Partitionnement en régions linéaires pour la vérification formelle de réseaux de neurones

Julien Girard-Satabin, Aymeric Varasse, Guillaume Charpiat, Zakaria Chihani, Marc Schoenauer

Partitionnement en régions linéaires pour la vérification formelle de réseaux de neurones

Julien Girard-Satabin[†12], Aymeric Varasse[†1], Guillaume Charpiat[2], Zakaria Chihani[1], and Marc Schoenauer[2]

[1] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
Les auteurs marqués d'un [†] ont apporté une contibution identique
julien.girard2@cea.fr, aymeric.varasse@cea.fr, zakaria.chihani@cea.fr
[2] INRIA TAU, LRI, France
guillaume.charpiat@inria.fr, marc.schoenauer@inria.fr

Résumé

La grande polyvalence et les résultats impressionnants des réseaux de neurones modernes viennent en partie de leur non-linéarité. Cette propriété fondamentale rend malheureusement très difficile leur vérification formelle, et ce, même si on se restreint à une structure linéaire par morceaux. Cependant, chacune de ces régions linéaires prise indépendamment est simple à analyser. Nous proposons dans cet article une méthode permettant de simplifier le problème de vérification en opérant une séparation en multiples sous-problèmes linéaires. Nous présentons également des résultats concernant la structure de ces régions linéaires ainsi que leur similarité. Ce travail en cours démontre déjà la faisabilité de l'approche sur des problèmes simples ainsi que quelques expériences face à l'état de l'art.

1 Introduction

Les réseaux de neurones profonds ont fait l'objet d'intenses recherches au cours des dernières années. De premiers travaux théoriques ont prouvé leur qualité d'approximateurs universels [12], c'est-à-dire leur capacité théorique à approximer n'importe quelle fonction. Plus récemment, des résultats pratiques impressionnants sur des tâches traitant des données à haute dimension (texte, image, son) ont abouti à leur utilisation dans de nombreuses applications industrielles (p. ex. détection d'objet, modification d'image, robotique). Tout efficaces qu'ils soient, les réseaux de neurones sont des programmes, et en tant que tels ne sont pas exempts de dysfonctionnements. Des travaux de recherche ont par exemple exhibé les exemples adverses, des perturbations imperceptibles volontairement construites pour tromper le réseau. Ces exemples peuvent être transférés entre programmes [16], permettent de synthétiser des discours [17] ou des vidéos [5] à même de tromper les réseaux au niveau de l'état de l'art, et ne sont pas systématiquement détectables [24]. D'autres travaux ont démontré la possibilité de reconstruire les paramètres du réseau [23] ou les données utilisées pour son entraînement [19] uniquement en se basant sur les sorties, ce qui soulève de graves problèmes de confidentialité des données, en particulier dans le domaine médical.

Les réseaux de neurones profonds sont structurés en couches, successions d'opérations linéaires séparées par des fonctions d'activation non-linéaires, l'une des plus populaires étant la *Rectified Linear Unit* ou *ReLU* : $x \to max(x, 0)$. Cette fonction est linéaire par morceaux : ReLU a un comportement linéaire sur $\mathbb{R}_-$ et sur $\mathbb{R}_+$. Les régions de l'espace d'entrée délimitant de tels comportements sont qualifiées de *régions linéaires*, ou *facettes*. Il est commun de considérer le nombre de facettes d'un réseau comme une mesure de son expressivité [18], informellement sa capacité à approximer des fonctions de plus en plus complexes. Dans le cas d'usage classique

d'un réseau, on applique les couches linéaires et non-linéaires aux entrées. Chaque fonction d'activation pouvant produire deux cas linéaires, une exploration naïve de l'espace de recherche doit donc traiter un nombre exponentiel de cas. Cette explosion combinatoire constitue l'un des obstacles majeurs à l'application de techniques de vérifications exhaustives (*e.g.,* calcul de Satisfiabilité, SAT et Satisfiabilité Modulo Théorie, SMT).

Pour surmonter cet obstacle, nous proposons d'exploiter les résultats récents d'analyse de régions linéaires. Les auteurs de [11] avancent ainsi que le nombre effectif de facettes pour les réseaux de $\mathbb{R}$ dans $\mathbb{R}$ est borné linéairement en le nombre de neurones. Dans la suite de leurs travaux [10], ce résultat est étendu à une classe de programmes plus large et plus représentative de réseaux réels, par une borne supérieure qui n'est pas exponentielle en le nombre de neurones. Une borne plus souple est également présentée dans d'autres travaux, tels que [18]. En réduisant un réseau de neurones à une union de facettes, vérifier une propriété de sûreté sur un réseau complexe devient plus simple, car revient à vérifier un ensemble de problèmes linéaires.

Notre contribution s'établit ainsi :

1. nous proposons différentes méthodes d'énumération explicite de facettes, appliquées à différents réseaux de neurones ;
2. nous proposons un algorithme de subdivision d'un problème initial en sous-problèmes linéaires plus simples à résoudre ;
3. nous évaluons la performance de notre approche en l'appliquant sur des réseaux de neurones et en comparant par rapport à l'état de l'art ; nous analysons également les caractéristiques de ces régions linéaires en fonction de différents paramètres.

2 Travaux connexes

La vérification formelle de réseaux de neurones est un domaine naissant qui a déjà produit de nombreuses techniques et outils. Reluplex [13], son successeur Marabou [14] et le solveur Planet [9] constituent ainsi les premières tentatives de méthodes de vérification exhaustives, utilisant le calcul SMT. Dans ces travaux, les auteurs proposent une réécriture de l'algorithme du simplexe permettant une évaluation paresseuse des ReLU afin d'élaguer plus rapidement l'espace de recherche. Notre travail s'inscrit dans cette ligne de recherche ; toutefois nous proposons une technique de pré-traitement plutôt qu'une analyse totale. Notre proposition et la leur sont orthogonales, et peuvent potentiellement être combinées. D'autres techniques d'évaluation exhaustives ont été proposées, telles qu'une formulation en programmation en nombre entiers [22], et une autre utilisant le paradigme *branch and bound* [4].

Des approches non exhaustives existent également, et passent généralement à l'échelle sur des réseaux plus larges, au détriment de la précision de l'analyse. La propagation symbolique de domaines numériques et l'interprétation abstraite sont les techniques les plus courantes, utilisées par exemple dans Reluval [25], CNN-Cert [3] et ERAN [21] ; nous utilisons cette technique également. La délimitation entre ces deux familles n'est pas gravée dans le marbre : il existe par exemple des travaux combinant programmation en nombres entiers et propagation symbolique [20].

L'étude des régions linéaires des réseaux de neurones à ReLU a fait l'objet de travaux théoriques, notamment concernant leur énumération [18]. Une extension du théorème d'approximateur universel a été proposée par [1] pour les réseaux certifiés. Enfin, l'idée d'utiliser les régions linéaires pour améliorer la procédure de vérification formelle a été explorée par les auteurs de [7], où ils proposent une technique d'apprentissage qui maximise le volume des régions linéaires, ce qui résulte en une amélioration de la robustesse du réseau. Notre approche ne nécessite pas le

réentraînement du réseau et ne modifie pas les régions linéaires ; elle pourrait ainsi se combiner avec la leur.

3 Définitions

3.1 Vecteurs d'activation et facettes

Prenons comme exemple un ensemble d'entrée $\mathcal{X}$ en $\mathbb{R}^{D_{\text{in}}}$ (p. ex., pour des images RGB carrées de taille $p \times p$, $D_{\text{in}} = 3p^2$) et un espace de sortie $\mathcal{Y}$ en $\mathbb{R}^{D_{\text{out}}}$. On désigne par $f : \mathcal{X} \to \mathcal{Y}$ un réseau de neurones entraîné à L couches, de $\mathcal{X}$ vers $\mathcal{Y}$. Chaque couche a des entrées et sorties multidimensionnelles, représentées par des tableaux dont chaque case est un neurone. Une couche l_i a une entrée en $\mathbb{R}^{D_{(i-1)}}$ et une sortie en $\mathbb{R}^{D_i}$, pour $i = 2 \dots L$, avec $D_1 = D_{\text{in}}$ et $D_N = D_{\text{out}}$. On notera par la suite une couche par l pour éviter de surcharger la lecture. Nous considérons ici des réseaux pour lesquels chaque couche l est composée d'une application linéaire de paramètres θ^l, suivie d'une activation ReLU (par facilité, nous appelons ainsi la généralisation de ReLU à une entrée multidimensionnelle, opérant sur chaque coordonnée indépendamment). On note C^l la composition de ces deux opérations. Notons que θ^l résulte de l'entraînement du réseau et ne change pas durant son utilisation : les seules variables sont les vecteurs de $\mathcal{X}$. Pour une entrée quelconque, chaque neurone ReLU de chaque couche aura une sortie nulle (resp. positive) si son entrée est négative ou nulle (resp. positive), auquel cas il sera dit *inactif* (resp. *activé*).

On note l'état d'activation, ou schéma d'activation, $\mathcal{S}^l_{\mathcal{F}}$ l'état d'activation des neurones ReLU pour une couche donnée l : un neurone actif est noté 1, un neurone inactif est noté 0. À titre d'exemple, pour le réseau présenté Figure 1, $\mathcal{S}^1_{\mathcal{F}} = (0, 1, 1)$.

On appelle une *facette* le sous-ensemble $\mathcal{F}$ de l'espace d'entrée qui génère un certain vecteur d'activation $\mathcal{S}^l_{\mathcal{F}}$. Toutes les entrées dans cette facette définissent un même vecteur d'activation dans le réseau quand celui-ci les traite. Les facettes définissent donc des régions où le comportement du réseau est *linéaire*, car toutes les ReLU sont dans un état défini. Une entrée $\vec{x}$ appartient à une facette $\mathcal{F}$ si et seulement si son passage dans le réseau produit les mêmes schémas d'activation que les points inclus dans la facette.

3.2 Construction de facettes

Considérons un neurone $n_{i,l}$ de la couche l. Si ce neurone est activé, sa valeur, résultat d'une opération linéaire impliquant des neurones de la couche précédente, est positive. Par exemple, pour une multiplication matricielle d'éléments $w_{i,j}$, pour les sorties des couches précédentes $y_{j,(l-1)}$, il vient une contrainte linéaire

$$n_{i,l} = \sum_j w_{i,j} y_{j,(l-1)} > 0$$

De même, un neurone inactif amène la contrainte suivante

$$n_{i,l} = \sum_j w_{i,j} y_{j,(l-1)} \leq 0$$

Chaque neurone porte une telle contrainte sur les entrées ; une facette est ainsi la conjonction de ces contraintes. Géométriquement, on peut considérer une facette comme un polyèdre convexe décrit par l'ensemble des contraintes linéaires des activations. Voir la Figure 1 pour une illustration sur un exemple jouet.

Pour construire ces contraintes, on opère une propagation symbolique depuis les entrées. Chaque variable d'entrée est représentée par un symbole s_n. On définit une expression symbolique ϕ_e de la forme $\sum_n \alpha_{n,e} \cdot s_n + b$ où $\forall n \in [\![1, N]\!], \alpha_n \in \mathbb{R}$ et $b \in \mathbb{R}$. L'opération $\cdot$ représente la multiplication entre un réel et un symbole. Ces expressions sont construites pour chaque neurone ReLU : elles représentent l'équation de l'hyperplan séparant l'espace des entrées entre l'état activé ou désactivé. La construction de ces équations se fait par application des opérations linéaires des couches successives sur les expressions symboliques. On définit plusieurs opérations élémentaires entre opérations symboliques. Ici, $*$ (respectivement $+$) représente l'opération de multiplication (respectivement d'addition) usuelle sur les réels.

Addition d'un réel à une expression

Pour une expression ϕ_e donnée et une valeur réelle r, on a

$$\phi_e + r = \sum_n \alpha_{n,e} \cdot s_n + (b + r)$$

Multiplication d'une expression par un réel

Pour une expression ϕ_e donnée et une valeur réelle λ, on a

$$\lambda \cdot \phi_e = \sum_n (\lambda * \alpha_{n,e}) \cdot s_n + \lambda * b$$

Addition de deux expressions

Pour deux expressions ϕ_e^1 et ϕ_e^2, on note l'addition $\oplus$ et on a

$$\phi_e^1 \oplus \phi_e^2 = \sum_n \alpha_{n,e}^1 \cdot s_n + b^1 \oplus \sum_n \alpha_{n,e}^2 \cdot s_n + b^2 = \sum_n (\alpha_{n,e}^1 + \alpha_{n,e}^2) \cdot s_n + (b^1 + b^2)$$

Ces opérations linéaires sont composées au cours de la propagation des expressions symboliques dans le réseau, permettant d'obtenir les équations des hyperplans pour chaque neurone ReLU. Leur état d'activation est ensuite modifié en fonction de la facette considérée.

Dans le reste de cet article, nous cherchons à vérifier formellement un réseau de neurones. Pour un réseau f, une précondition sur l'espace d'entrée $\mathcal{D} \subset \mathcal{X}$ et une postcondition sur l'espace de sortie $\mathcal{P} \subset \mathcal{Y}$, nous souhaitons prouver formellement que

$$\forall x \in \mathcal{D} \implies f(x) \in \mathcal{P}$$

La forme exacte des pré et post conditions varie selon la propriété à vérifier. Ainsi, pour la robustesse locale face aux exemples adverses autour d'un échantillon, la propriété peut s'écrire, pour un échantillon $x \in \mathcal{X}$: $\forall \eta \in \mathbb{R}^{D_{\text{in}}}$ t.q. $\|\eta\| < \varepsilon$, $f(x + \eta) = f(x)$. Pour les propriétés de sûreté définies dans [15], les préconditions sont des contraintes linéaires.

4 Recherche de facettes pertinentes

Les sous-régions linéaires du réseau sont simples à vérifier, du fait de l'absence d'opérations produisant des disjonctions de cas (opérations dues aux ReLU). En supposant que nous disposions d'une cartographie exhaustive des facettes *effectivement atteignables* par notre réseau, il serait alors possible d'effectuer la vérification sur chaque facette. Le nombre théorique de

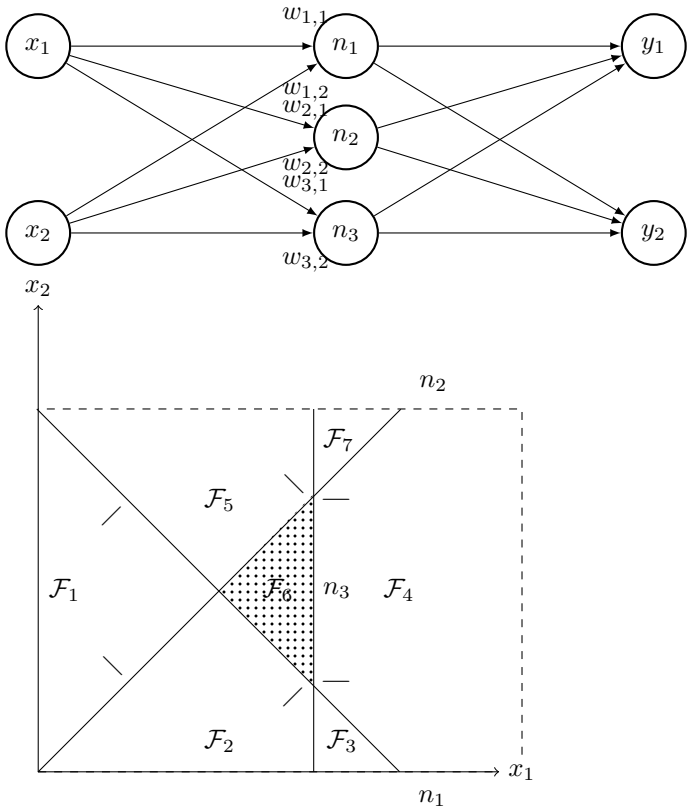

FIGURE 1 – Un réseau jouet où chaque n_i est un neurone ReLU, x_i et y_i sont des valeurs réelles, et $w_{i,j}$ des paramètres. En bas, les facettes sur l'espace d'entrée. L'état d'activation de chaque neurone génère un demi-espace. La zone en points définit la facette résultant en l'activité du neurone n_1 et l'inactivité des neurones n_2 et n_3.

facettes est exponentiel en le nombre de neurones ; toutefois un réseau n'exploite en pratique que rarement toutes ses régions linéaires. Pour une tâche simple, un réseau avec un grand nombre de neurones ne semble avoir besoin que d'une partition restreinte de l'espace d'entrée. Des facettes peuvent également disposer d'un support plus large dans l'espace d'entrée, ce qui peut par exemple indiquer que le réseau exploite beaucoup plus les informations présentes dans la région correspondante. Dans tous les cas, énumérer les facettes effectivement empruntables par notre réseau pour l'espace d'entrée considéré permettrait d'obtenir des informations intéressantes pour l'analyse. Notre procédure de subdivision se doit d'être exhaustive : si notre réseau est porteur d'un dysfonctionnement, notre procédure doit pouvoir l'exhiber. En d'autres termes, nous souhaitons trouver toutes les facettes $\mathcal{F}_i$, formant une partition $\bigcup_i \mathcal{F}_i = \mathcal{X}$.

Une première approche naïve consiste à énumérer les facettes voisines en partant d'une facette connue, déduite par exemple du calcul d'une entrée dans le réseau. Les facettes voisines peuvent alors être énumérées récursivement jusqu'à épuisement des voisins. Pour identifier les polytopes voisins, il est nécessaire d'identifier les neurones dont l'activation définit les arêtes du polytope courant. En effet, pour passer d'une facette $\mathcal{F}_i$ à une facette voisine $\mathcal{F}_j$, il suffit de changer l'activation d'un neurone, et de regarder si le polytope résultant est bien compris dans $\mathcal{X}$. En effet, deux facettes voisines sont séparées d'un hyperplan, hyperplan issu de la contrainte

linéaire d'un neurone. Il est donc nécessaire d'identifier quels hyperplans constituent les arêtes du polytope considéré, et changer l'état d'activation du neurone associé à cet hyperplan. Si le nouvel état d'activation respecte les contraintes de l'entrée, alors il constitue une nouvelle facette. L'Algorithme 1 résume cette procédure. Appliqué récursivement, il permet d'énumérer tous les voisins. Cette procédure exhaustive est cependant coûteuse en fonction de la procédure de résolution utilisée.

Data: An initial concrete facet $\mathcal{F}_{init}$, an input space domain $\mathcal{C}_{inputs}$
Result: a set of all valid hyperplanes delimiting $\mathcal{F}_{init}$

```
F_current = F_init ;
neighbours = ∅ ;
// take n random constraints to build an initial polyhedron
stack = TakeRandom(F_current) ⋃ C_inputs;
forall constraint ∈ F_current do
    // if there exists a point that respects existing constraints but does not respect the
       additional constraint, then it changes the form of the polyhedron
    if Solve(stack ⋃ ¬constraint) then
        // check if the obtained polyhedron is non-empty
        if Solve(stack ⋃ constraint) then
            stack = stack ⋃ constraint
        end
    end
end
return stack
```

Algorithme 1: Sélection d'un ensemble d'hyperplans contenant toutes les frontières d'une facette

Travail en cours

Une autre stratégie consiste à échantillonner des données issues de l'ensemble d'apprentissage du réseau, et de compter le nombre de facettes différentes auxquelles ces données appartiennent. On peut alors comparer ce nombre aux bornes théoriques proposées par exemple dans [10]. Cette méthode n'est cependant pas exhaustive et ne servirait qu'indicateur en haute dimensions : trouver une technique d'énumération plus efficace constitue un travail en cours. Une approche exploitant l'architecture en couche des réseaux est par exemple poursuivie, mais doit encore être améliorée pour des questions de performances.

5 Partitionnement en sous-problèmes linéaires

L'ensemble de toutes les facettes pertinentes $\bigcup_i \mathcal{F}_i$ obtenu constitue un partitionnement de l'espace d'entrée. Ainsi, vérifier la propriété sur chaque facette $\mathcal{F}_i$ est équivalent à vérifier la propriété sur $\mathcal{X}$ (en fait, les frontières sont vérifiées plusieurs fois). Ces problèmes linéaires étant indépendants, l'emploi de techniques de parallélisation est possible et permettrait une amélioration significative du temps de vérification. Plus formellement, considérons un ensemble de facettes $\bigcup_i \mathcal{F}_i$ pour un réseau f et un espace d'entrée $\mathcal{X}$. Le partitionnement consiste à ajouter au flot de contrôle de base du réseau f les contraintes sur les entrées $\mathcal{F}_i$, et à fixer l'état d'activation des neurones ReLU correspondant. Le flot de contrôle résultant n'est donc composé

que d'opérations linéaires : les opérations originales du flot de contrôle et les neurones ReLU activés ou désactivés en fonction du schéma d'activation $\mathcal{S}_{\mathcal{F}_i}$.

Travail en cours

Cette approche de partitionnement peut être raffinée en ciblant des régions linéaires à traiter en priorité par le solveur. Les facettes n'ont en effet pas toutes les mêmes caractéristiques : ainsi certains schémas d'activation peuvent être beaucoup plus présents que d'autres dans un ensemble de données particulier. À titre d'illustration, la figure 2 montre le nombre de schémas d'activation distincts pour un échantillonnage sur les réseaux présentés dans la Section 6.2. Nous constatons que dans certains cas, près de 40% des points de l'ensemble de données passent par la même facette. Il est dès lors envisageable de prioriser ces facettes pour obtenir le plus rapidement possible la plus large couverture des entrées, et détecter rapidement des contre-exemples.

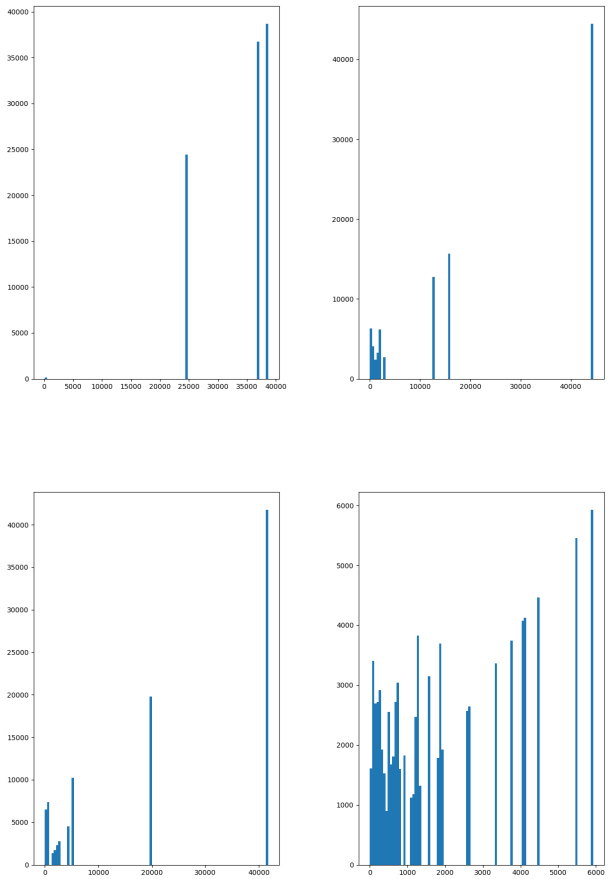

FIGURE 2 – Nombre de fois où une facette a été empruntée, pour 10000 échantillonnages. De gauche à droite et de haut en bas, réseaux de neurones différents avec dimension d'entrée croissante.

6 Évaluation

6.1 Implémentation

La technique de partitionnement est implémentée en OCaml, au sein de l'outil *Inter Standard Artificial Intelligence Encoding Hub* (ISAIEH)[1]. ISAIEH utilise le format standard de description de réseaux ONNX comme entrée principale. L'outil transcrit le flot de contrôle d'un réseau de neurones en formule SMT via le standard SMTLIB [2]. ISAIEH intègre également des techniques de réécriture pour simplifier des réseaux de neurones, dont celle que nous présentons dans cet article. La représentation intermédiaire employée par ISAIEH pour manipuler des réseaux de neurones est un graphe acyclique (N, A) orienté où les nœuds N représentent les opérations sur des tableaux multidimensionnels classiques en apprentissage profond (multiplications matricielles, convolutions, ReLU), et où les arêtes A, étiquetées avec le nom de tenseurs, définissent l'ordre de traitement des opérations. Chaque nœud N décrit l'opération courante, les entrées et les sorties ainsi que les paramètres éventuels nécessaires au calcul. Ce graphe est ensuite traité par un module d'écriture en SMT, qui se charge de convertir les opérations multidimensionnelles en format compréhensible par différentes théories SMT (QF_NRA, QF_LRA et partiellement QF_FP). L'ensemble des opérations supportées est un sous-ensemble des opérations définies par le standard ONNX[2]. Y figurent notamment la multiplication matricielle MatMul, l'application de la fonction ReLU et des opérations de *pooling* MaxPool.

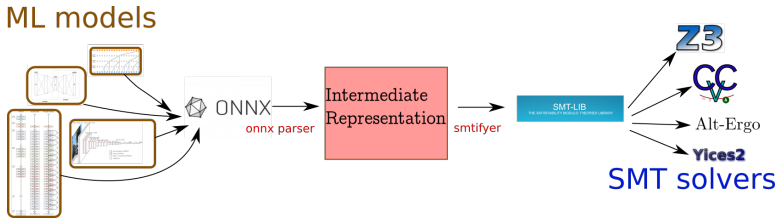

Figure 3 – Schéma de fonctionnement de ISAIEH

Les hyperplans sont obtenus par propagation symbolique au sein d'ISAIEH, ils sont ensuite convertis au format numpy array et manipulés via des scripts python. La bibliothèque python scikit-learn est utilisée pour résoudre le problème linéaire d'appartenance à une facette.

6.2 Expériences

L'hypothèse que l'on souhaite vérifier ici est la suivante : les neurones ReLU introduisent des non-linéarités qui imposent au solveur de séparer et vérifier chaque cas possible, et ceci augmente la complexité et le temps nécessaire à la résolution. En séparant le problème en différentes régions linéaires, ces non-linéarités ne sont plus présentes, et le temps de résolution s'en voit alors réduit, et inférieur au temps nécessaire à la vérification de la propriété pour la fonction non-linéaire directement sur l'ensemble de toutes les entrées possibles.

Les expériences ont été effectuées sur un ordinateur Dell Precision 5530, doté d'un processeur Intel Core i7-8850H cadencé à 2,6GHz, doté de 16 Go de mémoire RAM, avec pour système d'exploitation Ubuntu 18.04.1 LTS x86_64. Le multiprocessing est effectué grâce à la librairie multiprocess et exécutée par la librairie supbrocess, librairies standard de Python (3.8.5 64

1. https://git.frama-c.com/pub/isaieh
2. https://github.com/onnx/onnx/blob/master/docs/Operators.md

bits). La libraire Time de Python est utilisée afin de mesurer le temps nécessaire au solveur pour vérifier la propriété considérée sur chaque région linéaire du réseau de neurones. Pour la technique de partitionnement, les expériences comparent la vitesse de résolution de différents problèmes de sûreté classique pour différentes architectures.

On considère tout d'abord un réseau jouet de deux couches, avec des images en noir et blanc de taille $N \times N$ en guise d'entrée. Il y a $N/2$ neurones sur la première couche et $N/4$ sur la deuxième. Ce réseau doit déclencher une alarme quand il y a au moins un pixel blanc sur la moitié inférieure de l'image (ce qui symbolise la présence d'un obstacle) : il s'agit d'un problème de classification avec deux classes, "alarme" et "normal". Le signe de la sortie $f(x)$ du réseau désigne la classe prédite pour l'entrée x. Un maximum de trois pixels peuvent être allumés en même temps. Pour ces réseaux, le nombre de facettes peut être obtenu par énumération directe de toutes les images possibles ; les réseaux ne sont alors entraînés que sur une fraction de ce total. Le problème ϕ s'écrit alors $\forall x \in \mathcal{X}^{\text{unsafe}}, f(x) > 0$, où $\mathcal{X}^{\text{unsafe}}$ représente l'ensemble dénombrable des images contenant au moins un pixel blanc sur leur moitié inférieure. Le réseau est entraîné pour ne jamais se tromper, et le problème de vérification est UNSAT (il n'existe pas d'entrée obstacle telle que le réseau ne la détecte pas comme un obstacle). On vérifie également la propriété ϕ_4 de l'implémentation du Aircraft Collision Avoidance System, décrit dans [13]. L'objectif de ce réseau est de reconnaître les situations de risque de collisions entre deux aéronefs, et d'émettre des directives de changement de direction en cas de besoin. Le réseau compte six couches cachées pour un total de 300 neurones. Il compte cinq entrées, qui correspondent à des valeurs réelles issues de capteurs embarqués dans un avion, les capteurs sont supposés parfaits. Le réseau est entraîné pour émettre un choix de cinq directives : pas de changement, Strong Left, Strong Right, Weak Left, Weak Right. Les solveurs employés pour la vérification sont z3 [8] version 4.8.10, un solveur SMT état de l'art, Colibri [6], un solveur de programmation par contrainte et Marabou [14], un outil de vérification implémentant la procédure Reluplex.

La Table 1 synthétise les résultats de vérification. Le temps de vérification moyen d'une facette pour chaque réseau est indiqué quand il est disponible. Pour chaque réseau, la première ligne décrit les performances sans partitionnement linéaire, la deuxième (notée D&C) avec le partitionnement. La variante avec partitionnement s'exécute avec la logique QF_LRA, pour bénéficier au maximum des heuristiques de solveurs offertes par l'arithmétique linéaire. Pour les variantes sans partitionnement, on indique le temps en QF_LRA et QF_NRA pour comparaison.

Les résultats de vérification (SAT ou UNSAT) sont identiques sur les problèmes non partitionnés et tous les problèmes linéaires. La mention "multi process" indique les performances lorsque le solveur est lancé sur les différents problèmes indépendants via la bibliothèque de multiprocessing utilisée. Le timeout est à 12h. Il n'a pas été possible d'utiliser notre méthode sur Marabou à cause de problèmes de compatibilité de format. La génération des régions linéaires dure environ une dizaine de secondes pour un réseau 5×5.

La Table 2 donne des résultats préliminaires de dénombrement de facettes obtenus avec notre algorithme. Le nombre de facettes trouvé est comparé à la borne théorique proposée par [10] : $n^d/d!$ où n est le nombre de neurones et d la dimension de l'entrée.

6.3 Interprétation

Pour les variantes D&C, on observe que pour tous les réseaux jouets, la vérification de l'union des régions linéaires va systématiquement plus vite que l'application d'un solveur classique. La Table 2 montre pour les réseaux 3x3, 5x5 et 7x7 un nombre de facettes significativement plus faible que les bornes théoriques, ce qui constitue un résultat encourageant pour l'intérêt de

	z3	Marabou	Colibri
Net 5x5	34s (QF_NRA) 132s (QF_LRA)	0,05s	TIMEOUT
Net 5x5($D\&C$)	137s (single process) 23.7s (multi process) 0.05s (mean time to verify one facet)	–	1454s (multiprocess) 5.5s (mean time to verify one facet)
Net 7x7	391s (QF_NRA) TIMEOUT (QF_LRA)	0.02s	TIMEOUT
Net 7x7 (D&C)	1393s (multi process) 0.28s (mean time to verify one facet)	–	25326s (multi process) 7.3s (mean time to verify one facet)
ACAS ϕ_4	4280s (UNKNOWN, QF_NRA)	22s	TIMEOUT
ACAS ϕ_4($D\&C$)	0.35s (mean time to verify one facet)	–	2.18s (mean time to verify one facet)

Table 1 – Temps d'exécution pour différents problèmes

Réseau	nombre d'entrées	nombre de neurones	borne théorique	nombre de facettes
CAMUS 3x3	9	8	1067	4
CAMUS 5x5	25	18	$2e^9$	1007
CAMUS 7x7	49	36	$2e^{19}$	8560
CAMUS 9x9	81	62	$2.6e^{26}$	1859
ACAS	5	300	$2.02e^{10}$	–

Table 2 – Énumération de facettes

notre méthode.

On constate sur la Figure 2 que certaines mêmes facettes sont empruntées plus de 40% du temps, ce qui implique une faible diversité dans les schémas d'activation effectivement rencontrés.

La diminution du nombre de facettes pour le réseau 9x9 est un phénomène pour lequel nous n'avons pas encore d'explications : il appartiendra à la suite du travail d'élucider ce fait. Une hypothèse est que l'augmentation de la complexité de la tâche force le réseau à réorganiser ses régions linéaires durant l'entraînement, afin de traiter le problème avec un nombre réduit de facettes. Il n'a pas été possible d'obtenir le nombre exact de facettes sur un réseau ACAS, l'algorithme d'énumération ayant dysfonctionné sur ce réseau.

7 Discussion et perspectives

En nous appuyant sur des résultats théoriques récents, nous avons proposé une technique de partitionnement de problème de vérification de réseaux de neurones en sous-problèmes linéaires plus simples à vérifier. Cette technique emploie les régions linéaires de l'espace d'entrée générées par les fonctions d'activations, régions linéaires que nous dénombrons et analysons également.

Plusieurs pistes de poursuite de recherche sont envisagées. Compléter les expériences sur

un échantillon plus large de réseaux pour confirmer l'intérêt de notre technique en constitue une première. Le perfectionnement de notre algorithme d'énumération de voisins, notamment l'accélération de la procédure d'énumération de voisins en est une autre. L'approche purement géométrique que nous poursuivons pour l'instant risque de se heurter à la difficulté combinatoire observée en haute dimension ; il serait possible d'utiliser des raisonnements au niveau du réseau pour diminuer la complexité de l'algorithme. Enfin, de premiers éléments d'analyse tendent à montrer que les hyperplans varient relativement peu dans les domaines d'entrée considérés. Une possible amélioration de notre technique consisterait alors à modifier le réseau pour diminuer le nombre d'hyperplans et ainsi accélérer d'autant plus la vérification.

L'amélioration de l'outil ISAIEH, notamment par le support de plus d'opérateurs du standard ONNX et de formats de sortie, permettra d'utiliser notre technique sur plus de réseaux différents.

Remerciements

Ces travaux ont été soutenus financièrement par la Commission européenne par le biais du sous-projet SAFAIR du projet SPARTA, qui a reçu un financement du programme de recherche et d'innovation Horizon 2020 de l'Union européenne dans le cadre de la convention de subvention n° 830892. Ces travaux ont également reçu un soutien financier du projet CPS4EU, du financement de l'entreprise commune (EC) ECSEL dans le cadre de la convention de subvention n° 826276. L'EC reçoit le soutien du programme de recherche et d'innovation Horizon 2020 de l'Union européenne et de la France, de l'Espagne, de la Hongrie, de l'Italie et de l'Allemagne.

Références

[1] Maximilian Baader, Matthew Mirman, and Martin Vechev. Universal Approximation with Certified Networks. September 2019.

[2] Clark Barrett, Pascal Fontaine, and Aaron Stump. The SMT-LIB Standard. page 104.

[3] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. CNN-Cert : An Efficient Framework for Certifying Robustness of Convolutional Neural Networks. *arXiv :1811.12395 [cs, stat]*, November 2018. arXiv : 1811.12395.

[4] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. A Unified View of Piecewise Linear Neural Network Verification. *arXiv :1711.00455 [cs]*, November 2017. arXiv : 1711.00455.

[5] Shang-Tse Chen, Cory Cornelius, Jason Martin, and Duen Horng Chau. ShapeShifter : Robust Physical Adversarial Attack on Faster R-CNN Object Detector. *arXiv :1804.05810 [cs, stat]*, April 2018. arXiv : 1804.05810.

[6] Zakaria Chihani, Bruno Marre, François Bobot, and Sébastien Bardin. Sharpening Constraint Programming Approaches for Bit-Vector Theory. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, volume 10335, pages 3–20. Springer International Publishing, Cham, 2017.

[7] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In *the 22nd International Conference on Artificial Intelligence and Statistics*, pages 2057–2066, 2019.

[8] Leonardo De Moura and Nikolaj Bjørner. Z3 : An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Ruediger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. *arXiv :1705.01320 [cs]*, May 2017. arXiv : 1705.01320.

[10] Boris Hanin and David Rolnick. Deep ReLU Networks Have Surprisingly Few Activation Patterns. page 10.

[11] Boris Hanin and David Rolnick. Complexity of Linear Regions in Deep Networks. *arXiv :1901.09021 [cs, math, stat]*, January 2019. arXiv : 1901.09021.

[12] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2) :251–257, 1991.

[13] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex : An Efficient SMT Solver for Verifying Deep Neural Networks. *arXiv :1702.01135 [cs]*, February 2017. arXiv : 1702.01135.

[14] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, volume 11561, pages 443–452. Springer International Publishing, Cham, 2019.

[15] Guido Manfredi and Yannick Jestin. An introduction to acas xu and the challenges ahead. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–9. IEEE, 2016.

[16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in Machine Learning : from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv :1605.07277 [cs]*, May 2016. arXiv : 1605.07277.

[17] Yao Qin, Nicholas Carlini, Ian Goodfellow, Garrison Cottrell, and Colin Raffel. Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition. *arXiv :1903.10346 [cs, eess, stat]*, March 2019. arXiv : 1903.10346.

[18] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and Counting Linear Regions of Deep Neural Networks. *arXiv :1711.02114 [cs, math, stat]*, September 2018. arXiv : 1711.02114.

[19] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, San Jose, CA, USA, May 2017. IEEE.

[20] Gagandeep Singh and Timon Gehr. BOOSTING ROBUSTNESS CERTIFICATION OF NEURAL NETWORKS. *ICLR 2019*, page 12, 2019.

[21] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and Effective Robustness Certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10825–10836. Curran Associates, Inc., 2018.

[22] Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2018.

[23] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing Machine Learning Models via Prediction APIs. pages 601–618, 2016.

[24] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness May Be at Odds with Accuracy. *arXiv :1805.12152 [cs, stat]*, May 2018. arXiv : 1805.12152.

[25] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals. *arXiv :1804.10829 [cs]*, April 2018. arXiv : 1804.10829.