SAM : un outil de vérification de propriétés d'atteignabilité sur des classificateurs neuronaux

Arthur Clavière[1], Eric Asselin[1], Christophe Garion[2], and Claire Pagetti[3]

[1] Collins Aerospace, Toulouse, France
[2] ISAE-SUPAERO, Toulouse, France
[3] ONERA, Toulouse, France

Résumé

L'essor de l'utilisation de réseaux de neurones dans les systèmes critiques nécessite d'offrir des moyens de vérification adaptés à ces nouveaux types d'algorithme. L'outil SAM permet de vérifier des *propriétés d'atteignabilité* de *classificateurs neuronaux* en exploitant plusieurs solveurs existants. Dans la pratique, il fournit (1) une traduction automatique de la propriété à vérifier sur le classificateur en une propriété équivalente sur le ou les réseaux de neurones composant le classificateur, (2) une interface uniformisée vers les outils existants et (3) plusieurs heuristiques pour faciliter et accélérer la vérification. L'utilisation de SAM est illustrée sur l'étude de cas ACAS Xu – un système d'évitement de collision en vol.

1 Introduction

L'utilisation des réseaux de neurones, et notamment des *classificateurs neuronaux*, s'est répandue dans le monde de l'embarqué critique, avec plusieurs exemples d'applications tels qu'un système d'évitement de collision en vol [7] ou un régulateur de vitesse pour les véhicules autonomes [10]. Comme pour tout système critique, il faut apporter de la confiance et des garanties de bon fonctionnement. L'utilisation de vérification formelle est une réponse prometteuse. La vérification de *propriétés de sûreté* (ou *d'atteignabilité*) – à savoir qu'un système n'atteint pas un état indésirable – est un problème connu de longue date et de nombreux outils, tels que model checking, SAT/SMT ou interprétation abstraite pour ne citer qu'eux, permettent de vérifier de telles propriétés. Les spécificités des systèmes intégrant des briques de réseaux de neurones sont :

— l'absence d'états internes. Il s'agit d'une fonction de type calcul matriciel appliquée à un *tenseur* d'entrée (une matrice à plusieurs dimensions). En ce sens, le système est beaucoup plus simple qu'un automate ou un réseau de Petri ;

— la manipulation intensive de réels avec des matrices parfois de très grande taille. Or les outils classiques de vérification reposent plutôt sur l'analyse de Booléens et d'entiers.

C'est la raison pour laquelle, de nouveaux outils de vérification dédiés aux réseaux de neurones sont apparus tels que Reluplex/Marabou [8] ou Deeppoly [14]. Cependant les problèmes considérés étaient restreints à des réseaux de neurones [9, 12, 16, 1, 19, 20, 3] et non à des *classificateurs neuronaux*. La différence est que les classificateurs comportent un ou plusieurs réseaux, de fonctions de pre et post processing ainsi qu'une fonction de sélection de classe (souvent sous la forme d'un argmin). Notre contribution est donc la formalisation du problème d'atteignabilité pour un *classificateur neuronal* et le développement d'un outil appelé SAM (**S**afety **A**ssessment of **M**achine learning) reposant sur des solveurs existants et des stratégies d'appel efficaces.

La section 2 présente la notion de réseau de neurones, les méthodes et outils pour la vérification de ces réseaux ainsi que la notion de classificateur neuronal. Ensuite, la section 3 décrit le problème de vérification de propriétés d'atteignabilité sur des classificateurs neuronaux et montre comment ce type de problème peut être traduit en un problème de vérification de réseau de neurones, pouvant être résolu par un outil dédié. La section 4 donne une présentation détaillée de l'outil SAM et la section 5 montre comment SAM peut être utilisé afin de montrer la sûreté de fonctionnement d'un système d'évitement de collision en vol utilisant des classificateurs neuronaux (ACAS Xu).

2 Concepts élémentaires

L'objectif de cette partie est de rappeler les concepts nécessaires à la vérification de classificateurs neuronaux.

2.1 Réseaux de neurones

Nous nous restreignons à un sous-ensemble de réseaux de neurones à savoir les réseaux de neurones ReLU feed-forward fully-connected. A noter que cette hypothèse est moins restrictive qu'il n'y paraît car de nombreux types de couche de réseaux de neurones, telles que convolution ou max pool sont équivalentes fonctionnellement à des réseaux de neurones ReLU feed-forward fully-connected [11].

Définition 1 (Réseau de neurones ReLU feed-forward fully-connected)**.** *Un* réseau de neurones ReLU feed-forward fully-connected *est une fonction* $F_{\mathcal{N}}$ *définie par les paramètres* $\mathcal{N} = (L, T, W, B)$ *où :*

- $L > 2$ *est la* profondeur *du réseau ;*
- $T = (k_1, \ldots, k_L) \in \mathbb{N}^L$ *avec* L *un entier, est la* topologie *du réseau.* k_1*, respectivement* k_L*, est la dimension de l'entrée du réseau, respectivement de la sortie du réseau ;*
- $W = (\mathbf{W}_2, \ldots, \mathbf{W}_L) \in \mathbb{R}^{k_1 \times k_2} \times \ldots \times \mathbb{R}^{k_{L-1} \times k_L}$ *est un ensemble de* $L-1$ *matrices, appelées* poids du réseau *;*
- $B = (\mathbf{B}_2, \ldots, \mathbf{B}_L) \in \mathbb{R}^{k_2} \times \ldots \times \mathbb{R}^{k_L}$ *est un ensemble de* $L-1$ *vecteurs, appelés* biais du réseau.

Le réseau de neurones $F_{\mathcal{N}}$ *défini par ces paramètres est la fonction :*

$$F_{\mathcal{N}} : \begin{cases} \mathbb{R}^{k_1} \to \mathbb{R}^{k_L} \\ \mathbf{x} \mapsto F_L \circ \sigma_{k_{L-1}} \circ F_{L-1} \circ \ldots \circ \sigma_{k_2} \circ F_2(\mathbf{x}) \end{cases} \tag{1}$$

où F_l *est une transformation affine :*

$$\forall l \in [\![2, L]\!], \quad F_l : \begin{cases} \mathbb{R}^{k_{l-1}} \to \mathbb{R}^{k_l} \\ \mathbf{x} \mapsto \mathbf{B}_l + \mathbf{W}_l \cdot \mathbf{x} \end{cases} \tag{2}$$

et σ_{k_l} *est la fonction ReLU définie par :*

$$\forall l \in [\![2, L-1]\!], \quad \sigma_{k_l} : \begin{cases} \mathbb{R}^{k_l} \to \mathbb{R}^{k_l} \\ \mathbf{x} \mapsto (max(0, \mathbf{x}_1), \ldots, max(0, \mathbf{x}_{k_l})) \end{cases} \tag{3}$$

2.2 Propriétés d'atteignabilité des réseaux de neurones

Plusieurs travaux récents se sont intéressés à la vérification de *propriétés d'atteignabilité sur des réseaux de neurones* [9, 12, 16, 1, 8, 4, 17, 15, 13, 14, 18]. Ces propriétés s'expriment sous la forme suivante : "*si l'entrée du réseau appartient à un certain ensemble X, alors la sortie doit appartenir à un certain ensemble Y*".

Définition 2 (Propriété d'atteignabilité sur un réseau [9, 12])**.** *Une propriété d'atteignabilité sur un réseau est un n-uplet $\mathcal{P}_r = (F_\mathcal{N}, X, Y)$ où $F_\mathcal{N}$ est un réseau ; X un sous-ensemble de $\mathbb{R}^p$ avec p la dimension de l'espace d'entrée de $F_\mathcal{N}$ et Y un sous-ensemble de $\mathbb{R}^m$ avec m la dimension de l'espace de sortie de $F_\mathcal{N}$. La propriété $\mathcal{P}_r$ est satisfaite si et seulement si :*

$$F_\mathcal{N}(X) \subseteq Y \tag{4}$$

où $F_\mathcal{N}(X) = \{F_\mathcal{N}(\mathbf{x}) \mid \mathbf{x} \in X\}$ est l'ensemble des sorties atteignables *depuis X.*

Exemple 1 (Propriété valide $\mathcal{P}_r^1$)**.** *La Figure 1 illustre le cas d'une propriété $\mathcal{P}_r^1 = (F_{\mathcal{N}_1}, X, Y)$ sur un réseau $F_{\mathcal{N}_1} : \mathbb{R}^2 \to \mathbb{R}^2$. Cette propriété est valide car $F_{\mathcal{N}_1}(X) \subseteq Y$.*

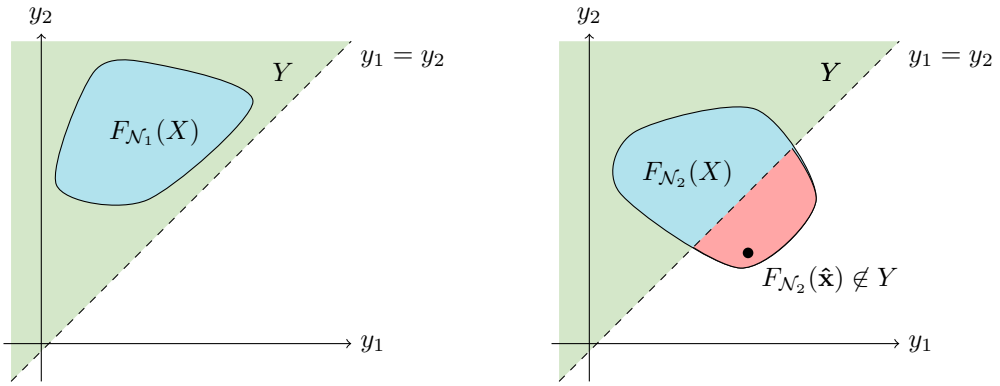

FIGURE 1: Un exemple de propriété valide $\mathcal{P}_r^1 : F_{\mathcal{N}_1}(X) \subseteq Y$ (à gauche) et un contre-exemple $\hat{\mathbf{x}}$ à une propriété invalide $\mathcal{P}_r^2 : F_{\mathcal{N}_2}(X) \subseteq Y$ (à droite)

Soit une propriété d'atteignabilité donnée $\mathcal{P}_r = (F_\mathcal{N}, X, Y)$. Vérifier la propriété $\mathcal{P}_r$ signifie décider si $\mathcal{P}_r$ est valide ou non. Ce problème de décision est habituellement résolu par un *vérificateur* qui prend en entrée la propriété $\mathcal{P}_r$ et retourne un résultat dans $\{\texttt{True}, \texttt{False}, \texttt{Unknown}, \texttt{Timeout}\}$. Afin de caractériser le degré de confiance qui peut être accordé au résultat fourni par un vérificateur, les notions de vérificateur *correct* et de vérificateur *complet* sont couramment utilisées :

Définition 3 (Vérificateur correct [9, 12, 8])**.** *Un vérificateur est* correct *si,* pour toute *propriété $\mathcal{P}_r$, il retourne* `True` *seulement si $\mathcal{P}_r$ est valide.*

Définition 4 (Vérificateur complet [9, 12, 8])**.** *Un vérifiateur est* complet *si,* pour toute *propriété $\mathcal{P}_r$:*

- *il retourne un résultat dans* $\{\texttt{True}, \texttt{False}, \texttt{Timeout}\}$ *;*
- *il retourne* `False` *seulement si $\mathcal{P}_r$ est invalide.*

Etant données ces définitions, un vérificateur correct *et* complet fournira le résultat de plus grande confiance. Un tel vérificateur pourra cependant ne pas terminer en temps raisonnable, étant donné le caractère NP-difficile du problème de vérification [8]. Ci-dessous, nous présentons deux types de vérificateurs, correspondant à deux approches possibles. Du fait de la difficulté à être simultanément correct, complet et rapide, l'un de ces aspects est souvent laissé de côté, par exemple un vérificateur correct et qui termine en temps raisonnable mais sans être complet.

2.2.1 Falsification

Une première approche, appelée *falsification*, consiste à étudier la *négation* de $\mathcal{P}_r$. Plus précisément, il s'agit de décider s'il existe un *contre-exemple* à $\mathcal{P}_r$ *i.e.,* un élément $\hat{\mathbf{x}} \in X$ tel que $F_{\mathcal{N}}(\hat{\mathbf{x}}) \notin Y$.

Exemple 2 (Contre-exemple)**.** *Soit $\mathcal{P}_r^2 = (F_{\mathcal{N}_2}, X, Y)$. Cette propriété étant invalide, il existe au moins un contre-exemple, comme celui illustré en Figure 1.*

Une méthode classique pour résoudre ce problème de falsification consiste à exprimer l'existence d'un contre-exemple via un ensemble de contraintes. Plusieurs vérificateurs implantent cette méthode *i.e.,* cherchent à déteminer si ces contraintes sont satisfaites afin de décider de l'existence ou non d'un contre-exemple. Parmi ces verificateurs, on peut citer Reluplex/Marabou [8] ou Planet [4]. Ces vérificateurs sont à la fois *corrects et complets*, au détriment d'un temps de résolution souvent long pour des réseaux de grande taille grande ou des ensembles d'entrée X grands.

2.2.2 Analyse d'atteignabilité

Une seconde approche, appelée *analyse d'atteignabilité*, consiste à calculer l'ensemble $F_{\mathcal{N}}(X)$ des sorties atteignables depuis X. Il est ensuite possible de vérifier si la relation d'inclusion $F_{\mathcal{N}}(X) \subseteq Y$ est satisfaite ou non. Cependant, le calcul exact de $F_{\mathcal{N}}(X)$ peut s'avérer très coûteux et une solution alternative consiste à calculer une *sur-approximation* des sorties atteignables *i.e.,* un ensemble $\widetilde{F_{\mathcal{N}}}(X)$ tel que $\widetilde{F_{\mathcal{N}}}(X) \supseteq F_{\mathcal{N}}(X)$.

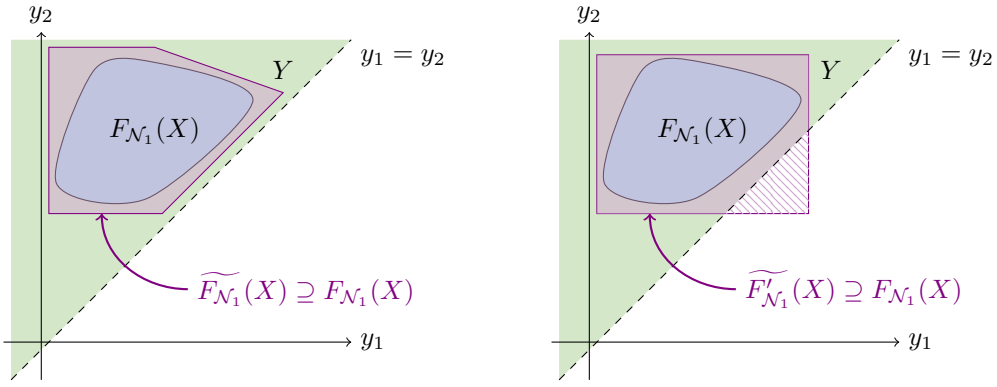

Figure 2: Un exemple de sur-approximation $\widetilde{F_{\mathcal{N}_1}}(X) \subseteq Y$ (à gauche) et un exemple de sur-approximation $\widetilde{F'_{\mathcal{N}_1}}(X) \not\subseteq Y$ telle que $\widetilde{F'_{\mathcal{N}_1}}(X) \cap Y \neq \emptyset$ (à droite)

Exemple 3. *Soit la propriété valide $\mathcal{P}_r^1 = (F_{\mathcal{N}_1}, X, Y)$. La Figure 2 illustre $F_{\mathcal{N}_1}$ et une sur-approximation possible $\widetilde{F_{\mathcal{N}_1}}$. Nous avons $\widetilde{F_{\mathcal{N}_1}} \subseteq Y \Rightarrow \mathcal{P}_r^1$ est valide.*

Plusieurs vérificateurs implantent cette méthode *i.e.,* cherchent à calculer une sur-approximation des sorties atteignables, par exemple DEEPPOLY [14] ou *NNV* [15]. Ces vérificateurs sont *corrects* mais *pas complets* et peuvent donc renvoyer le résultat `Unknown`. Cependant, ces vérificateurs ont souvent un coût en temps de calcul très largement inférieur à celui des vérificateurs s'appuyant sur des méthodes de falsification complètes.

Exemple 4. *Soit la propriété valide* $\mathcal{P}_r^1 = (F_{\mathcal{N}_1}, X, Y)$. *La Figure 2 illustre une sur-approximation possible* $\widetilde{F'_{\mathcal{N}_1}}$. *Cette sur-approximation est telle que* $\widetilde{F'_{\mathcal{N}_1}} \not\subseteq Y$ *et* $\widetilde{F'_{\mathcal{N}_1}} \cap Y \neq \emptyset$. *Avec cette seule connaissance, le vérificateur ne peut conclure quant à la validité de la propriété et renverra le résultat* `Unknown`.

2.3 Classificateurs neuronaux

Les réseaux de neurones présentés en section 2.1 peuvent être composés avec d'autres fonctions afin de réaliser une tâche de *classification*, c'est-à-dire produire une sortie parmi un *ensemble fini* de sorties possibles, aussi appelées *classes*. Nous appelons ce type de fonction *classificateur neuronal*, dont une forme typique est donnée ci-après :

Définition 5 (Classificateur neuronal)**.** *Un classificateur neuronal à* m *classes est une fonction* $F_{\mathcal{C}_t}$ *définie par les paramètres* $\mathcal{C}_t = (\mathcal{N}, I, pre, post, \pi)$ *où :*

- $\mathcal{N}$ *définit un réseau de neurones* $F_{\mathcal{N}} : \mathbb{R}^p \to \mathbb{R}^m$ *;*
- $I \subset \mathbb{R}^p$ *est un ensemble d'entrée ;*
- $pre : \mathbb{R}^p \to \mathbb{R}^p$ *est une fonction de pré-traitement, par exemple une normalisation min-max ou standard ;*
- $post : \mathbb{R}^m \to \mathbb{R}^m$ *est une fonction de post-traitement, par exemple une fonction softmax ou softmin ;*
- $\pi : [\![1, m]\!] \to \mathcal{C}$ *est une bijection de* $[\![1, m]\!]$ *dans* C *où* $C = (c_1, \ldots, c_m)$ *est l'ensemble fini des* m *sorties possibles du classificateur, aussi appelées* classes.

Le classificateur neuronal $F_{\mathcal{C}_t}$ *est la fonction définie par :*

$$F_{\mathcal{C}_t} = \pi \circ argmin \circ post \circ F_{\mathcal{N}} \circ pre \tag{5}$$

où $argmin : \mathbb{R}^m \to [\![1, m]\!]$ *est défini par* $argmin(\mathbf{y}) := argmin_i\ \mathbf{y}_i$.

3 Propriétés d'atteignabilité des classificateurs neuronaux

Nos travaux se concentrent sur l'étude de propriétés d'atteignabilité sur des classificateurs neuronaux. Ce type de propriété est plus complexe qu'une propriété d'atteignabilité sur un réseau de neurones, car elle implique une fonction plus complexe. En particulier, les fonctions, *pre*, *post*, *argmin* et π impliquées dans le classificateur neuronal et mentionnées en section précédente doivent être prises en compte.

Définition 6 (Propriété d'atteignabilité sur un classificateur neuronal)**.** *Une propriété d'atteignabilité sur un classificateur neuronal* $F_{\mathcal{C}_t}$ *est un n-uplet* $\mathcal{P}_{\mathcal{C}_t} = (F_{\mathcal{C}_t}, X, C^{des})$ *où :*

- $X \subseteq I$ *est un sous-ensemble de l'espace d'entrée* I *;*
- $C^{des} = \{c_j\}_{j \in \mathcal{J}}$ *avec* $\mathcal{J} \subseteq [\![1, m]\!]$ *un sous-ensemble de l'espace de sortie discret* $C = \{c_1, \ldots, c_m\}$.

La propriété $\mathcal{P}_{\mathcal{C}_t}$ est satisfaite si et seulement si :

$$F_{\mathcal{C}_t}(X) \subseteq C^{des} \tag{6}$$

Exemple 5 (Propriété valide $\mathcal{P}^1_{\mathcal{C}_t}$)**.** *La Figure 3 est un exemple de propriété satisfaite $\mathcal{P}^1_{\mathcal{C}_t} = (F^1_{\mathcal{C}_t}, X, C^{des})$ sur un classificateur $F^1_{\mathcal{C}_t} : \mathbb{R}^2 \to C$. La Figure représente les ensembles X, $C = \{c_1, c_2, c_3, c_4, c_5\}$, $C^{des} = \{c_1, c_2, c_5\}, F^1_{\mathcal{C}_t}(X) = \{c_2, c_5\}$. La propriété $\mathcal{P}^1_{\mathcal{C}_t}$ est satisfaite car $F^1_{\mathcal{C}_t}(X) \subseteq C^{des}$.*

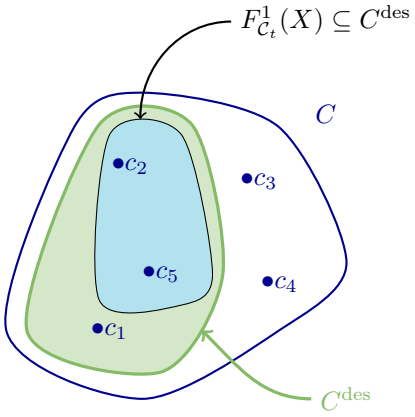

FIGURE 3: Une propriété valide $F^1_{\mathcal{C}_t}(X) \subseteq C^{\text{des}}$

Notre première contribution consiste à *traduire* la propriété (6) afin de nous ramener à une propriété sur le réseau de neurones $F_{\mathcal{N}}$, via des transformations pour prendre en compte d'une part la fonction *pre* en amont du réseau $F_{\mathcal{N}}$ et d'autre part les fonctions *post*, *argmin* et π en aval du réseau $F_{\mathcal{N}}$, comme illustré en Figure 4. En se ramenant à une propriété sur $F_{\mathcal{N}}$, il est alors possible d'utiliser les vérificateurs présentés en section 2.2.

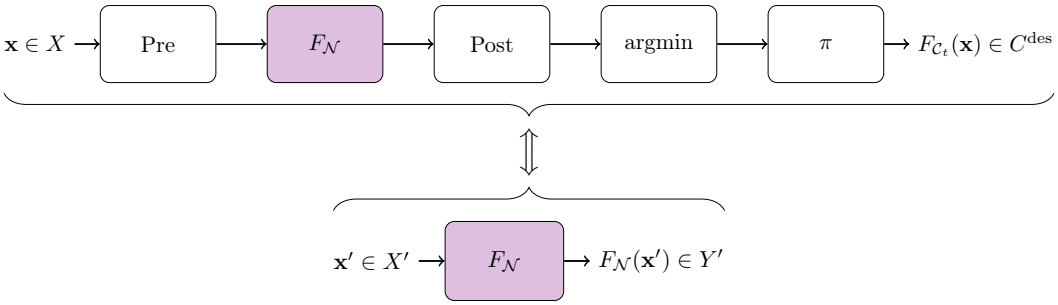

FIGURE 4: L'équivalence $\mathcal{P}_{\mathcal{C}_t} = (F_{\mathcal{C}_t}, X, C^{\text{des}}) \iff \mathcal{P}_r = (F_{\mathcal{N}}, X', Y')$

Dans la suite, nous faisons l'hypothèse que la fonction *post* préserve l'ordre relatif entre les composantes de son entrée, ce qui s'avère légitime car les fonctions couramment utilisées, telles que *softmax*, respectent cette hypothèse. Plus précisément, nous supposons que :

$$\forall \mathbf{x} \in \mathbb{R}^m, \forall (i,k) \in [\![1,m]\!]^2, i \neq k, \quad (\mathbf{x}_i \leq \mathbf{x}_k) \Rightarrow (\text{post}(\mathbf{x}_i) \leq \text{post}(\mathbf{x}_k)) \tag{7}$$

Sous cette hypothèse, le théorème suivant permet de traduire la propriété $\mathcal{P}_{\mathcal{C}_t}$ en une propriété d'atteignabilité sur le réseau de neurones $F_{\mathcal{N}}$ composant $\mathcal{P}_{\mathcal{C}_t}$:

Théorème 1. *La propriété d'atteignabilité $\mathcal{P}_{\mathcal{C}_t} = (F_{\mathcal{C}_t}, X, C^{des})$ est équivalente à la propriété d'atteignabilité $\mathcal{P}_r = (F_{\mathcal{N}}, X', Y')$ sur le réseau $F_{\mathcal{N}}$, comme illustré en Figure 4 :*

$$\forall \mathbf{x}' \in X',\ F_{\mathcal{N}}(\mathbf{x}') \in Y' \tag{8}$$

où :

— $X' = pre(X)$;

— $Y' = \left\{ \mathbf{y}' \in \mathbb{R}^m \mid \bigwedge_{i \in [\![1,m]\!] \setminus \mathcal{J}} \left(\bigvee_{j \in \mathcal{J}} \quad \mathbf{y}'_{\pi^{-1}(c_j)} \leq \mathbf{y}'_{\pi^{-1}(c_i)} \right) \right\}$

4 SAM : un vérificateur de classificateurs neuronaux

Pour vérifier des classificateurs neuronaux, nous n'avons pas cherché à développer de nouveaux vérificateurs mais plutôt à utiliser les vérificateurs existants, ce que permet l'application du Théorème 1. Notre seconde contribution est un outil nommé SAM qui suit cette approche *i.e.,* s'appuie sur des vérificateurs existants comme par exemple Reluplex/Marabou ou Deeppoly pour vérifier des propriétés d'atteignabilité sur des classificateurs neuronaux.

4.1 Architecture

Tout d'abord, SAM fonctionne sous les hypothèses suivantes :

Hypothèse 1. *La fonction* Pre *composant le classificateur $F_{\mathcal{C}_t}$ est supposée s'écrire sous la forme suivante :*

$$Pre(\mathbf{x})_i = a_i \cdot \mathbf{x}_i + b_i \tag{9}$$

où $(a_i, b_i) \in \mathbb{R}^2$.

Remarque 1. *Cette hypothèse n'est pas restrictive puisqu'elle est satisfaite par la plupart des fonctions de pré processing utilisée aujourd'hui, telles que les fonctions de* normalisation *ou de* standardisation *[5].*

Hypothèse 2. *L'ensemble X est un hyper-rectangle.*

Remarque 2. *A nouveau, cette hypothèse est pertinente puisqu'elle correspond aux propriétés étudiées usuellement dans la littérature e.g., [8, 1, 17].*

Comme illustré en Figure 5, notre outil a les entrées et sorties suivantes :

— **Entrées :**
  — $\mathcal{P}_{\mathcal{C}_t} = (F_{\mathcal{C}_t}, X, C^{\text{des}})$: une propriété d'atteignabilité sur un classificateur $F_{\mathcal{C}_t}$ où $X = \times_{i \in [\![1,p]\!]} [\underline{\mathbf{x}}_i, \overline{\mathbf{x}}_i]$ est un produit d'intervalles *i.e.,* un hyper-rectangle ;
  — `Config` : est un ensemble de paramètres permettant de configurer SAM, et dont le détail est donné plus tard dans cette section.

— **Sorties :**
  — `Result` $\in$ {`true`, `false`, `unknown`, `timeout`} : une réponse au problème de vérification.

Comme illustré en Figure 5, SAM se décompose en plusieurs blocs fonctionnels, décrits ci-dessous.

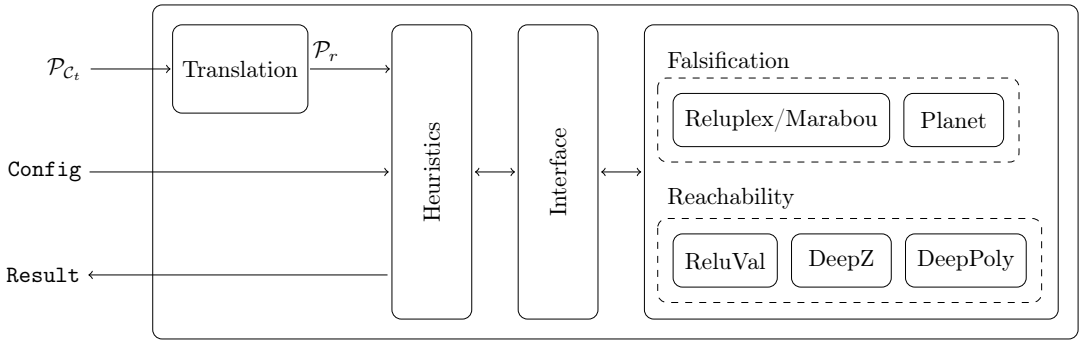

FIGURE 5: Architecture de SAM

4.2 Translation

Ce bloc traduit la propriété $\mathcal{P}_{\mathcal{C}_t} = (F_{\mathcal{C}_t}, X, C^{\text{des}})$ en une propriété équivalente $\mathcal{P}_r = (F_{\mathcal{N}}, X', Y')$ via une application directe du Théorème 1. La propriété équivalente $\mathcal{P}_r$ est calculée en deux étapes :

(1) X' est calculé par interprétation abstraite, en s'appuyant sur le domaine abstrait des intervalles [6]. En effet, cette technique permet de calculer de manière exacte l'ensemble X' dans le cas où l'hypothèse 1 est satisfaite :

$$X' = \text{Pre}^{\#}(X) \tag{10}$$

où $\text{Pre}^{\#}$ est appelé *transformeur abstrait* de la fonction *Pre* et est défini par :

$$\text{Pre}^{\#}(\times_{i\in[\![1,p]\!]}[\underline{\mathbf{x}}_i, \overline{\mathbf{x}}_i]) = \times_{i\in[\![1,p]\!]}[\underline{\mathbf{x}}'_i, \overline{\mathbf{x}}'_i] \tag{11}$$

où

$$\begin{cases} \underline{\mathbf{x}}'_i = (a_i \geq 0)\cdot a_i \cdot \underline{\mathbf{x}}_i + (a_i < 0)\cdot a_i \cdot \overline{\mathbf{x}}_i + b_i & (12) \\ \overline{\mathbf{x}}'_i = (a_i \geq 0)\cdot a_i \cdot \overline{\mathbf{x}}_i + (a_i < 0)\cdot a_i \cdot \underline{\mathbf{x}}_i + b_i & (13) \end{cases}$$

et $\text{Pre}(\mathbf{x})_i = a_i \cdot \mathbf{x}_i + b_i$ (voir Hypothèse 1).

(2) Y' est calculé comme l'ensemble de contraintes données au Théorème 1.

4.3 Interface

Ce bloc implante une fonction `solve` permettant d'appeler un vérificateur de réseau de neurones sur une propriété d'atteignabilité $\mathcal{P}_r$ donnée. Ses entrées/sorties sont les suivantes :

— **Entrées :**
  — $\mathcal{P}_r = (F_{\mathcal{N}}, X', Y')$: une propriété d'atteignabilité telle que :
    — $X' = \times_{i\in[\![1,p]\!]}[\underline{\mathbf{x}}'_i, \overline{\mathbf{x}}'_i]$ est un hyper-rectangle ;
    — Y' est de la même nature que l'ensemble Y' produit par le bloc *Translation i.e.,* décrit par une *conjonction de disjonctions* de contraintes d'inégalité linéaires : $Y' = \bigwedge_{i\in[\![1,m]\!]\setminus\mathcal{J}} \left(\bigvee_{j\in\mathcal{J}} \mathbf{y}'_{\pi^{-1}(c_j)} \leq \mathbf{y}'_{\pi^{-1}(c_i)}\right)$;

— `Verifier` ∈ {RELUPLEX/MARABOU, PLANET, RELUVAL, DEEPZ, DEEPPOLY } : le vérificateur à appeler.

— **Sorties :**

— `Output` ∈ {`true`, `false`, `unknown`} : la sortie du vérificateur choisi.

4.3.1 Cas des méthodes de falsification

Les méthodes de falsification *i.e.,* {RELUPLEX/MARABOU, PLANET } supposent que :

— X' est un polyèdre convexe ;

— Y' est le complémentaire d'un polyèdre.

Dans notre cas, X' est effectivement un polyèdre convexe (car c'est un hyper-rectangle), mais Y' n'est pas le complémentaire d'un polyèdre (qui s'écrit comme une disjonction de conjonctions de contraintes d'inégalités linéaires). Un effort supplémentaire est donc nécessaire afin d'utiliser les méthodes de falsification et les outils associés. Pour ce faire, nous nous intéressons à la négation de la propriété $\mathcal{P}_r$, négation qui satisfait le théorème suivant :

Théorème 2. *La négation de* $\mathcal{P}_r = (F_{\mathcal{N}}, X', Y')$ *satisfait l'équivalence :*

$$\neg\mathcal{P}_r \iff \bigvee_{i\in[\![1,m]\!]\setminus\mathcal{J}} \neg\mathcal{P}_r^i \tag{14}$$

où $\mathcal{P}_r^i = (F_{\mathcal{N}}, X', Y_i')$ *est une propriété d'atteignabilité sur* $F_{\mathcal{N}}$ *avec* Y_i' *tel que :*

$$Y_i' = \left\{ \mathbf{y}' \in \mathbb{R}^m \mid \bigvee_{j\in\mathcal{J}} \mathbf{y}'_{\pi^{-1}(c_j)} \leq \mathbf{y}'_{\pi^{-1}(c_i)} \right\} \tag{15}$$

L'ensemble Y_i' définissant chaque propriété $\mathcal{P}_r^i$ est maintenant le complémentaire d'un polyèdre, ce qui rend possible la vérification de $\mathcal{P}_r^i$ via les méthodes de falsification :

— si chaque propriété $\mathcal{P}_r^i$ est valide, alors $\bigvee_{i\in[\![1,m]\!]\setminus\mathcal{J}} \neg\mathcal{P}_r^i$ est fausse et la propriété qui nous intéresse en définitive $\mathcal{P}_r$ est valide ;

— sinon, et s'il existe un indice $i \in [\![1,m]\!]\setminus\mathcal{J}$ tel que $\mathcal{P}_r^i$ est fausse, alors $\mathcal{P}_r$ est fausse elle aussi.

Théorème 3. *La fonction* `solve` *est* correcte et complète *i.e.,* `output` ∈ {`true`, `false`} *et* `output` = `true` ⇒ $\mathcal{P}_r$ *est valide,* `output` = `false` ⇒ $\mathcal{P}_r$ *est non valide.*

4.3.2 Cas des méthodes d'analyse d'atteignabilité

Dans le cas des méthodes d'analyse d'atteignabilité {RELUVAL, DEEPZ, DEEPPOLY }, il est nécessaire que les ensembles X', Y' soient des hyper-rectangles, des zonotopes ou des polyèdres. Alors que l'ensemble X' est effectivement un hyper-rectangle, Y' n'est pas de cette nature. Encore une fois, une procédure particulière est nécessaire pour permettre l'utilisation des méthodes s'appuyant sur une analyse d'atteignabilité. Pour ce faire, l'analyse d'atteignabilité est utilisée afin de calculer un produit d'intervalles $F_{\mathcal{N}}^{\#}(X') = \times_{i\in[\![1,m]\!]}[\underline{\mathbf{y}}'_i, \overline{\mathbf{y}}'_i]$ représentant une sur-approximation des sorties atteignables depuis X', comme illustré en Figure 6.

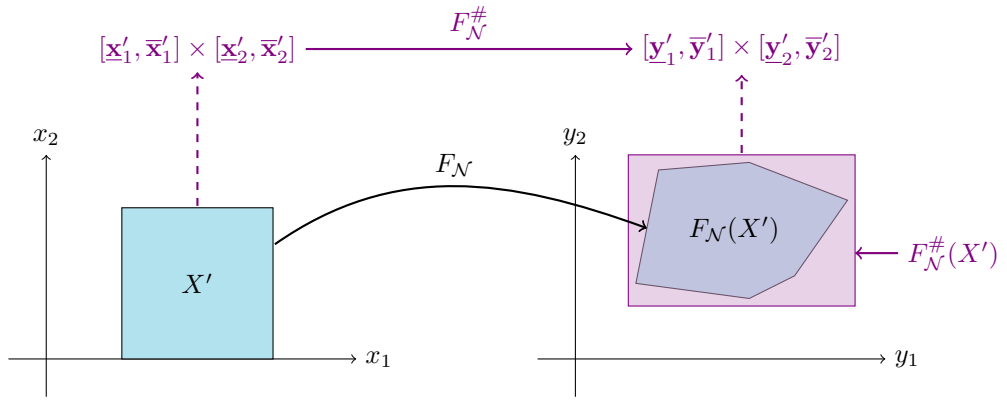

FIGURE 6: Approximation des sorties atteignables de $F_{\mathcal{N}}$

En s'appuyant sur l'élément $F^{\#}_{\mathcal{N}}(X')$ calculé par ce type de méthode, on peut ensuite raisonner sur la propriété $\mathcal{P}_r$ via le théorème suivant :

Théorème 4. *La propriété $\mathcal{P}_r = (F_{\mathcal{N}}, X', Y')$ est valide si :*

$$max_{j \in \mathcal{J}}(\overline{\mathbf{y}}'_{\pi^{-1}(c_j)}) \leq min_{i \in [\![1,m]\!] \setminus \mathcal{J}}(\underline{\mathbf{y}}'_{\pi^{-1}(c_i)}) \tag{16}$$

Théorème 5. *La fonction* `solve` *est* correcte *mais* pas complète *i.e.,* `output` $\in \{$`true`, `unknown`$\}$ *et* `output` $=$ `true` $\Rightarrow \mathcal{P}_r$ *est valide.*

4.4 Heuristiques

Il est possible soit de vérifier directement la propriété $\mathcal{P}_r$ produite par le bloc *Translation*, soit d'appeler des heuristiques afin d'accélérer la vérification. Ces heuristiques sont implantées dans une fonction `verify` avec les entrées et sorties suivantes :

- **Entrées :**
  - $\mathcal{P}_r = (F_{\mathcal{N}}, X', Y')$: la propriété produite par le bloc *Translation* ;
  - `config` $=<$ (`Verifier`$_1$, `Verifier`$_2$), `depth`$_{\max}$, `time` $>$: un ensemble de paramètres
    - `Verifier`$_1 \in \{$RELUVAL, DEEPZ, DEEPPOLY $\}$ un vérificateur correct s'appuyant sur une analyse d'atteignabilité ;
    - `Verifier`$_2 \in \{$RELUPLEX/MARABOU, PLANET $\}$ un vérificateur correct et complet s'appuyant sur la falsification ;
    - `depth`$_{\max} \in \mathbb{N}$ une profondeur maximale pour la stratégie *diviser pour mieux régner* (voir section 4.4.2) ;
    - `time` $\in \mathbb{R}$ un timeout pour la vérification de $\mathcal{P}_r$;
- **Sortie :**
  - `Result` $\in \{$`true`, `false`, `unknown`, `timeout`$\}$: une réponse au problème de vérification.

4.4.1 Heuristique #1 : Vérification hybride

Le but de cette première heuristique est de combiner les avantages de chacun des deux types de vérificateurs possibles :

(1) Dans un premier temps, la vérification de $\mathcal{P}_r$ est effectuée avec $\texttt{Verifier}_1$, réalisant une analyse d'atteignabilité par approximation des sorties atteignables. Si cette première tentative produit le résultat `true`, alors la vérification a pu être réalisée à moindre coût, du fait de l'approximation ;

(2) Dans la cas contraire *i.e.,* si l'étape (1) produit le résultat `unknown`, alors le vérificateur $\texttt{Verifier}_2$ est utilisé afin de savoir si la propriété est effectivement invalide ou si l'approximation est responsable du résultat `unknown`. Le coût plus élevé de cette seconde tentative n'intervient que si nécessaire.

4.4.2 Heuristique #2 : Diviser pour mieux régner (DnC)

La seconde heuristique a pour but d'améliorer la vérification via l'approximation des sorties atteignables avec $\texttt{Verifier}_1$. Comme démontré dans [17], plus l'ensemble d'entrée X' est petit, plus l'approximation produite par $\texttt{Verifier}_1$ est précise et donc plus les chances de vérifier la propriété $\mathcal{P}_r$ sont grandes. En conséquence, nous proposons une heuristique classique de division récursive de l'ensemble d'entrée X'. Tant que le vérification avec `Verifier1` ne retourne pas `true`, et tant que la profondeur maximale $\texttt{depth}_{\max}$ n'est pas atteinte, alors une bissection de X' est effectuée selon toutes les dimensions, produisant en nouvel ensemble de problèmes de vérification. Ceux-ci sont vérifiés par `Verifier1`.

5 Application à l'ACAS Xu

L'ACAS Xu est une *fonction* utilisée pour l'évitement de collision entre deux avions : un premier avion appelé *ownship* et un second avion appelé *intruder*. Dans la suite, ces deux avions sont supposés être dans le même plan horizontal c'est-à-dire à la même altitude. La fonction d'évitement de collision est alors une une fonction $h^*_{\text{ACAS Xu}}$ à 6 entrées et 1 sortie, définie comme la composition de tables logiques avec une interpolation par plus proches voisins. Plus précisément, l'unique sortie de $h^*_{\text{ACAS Xu}}$ est une *manœuvre d'évitement horizontale*, prise parmi cinq manœuvres possibles : *Clear-Of-Conflict* (COC), *Weak Left* (WL), *Weak Right* (WR), *Strong Left* (SL) et *Strong Right* (SR).

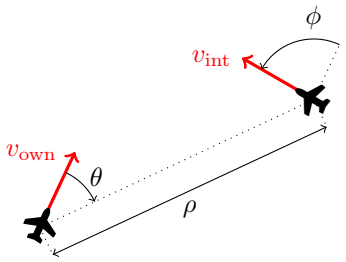

FIGURE 7: Les variables en entrée de la fonction $h^*_{\text{ACAS Xu}}$

Par ailleurs, les 6 entrées sont les suivantes :

— 5 variables décrivant la rencontre entre les deux avions du point de vue du ownship, (voir Figure 7), à savoir la distance ρ et l'angle θ représentants les coordonnées cylindriques de l'intruder relativement au ownship, le cap ϕ de l'intruder relatif au cap du ownship et les vitesses des deux avions v_{own} et v_{int}.
— la dernière manœuvre effectuée par l'ownship (parmi les cinq manœuvres possibles).

Dans une démarche de réduction de l'empreinte mémoire de la fonction $h^*_{\text{ACAS Xu}}$, rendue pertinente par le contexte embarqué du cas d'étude (la fonction d'évitement de collision a vocation à être embarquée à bord des avions avec une période de 1s), une version alternative a été proposée par [7]. Cette version alternative, notée $h_{\text{ACAS Xu}}$, a pour but d'approximer la fonction d'origine $h^*_{\text{ACAS Xu}}$. Elle est composée :

— de 5 classificateurs neuronaux $F^{\text{COC}}_{\mathcal{C}_t}, F^{\text{WL}}_{\mathcal{C}_t}, F^{\text{WR}}_{\mathcal{C}_t}, F^{\text{SL}}_{\mathcal{C}_t}, F^{\text{SR}}_{\mathcal{C}_t}$, où chaque classificateur a 5 entrées $(\rho, \theta, \phi, v_{\text{own}}, v_{\text{int}})$ et une sortie (la manœuvre d'évitement à réaliser) ;
— un mécanisme de *commutation* permettant de prendre en compte la sixième entrée *i.e.,* la dernière manœuvre effectuée par l'ownship : si cette manœuvre est $u_{\text{prev}} \in \{\text{COC}, \text{WL}, \text{WR}, \text{SL}, \text{SR}\}$, alors le classificateur $F^{u_{\text{prev}}}_{\mathcal{C}_t}$ est appelé sur les 5 autres entrées.

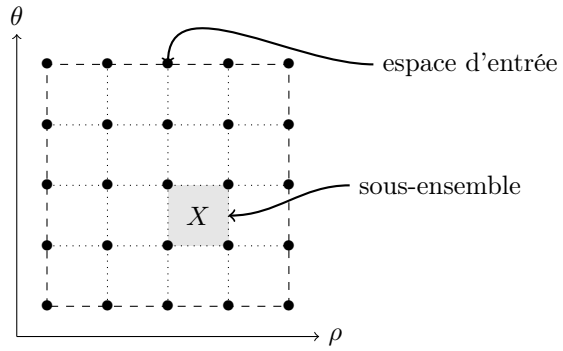

FIGURE 8: Illustration du partitionnement de l'espace d'entrée selon les entrées ρ et θ

Afin de démontrer que cette fonction alternative offre des garanties de sûreté similaires à la fonction d'origine $h^*_{\text{ACAS Xu}}$, les auteurs de [2] ont proposé l'approche suivante :

— en s'appuyant sur le maillage induit par les tables logiques de la fonction d'origine $h^*_{\text{ACAS Xu}}$, partitionner l'ensemble des entrées possibles des classificateurs neuronaux en une collection d'hyper-rectangles $\{X_k\}_k$, comme illustré en Figure 8 ;
— pour chaque hyper-rectangle X_k et pour chaque u_{prev}, montrer que l'ensemble des manœuvres en sortie de $F^{u_{\text{prev}}}_{\mathcal{C}_t}$ depuis X_k est inclus dans l'ensemble des manœuvres qu'aurait produit la fonction d'origine $h^*_{\text{ACAS Xu}}$, ce qui se traduit par la relation d'inclusion :

$$F^{u_{\text{prev}}}_{\mathcal{C}_t}(X_k) \subseteq h^*_{\text{ACAS Xu}}(X_k, u_{\text{prev}}) \tag{17}$$

L'Equation 17 est une propriété d'atteignabilité sur un classificateur neuronal, qui peut donc être vérifiée par SAM. Afin de réaliser cette vérification, nous avons considéré :

— deux types d'expériences :

  - **3D** : dans une première expérience, nous avons fixé les vitesses : $v_{\text{own}} = 438$ ft/s et $v_{\text{int}} = 414$ ft/s, ce qui revient à considérer que l'espace d'entrée des classificateurs neuronaux n'est plus que de dimension 3 (ρ, θ, ϕ). Cette hypothèse est pertinente dans la mesure où les avions en vol de croisière demeurent à une vitesse fixe ;
  - **5D** : dans une seconde expérience, nous avons considéré l'ensemble des valeurs possibles pour les 5 entrées des classificateurs neuronaux *i.e.,* un espace d'entrée 5D.
- trois types de classificateurs neuronaux, impliquant des réseaux de neurones de tailles similaires *i.e.,* avec une profondeur et une topologie proches :
  - `julian` : 5 classificateurs issus de la littérature et développés par [7] ;
  - `ducoffe` : 5 classificateurs développés par Mélanie Ducoffe, appris avec des techniques adverses dans le but de satisfaire la propriété 17 ;
  - `mamalet` : 5 classificateurs développés par Franck Mamalet, appris pour contraindre la constante de Lipschitz des réseaux de neurones à 1, dans le but de satisfaire la propriété 17.

5.1 Résultats 3D

En suivant l'approche de [2], l'espace d'entrée 3D des classificateurs neuronaux a été partitionné en **60 800** hyper-rectangles. La vérification de chaque classificateur conduit donc à la vérification de $5 \times$ **60 800** = **304 000** propriétés d'atteignabilité. Les résultats sont donnés en Table 1.

	temps de calcul	nombre de `true`	nombre de `false` ou `unknown`	taux de succès
`julian`	**5 jours**	254,670	49,330	**84%**
`ducoffe`	**17 heures**	286,023	17,977	**94%**
`mamalet`	**26 minutes**	280,000	24,000	**92%**

Table 1: Résultat de la vérification avec SAM en 3D

Ces résultats montrent que toutes les propriétés d'atteignablité ne sont pas satisfaites (taux de succès $< 100\%$) et ce quelque soit le type des classificateurs considérés. En revanche, les taux de succès sont relativement élevés, aux alentours de 90% et l'utilisation de notre outil permet d'identifier formellement les zones de l'espace d'entrée où les propriétés sont valides ou pas, ce qui représente une information utile du point de vue de l'utilisation pratique des classificateurs, permettant notamment le développement d'un contrôleur hybride comme décrit dans [2]. Par ailleurs, on note que les temps de vérification et taux de succès sont bien meilleurs pour les classificateurs ayant été entraînés dans le but de satisfaire la propriété considérée (`ducoffe` et `mamalet`).

5.2 Résultats 5D

De la même manière, l'espace d'entrée 5D est partitionné en **7 356 800** hyper-rectangles, conduisant à la vérification de $5 \times$ **7 356 800** = **36 784 000** propriétés d'atteignabilité. Les résultats sont donnés en Table 2.

En comparaison de l'expérience en 3D, du fait à la fois de la plus grande complexité des propriétés à vérifier (hyper-rectangles en entrée en 5D) et du fait du nombre plus important

	temps de calcul	nombre de `true`	nombre de `false` ou `unknown`	taux de succès
`julian`	-	-	-	-
`ducoffe`	**∼6 mois**	34 352 549	2 431 451	**93.4%**
`mamalet`	**∼1 mois**	34 173 698	2 610 302	**92.9%**

TABLE 2: Résultat de la vérification avec SAM en 5D

de propriétés à vérifier, les temps de vérification sont bien plus longs, et au delà d'un an pour les classificateurs `julian`. On retrouve les tendances de l'expérience en 3D en terme de temps de vérification (`julian` > `ducoffe` > `mamalet`) et de taux de succès (`mamalet` > `ducoffe`). On note aussi que si les performances relatives demeurent les mêmes, les taux de succès pour `ducoffe` et `mamalet` demeurent quasiment identiques, avec même une amélioration du taux de succès pour `mamalet`.

6 Conclusion

Dans cet article, nous avons proposé un outil capable de vérifier des propriétés d'atteignabilité sur des classificateurs neuronaux tout en s'appuyant sur les outils existants pour l'analyse de réseaux de neurones. Nous avons illustré l'applicabilité de cet outil à un cas d'usage aéronautique, l'ACAS Xu, où notre outil a permis d'étudier la sûreté de fonctionnement du système. En outre, nous avons observé que la prise en compte des propriétés à vérifier dès le développement des classificateurs neuronaux améliorait de façon significative à la fois le temps de vérification et le taux de preuve. Ce cas d'usage utilise néanmoins des classificateurs avec des réseaux relativement petits (quelques centaines de neurones, 5 entrées et 5 sorties). En termes de perspectives, il est intéressant d'étudier le passage à l'échelle de cet outil sur des classificateurs de plus grande complexité (par exemple YOLO), et l'éventuelle nécessité de nouvelles heuristiques pour traiter cette complexité accrue.

Références

[1] Stanley Bak, Changliu Liu, and Taylor T. Johnson. The Second International Verification of Neural Networks Competition (VNN-COMP 2021) : Summary and Results. *ArXiv*, abs/2109.00498, 2021.

[2] Mathieu Damour, Florence De Grancey, Christophe Gabreau, Adrien Gauffriau, Jean-Brice Ginestet, Alexandre Hervieu, Thomas Huraux, Claire Pagetti, Ludovic Ponsolle, and Arthur Clavière. Towards Certification of a Reduced Footprint ACAS-Xu System : A Hybrid ML-Based Solution. In Ibrahim Habli, Mark Sujan, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 34–48, Cham, 2021. Springer International Publishing.

[3] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

[4] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis*, 2017.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[6] T. Hickey, Q. Ju, and M. H. Van Emden. Interval arithmetic : From principles to implementation. *J. ACM*, 48(5) :1038–1068, sep 2001.

[7] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 42(3) :598–608, 2019.

[8] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex : An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[9] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. *Found. Trends Optim.*, 4(3–4) :244–404, feb 2021.

[10] Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Souradeep Dutta, Taylor J. Carpenter, Radoslav Ivanov, and Taylor T. Johnson. ARCH-COMP19 category report : Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systemsi, part of CPS-IoT Week 2019, Montreal, QC, Canada, April 15, 2019*, pages 103–119, 2020.

[11] Wei Ma and Jun Lu. An equivalence of fully connected layer and convolutional layer. *CoRR*, abs/1712.01252, 12 2017.

[12] Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. Adversarial robustness of deep neural networks : A survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2022.

[13] 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*, volume 31, pages 10802–10813. Curran Associates, Inc., 2018.

[14] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), 2019.

[15] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV : the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, pages 3–17, 2020.

[16] Caterina Urban and Antoine Miné. A review of formal methods applied to machine learning. *CoRR*, abs/2104.02466, 2021.

[17] Shiqi Wang, Kexin Pei, Justine Whitehouse, Junfeng Yang, and Suman Sekhar Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, 2018.

[18] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Sekhar Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown : Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *Neural Information Processing Systems*, 2021.

[19] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *CoRR*, abs/2208.05740, 2022.

[20] Huan Zhang, Shiqi Wang, Kaidi Xu, Yihan Wang, Suman Jana, Cho-Jui Hsieh, and Zico Kolter. A branch and bound framework for stronger adversarial attacks of ReLU networks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 26591–26604. PMLR, 17–23 Jul 2022.