments and corrections.

REFERENCES

[1] [n. d.]. Fake News Detector AI | Detect Fake News Using Neural Networks. http://www.fakenewsai.com/

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[3] Mahdieh Abbasi and Christian Gagn. 2017. Robustness to Adversarial Examples through an Ensemble of Specialists. *arXiv:1702.06856 [cs]* (Feb. 2017). http://arxiv.org/abs/1702.06856 arXiv: 1702.06856.

[4] N. Akhtar and A. Mian. 2018. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *IEEE Access* 6 (2018), 14410–14430. https://doi.org/10.1109/ACCESS.2018.2807385

[5] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420* (2018).

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. *arXiv:1604.07316 [cs]* (April 2016). arXiv: 1604.07316.

[7] Wieland Brendel and Matthias Bethge. 2017. Comment on "Biologically inspired protection of deep networks from adversarial attacks". *arXiv:1704.01547 [cs, q-bio, stat]* (April 2017). arXiv: 1704.01547.

[8] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. 2018. THERMOMETER ENCODING: ONE HOT WAY TO RESIST ADVERSARIAL EXAMPLES. (2018), 22.

[9] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. *2017 IEEE Symposium on Security and Privacy (SP)* (2017), 39–57.

[10] François Chollet et al. 2015. Keras. https://keras.io.

[11] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830 [cs]* (Feb. 2016). http://arxiv.org/abs/1602.02830 arXiv: 1602.02830.

[12] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, and D. H. Chau. 2017. Keeping the Bad Guys Out: Protecting and Vaccinating Deep Learning with JPEG Compression. *ArXiv e-prints* (May 2017). arXiv:cs.CV/1705.02900

[13] H. Drucker and Y. Le Cun. 1991. Double backpropagation increasing generalization performance. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, Vol. ii. 145–150 vol.2. https://doi.org/10.1109/IJCNN.1991.155328

[14] Gamaleldin F. Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alex Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. 2018. Adversarial Examples that Fool both Computer Vision and Time-Limited Humans. *arXiv:1802.08195 [cs, q-bio, stat]* (Feb. 2018). http://arxiv.org/abs/1802.08195 arXiv: 1802.08195.

[15] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. 2017. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542, 7639 (2017), 115.

[16] Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B. Gardner. 2017. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410* (2017).

[17] Justin Gilmer, Luke Metz, Fartash Faghri, Samuel S. Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian Goodfellow. 2018. Adversarial Spheres. *arXiv:1801.02774 [cs]* (Jan. 2018). http://arxiv.org/abs/1801.02774 arXiv: 1801.02774.

[18] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. In *arXiv:1412.6572 [cs, stat]*. http://arxiv.org/abs/1412.6572 arXiv: 1412.6572.

[19] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).

[20] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2016. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *Proceedings of the 2017 European Symposium on Research in Computer Security* (June 2016). http://arxiv.org/abs/1606.04435 arXiv: 1606.04435.

[21] Shixiang Gu and Luca Rigazio. 2014. Towards Deep Neural Network Architectures Robust to Adversarial Examples. (Dec. 2014).

[22] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *arXiv:1708.06733 [cs]* (Aug. 2017). arXiv: 1708.06733.

[23] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. 2017. Countering Adversarial Images using Input Transformations. *arXiv:1711.00117 [cs]* (Oct. 2017). http://arxiv.org/abs/1711.00117 arXiv: 1711.00117.

[24] Dan Hendrycks and Kevin Gimpel. 2016. Early methods for detecting adversarial images. *arXiv preprint arXiv:1608.00530* (2016).

[25] Weiwei Hu and Ying Tan. 2017. Black-Box Attacks against RNN based Malware Detection Algorithms. *arXiv:1705.08131 [cs]* (May 2017). arXiv: 1705.08131.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov. 1998), 2278–2324. https://doi.org/10.1109/5.726791

[27] Xin Li and Fuxin Li. 2016. Adversarial examples detection in deep networks with convolutional filter statistics. *CoRR, abs/1612.07767* 7 (2016).

[28] Jiajun Lu, Theerasit Issaranon, and David Forsyth. 2017. Safetynet: Detecting and rejecting adversarial examples robustly. *CoRR, abs/1704.00103* (2017).

[29] Yan Luo, Xavier Boix, Gemma Roig, Tomaso A. Poggio, and Qi Zhao. 2015. Foveation-based Mechanisms Alleviate Adversarial Examples. *CoRR* abs/1511.06292 (2015).

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv:1706.06083 [cs, stat]* (June 2017). http://arxiv.org/abs/1706.06083 arXiv: 1706.06083.

[31] Dongyu Meng and Hao Chen. 2017. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 135–147.

[32] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267* (2017).

[33] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2015. DeepFool: a simple and accurate method to fool deep neural networks. *arXiv:1511.04599 [cs]* (Nov. 2015). arXiv: 1511.04599.

[34] A. Nayebi and S. Ganguli. 2017. Biologically inspired protection of deep networks from adversarial attacks. *ArXiv e-prints* (March 2017). arXiv:stat.ML/1703.09202

[35] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. 2018. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv preprint arXiv:1610.00768* (2018).

[36] Nicolas Papernot and Patrick McDaniel. 2017. Extending Defensive Distillation. *arXiv:1705.05264 [cs, stat]* (May 2017).

[37] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2016. Practical Black-Box Attacks against Machine Learning. *arXiv:1602.02697 [cs]* (Feb. 2016). http://arxiv.org/abs/1602.02697 arXiv: 1602.02697.

[38] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2015. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. *arXiv:1511.04508 [cs, stat]* (Nov. 2015). http://arxiv.org/abs/1511.04508 arXiv: 1511.04508.

[39] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James A. Storer. 2018. Deflecting Adversarial Attacks with Pixel Deflection. *CoRR* abs/1801.08926 (2018).

[40] Andrew Slavin Ross and Finale Doshi-Velez. 2017. Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing their Input Gradients. *arXiv:1711.09404 [cs]* (Nov. 2017). http://arxiv.org/abs/1711.09404 arXiv: 1711.09404.

[41] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv:1312.6199 [cs]* (Dec. 2013). http://arxiv.org/abs/1312.6199 arXiv: 1312.6199.

[42] D. Tingle, Y.E. Kim, and D. Turnbull. 2010. Exploring automatic music annotation with acoustically-objective tags. In *Proceedings of the international conference on Multimedia information retrieval*. ACM, 55–62.

[43] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. *arXiv:1704.01155 [cs]* (2018). https://doi.org/10.14722/ndss.2018.23198 arXiv: 1704.01155.

[44] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17* (2017), 363–376. https://doi.org/10.1145/3133956.3134018 arXiv: 1708.06525.