Tolérance aux fautes pour détecter les comportements indésirables des réseaux de neurones

Kaoutar Rhazali, Benjamin Lussier, Walter Schön, Stéphane Géronimi

Tolérance aux fautes pour détecter les comportements indésirables des réseaux de neurones

Kaoutar Rhazali†*, Benjamin Lussier†, Walter Schön† et Stéphane Geronimi*
†Sorbonne Universités, Université de Technologie de Compiègne, CNRS, UMR 7253, Heudiasyc CS 60 319, 60203 Compiègne, France
Email : kaoutar.rhazali@hds.utc.fr, benjamin.lussier@hds.utc.fr, walter.schon@hds.utc.fr
*Groupe PSA, Direction de la recherche et de l'innovation automobile, Route de Gisy 78943 Vélizy Villacoublay Cedex, France
Email : kaoutar.rhazali@mpsa.com, stephane.geronimi@mpsa.com

***Résumé*—Suite au progrès fulgurant que les réseaux de neurones artificiels connaissent actuellement, leur implémentation s'étend à plusieurs domaines. En revanche, leur utilisation n'est pas autorisée dans les applications critiques car leur comportement est considéré comme imprévisible et non sûr. Dans cet article, nous présentons deux approches visant à apporter de la tolérance aux fautes logicielles dans les réseaux de neurones afin d'améliorer leur sécurité-innocuité. Notre objectif est de développer des réseaux de neurones capables de détecter les situations inconnues qui diffèrent de celles apprises. L'une des approches consiste à utiliser une redondance diversifiée au niveau des réseaux, et la deuxième consiste à ajouter une nouvelle sortie au réseau capable de reconnaitre les entrées aberrantes.**

I. Introduction

L'intelligence artificielle (IA) a connu ces dernières années un grand intérêt, que ce soit pour son application à l'autonomie des systèmes (comme les véhicules autonomes) ou le développement de réseau d'apprentissage profond ou *deep learning*. En particulier, les progrès réalisées dans le domaine de l'apprentissage automatique ou *Machine Learning*, ont donné une piste de recherche prometteuse et qui reflète très bien ce regain d'intérêt pour l'IA. Le principe du machine learning est de développer des algorithmes capables d'apprendre seuls à résoudre des problèmes au moyen d'expériences. Dans ce contexte, le joueur d'échecs virtuel est l'un des exemples les plus illustratifs marquant le succès du machine learning, où l'ordinateur a largement devancé les capacités des humains en termes de rapidité et de performance. Ce joueur apprend à anticiper tous les futurs mouvements possibles de l'adversaire afin de choisir celui qui sera meilleur pour lui, grâce à un modèle appris par apprentissage et une base de données de très grand nombre de coups.

Le *machine learning* englobe plusieurs techniques telles que les réseaux de neurones artificiels (RNA), les arbres de décisions, la régression linéaire, les forêts d'arbres décisionnels, les machines à vecteurs, etc. [1]. Dans cet article nous allons nous intéresser particulièrement aux réseaux de neurones artificiels, dont le principe de fonctionnement est inspiré de celui du neurone biologique. En effet, un réseau de neurones est composé d'un ensemble de noeuds interconnectés et qui sont capables de communiquer entre eux par l'envoi de signaux à travers des connexions pondérées. Cette technique a donné naissance à l'apprentissage profond ou "Deep Learning" [2], qui a révolutionné le domaine de l'apprentissage, notamment en termes de résolution de problèmes liés à la reconnaissance et à la prédiction. Ce type d'algorithmes, comme plusieurs autres, apprend à partir d'exemples connus et grâce à l'entraînement. En tenant compte des différences entre ses prédictions et les sorties correctes, l'algorithme accorde ses pondérations afin d'optimiser la précision de ses prédictions.

Bien que les RNA aient mené à d'excellents résultats dans divers domaines, leur usage est actuellement limité aux applications qui n'imposent pas d'exigences de sûreté de fonctionnement (SdF). En effet, à l'heure actuelle, nous n'arrivons pas à interpréter théoriquement les résultats obtenus par ces outils, ni à les justifier ou prédire leur comportement. De ce fait, sans possibilité de garantir leur comportement, ils ne peuvent pas être utilisés dans les applications critiques.

Dans cet article nous allons nous intéresser à l'étude de la sûreté de fonctionnement des RNA par l'application de méthodes de tolérance aux fautes afin d'améliorer leur sécurité-innocuité. L'intérêt de cette étude est d'analyser les faiblesses de ces réseaux d'une part, et de trouver des solutions potentielles qui permettront de détecter les sorties erronées dans le but de garantir un comportement correct lors du fonctionnement du RNA d'autre part.

La deuxième section, présente un état de l'art sur les RNA, la tolérance aux fautes, et la SdF dans les RNA. La troisième section décrit les approches proposées visant à améliorer la tolérance aux fautes dans ces algorithmes. La quatrième section met en évidence la question de diversification des réseaux, et la cinquième section met en application les approches proposées et expose les résultats obtenus. L'article se termine par une conclusion et des perspectives.

II. Etat de l'art

Dans cette section, nous présentons une étude bibliographique sur les principaux axes de notre étude. Tout d'abord, nous introduisons les réseaux de neurones, ensuite nous donnons un aperçu sur la tolérance aux fautes, et enfin nous exposons quelques travaux réalisés dans le domaine de la SdF des RNA.

A. Réseaux de neurones artificiels

Un réseau de neurones artificiels (RNA) est construit à partir de plusieurs neurones interconnectés sous forme de noeuds. Le premier neurone, appelé "perceptron", a été développé par F. Rosenblatt en 1957. Le perceptron a la particularité de

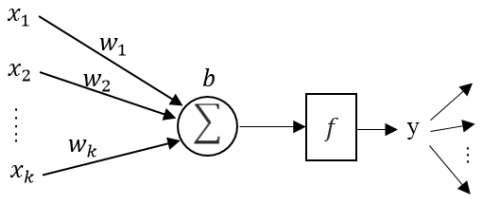

FIGURE 1. Représentation de l'architecture d'un neurone formel

ne prendre en entrée que des valeurs binaires et de générer une valeur binaire en sortie également, alors qu'un neurone ordinaire peut recevoir et produire des valeurs réelles. Dans un réseau, chaque neurone constitue un noeud, d'où vient l'appellation *Multi-Layer Percpetron* ou MLP qui est utilisée également comme référence aux réseaux de neurones même s'ils ne sont pas forcément composés de perceptrons. Le neurone peut être traduit mathématiquement par une fonction qui calcule l'image de la somme pondérée de ses entrées au moyen d'une fonction de transformation afin d'obtenir la sortie :

$$y = f(x_1, x_2, ..., x_k) = f(\sum_{i=1}^{k} w_i x_i + b)$$

La fonction de transformation est appelée fonction d'activation et elle peut avoir plusieurs formes (sigmoïde, tangente hyperbolique, ReLU, ...). La figure 1 donne une représentation graphique d'un neurone, les x_i représentent les entrées, les w_i les poids, b le biais, et f la fonction d'activation.

Un MLP est une succession de plusieurs couches de neurones, où chacun reçoit les données envoyées par tous les neurones de la couche précédente, effectue les calculs nécessaires, et envoie à son tour le résultat à tous les neurones de la couche suivante. Donc, pour construire un RNA, il faut au moins trois couches : une première couche appelée couche d'entrée, une deuxième couche appelée couche cachée, et une dernière couche appelée couche de sortie. Le nombre des couches cachées dans un réseau détermine sa profondeur. L'intérêt de celles-ci est de donner au réseau la capacité d'interpréter implicitement les données afin d'en tirer les caractéristiques (appelées *features*) nécessaires lui permettant de prévoir les sorties adéquates.

Lors de la phase d'entraînement, le réseau de neurones adapte ses paramètres (poids et biais) afin de produire des sorties égales aux résultats corrects. L'ajustement se fait en plusieurs passages. Chaque passage est réalisé en trois étapes : tout d'abord, le réseau calcule à partir d'un exemple connu sa valeur de sortie, ensuite au moyen d'une fonction de coût, il évalue l'erreur en comparant la sortie réelle avec celle qui a été calculée, et en dernière étape, il effectue un passage dans le sens inverse afin de corriger ses paramètres en utilisant une rétro-propagation (*Backpropagation*) [3], dans le but de minimiser la fonction de coût au moyen de l'algorithme *Gradient Descent*. Il existe d'autres algorithmes que le *Gradient Descent* mais qui sont moins répandus et plus complexes, tels que le *Conjugate Gradient* [4] et le *BFGS* [5].

La conception des réseaux de neurones est une discipline complexe : puisque le comportement des réseaux est difficile à prédire, les architectures sont principalement développées de façon empirique. Ainsi de nombreux chercheurs proposent différentes architectures, et plusieurs compétitions sont organisées pour les comparer tout au long de l'année. La plateforme web "kaggle.com" expose tous les concours planifiés. Parmi ceux-ci, la compétition annuelle de reconnaissance visuelle ILSVRC (ImageNet Large Scale Visual Recognition Challenge) [6] est la plus populaire.

Nous pouvons distinguer deux grandes catégories de réseaux de neurones selon leur topologie [7] : Les réseaux à propagation directe ou *Feed-forward Neural Networks*, et les réseaux récurrents, connus sous le nom RNN *Reccurent Neural Networks*. La deuxième catégorie dispose de connexions cycliques et elle est généralement utilisée dans la modélisation de systèmes dynamiques. La première catégorie est acyclique et possède seulement des connexions unidirectionnelles. Les approches modernes s'intéressent plus à ce type de RNA, vu leur puissance, leur polyvalence et leur simplicité. Dans notre travail, nous allons nous concentrer particulièrement sur les réseaux de neurones de la première catégorie, et plus précisément sur les MLP et les réseaux de convolution ou CNN (*Convolutional Neural Network*) [2].

Plusieurs éléments entrent dans la composition d'un réseau de neurones, notamment le nombre de couches cachées, le nombre de neurones dans chaque couche, le taux d'apprentissage, la fonction d'activation, la fonction de coût, et la méthode choisie pour l'initialisation des poids et des biais. Tous ces éléments peuvent impacter considérablement la performance et la robustesse du réseau, ainsi que sa capacité à converger vers la solution optimale. En revanche, il n'existe pas de processus de développement qui permette de déterminer le bon réglage de ces paramètres. Néanmoins, plusieurs chercheurs ont proposé des recommandations dans ce contexte. Dans [8], les auteurs introduisent un nouvel algorithme d'apprentissage appelé *greedy algorithm*, utilisé spécialement pour initialiser les poids et les biais des réseaux profonds DBN (*Deep Belief Network*). Il s'agit de pré-entraîner le réseau couche par couche au moyen d'un apprentissage non supervisé et de le raffiner par la suite d'une façon supervisée. Cette technique s'est étendue aux auto-encodeurs [9] et aux réseaux de convolutions [10], où elle a prouvé son efficacité. Dans [11] est proposée une méthode de réglage automatique du taux d'apprentissage, cette dernière ajustant ce paramètre suivant les variations locales du gradient au cours de l'entraînement.

La profondeur d'un réseau affecte positivement sa capacité d'apprentissage, cependant elle peut générer le problème de surapprentissage ou *overfitting*. Ce problème caractérise le cas où le réseau a de bons résultats sur l'échantillon d'entraînement alors qu'il n'arrive pas à généraliser sur les exemples de test. Parmi les méthodes proposées pour éviter ce phénomène, nous pouvons citer la technique de *dropout* introduite récemment dans [12] et évaluée dans [13], ainsi que les techniques de régularisation de type L1 et L2, ou la limitation de la taille du réseau. Nous proposons comme références sur le sujet d'optimisation des paramètres d'un réseau de neurones, les travaux suivants : [14] [15] [16] et [17].

B. Tolérance aux fautes

Notre objectif est d'élaborer des méthodes qui pourront rendre les réseaux de neurones tolérants aux fautes, ce qui

améliorera la SdF de ces mécanismes et permettra leur utilisation dans des applications critiques, comme les véhicules autonomes.

La tolérance aux fautes est l'un des moyens de SdF dont l'objectif est de diminuer la probabilité de défaillance malgré la présence éventuelle de fautes. Cette technique se fait généralement en deux grandes étapes : la détection d'erreur et le rétablissement du système [18]. En premier lieu, la détection d'erreur permet de provoquer la levée d'un signal ou d'un message d'erreur à l'intérieur du système lors de l'identification de la présence d'erreur, ensuite le rétablissement du système est déclenché par le signal ou le message pour substituer un état jugé exempt d'erreurs et de fautes.

Les formes de détection d'erreur les plus couramment utilisées sont les suivantes [19] :

- **Duplication et comparaison** : l'utilisation de plusieurs unités redondantes indépendantes fournissant le même service vis-à-vis du processus de création et d'activation de fautes.
- **Contrôle temporel** : l'utilisation de temporisateurs réglés pour vérifier si le temps de réponse ne dépasse pas un seuil prédéfini.
- **Contrôle de vraisemblance** : l'utilisation d'invariants à partir de règles de sécurité afin de vérifier la conformité des entrées et des sorties du système.

Le rétablissement se fait généralement en deux phases : le traitement d'erreur pour éliminer l'erreur et un traitement de faute pour prévenir la réactivation de la faute. Au niveau du traitement d'erreur, nous pouvons citer trois approches :

- **Reprise** : elle permet de ramener le système dans un état antérieur sauvegardé appelé point de reprise.
- **Poursuite** : c'est une approche alternative ou complémentaire à la reprise. Elle permet de chercher un nouvel état acceptable pour le système à partir duquel celui-ci pourra fonctionner.
- **Compensation** : elle est basée sur le principe d'utilisation d'une redondance suffisante de manière à masquer l'erreur.

C. La sûreté de fonctionnement dans les réseaux de neurones artificiels

Malgré les efforts déployés par les chercheurs de la communauté de l'IA, ceux-ci n'arrivent ni à comprendre ni à prédire naturellement le comportement des RNA. Cela est dû à la complexité de ces algorithmes liée d'une part à leur topologie qui implique généralement un grand nombre de paramètres, et d'autre part au comportement émergent de leur fonction finale. En effet, les développeurs peuvent réussir à entraîner un réseau de neurones à classer parfaitement le jeu de données de test sans comprendre comment il a adapté ses poids et ses biais, comment il a réparti son espace de recherche pour optimiser sa fonction finale, et sur quelles informations il s'est focalisé pour réussir son objectif. Ce manque d'information fait du réseau de neurones une boite noire impossible à déchiffrer. Actuellement, pour évaluer la performance d'un réseau de neurones, les développeurs se limitent à calculer quelques paramètres statistiques sur un échantillon de test, tels que le taux d'erreur, la précision, l'exactitude *accuracy*, ou le rappel *recall*. Cette évaluation reste superficielle car il est difficile de généraliser le comportement de ces mécanismes en se basant seulement sur un échantillon de test, dont la taille est généralement très limitée, et qui de plus, peut ne pas refléter toute la réalité, notamment lorsque la fonction est développée pour opérer dans un environnement ouvert. Par conséquent, on ignore complétement la réaction d'un réseau de neurones face à une entrée qui dévie très légèrement du jeu de données de l'apprentissage, voire une entrée inconnue.

Pour comprendre le comportement, seule une analyse mathématique exacte d'un réseau entraîné pourrait permettre de connaître comment il fonctionne afin de pouvoir le valider formellement par la suite. Cette analyse peut être très fastidieuse et complexe, voire quasi-impossible pour des réseaux profonds (les réseaux de convolutions, les réseaux récurrents, ...) vu le nombre important de paramètres qu'ils nécessitent. De plus, la difficulté de prédire le comportement de ces mécanismes rend leur vérification par test très difficile, car d'une part on ne peut pas garantir que deux entrées similaires aient la même sortie, et d'autre part les domaines d'entrées sont généralement très importants et impossibles à traiter par programmation impérative, ce qui justifie par ailleurs l'utilisation des réseaux de neurones pour résoudre le problème. Enfin, ce manque au niveau de la compréhension de ces algorithmes, fait que leur développement est principalement empirique.

Pour toutes ces raisons, l'utilisation des réseaux de neurones pour les applications critiques est actuellement fortement non recommandée dans la plupart des normes. Il est ainsi fondamental d'améliorer leur sécurité-innocuité pour permettre leur utilisation pour des applications où la programmation impérative ne peut apporter de solutions, comme la reconnaissance de situation ou la prise de décision dans un environnement ouvert. Dans cette perspective, notre but est de développer des approches visant à améliorer la sécurité-innocuité de ces algorithmes par le biais de la détection des comportements indésirables. Ce point sera détaillé dans la section III.

La littérature sur les réseaux de neurones est très abondante, en revanche, très peu de publications s'intéressent au sujet de la SdF de ces réseaux. Dans [20] est proposée une méthodologie basée sur la diversification pour favoriser la génération de réseaux de neurones fiables. La procédure peut se résumer en trois étapes. Tout d'abord, il faut produire des versions multiples de réseaux en faisant varier plusieurs paramètres (la structure du réseau, le nombre des nœuds cachés, les jeux de données, et l'initialisation des poids) et en utilisant deux types de réseaux : les MLP et les RBM (*Restricted Boltzmann Machines*). Ensuite, il faut choisir les réseaux performants par le biais d'une technique d'optimisation, et finalement, il faut rassembler les réseaux choisis pour constituer un seul réseau, dont la sortie est déterminée par vote majoritaire. Dans [21], les auteurs combinent les sorties de plusieurs réseaux en une seule sortie par le calcul de la moyenne simple ou la moyenne pondérée. Ensuite, après le calcul de l'erreur au niveau de la sortie commune, une rétro-progation est effectuée sur tous les réseaux. Les auteurs affirment que cette approche permet d'améliorer l'apprentissage, la généralisation et la tolérance aux fautes. Dans la même optique, en se basant sur le fait que

lorsque des programmes sont développés indépendamment, ils ont plus tendance à tomber en panne indépendamment, il est possible par la suite de renforcer la fiabilité. Le travail réalisé dans [22], étudie le concept de la création de réseaux divers, en faisant varier les paramètres initiaux, les données d'entraînement, et la nature des entrées. L'idée est de combiner les sorties de divers réseaux afin de produire une seule sortie correcte et fiable. Les auteurs ont comparé la corrélation entre les réseaux générés par chaque méthode afin d'évaluer la qualité de la diversification. L'utilisation de mesures différentes au niveau des entrées s'est avérée comme la technique la plus performante en matière de création de la diversité, alors que la méthode de variation des conditions initiales était la moins efficace. La référence [23] définit des critères de sécurité *Safety Criteria* qui pourraient contribuer à la justification de la sécurité-innocucité des réseaux de neurones. Les critères sont specifiés sous forme d'exigences de sécurité-innocuité, sur le comportement du réseau, qui doivent être respectées lors du développement des réseaux de neurones. L'article ne donne toutefois ni techniques ni explications sur la procédure de l'implémentation de ces exigences dans le réseau. Les mêmes auteurs, présentent dans [24] et [25], une méthodologie permettant d'améliorer la sécurité-innocuité du cycle de vie des RNA. L'objectif du cycle de vie introduit est de contrôler le comportement des réseaux de neurones et d'assurer un niveau acceptable de sécurité-innocuité. Ils introduisent un modèle de réseau de neurones basée sur la représentation de l'apprentissage sous forme symbolique. Ils divisent l'espace en deux zones, une zone de représentation symbolique et une zone de représentation neuronale. L'idée est d'insérer un problème symbolique dans un réseau de neurones, et d'extraire les connaissances sous forme symbolique après apprentissage. Ces papiers sont extrêmement théoriques, car les réseaux neuronaux symboliques sont des mécanismes extrêmement difficiles à concevoir, et les applications pratiques des réseaux de neurones utilisent quasi-unanimement une représentation valuée. Les deux références [26] et [27], présentent des processus de développement d'un RNA qui peuvent être utilisés dans des applications critiques. Le premier article répartit le processus en 5 étapes, et insiste sur le point de la mise en œuvre de documentations pour chaque étape afin de garder une traçabilité. Le deuxième article expose deux méthodes d'évaluation du réseau de neurones. L'une consiste à vérifier si le réseau de neurones a été développé de façon contrôlée et planifiée, et la seconde consiste à vérifier si le réseau a réussi les tests et s'il est conforme aux spécifications. Les étapes spécifiées dans ces deux références sont génériques et peuvent être considérées comme des ensembles de bonnes pratiques à suivre lors du développement d'un réseau de neurones, et bien que renforçant leur sûreté de fonctionnement par prévention des fautes, elles n'apportent pas de garanties quant à leur comportement. Une autre référence récente [28] traite le problème de la sécurité-innocuité dans l'apprentissage par renforcement, spécialement lors de l'application de ce type d'apprentissage pour la génération de stratégies de conduite à long terme. L'article met l'accent sur deux types de défis qui freinent le développement de la conduite autonome, à savoir la sécurité-innocuité fonctionnelle dans la politique de conduite et le comportement des obstacles (véhicules et piétons) qui est souvent imprévisible, ce qui rend le processus de décision markovien utilisé dans ce type d'apprentissage problématique. L'article décompose la politique de conduite en deux parties : une politique de désirs qui doit être apprise par expérience, et une politique de planification de trajectoire en utilisant des contraintes programmées de façon impératives. La seconde partie consiste à optimiser une fonction de coût définie par la polititique de désirs. Dans cet article, l'aspect de la sécurité-innocuité apparaît seulement lors de l'application des contraintes au niveau de la planification de trajectoire, pour une tâche qui, de notre point de vue, fait seulement appel à la résolution d'un problème d'optimisation et n'implique aucun algorithme d'apprentissage.

III. Tolérance aux fautes dans les réseaux de neurones

Les RNA ne sont généralement pas utilisés pour des applications critiques, en particulier du fait que les standards de sécurité-innocuité ont extrêmement limité les recommandations de leur utilisation pour de telles applications. Ces algorithmes peuvent en effet être utilisés seulement dans les applications exigeant le plus bas niveau d'intégrité de sécurité (SIL1) [24].

Les développeurs des réseaux de neurones s'intéressent généralement à la tolérance aux fautes en considérant les deux points suivants : leur aptitude à tolérer les fautes internes liées aux neurones, et leur aptitude à tolérer les fautes externes liées à la présence de données bruitées. Les architectures des RNA développées jusqu'à l'heure actuelle ont bien montré leur capacité à satisfaire ce deuxième point qui est lié à la caractéristique de robustesse (tolérance aux fautes externes) fondamentale dans le domaine de la robotique. Leur objectif principal est rigoureusement lié à l'amélioration de la précision, et par suite l'augmentation de la fiabilité sans tenir compte de l'aspect de la sécurité-innocuité, qui s'avère plus important quand ces réseaux sont développés pour être implémentés dans un système critique. Dans le présent article nous étudions la tolérance aux fautes des RNA pour permettre de détecter quand le réseau neuronal atteint ou dépasse les limites de son domaine de bon fonctionnement et risque de juger des entrées pour lesquelles il n'a pas été entraîné. Autrement dit, nous cherchons à rendre le RNA capable de différencier entre les cas qu'il a déjà vus et les cas qu'il n'a jamais vus afin d'éviter une mauvaise interprétation par la suite, ce qui peut engendrer de graves conséquences. Ainsi, notre première préoccupation est d'assurer la sécurité-innocuité au sein de ces réseaux, plutôt que la fiabilité.

Nous proposons deux approches : l'une qui vise l'utilisation de la redondance diversifiée en faisant appel à plusieurs réseaux de neurones développés indépendamment, et une autre qui consiste à intégrer dans le réseau une sortie supplémentaire, afin de séparer les cas ordinaires et les cas aberrants. Certes, la redondance diversifiée est très présente dans la littérature comme nous avons pu le constater dans la section II-C, mais dans les références mentionnées, cette technique a été utilisée spécialement dans le but d'améliorer la précision, ce qui permet d'assurer le critère de fiabilité, mais aucun de ces articles n'a abordé la problématique de détection d'entrées hors des spécifications et jeux d'entraînement, pour lesquelles le système ne serait pas capable de décider correctement.

Notre première approche est basée sur l'idée que des réseaux de neurones développés indépendamment n'apprennent

pas forcément de la même manière. Nous visons à créer des réseaux divers capables d'agir différemment envers une nouvelle situation non apprise. En effet, lors de la phase d'apprentissage chaque réseau trace les limites de ses classes en essayant de trouver la solution optimale, et comme la dimension des variables d'entrée est très large, les différents réseaux sont généralement figés dans des fonctions et des optima locaux différents. Par exemple, la classification d'images en blanc et noir de taille 28*28 pixels, nécessite un espace de recherche à 256 dimensions où chaque variable peut avoir des valeurs comprises entre 0 et 255, ce qui fait au total 256^{256} entrées possibles. Ces réseaux divers ne vont pas avoir forcément le même comportement envers un exemple qu'ils n'ont pas appris, car tout simplement chacun d'eux s'est focalisé sur l'optimisation de sa sortie en tenant compte seulement des données d'entraînement, et en corrigeant les paramètres de son architecture propre.

En considérant que les réseaux ont été bien entraînés, leurs sorties doivent être similaires dans le cas d'une entrée qui ressemble aux classes considérées. Cela nous permettra de détecter une situation aberrante dans le cas où les sorties sont en désaccord et de l'isoler afin d'empêcher le système de l'interpréter de manière incorrecte. Des méthodes de rétablissement peuvent être implémentées selon le type de l'application. L'architecture du réseau global est présentée dans la figure 2. Il faut noter que cette approche ne donne aucune garantie sur le bon discernement d'entrées hors du domaine de spécification du réseau. En effet, rien ne garantit que des réseaux apparemment divers sur des échantillons utilisés pour leur validation auront un comportement différent face à la même entrée aberrante. Cependant, même sans garantie, cette approche permet certainement d'améliorer la tolérance aux fautes des réseaux neuronaux, et la combiner avec d'autres techniques pourrait permettre d'atteindre un niveau de sûreté de fonctionnement suffisant pour leur utilisation dans des systèmes critiques.

Dans notre deuxième approche, nous souhaitons entraîner un réseau de neurones qui possède, en plus de ses sorties ordinaires, une nouvelle sortie "Inconnue" (déterminant une classe d'entrée inconnue pour lui), capable de détecter les cas aberrants. Ce réseau doit être entraîné avec une nouvelle base de données avec un échantillon contenant des exemples qui sont diversifiés et qui ne ressemblent pas aux autres classes afin de favoriser l'apprentissage des cas aberrants. La figure 3 illustre cette approche. Il faut veiller à avoir des classes équilibrées, c'est-à-dire de même taille afin d'éviter le problème de surapprentissage. Cette technique n'apporte pas non plus de garantie sur le bon classement d'une entrée aberrante dans la catégorie inconnue. En effet, le comportement imprédictible du réseau neuronal fait qu'on ne peut pas prévoir le résultat d'une entrée. Elle peut cependant, de façon complémentaire à la première, améliorer la tolérance aux fautes du système.

En particulier, [29] laisse penser qu'il est impossible de garantir un bon classement de toutes les entrées possibles d'un réseau neuronal. Au moyen d'algorithmes évolutionnaires, les auteurs génèrent des images qui sont complètement non reconnaissables par l'humain tandis que le réseau de neurones réussit à les classer erronément avec une confiance qui va au-delà de 99%. Le papier démontre également que le réentraînement du réseau avec les images non reconnaissables n'a pas empêché le

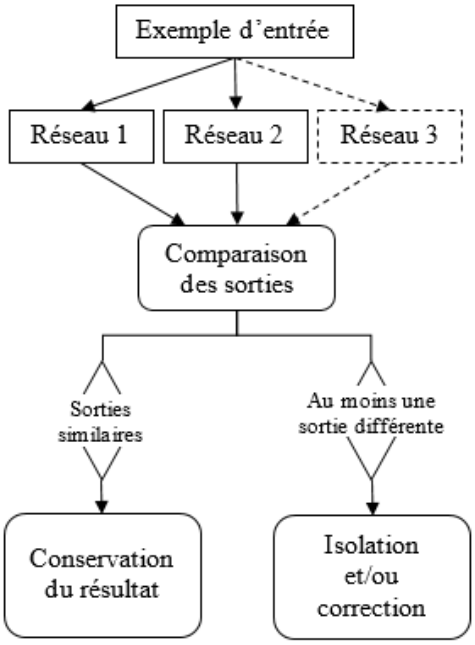

FIGURE 2. Approche de diversification des réseaux pour la détection des situations aberrantes

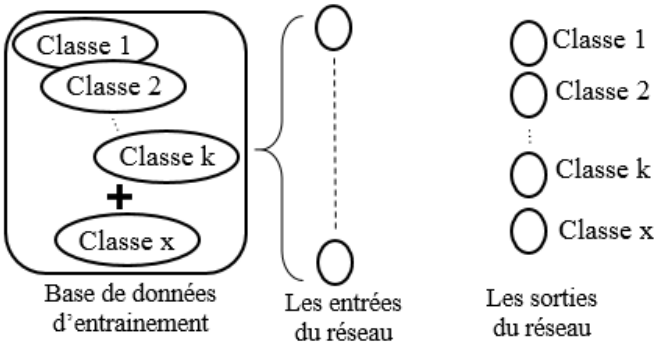

FIGURE 3. Un réseau de neurones avec une sortie supplémentaire. La classe x refère à la classe "inconnue". Le réseau doit être entraîné avec une nouvelle base de données contenant des exemples diversifiés affectés à la classe x.

réseau de se tromper pour d'autres images. Cela suggère donc que le comportement des réseaux de neurones est impossible à garantir et peut être aisément dupé. Cependant, l'utilisation de réseaux diversifiés peut aider à résoudre ce problème puisque chaque réseau a une fonction différente, et potentiellement un comportement différent face aux entrées aberrantes.

IV. DIVERSIFICATION DES RÉSEAUX DE NEURONES

Pour que la première approche de tolérance aux fautes proposée dans la section précédente soit efficace, il est important que les RNA utilisés soient bien diversifiés. La diversification peut être obtenue en jouant sur plusieurs éléments. Suite aux expériences que nous avons réalisées, il s'est avéré que la variation de l'ordre dans lequel le réseau utilise les données d'entraînement semble l'aspect le plus efficace pour générer des réseaux diversifiés. Nous avons également testé d'autres techniques qui ont été mentionnées dans [20] et [22], telles que la variation de l'état initial du réseau, de sa structure

(nombre de nœuds et nombre de couches), de son architecture (MLP, réseau de convolution, ...), et de l'échantillon d'apprentissage. Ces techniques peuvent également être efficaces, mais rien n'empêche deux réseaux d'architecture différentes de développer la même fonction, ou des fonctions très proches. Une technique qui parait efficace, et qui est utilisée souvent par les développeurs des réseaux de neurones dans le but de renforcer leurs réseaux à s'adapter aux petites variations est l'utilisation d'images augmentées. Cette méthode a montré son efficacité en terme d'amélioration de la performance des réseaux de neurones [30]. Elle consiste à altérer légèrement les images de chaque batch avant de les injecter dans le réseau. Une fois choisie pour l'entraînement, l'image subit des modifications suivant des degrés d'altération qui sont déterminés aléatoirement entre des valeurs min et max. Les modifications peuvent être réalisées en variant plusieurs paramètres : la luminosité, le contraste, la rotation, l'orientation, le recadrage, la netteté, etc. Dans la même optique, nous pouvons privilégier la diversification en s'inspirant des deux techniques : le *boosting* et le *bagging* [31] [32], de telle manière à attribuer des poids aux exemples d'entraînement. En effet, l'algorithme de *boosting* se base sur la performance de ses classificateurs pour générer les distributions de probabilités sur sa base de données alors que l'algorithme de *bagging* ne tient pas compte de la performance.

Plusieurs obstacles apparaissent quand on veut comparer ces méthodes de façon objective. Tout d'abord, le processus de développement des réseaux neuronaux étant encore empirique, l'influence de ces différents facteurs sur la fonction finale est mal comprise, et la diversification que nous allons obtenir va ainsi être très aléatoire. De plus, il est nécessaire de pouvoir mesurer la diversification entre deux réseaux, ce qui n'est pas un problème trivial. En effet, la diversification ne consiste pas seulement à avoir des réseaux qui n'ont pas les mêmes poids ou qui n'ont pas la même structure, mais plutôt à produire des réseaux dont la fonction de traitement des entrées est différente. Or, à moins d'identifier mathématiquement la fonction d'un réseau en effectuant la combinaison des différents neurones un à un, il n'est pas possible d'avoir une comparaison précise. En effet, une différence entre deux architectures ne garantit pas une différence entre leurs fonctions, et une comparaison sur les sorties ne pourra être que parcellaire considérant la taille considérable du domaine d'entrée.

La technique de déconvolution *Deconvolutional Network* introduite récemment dans [33] permet de visualiser les caractéristiques (ou *features*) apprises au niveau de chaque couche du réseau de neurones, et particulièrement les informations apprises par les filtres des réseaux de convolution. Cela peut permettre de décomposer la fonction complète d'un réseau en différentes caractéristiques, qui pourraient être plus faciles à comparer. Nous identifions deux autres approches qui peuvent être utilisées dans la mesure de la diversité, où l'une est basée sur la comparaison des gradients des réseaux, et l'autre utilise malgré les inconvénients présentés la comparaison des sorties pour de mêmes entrées données. En ce qui concerne la première approche, chaque sortie y_j d'un réseau R_k peut être exprimée par une fonction f_{kj} qui prend en arguments les entrées x_i du réseau :

$$y_j = f_{kj}(x_1, x_2, ..., x_n)$$

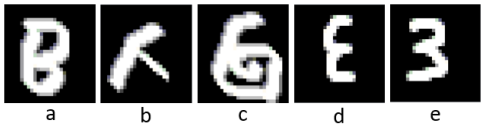

FIGURE 4. Les images aberrantes utilisées dans l'évaluation de l'approche de diversification

Alors pour chaque sortie y_j et pour chaque réseau R_k, il est possible de calculer un vecteur de gradient V_{kj} :

$$V_{kj} = (\frac{\partial f_{kj}}{\partial x_1}, \frac{\partial f_{kj}}{\partial x_2}, ..., \frac{\partial f_{kj}}{\partial x_n})$$

Les vecteurs de gradient nous donneront aussi une idée sur les entrées qui stimulent le plus une sortie donnée au niveau de chaque réseau, et par la suite nous pourrions estimer la diversification quantitativement. La deuxième approche est plus facile et rapide à mettre en oeuvre, et vise à étudier la corrélation entre les différents réseaux générés en faisant des tests sur plusieurs échantillons contenant des entrées diverses, aberrantes ou non. Les réseaux qui ont tendance à classer le plus d'exemples de la même manière, sont jugés alors faiblement diversifiés. Dans notre application, nous allons utiliser la deuxième approche vu sa simplicité et malgré sa subjectivité en regard de l'échantillon d'entrée testé.

V. APPLICATION

Dans cette application, nous utilisons la base de données MNIST [34] qui contient 50 000 images de chiffres manuscrits pour l'entraînement, 10 000 pour la validation, et 10 000 pour le test. Les images sont en blanc et noir et ont une taille de 28*28 pixels. Nous exploitons cette base de données pour entraîner plusieurs réseaux qui vont être utilisés par la suite dans l'évaluation de nos deux approches de tolérance aux fautes. Dans la première expérience, nous allons essayer de générer des réseaux diversifiés en jouant sur trois paramètres : l'ordre d'apprentissage (l'ordre dans lequel le réseau utilise les données lors de la phase d'entraînement), les paramètres initiaux, et la structure. Dans la deuxième expérience, nous allons entraîner un seul réseau avec une sortie supplémentaire pour la détection des entrées inconnues. Le comportement de l'ensemble des réseaux générés sera testé sur un échantillon de 5 images aberrantes, illustrées dans la figure 4.

A. Approche de diversification des réseaux de neurones

Nous avons produit 8 réseaux différents notés de $R1$ à $R8$. Tous les réseaux ont une architecture MLP avec une seule couche cachée de 30 neurones sauf le réseau $R8$ qui dispose de 60 neurones au niveau de sa couche cachée. Les réseaux sont générés comme suit :

- les réseaux $R1$ et $R2$ sont entraînés avec des biais et des poids initiaux différents et un ordre d'apprentissage différent. En effet, les paramètres du réseau $R1$ (b_1, w_1) ont été définis aléatoirement et ont contribué à la génération des paramètres du réseau $R2$ (b_2, w_2) suivant les équations :
$$b_2 = 1 - |b_1| \text{ et } w_2 = 1 - |w_1|$$
- les réseaux $R3$ et $R4$ sont entraînés avec des paramètres initiaux différents et un ordre d'apprentissage

TABLE I. LES CLASSES ATTRIBUÉES AUX IMAGES ABERRANTES PAR LES 8 RÉSEAUX GÉNÉRÉS ET LEURS TAUX DE CONFIANCE.

		Référence de l'image				
Réseau	**Taux d'erreur**	**a**	**b**	**c**	**d**	**e**
$R1$	4.71%	8 (99.91%)	0 (99.93%)	6 (98.94%)	2 (91.99%)	3 (98.79%)
$R2$	7.08%	8 (97.98%)	0 (94.53%)	6 (98.53%)	6 (56.01%)	3 (93.24%)
$R3$	5.11%	8 (99.98%)	0 (53.64%)	8 (99.08%)	8 (99.24%)	3 (98.82%)
$R4$	6.87%	8 (98.61%)	2 (53.13%)	2 (64.15%)	6 (99.54%)	3 (99.11%)
$R5$	4.54%	8 (99.51%)	2 (86.48%)	6 (99.97%)	5 (83.39%)	3 (99.98%)
$R6$	4.55%	8 (99.51%)	0 (99,91%)	6 (69.83%)	6 (88.76%)	3 (99.99%)
$R7$	4.93%	8 (99.77%)	0 (99.96%)	5 (68.19%)	8 (63.57%)	3 (99.88%)
$R8$	21.96%	8 99.98%	0 (98,43%)	6 (97.80%)	6 (95.31%)	3 (99.85%)

fixe. Les paramètres initiaux ont été générés de la même manière que dans les réseaux $R1$ et $R2$.

- les réseaux $R5$ et $R6$ sont entraînés avec les mêmes paramètres initiaux générés aléatoirement, et un ordre d'apprentissage différent.

- les réseaux $R7$ et $R8$ ont des structures différentes. Le $R7$ dispose d'une couche cachée de 30 neurones et le $R8$ dispose d'une couche cachée de 60 neurones. Ils sont entraînés avec des paramètres initiaux générés aléatoirement, et un ordre d'apprentissage différent.

Le tableau I présente les classes affectées aux images aberrantes par les 8 réseaux et leurs taux de confiance. Les taux d'erreur relatifs à chaque réseau sont calculés sur l'échantillon de test de 10 000 images et sont également exposés dans le tableau.

Nous constatons que tous les réseaux ont affecté l'image "a" à la classe "8" et l'image "e" à la classe "3" avec une grande confiance, car effectivement ces deux images ressemblent beaucoup aux chiffres "8" et "3". Ce résultat affirme la capacité des réseaux de neurones à s'adapter aux petits changements. Cependant, les autres images ont été classées différemment par la majorité des réseaux, ce qui confirme l'hypothèse que les réseaux de neurones diversifiés peuvent avoir des résultats différents pour la même entrée hors de leur spécification.

L'image "c" a été détectée comme un chiffre 6 par 4 réseaux. En regardant l'image, nous pouvons comprendre ce comportement vu la ressemblance qui existe. Quant à l'image "b", la majorité des réseaux l'ont affecté à la classe 0, ce qui affirme l'imprévisibilité de leur comportement et la limite de cette première approche, que nous avons identifiée dans la section III. En revanche, l'image "d" a été affectée à des classes différentes par tous les réseaux en les comparant deux par deux. Selon cet échantillon, les réseaux qui paraissent plus diversifiés sont les réseaux $R3$ et $R4$, du fait qu'ils ont toujours des points de vues différents envers les images "b", "c" et "d".

Ces expériences montre que le changement d'un seul paramètre peut impacter d'une manière significative l'apprentissage des neurones et par la suite établir de la diversification.

TABLE II. LES CLASSES ATTRIBUÉES AUX IMAGES ABERRANTES PAR LE RÉSEAU ENTRAINÉ AVEC LA CLASSE "INCONNUE".

Référence de l'image	**a**	**b**	**c**	**d**	**e**
Classe	Inconnue (100%)	Inconnue (99.01%)	Inconnue (100%)	Inconnue (93.45%)	3 (99.92%)

B. Approche d'utilisation de la classe "Inconnue"

Pour entraîner la classe "Inconnue", il faut ajouter un jeu de données diversifié contenant au moins 6000 images afin que les classes soient équilibrées et pour ne pas biaiser les résultats. En raison de manque de temps et de données suffisantes pour générer les exemples de cette classe, nous avons entraîné un réseau de neurones avec la même base de données MNIST mais avec seulement 6 sorties au lieu de 10, étiquetées comme suit : "0", "1", "2", "3", "4" et "Inconnue". En effet, nous avons gardé les données des 5 premières classes (les données des chiffres manuscrits allant de "0" à "4"), et nous avons extrait 7000 images parmi celles des classes restantes (les données des chiffres manuscrits allant de "5" à "9") pour générer l'ensemble d'entraînement de la classe "Inconnue". Le réseau entraîné a une architecture MLP avec une seule couche cachée de 30 neurones. Le tableau II présente les classes attribuées aux 5 images aberrantes par ce réseau, et leurs taux de confiance.

Nous remarquons que le réseau a classé les images "a", "b", "c" et "d" comme "Inconnue", et il a affecté l'image "e" à la classe "3" avec une grande confiance qui va au-delà de 99% pour la majorité des images. Effectivement, le réseau a utilisé les données d'entraînement du chiffre "8" pour entraîner sa classe "Inconnue", ce qui justifie l'attribution de l'image "a" à cette classe. La même chose pour l'image "c" qui a été classée comme "Inconnue" peut être parce que le réseau n'a pas été entraîné sur le chiffre "6". Quant à l'image "e", elle a été reconnue comme le chiffre "3" car tout simplement le réseau a été entraîné avec cette classe. En ce qui concerne les autres images ("b" et "d"), nous constatons que le réseau les a bien détectées comme aberrantes en les mettant dans la classe "Inconnue". En effet, malgré l'entraînement du réseau avec une classe "Inconnue" peu diversifiée, ce dernier a pu généraliser sa fonction en reconnaissant les images qui ne ressemblent pas à ces classes principales comme aberrantes.

Ces premiers résultats sont encourageants, et ils soutiennent d'une manière significative nos deux approches : l'approche visant à produire des réseaux de neurones diversifiés capables d'agir différemment envers des jeux de tests inconnus, et l'approche visant à générer un réseau de neurones avec une sortie supplémentaire pour classer les entrées qui différent de ces classes ordinaires. Nous sommes en train d'appliquer cette technique sur des réseaux profonds, tout en essayant d'identifier les techniques de diversification les plus efficaces.

VI. CONCLUSION

Bien que les réseaux de neurones aient prouvé leur efficacité dans plusieurs domaines, leur utilisation reste limitée aux applications peu critiques. Cela est dû tout simplement au manque de méthodologie et d'outils qui peuvent justifier et valider leur SdF. Dans ce contexte, nous avons établi deux approches dans le but de renforcer la SdF de ces algorithmes. Notre principal objectif est de permettre à ces algorithmes de tolérer les fautes afin d'améliorer leur sécurité-innocuité. Les

méthodes proposées permettent de détecter les entrées non apprises par le réseau de neurones dans le but d'éviter les mauvaises interprétations par ce dernier. La première approche consiste à utiliser une redondance diversifiée par la génération de plusieurs RNA. En effet, les réseaux de neurones qui n'ont pas subi le même entraînement ont tendance à converger vers des optima locaux différents, vu la grande taille de l'espace de recherche. Par conséquent, ces réseaux peuvent générer des comportements divers envers une situation non apprise. Notre approche exploite cette spécificité afin de reconnaître les entrées aberrantes dans le cas où les versions diversifiées sont en désaccord. Notre deuxième approche consiste à utiliser un seul réseau de neurones en rajoutant à ses sorties, une nouvelle sortie supplémentaire qui devrait s'entraîner à classer les cas non reconnus. Pour améliorer son efficacité, il faut insérer dans son jeu de données d'apprentissage des exemples bien diversifiés et étiquetés comme situations inconnues. D'après les expériences réalisées, il s'est avéré que cette technique permet au réseau de classer les exemples qui n'appartiennent à aucune de ses classes ordinaires dans la nouvelle classe étiquetée "Inconnue".

L'efficacité de la première approche repose significativement sur la qualité de la diversification : plus la diversification est importante entre les réseaux, plus ils sont capables d'avoir des comportements différents envers les exemples inconnus. A cet effet, nous avons proposé plusieurs techniques permettant de favoriser la diversification entre réseaux, telles que la variation des paramètres des réseaux, l'utilisation des images augmentées, et l'attribution des poids aux exemples d'entraînement.

Les résultats des expériences réalisées sur la base de données MNIST sont encourageants et ils nous poussent à tester nos approches sur d'autres jeux de données et d'essayer d'identifier les méthodes de diversification les plus efficaces. Nous sommes également en train d'appliquer ces techniques au domaine de l'apprentissage profond.

RÉFÉRENCES

[1] Tom M Mitchell et al. Machine learning. wcb, 1997.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http ://www.deeplearningbook.org.

[3] DRGHR Williams and GE Hinton. Learning representations by back-propagating errors. *Nature*, 323(6088) :533–538, 1986.

[4] Erik M Johansson, Farid U Dowla, and Dennis M Goodman. Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. *International Journal of Neural Systems*, 2(04) :291–301, 1991.

[5] Roberto Battiti and Francesco Masulli. Bfgs optimization for faster and automated supervised learning. In *International neural network conference*, pages 757–760. Springer, 1990.

[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3) :211–252, 2015.

[7] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks : A tutorial. *Computer*, 29(3) :31–44, 1996.

[8] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7) :1527–1554, 2006.

[9] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks.

[10] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 1137–1144, Cambridge, MA, USA, 2006. MIT Press.

[11] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. *ICML (3)*, 28 :343–351, 2013.

[12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv :1207.0580*, 2012.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb) :281–305, 2012.

[15] S Sathiya Keerthi, Vikas Sindhwani, and Olivier Chapelle. An efficient method for gradient-based adaptation of hyperparameters in svm models. *Advances in neural information processing systems*, 19 :673, 2007.

[16] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks : Tricks of the trade*, pages 437–478. Springer, 2012.

[17] Y Lecun, L Bottou, GB Orr, and K-R Müller. Efficient backprop. *Lecture notes in computer science*, pages 9–50, 1998.

[18] Algirdas Avižienis, Jean-Claude Laprie, and Brian Randell. Dependability and its threats : a taxonomy. In *Building the Information Society*, pages 91–120. Springer, 2004.

[19] Benjamin Lussier. *Tolérance aux fautes dans les systèmes autonomes*. PhD thesis, Institut National Polytechnique de Toulouse, 2007.

[20] Derek Partridge and William B Yates. Engineering multiversion neural-net systems. *Neural Computation*, 8(4) :869–893, 1996.

[21] William P Lincoln and Josef Skrzypek. Synergy of clustering multiple back propagation networks. In *NIPS*, pages 650–657, 1989.

[22] AJC Sharkey, NE Sharkey, and OC Gopinath. Diversity, neural nets and safety critical applications. *Current trends in connectionism*, pages 165–178, 1995.

[23] Zeshan Kurd and Tim Kelly. Establishing safety criteria for artificial neural networks. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 163–169. Springer, 2003.

[24] Zeshan Kurd, Tim Kelly, and Jim Austin. Safety criteria and safety lifecycle for artificial neural networks. In *Proc. of Eunite*, volume 2003, 2003.

[25] Zeshan Kurd and Tim Kelly. Safety lifecycle for developing safety critical artificial neural networks. In *International Conference on Computer Safety, Reliability, and Security*, pages 77–91. Springer, 2003.

[26] David M Rodvold. A software development process model for artificial neural networks in critical applications. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 5, pages 3317–3322. IEEE, 1999.

[27] Ian T Nabney, Mickael JS Paven, Richard C Eldridge, and Clive Lee. Practical assessment of neural network applications. In *Safe Comp 97*, pages 357–368. Springer, 1997.

[28] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv :1610.03295*, 2016.

[29] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled : High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[30] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962. Citeseer, 2003.

[31] Robert E Schapire. A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406, 1999.

[32] Eric Bauer and Ron Kohavi. An empirical comparison of voting

classification algorithms : Bagging, boosting, and variants. *Machine learning*, 36(1) :105–139, 1999.

[33] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[34] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.