Certification formelle des réseaux neuronaux profonds : un état de l'art en 2019

Hugo Bazille[1], Eric Fabre[1], and Blaise Genest[2]

[1] Univ Rennes, INRIA, Rennes, France
hugo.bazille@inria.fr, eric.fabre@inria.fr
[2] Univ Rennes, CNRS, IRISA, Rennes, France
blaise.genest@irisa.fr

Résumé Les réseaux neuronaux profonds, *DNNs* pour deep neural networks en anglais, sont tout aussi efficaces dans leurs tâches respectives que difficilement compréhensibles par un humain. Pour pouvoir les utiliser dans des applications critiques, à défaut d'être compris, ils doivent être certifiés. Nous allons passer en revue un grand nombre de travaux récents visant à certifier formellement des réseaux neuronaux profonds obtenus par apprentissage automatique profond. La plupart des travaux actuels se concentrent sur des réseaux à propagations avant (feed-forward), et le problème de certifier leur robustesse.

1 Introduction

Apprendre à une machine à résoudre un problème à partir d'exemples est un outil très puissant qui a de nombreuses applications. Ces dernières années, la révolution due à *l'apprentissage profond* a produit automatiquement des modèles, appelés réseaux neuronaux profonds, *DNN* pour deep neural networks en anglais, dans de nombreux champs différents (classification d'images, reconnaissance de la langue naturelle, traduction automatique, mais aussi conception de lois de contrôle, par exemple pour des drones ...). Alors que ces DNNs produisent des résultats excellents dans presque tous les cas, il arrive de les voir pris en défaut, ce qui a eu des conséquences parfois dramatiques (voir l'accident des voitures autonomes UBER, etc). Pire, il est souvent facile de trouver des attaques contre ces DNNs, en modifiant imperceptiblement les entrées [20]. Par exemple, on peut trouver des attaques contre des réseaux contrôlant une voiture [14], dans le domaine de la compréhension de texte [12] ou audio [4] ... Un consensus s'est ainsi formé autour de la nécessité d'analyser les DNNs produits par les algorithmes d'apprentissage profond, afin de pouvoir les certifier de manière formelle.

La difficulté la plus importante pour la certification des DNNs est qu'ils sont très difficiles à être compris par un humain : si l'on peut comprendre le raisonnement derrière l'utilisation des différents types de couches d'un DNN (qui sont d'ailleurs conçues par des experts), les poids sur les arcs (qui sont obtenus de manière automatique) ne font que peu de sens. En outre, le très grand nombre des neurones d'un DNN rend encore plus difficile son étude. Cela a

donné naissance au concept d'explicabilité en Intelligence Artificielle : on aimerait au minimum comprendre pourquoi un DNN renvoie un certain résultat, par exemple en mettant en évidence quelques patterns de l'entrée (des pixels d'une image ...) qui ont eu la plus grande contribution à la réponse produite. Cela peut rassurer l'utilisateur sur le fait que le DNN ne s'est pas focalisé sur un détail futile. Néanmoins, cela conduit à deux écueils. Premièrement, que faire si l'utilisateur n'est pas rassuré par cette réponse ? Et s'il l'est, est-ce suffisant pour avoir une confiance totale dans la réponse apportée ? Récemment, des travaux sur la certification formelle des DNNs ont ainsi été entrepris, qui ne se destinent pas à convaincre l'utilisateur d'une réponse, mais à apporter la preuve formelle que certaines propriétés sont vérifiées quelle que soit l'entrée. Il serait néanmoins naïf de croire que les propriétés vérifiables actuellement sont pleinement satisfaisantes, mais cela donne un cadre solide pour arriver à terme à des solutions convaincantes, en étendant les méthodes actuelles.

Dans ce papier, nous allons passer en revue les algorithmes disponibles en 2019 pour certifier formellement les DNNs. Parce que la vaste majorité des travaux portent sur les réseaux à propagation avant (feed-forward), nous nous concentrerons sur ceux-ci. On ne s'intéressera pas aux algorithmes permettant d'entraîner ces réseaux, mais à l'analyse et la certification de la robustesse d'un tel réseau, une fois entraîné. Nous rappellerons néanmoins la forme de ces DNNs, afin de décrire les différentes techniques pour les analyser. Il est à noter qu'un premier mini survey [3] a été réalisé en 2017, mais celui-ci ne pouvait pas prendre en compte les méthodes incomplètes (postérieures), décrites en Section 5.

2 Une rapide description des Réseaux Neuronaux Profonds (DNNs)

Dans cet état de l'art, nous allons nous restreindre au cas des DNNs à propagation en avant (*feed-forward*), c'est à dire tels que des entrées sont injectées dans les DNNs, sont utilisées récursivement pour calculer la sortie des couches cachées intermédiaires successives, avant de produire un résultat qui est retourné en sortie (voir Figure 1). En particulier, il n'y a pas de boucles dans de tels réseaux. La plupart des travaux sur la certification formelle des DNNs se focalisent sur les DNNs feed-forward. Concernant les réseaux récurrents (avec des boucles), quelques travaux proposent de les approcher automatiquement par des réseaux feed-forward et de certifier ces derniers [1].

Un réseau feed-forward est défini par les données suivantes :

— Des couches de 1 à n. La couche 1 est constituée des entrées. La couche n est constituée des sorties.
— D'un ensemble N de noeuds, chaque noeud n étant dans une couche $c(n)$.
— Une relation $R \subseteq N \times N$ entre les noeuds d'une couche i et les noeuds d'une couche $i+1$ auxquels ils sont connectés.
— Pour chaque arc $(n_i, n_{i+1}) \in R$ entre un noeud n_i de la couche i et un noeud n_{i+1} de la couche $i+1$, un poids $w(n_i, n_{i+1})$.

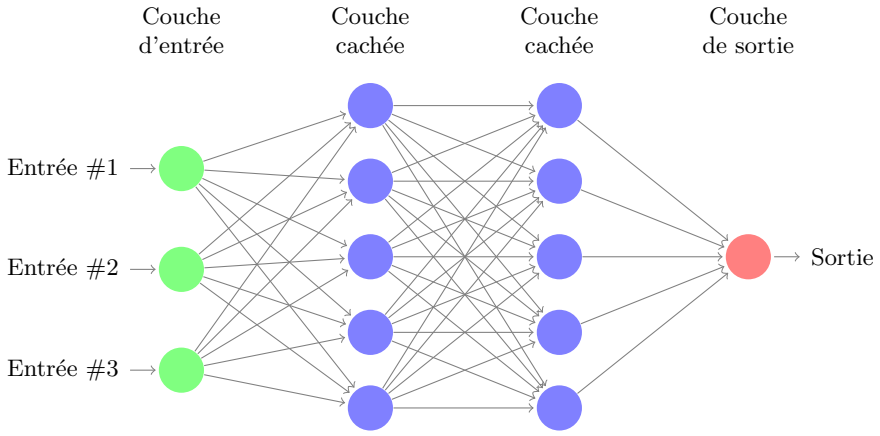

FIGURE 1: un DNN à propagation avant

— Pour chaque noeud n_{i+1}, une fonction d'activation $f : \mathbb{R}^n \to \mathbb{R}$ qui donne la valeur $v(n_{i+1})$ du noeud n_{i+1} en fonction des entrées $v(n_i)$ pondérées par $w(n_i, n_{i+1})$ reçues par les arcs (n_i, n_{i+1}).

Les couches utilisées, le nombre de noeuds et leurs connexions sont choisis en fonction de l'application (réseaux convolutifs pour la classification d'image avec des couches convolutives, puis de pooling, puis de correction, etc.). Les poids associés aux arcs sont quant à eux appris automatiquement par des techniques de rétropropagation du gradient d'erreur à partir des données annotées que l'on veut reproduire puis généraliser.

En pratique, la fonction d'activation dépend des couches utilisées (convolutive, pooling, de correction, de perte ...). Il peut s'agir des fonctions :

1. Identité : $f(x) = x$ pour tout x.
2. ReLU (rectified linear unit), $f(x) = x$ pour $x > 0$, $f(x) = 0$ sinon
3. Pooling, souvent $f(x_1, \ldots, x_n) = \max(x_i)$, ou bien leur moyenne,
4. Sigmoïde, $f(x) = \frac{1}{1+e^{-x}}$.
5. Tangente hyperbolique $f(x) = tanh(x)$.

Comme on peut le voir cette fonction d'activation peut être linéaire (identité, moyenne) ou non (ReLU, etc).

3 Certification des DNNs

3.1 Tests statistiques usuels et exemples adverses

La méthode la plus commune pour évaluer un DNN est de faire des tests avec des données annotées qui *n'ont pas été utilisées* pour apprendre le DNN. On peut ainsi comparer les réponses apportées par le DNN avec les réponses attendues, et en déduire un degré de précision, par exemple sous forme de courbe ROC ou de matrice de confusion pour les problèmes de classification. Un algorithme assez extensif de génération de tests est DeepXplore [14].

Si ces tests sont nécessaires, ils sont loin d'être suffisants. En effet, une caractéristique connue des DNNs [20] est leur manque de robustesse : un changement infime sur l'entrée peut avoir de grandes conséquences sur la sortie. Pire, un réseau peut avoir une très grande précision sur des données prises au hasard, mais être facilement attaquable : étant donnée une entrée, il est souvent aisé de la modifier d'une manière imperceptible et d'induire une sortie du DNN complètement différente [9].

Des méthodes ont été mises au point pour renforcer la robustesse des DNNs : D'abord, en générant automatiquement des exemples adverses afin de les apporter en entrée du DNN pour l'améliorer. D'autres méthodes restreignent la forme des réseaux pour assurer qu'ils soient structurellement plus robustes, en utilisant en particulier leur caractère Lipchitzien [5].

Si ces travaux sont utiles, en améliorant un des défauts principaux des DNNs, ils ne produisent aucune garantie, autre qu'une garantie statistique qui n'est pas suffisante pour des applications critiques des DNNs. Récemment, des travaux complétant ces tests statistiques ont été réalisés autour de la certification formelle de la robustesse des DNNs, travaux que nous allons décrire dans la suite.

3.2 Spécification formelle du problème

Il existe plusieurs types de certifications formelles possibles pour les DNNs. On se fixe un DNN à certifier. Les spécifications usuelles sont les suivantes :

- *satisfiabilité :* Etant donné des contraintes linéaire U, V, savoir si un couple d'entrée/sortie (u, v) satisfaisant U, V est réalisable par le DNN. Par exemple, U peut spécifier que l'entrée u contient une image de feu rouge, et V spécifie que la classification v est "feu vert". Montrer que ce couple n'est pas satisfaisable (UNSAT) certifierait qu'aucune entrée satisfaisant U (contenant un feu rouge) ne peut être classifiée comme feu vert.
- *robustesse* : Savoir si le DNN est *robuste*, c'est à dire si rajouter un bruit à l'entrée u peut faire changer la sortie v. Pour cela, on définit une région adverse, souvent une boule autour d'une entrée u, définie par rapport à la norme L_1 ou L_∞ pour avoir une spécification linéaire. Par exemple, toutes les images obtenues à partir d'une image u en changeant l'intensité de chaque pixel de quelques pourcents.

La robustesse peut être vue comme une question de satisfiabilité particulière, où U est défini comme une boule autour d'un point, et V doit être une constante

(spécification qui est linéaire). Dans la littérature, c'est la question de la satisfiabilité qui est résolue théoriquement, mais c'est la spécification de robustesse qui est le plus souvent considérée, parce que en pratique, il est quasi impossible de spécifier formellement une propriété voulue par une contrainte linéaire : comment décrire toutes les images de feux "rouges" ou de "feux verts" ?

La robustesse n'est en elle-même pas suffisante, parce qu'on ne demande pas que la sortie soit bonne, mais uniquement qu'elle ne change pas. C'est en fait une qualité plus qu'un défaut : cela évite d'avoir à spécifier les couples entrée/sortie, ce qui est souvent difficile (cf. plus haut). Aussi, le défaut est facilement résolu par des tests statistiques mentionnés plus haut, qui assure que la réponse soit bonne. Ainsi, c'est deux parties (tests statistiques et preuve formelle de robustesse) sont complémentaires.

La robustesse a aussi ses limites : si le DNN était robuste en tout point, alors sa sortie serait constante, et donc le DNN serait inutile. Une question intéressante est de déterminer automatiquement les points sur lesquels on demande la robustesse. Récemment, des méthodes telles que DeepSafe [10] ont été développées pour spécifier la robustesse d'un DNN d'une manière plus générale qu'explorer le voisinage d'une entrée u donnée. L'idée est d'utiliser des algorithmes de clustering sur les exemples donnés pour obtenir des clusters avec la même réponse. La question devient ainsi : "est ce que le DNN va apporter cette réponse pour toute entrée dans le cluster ?"

Il existe également quelques travaux qui traitent de spécifications non linéaires, tels que [16], qui utilisent des spécifications convexes.

3.3 Un problème de programmation linéaire mixte

Tout d'abord, il est utile de commencer par remarquer que si toutes les fonctions d'activation utilisées sont linéaires, alors on peut utiliser des algorithmes très efficaces de programmation linéaire pour répondre aux questions de satisfiabilité et de robustesse de manière exacte.

Cependant, les réseaux usuels utilisent plusieurs couches non linéaires, avec généralement des fonctions d'activation ReLU ou de Pooling. En considérant des fonctions d'activation linéaires par morceaux (ReLU, Max Pooling, mais ni tanh, ni sigmoïde) les problèmes de satisfiabilité et de robustesse deviennent ainsi [2] des problèmes de *programmation linéaire mixte*, mêlant valeurs continues et valeurs discrètes. Les entrées et les noeuds internes sont continues, tandis que le mode des fonctions d'activation linéaires par morceaux est discret : on veut connaitre le morceau linéaire d'intérêt, par exemple pour ReLU il faut savoir si l'entrée x est positive $x \geq 0$ ou négative $x < 0$. Le problème a ainsi été prouvé *NP-complet* [13], c'est à dire qu'il est impossible d'avoir un algorithme en temps polynomial pour résoudre le pire cas, à moins que P=NP. Comme les DNNs sont très grands, ce résultat semble indiquer l'infaisabilité de la certification.

Néanmoins, en pratique, le pire cas est très improbable. Ces dernières années, des algorithmes efficaces de recherche de certification sûre ont ainsi été développés. Il existe deux grands types complémentaires de méthodes :

— les méthodes complètes, qui produisent toujours un résultat : soit la certification, soit un contre-exemple d'entrée/sortie (u, v) qui viole la spécification.
— les méthodes incomplètes, qui peuvent retourner : "ne sait pas".

Les méthodes complètes sont plus précises que les méthodes incomplètes. Par exemple, elles sont capables de répondre exactement quel est le rayon de robustesse ε d'un DNN atour d'une entrée u. Les méthodes incomplètes ne seront capables que de donner une borne inférieure qui garantit la robustesse, et une borne supérieure au-delà de laquelle la robustesse est violée. Ces méthodes incomplètes sont néanmoins beaucoup plus rapides et passent mieux à l'échelle (des dizaines de milliers de noeuds plutôt que jusqu'à des milliers pour les premières). Elles doivent être utilisées en premier lieu. Si elles échouent à donner une réponse, il faudra alors essayer une méthode complète.

4 Méthodes complètes

Les méthodes complètes s'attaquent toutes à certifier un DNN avec des fonctions linéaires par morceaux, en développant différents algorithmes de recherche pour le problème de programmation linéaire mixte.

Un des premiers travaux dans ce sens est RELUPLEX [13], qui développe un algorithme de recherche paresseux sur les modes d'activation des ReLU : comme ces modes d'activation sont les seules variables difficiles à décider, l'algorithme repousse leurs études le plus loin possible, espérant être capable de conclure sans les considérer. Au départ, l'algorithme suppose que les entrées des noeuds ReLU sont complètement décorrélées des sorties. Un algorithme très efficace de programmation linéaire (continue) peut donc être utilisé. Cet algorithme va guider vers les noeuds ReLU qui doivent être explorés, en ignorant les autres noeuds. Pour les noeuds explorés, un mode d'activation est considéré pour relancer l'al-

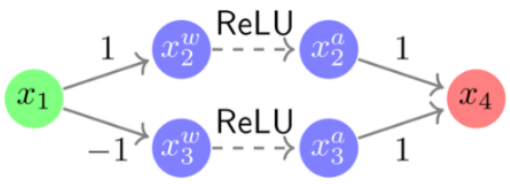

FIGURE 2: Décorrélation entre sortie pondérée x_2^w, x_3^w et activation ReLU x_2^a, x_3^a de 2 noeuds x_2, x_3. La méthode du simplexe trouve une solution $x_1 = x_2^w = x_3^w = 0$, $x_3^a = 0, x_4 = x_2^a = 0.5$ à la requête $x_4 \geq 0.5$ avec $x_1 \in [0, 1]$. ReLUplex se rend compte que cette solution viole le noeud ReLU x_2. Comme $x_2^w \geq 0$, il rajoute la contrainte $x_2^a = x_2^w$ et trouve la solution $x_1 = 0.5$ implique $x_4 = 0.5$. Il n'a pas eu besoin de considérer la fonction d'activation pour le noeud x_3 (les valeurs x_3^a, x_3^w sont restées décorrélées).

gorithme de programmation linéaire. Enfin, un algorithme de backtracking est utilisé si le mode sélectionné n'était pas consistant avec d'autres variables.

La partie délicate est de limiter l'exploration des phases d'activation de tous les noeuds ReLU, qui provoque une explosion combinatoire du nombre de cas à considérer (2 puissance le nombre de noeuds). ReLUPLEX limite cela en n'analysant que les noeuds vraiment nécessaires. Des variantes existent : dans le solveur Planet [7], une préanalyse du DNN en surapproximant la sortie des noeuds non linéaires en des fonctions linéaires est utilisée afin de connaître les modes d'activation de la plupart de ces noeuds. Cela permet d'éviter de considérer des modes inutiles. Dans le solveur BAB [3], des techniques de "Branch and Bound" sont utilisées pour explorer l'espace d'activation des noeuds ReLU.

Enfin, pour les DNNs destinés à contrôler des systèmes complexes, le solveur Sherlock [6] a été développé. Sherlock s'intéresse à trouver les valeurs extrémales de la sortie. Comparé à RELUPLEX, une recherche locale est ajoutée, utilisant une descente de gradient pour trouver un extrémum *local* de la sortie. Cette recherche alterne avec une phase de programmation linéaire mixte, utilisée pour trouver une meilleure valeur (si possible) que l'extrémum local, etc.

5 Méthodes incomplètes

Les méthodes incomplètes se basent toutes sur l'approximation des noeuds non linéaires du DNN. L'idée est d'utiliser deux fonctions linéaires, telles que la fonction d'entrée/sortie d'un neurone (non linéaire) soit comprise entre ces deux fonctions. Ainsi, le problème se ramène à un simple problème de programmation

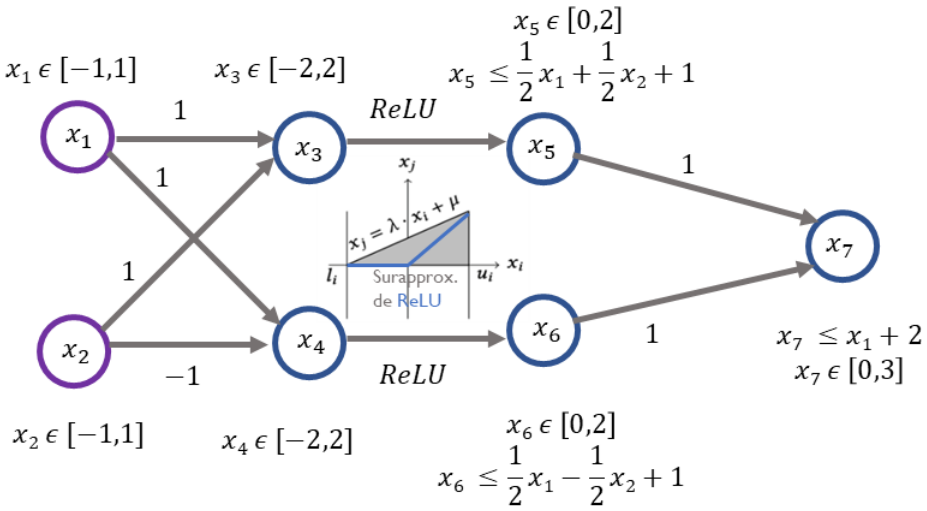

FIGURE 3: Zonotope utilisé dans ERAN : La fonction ReLU est surapproximée par une fonction linéaire. Cela permet d'être sûr que $x_7 \leq 3$, ce qui ne serait pas le cas si on ne considère que les valeurs extrémales ($x_7 = 4$ semblerait possible).

linéaire. Si le DNN ainsi surapproximé est certifié, alors le DNN initial est aussi certifié. De plus, si un contre-exemple (u, v) est trouvé, il peut être testé dans le DNN original : si l'entrée u produit effectivement une sortie v' (qui peut être différent de v) violant les conditions, une attaque du DNN a été trouvée. Sinon, l'algorithme répond qu'il ne sait pas.

Le premier travail implémentant cette technique est le solveur NEVER [15], qui approche des noeuds sigmoïdes par des fonctions linéaires de plus en plus raffinées, mais avec des performances très réduites. Cette technique a également été utilisée pour les fonctions d'activation ReLU [21], beaucoup plus faciles à approcher, avec de bien meilleurs performances.

Plus récemment, les fonctions non linéaires d'activation ont été approchées à l'aide de polyèdres (toujours définis par des fonctions linéaires), d'abord dans AI2 [8], puis dans ERAN, en utilisant des zonotopes (polyèdres avec symétrie) [18] et des polyèdres restreints DeepPoly [19], pour aider au passage à l'échelle. Ces techniques permettent d'analyser des DNNs avec plusieurs dizaines de milliers de noeuds, pour tout type de fonctions d'activation.

D'autres travaux ont aussi été réalisés pour des entrées de petite taille, avec des techniques indépendantes de la taille du DNN. L'analyse est réalisée couche par couche, en bornant les valeurs possibles des noeuds de la couche considérée. Le premier solveur de ce type est DLV [11], qui discrétise les entrées et réalise une recherche exhaustive parmi toutes les valeurs d'entrée. Des techniques d'analyse sont également possibles sans discrétiser les entrées, en utilisant le caractère Lipchitzien de certaines fonctions d'activation (ReLU, etc) [17],[21], ce qui par raffinements successifs permet de borner de manière fine la réponse des neurones.

6 Conclusion et Discussion

Nous avons réalisé un tour d'horizon des différentes techniques de certification de DNNs en 2019. En l'espace de quelques années, ces nouvelles techniques ont permis de gagner quelques ordres de grandeur. Actuellement, les techniques incomplètes, utilisant approximation et abstraction, sont les plus efficaces, et permettent de prendre en compte des DNNs de taille relativement grande (des dizaines de milliers de noeuds). Elles ne sont néanmoins pas toujours capables de décider si un DNN est sûr. Des techniques de raffinement automatique doivent être développées afin de répondre dans bien plus de cas.

Enfin, les DNNs ne sont pas les seuls modèles efficaces mais peu compréhensibles produits par l'intelligence artificielle. L'apprentissage par renforcement présente les mêmes défis de certification formelle qu'il faudrait relever à l'avenir.

Remerciements : Nous aimerions remercier les rapporteurs pour leurs suggestions constructives.

Références

1. Michael E. Akintunde, Andreea Kevorchian, Alessio Lomuscio, and Edoardo Pirovano. Verification of rnn-based neural agent-environment systems. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pages 6006–6013. AAAI Press, 2019.
2. Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Advances in Neural Information Processing Systems 29 : Annual Conference on Neural Information Processing Systems 2016.*, pages 2613–2621, 2016.
3. Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Piecewise linear neural network verification : A comparative study. *Technical Report, CoRR*, abs/1711.00455, 2017.
4. Nicholas Carlini and David A. Wagner. Audio adversarial examples : Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops, SP Workshops 2018.*, pages 1–7. IEEE Computer Society, 2018.
5. Moustapha Cissé, Piotr Bojanowski, Edouard Grave, Yann N. Dauphin, and Nicolas Usunier. Parseval networks : Improving robustness to adversarial examples. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 854–863. PMLR, 2017.
6. Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods - 10th International Symposium, NFM 2018, Proceedings*, pages 121–138, 2018.
7. Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Proceedings*, pages 269–286, 2017.
8. Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2 : safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings.*, pages 3–18, 2018.
9. Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
10. Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark W. Barrett. Deepsafe : A data-driven approach for assessing robustness of neural networks. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Proceedings*, volume 11138 of *LNCS*, pages 3–19. Springer, 2018.
11. Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017*, volume 10426 of *LNCS*, pages 3–29. Springer, 2017.

12. Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In *2017 Conference on Empirical Methods in Natural Language, EMNLP 2017.*, pages 2021–2031. Association for Computational Linguistics, 2017.
13. Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex : An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Proceedings*, volume 10426 of *LNCS*, pages 97–117. Springer, 2017.
14. Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore : Automated whitebox testing of deep learning systems. In *26th Symposium on Operating Systems Principles 2017.*, pages 1–18. ACM, 2017.
15. Luca Pulina and Armando Tacchella. Never : a tool for artificial neural networks verification. *Ann. Math. Artif. Intell.*, 62(3-4) :403–425, 2011.
16. Chongli Qin, Krishnamurthy Dvijotham, Brendan O'Donoghue, Rudy Bunel, Robert Stanforth, Sven Gowal, Jonathan Uesato, Grzegorz Swirszcz, and Pushmeet Kohli. Verification of non-linear specifications for neural networks. In *Seventh International Conference on Learning Representations, ICLR 2019*, 2019.
17. Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018.*, pages 2651–2659, 2018.
18. Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems 31 : Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018.*, pages 10825–10836, 2018.
19. Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *PACMPL*, POPL(3) :41 :1–30, 2019.
20. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.
21. Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. Towards fast computation of certified robustness for relu networks. In *35th International Conference on Machine Learning, ICML 2018*, pages 5273–5282, 2018.