Verified Classification using Deep Neural Networks with Perturbation Analysis

PATRICK DEUTSCHMANN and LUKAS TIMPL

This report describes our submission to the *Siemens AI Dependability Assessment*. It solves a binary classification task using a deep neural network and provides security guarantees for local robustness using perturbation analysis based on linear relaxation. Our model displays high predictive performance, gives provable safety guarantees, scales well to more complex data sets, and lets domain experts dynamically configure the class-wise cost of misclassification.

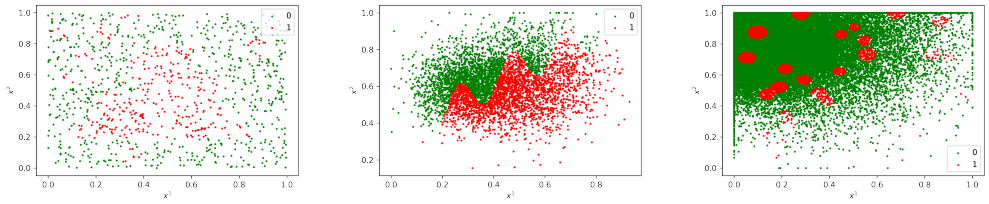
Contents

1 INTRODUCTION

The challenge task of the *Siemens AI Dependability Assessment*[1] was to solve a 2D binary classification problem, i.e. a binary function $f(\mathbf{x_i})$ classifies a data point $\mathbf{x_i} = (x_i^1, x_i^2)$ as either $l_i = 0$ (green) or $l_i = 1$ (red). With only two dimensions, the data sets are rather simple, yet the approach should also scale to inputs of higher dimensions. Most importantly, however, $f(\mathbf{x}_i)$ should provide safety guarantees. This means that the ML model must not only have high predictive capabilities but also must be able to produce provably reliable results under certain assumptions. Furthermore, the setting is chosen such that misclassification costs are not equal for the two classes. Following the analogy of traffic lights, it is more dangerous (costly) to classify a red sample as green than classifying a green sample as red. It is given that the training data is labelled correctly and that the classes do not overlap.

[1] https://ecosystem.siemens.com/ai-da-sc

Authors' address: Patrick Deutschmann, patrick.deutschmann@student.tugraz.at; Lukas Timpl, lukas.timpl@student.tugraz.at.



(a) 1,000 samples (b) 5,000 samples (c) 50,000 samples

Fig. 1. The data sets of the challenge differ significantly in size, class balance and separability.

Our solution aims at striking the ideal balance of these criteria in that it displays high predictive performance, gives provable safety guarantees, scales well to more complex data sets, and lets domain experts dynamically configure the class-wise cost of misclassification.

The architecture we chose is a deep neural network trained with cost-weighted binary cross-entropy loss. The safety guarantees follow the assumption of reliable training inputs. Using Linear Relaxation Based Perturbation Analysis (LiRPA), we can guarantee stable predictions in the ϵ-neighbourhood of previously seen samples. [2]

1.1 Data sets

Fig. 1 shows the three data sets of the challenge. Before we started our experiments, we analysed them and found that they differ in multiple critical aspects.

(1) **Size**: The data sets vary in size, with A being the smallest and C the biggest.
(2) **Class balance**: While data set B is relatively balanced, A and C show significant imbalance with a ratio of roughly 2:1 and 10:1, respectively, where class 0 (green) is the overrepresented one.
(3) **Separability**: It is easy to see that data set B can be separated by drawing a simple sine-like wave across the two dimensions and data set C by using multiple ellipses. For data set A, however, it is harder to make out a clear trend. There are several cases where green and red samples are very close to each other, and accurately separating classes will prove difficult.

Thus, while we employ the same general approach for all three data sets, we tune our hyper-parameters separately to obtain ideal performance.

2 POSSIBLE SOLUTIONS AND RELATED WORK

Before choosing an approach, we analysed the core aspect of safety guarantees and possible assumptions. In particular, we looked at two possibilities of formulating guarantees for our problem: (a) expressing uncertainty for a given sample and (b) robustness against local perturbations.

The former addresses the problem that most machine learning approaches always produce an answer, even if they encounter inputs the like of which they have never seen before. In that case, they should be highly unsure about their prediction but mostly fail to express that. One way to counter this are Bayesian Neural Networks (BNNs) [2]. In a probabilistic manner, they place a distribution over the network parameters. They can, hence, express their uncertainty in case they see challenging inputs. This could be very useful in a security-critical environment and would, for instance, allow an autonomous vehicle to alert its driver if it is unable to deal with a particular situation. However, it does not strictly correspond to the challenge description where security

[2] All code is provided as a ZIP file attached to this report.

	Data set	A				B				C			
	Model	*LR*	*NB*	*SVM*	*GB*	*LR*	*NB*	*SVM*	*GB*	*LR*	*NB*	*SVM*	*GB*
Train	Acc.	0.67	0.77	0.87	**0.96**	0.85	0.85	0.89	**0.996**	0.91	0.91	0.91	**0.96**
	F1	0.81	0.73	0.87	**0.96**	0.85	0.85	0.89	**0.996**	0.95	0.95	0.95	**0.96**
Val.	Acc.	0.67	0.75	0.84	**0.92**	0.88	0.88	0.90	**0.984**	0.91	0.91	0.91	**0.96**
	F1	0.80	0.71	0.83	**0.92**	0.88	0.88	0.90	**0.984**	0.95	0.95	0.95	**0.96**

Table 1. Results of baseline models

guarantees for *unseen* inputs should be made. A BNN would require a concrete input $\mathbf{x}_i$ to express its certainty.

Therefore, we to decided pursue the goal of (b), meaning that we made our model robust against local perturbations. Simply put, this means that the model should reliably label unseen samples that lie near known training samples with the same class as the known sample. We put forward a more formal definition in Subsection 3.2.

It is trivial to give such guarantees for simple models: For logistic regression and k-nearest-neighbours, the decision boundary can be used to derive them. For naive Bayes classifiers, confidence intervals can be directly computed. Some of these classifiers would also perform relatively well on the given data sets, which is why we use them as baselines for our final model (see Subsection 2.1). However, they would not scale well to higher dimensions and more complex problems.

More complicated models include support vector machines (SVM) and Gaussian Processes (GP) for classification. SVM can be powerful, especially with RBF kernels, but cannot provide the guarantees we require. GP would provide us with empirical confidence intervals but becomes inefficient in higher dimensions. Furthermore, both approaches are not leading in state-of-the-art predictive capabilities for challenging tasks, especially in the vision domain.

Another powerful class of machine learning models are decision trees and random forests. They display high predictive capabilities and can also be verified, but as [7] found, verification time does not scale very well to larger models. However, there might very well exist verification techniques for tree-based models that could suit this challenge but that we are unaware of at this time.

Finally, we arrive at deep neural networks, which have received much attention in recent machine learning history. They scale well to very complex problems but used to be considered textbook examples for black-box models. However, recently, there has been a lot of work focused on explainability and defence against adversarial samples. It started with approaches that provided exact upper and lower bounds for neural network outputs given certain input perturbations. Reluplex [5] is an example that, however, only works for networks with ReLU activations. Further research introduced methods that work with arbitrary activations and use linear relaxations that compute tight approximations to significantly improve performance, allowing these techniques to be scaled to larger networks. Such work includes IBP [3], CROWN [9] and DeepPoly [6]. We decided to use AutoLiRPA [8], a framework that applies these techniques to general neural networks and allows for automatic analysis on computational graphs.

2.1 Baselines

To warrant the use of a complex deep neural network, we made sure that it exceeds the performance of simpler models. We established baselines with logistic regression *(LR)*, Gaussian naive Bayes *(NB)*, SVM with an RBF kernel *(SVM)* and a gradient boosting classifier *(GB)* as depicted in Table 1. We observe that gradient boosting performs best for all data sets, followed by SVM. Data set A seems to be the most challenging one, probably due to the small sample size and irregular separability.

True \ Predicted	0 (green)	1 (red)
0 (green)	$c_{0,0} = 0$	$c_{0,1}$
1 (red)	$c_{1,0}$	$c_{1,1} = 0$

Table 2. Cost matrix for classifications

3 APPROACH

We model the safety criticality of our classifier's decisions using a cost matrix illustrated in Table 2. Correctly classifying a sample obviously does not incur any cost, i.e. $c_{0,0} = c_{1,1} = 0$. In our particular example, assigning label 0 (green) to a data point with true label 1 (red) is safety-critical and therefore the associated cost $c_{1,0}$ should greatly exceed the cost $c_{0,1}$ of assigning label 1 (red) to true label 0. Concrete values for $c_{0,1}$ and $c_{1,0}$ can be set by the users of our approach. The algorithm then minimises the total cost. If $c_{0,1} = c_{1,0}$, this is equivalent to maximising accuracy.

3.1 Model

The model we use is a deep neural network implemented in PyTorch. It receives as input a 2D training sample, processes it through several hidden layers with ReLU activations and finally outputs a single scalar, activated through the sigmoid function $\sigma(a_i)$. If the output is smaller than 0.5, class 0 is assigned, class 1 otherwise. We use double weighted binary cross-entropy as our loss function, where we define the loss for one sample i as

$$l_i = -w_i[py_i \log \sigma(a_i) + (1 - y_i) \log(1 - \sigma(a_i))]. \tag{1}$$

a_i is the output of the neural network before activated in sigmoid and y_i is the correct label. To counter class imbalance, we set

$$w_i = \begin{cases} \frac{\#\text{ class 0 samples}}{\#\text{ class 1 samples}} & \text{if } y_i = 1, \\ 1 & \text{otherwise.} \end{cases} \tag{2}$$

This will cause the training samples of the underrepresented class to be weighted higher and thereby offset the imbalance. Additionally to that, we set the weight for positive examples $p = \frac{c_{1,0}}{c_{0,1}}$ to account for the costs of misclassification that differ per class.

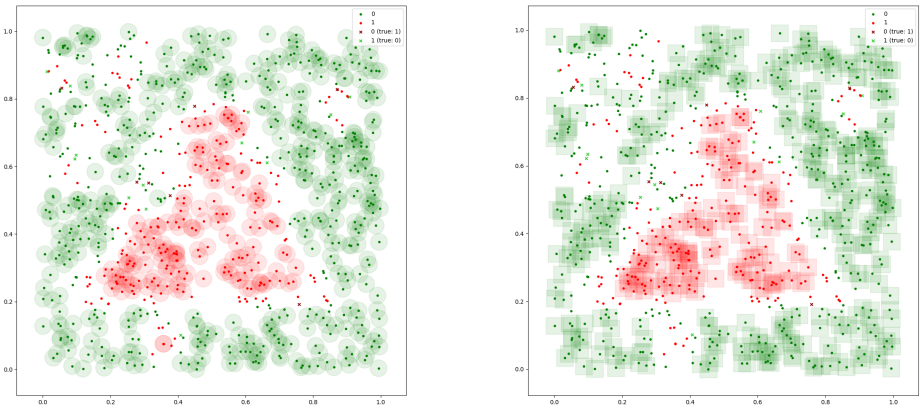
3.2 Verification

Relying on the assumption that our training samples are correctly labelled, we want to guarantee for a certain number of samples, which we call *verified*, that all inputs surrounding them are assigned the same class as the centre. Formally, we define a neighbourhood around training sample $\mathbf{x}_i$ as an L_p-ball with radius ϵ. We then guarantee for all possible inputs that lie within this ball $B_\epsilon(\mathbf{x}_i) = \{\mathbf{y} \in \mathbb{R} : ||\mathbf{x}_i - \mathbf{y}||_p < \epsilon\}$ that they are assigned l_i. In our implementation, we support both the L_2 and the L_∞-norm. Under the assumption that an unseen data point lies within the defined neighbourhood of a known training sample, we can give a probability that this new data point will be correctly classified based on the percentage of training samples that are verified, i.e.

$$\text{acc}_{\text{verified}}(\epsilon) := p(f(\mathbf{y}_i) = l_i | \exists \mathbf{x}_j : \mathbf{y}_i \in B_\epsilon(\mathbf{x}_j) \wedge l_j = l_i) \tag{3}$$

$$= \frac{\#\text{ verified and correctly classified samples}}{\#\text{ samples}} \tag{4}$$

where $\mathbf{x}_j$ is a sample of the training data set.



(a) L_2-norm (b) L_∞-norm

Fig. 2. Illustration of verified samples for $\epsilon = 0.025$. The boxes or circles (for L_∞-norm or L_2-norm respectively) denote the area of which we can guarantee that all samples within will be assigned the same class as the training sample in the middle, i.e. all inputs in red boxes/circles will be classified as 1, all inputs in green boxes/circles as 0.

We call this metric *verified accuracy*, inspired by the *verified accuracy under IBP verification* as proposed in [4]. The smaller the ϵ-neighbourhood becomes, the closer the verified accuracy approaches the accuracy, i.e. $\lim_{\epsilon \to 0} \text{acc}_{\text{verified}}(\epsilon) = \text{acc}$. The concept is illustrated in Figure 2.

The challenging aspect of this verification is to show for all data points $\mathbf{y} \in B_\epsilon(\mathbf{x}_i)$ that they indeed are labelled with the same class as $\mathbf{x}_i$. As there exist infinitely many such inputs, it is impossible to let the model classify them all and check empirically. We, therefore, use the AutoLiRPA framework [8] to compute bounds for the outputs of our network. It computes for a set of inputs (which we provide as a training sample and its ϵ-neighbourhood) the worst-case lower and upper bounds for the output of a neural network. It achieves this by simplifying the network to a linear function and using upper and lower bounds to replace non-linear units. Using this, it can propagate the bounds through the model and, in the end, provide the bounds for the entire network.

For our verification, we compute upper (a_{upper}) and lower (a_{lower}) bounds for every sample in our training data set, compute the respective sigmoid activations and check whether both the upper and the lower bound would result in the same prediction. If they do, we can guarantee that all inputs $\mathbf{y} \in B_\epsilon(\mathbf{x}_i)$ are classified as l_i. Otherwise, we cannot make that statement, and somewhere in the ϵ-neighbourhood of $\mathbf{x_i}$ there exist inputs that our model would classify as $1 - l_i$.

3.3 User configuration

To summarise, we ask the users of our model to configure the following critical parameters:

(1) **Perturbation** ϵ: The perturbation controls for how large a neighbourhood of training samples we can assume that the classification should be the same as the centre sample. Low values can be chosen if the sensors with which the data was measured are very accurate, high values otherwise. A low ϵ will result in a verified accuracy close to the general accuracy; a high one will lead to more considerable divergence.

(2) **Norm (L_2 or L_∞)**: The perturbation norm controls the p-norm that is used for the L_p-ball of the neighbourhood. If we assume that all sensors will diverge to degree ϵ, we could, for example, set this to L_∞. If the total deviations are expected to be in that range, L_2-norm might be better suited. Note that while we have only implemented L_2 or L_∞, all p-norms can theoretically be used.
(3) **Costs of misclassification $c_{0,1}$ and $c_{1,0}$**: These values should reflect the class-wise cost associated with misclassification as shown in Table 2.

4 EXPERIMENTS

After fixing the approach, we conducted numerous experiments to find the ideal network architecture that performs best for the individual data sets.

4.1 Metrics

We use various metrics, which allow us to compare the results given the imbalanced data sets, the verification capability, and the safety-related cost associated with misclassification.

- **Precision, Recall, F1**: For classification tasks on imbalanced data sets, precision, recall, and F1 score are commonly used to compare model performance. They help to capture the trade-off between true-positive and false-positive predictions better than accuracy. We use the F1 score during training to monitor our model's performance and compare final models. Additionally to the individual class scores, we compute the macro average over the F1 score of both classes. This means both labels are valued equally even though they might have a different number of samples. Therefore, the score would be low if the model only performed well on the class representing the majority of the data points, which is not the case for standard accuracy. For brevity reasons, we only report the combined score in our results.
- **Classification cost**: Our evaluation also captures the varying misclassification costs of the two classes as *total classification cost*. It is defined as the sum of all misclassifications weighted with the associated cost. Furthermore, we introduce the metric *classification cost per sample* which allows us to compare the metric on data sets with a different number of samples.
- **Verified Accuracy**: The verified accuracy which we introduced in Section 3 can quantify the probability of classifying an unseen data point correctly. This metric helps us evaluate the dependability of our model and provides a lower bound for the achieved accuracy of our model.
- **Misclassification Error**: Under the above assumptions, the misclassification error is simply $1 - \text{acc}_{\text{verified}}$.

In the end, we optimise our model for minimal classification cost. Depending on the ϵ chosen by the user, we achieve varying *verified accuracies* and *misclassification errors*.

4.2 Architecture Search

Due to the simplicity of the data sets, we only use simple multilayer perceptron (MLP) architectures. We experiment with between 1 and 4 hidden layers and different numbers of neurons as well as regularisation techniques such as weight decay and dropout regularisation. We train the model using Adam with a batch size of 32 and varying learning rates. To perform the architecture search, we use the metric *classification cost per sample*. This metric helps us find a suitable model with respect to our security metrics. We utilise Weights & Biases [1] for tracking our experiments and for visualising the results.[3]

[3] An interactive report of our model's results can be found here.

A big problem for neural networks is overfitting, meaning that the network will end up fitting the training data very well but will fail to generalise to new unseen data. To prevent this, we use 20% of the given data as validation data set. During training, we monitor the validation loss and use early stopping to prevent the network from overfitting.

4.3 Results

In this section, we list our achieved results on the data sets under our previously defined performance metrics. Amongst our metrics, the cost per sample gives a good indication of the performance with respect to the misclassification cost, while the verified accuracy provides a lower bound for the achieved accuracy under the respective ϵ-assumption. The corresponding misclassification error can be computed as one minus the verified accuracy.

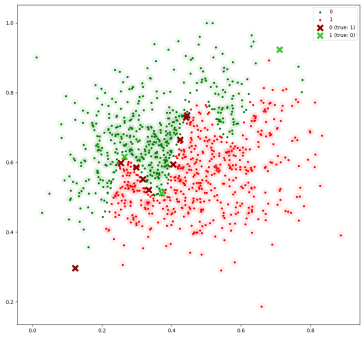
We perform experiments where we change the defined costs to be in line with the security aspect of the challenge. Namely, while keeping the cost $c_{0,1}$ (not security-critical) constant to 1, we alternate between cost values 1, 10 and 50 for $c_{1,0}$ (security-critical). For each data set, we are able to match or exceed the established baselines for a cost of 1:1. In Figure 3 we illustrate the performance of the trained model on data set B for varying costs. While for $c_{1,0} = 1$ there are a number of security-critical misclassifications (marked as red crosses), for $c_{1,0} = 10$ we only observe two such misclassification and for $c_{1,0} = 50$ the model does not make any. Furthermore, these results are achieved on the validation set, indicating the generalised capability of the model to respect the defined safety cost.

Data set A. For data set A, we use four hidden layers, each with 200 neurons and a learning rate of 0.002. We also use weight decay with $\lambda = 0.0001$ to prevent overfitting. In our experience, it was the most challenging data set. The results are listed in Table 3. We exceed the baseline accuracy for cost $c_{1,0} = 1$ and still achieve good performance with cost $c_{1,0} = 10$. Only for $c_{1,0} = 50$ the performance starts to degrade as the network gets more careful, and there is a safety-accuracy trade-off.

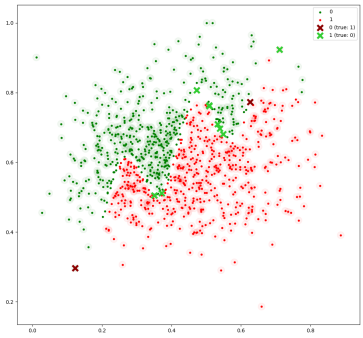
	Cost	$c_{1,0} = 1$			$c_{1,0} = 10$			$c_{1,0} = 50$		
	ϵ	**0.001**	**0.01**	**0.025**	**0.001**	**0.01**	**0.025**	**0.001**	**0.01**	**0.025**
Full	**Verified Acc.**	0.9680	0.9090	0.7250	0.9080	0.8540	0.7210	0.69500	0.6540	0.5850
Train	**Accuracy**	0.9750			0.9200			0.6975		
	F1	0.9748			0.8881			0.7536		
	Cost/Sample	0.0250			0.0800			0.3025		
Validation	**Accuracy**	0.9450			0.8800			0.7050		
	F1	0.9384			0.8427			0.7158		
	Cost/Sample	0.0550			0.3000			0.5400		

Table 3. Results for data set A

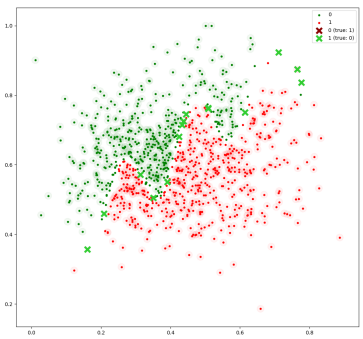
Data set B. For data set B, we use two hidden layers, each with 64 neurons and a learning rate of 0.0003. The results are listed in Table 4. In our experiments, data set B was the least challenging one, and we achieve excellent scores across all metrics with accuracy scores of around 99%.



(a) $c_{1,0} = 1$



(b) $c_{1,0} = 10$



(c) $c_{1,0} = 50$

Fig. 3. Predictions for data set B on the validation set for $\epsilon = 0.01$ with varying security-critical misclassification costs $c_{1,0}$

	Cost	$c_{1,0} = 1$			$c_{1,0} = 10$			$c_{1,0} = 50$		
	ϵ	**0.001**	**0.01**	**0.025**	**0.001**	**0.01**	**0.025**	**0.001**	**0.01**	**0.025**
Full	**Verified Acc.**	0.9864	0.8804	0.6836	0.9854	0.8880	0.7052	0.9760	0.8802	0.6514
Train	**Acc.**	0.9915			0.9918			0.9810		
	F1	0.9859			0.9813			0.9521		
	Cost/Sample	0.0085			0.0105			0.0190		
Val	**Acc.**	0.9880			0.9890			0.9850		
	F1	0.9888			0.9892			0.9869		
	Cost/Sample	0.0012			0.0290			0.0150		

Table 4. Results for data set B

Data set C. For data set C, we used two hidden layers with 256 neurons and a learning rate of 0.001. We list the results in Table 5. The biggest challenge with this data set was the high imbalance in the number of labels between the classes. However, as we adapted our loss function accordingly, we achieved rather good results around 98% accuracy for $c_{1,0} = 1$. As the cost increases, we can see a safety-accuracy trade-off where the system gets very cautious, and the accuracy does suffer a bit.

	Cost	$c_{1,0} = 1$			$c_{1,0} = 10$			$c_{1,0} = 50$		
	ϵ	**0.001**	**0.01**	**0.025**	**0.001**	**0.01**	**0.025**	**0.001**	**0.01**	**0.025**
Full	**Verified Acc.**	0.9749	0.8854	0.6563	0.9408	0.8636	0.6316	0.8769	0.8055	0.5660
Train	**Acc.**	0.9806			0.9467			0.8837		
	F1	0.8880			0.7830			0.7247		
	Cost/Sample	0.0194			0.0526			0.1163		
Val.	**Acc.**	0.9785			0.9483			0.8833		
	F1	0.9229			0.8335			0.7591		
	Cost/Sample	0.0215			0.0526			0.1216		

Table 5. Results for data set C

5 DISCUSSION

In this work, we have proposed a solution for estimating the dependability of neural networks and giving safety guarantees for local robustness in a classification setting.

While we only applied our approach to simple two-dimensional input data, it can easily be scaled to higher dimensions by using more complex models. This is also possible because bounds can be computed rather quickly as compared to the exact methods described in Section 2. Due to the high predictive capabilities of neural networks in various domains, the proposed solution can be applied to a multitude of different problems. Furthermore, the library we used, AutoLiRPA [8], can also be used for more complex architectures such as convolutional networks, recurrent neural networks, or transformers and thereby verify results for state-of-the-art models for challenging problems in, for example, computer vision.

The main limitations of our approach lie in the necessary assumptions, i.e. the requirement of correctly labelled input data and the premise that new samples are located within the ϵ-neighbourhood of existing training samples, as discussed in Section 3. We cannot give guarantees for inputs, the like of which our model has never seen before. That, however, lies in the nature of data-driven prediction models. While for these simple data sets other approaches might be less resource-intensive, they do not possess the predictive capabilities of neural networks and do not perform as well on other, more complex domains.

The definition of an ϵ-neighbourhood can be used to estimate the dependability of our system and can be adapted to the requirements of a given application. In conjunction with the cost matrix, which allows us to configure the desired safety-accuracy trade-off, this renders our system adaptable to a variety of safety-critical applications. We, hence, believe that our model strikes a good balance in this environment of demanding requirements and can be incorporated well into safety verification solutions.

REFERENCES

[1] Lukas Biewald. 2020. Experiment Tracking with Weights and Biases. https://www.wandb.com/ Software available from wandb.com.

[2] Ethan Goan and Clinton Fookes. 2020. Bayesian Neural Networks: An Introduction and Survey. *Lecture Notes in Mathematics* (2020), 45–87. https://doi.org/10.1007/978-3-030-42553-1_3

[3] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. 2019. On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. arXiv:1810.12715 [cs.LG]

[4] Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. 2019. Achieving verified robustness to symbol substitutions via interval bound propagation. *arXiv preprint arXiv:1909.01492* (2019).

[5] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

[6] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–30.

[7] John Törnblom and Simin Nadjm-Tehrani. 2018. Formal verification of random forests in safety-critical applications. In *International Workshop on Formal Techniques for Safety-Critical Systems*. Springer, 55–71.

[8] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. arXiv:2002.12920 [cs.LG]

[9] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. *arXiv preprint arXiv:1811.00866* (2018).