Annales Mathematicae et Informaticae
60 (2024) pp. 141–150
DOI: 10.33039/ami.2024.02.007
URL: https://ami.uni-eszterhazy.hu

Effective inclusion methods for verification of ReLU neural networks

Attila Szász, Balázs Bánhelyi

University of Szeged, Institute of Informatics
{szasz,banhelyi}@inf.u-szeged.hu

Abstract. The latest machine learning models are sensitive to adversarial inputs, i.e., the neural network can give incorrect results even with small changes in the learning case. To avoid this, techniques are used during learning, or verification is also possible. In many cases, these methods use interval arithmetic, whose usefulness is severely limited by overestimation. In this paper, we present and compare such methods that can handle this problem.

Keywords: artificial neural network, verification, interval arithmetic, symbolic calculations

AMS Subject Classification: 68T07, 68N30, 65G30

1. Introduction

One of the most important topics in artificial intelligence research today is the verification of neural networks. The accuracy of the networks has continuously increased over the years, so more and more complex neural networks have been created and many tasks could be solved by them. Many modern teaching methods have been developed that have improved the quality of the networks. In certain fields, it is inevitable to be precise and to have fast networks.

In many works, it has been shown that these nets, which are considered to be safe, can also be wrong [10]. In many cases, noise on the input that is invisible to the human eye can lead to a wrong classification. There are many methods developed by researchers to solve these problems. The methods are mainly divided into 2 classes: robust learning and adversary example detection.

For this reason, neural network verification is an important topic in today's artificial intelligence research. The neural network technique focuses on speed and typically uses floating point arithmetic, while others prefer symbolic methods [15]

Submitted: July 17, 2023
Accepted: February 6, 2024
Published online: February 14, 2024

used for reliability. Other important family of deep neural networks, the Binarized Neural Networks (BNN) [4, 6], that are similar to regular feedforward neural networks. One difference is that the weights and activations in a BNN are constrained to be only two values: 1 and −1, which implied other verification technique.

The standard numerical systems often have significantly longer run times [2, 11, 13]. In this paper, other methods have been described using the numerical result. These methods have a correct evaluation and a manageable runtime. The system we wrote defines not only the inputs but also the interval of values for the outputs of the given network. When verifying the robustness of a neural network, these intervals must be as small as possible. During the evaluation, in addition to the nets with the ReLU activation function, the output widths and the running times were also compared.

2. Motivation

During our work, we have developed a system based on reliable network assessment. The system supports multiple evaluation methods, in both CPU and GPU environments. Many current contemporary systems use floating-point arithmetic with an emphasis on speed in evaluation. The big disadvantage of this is that some numerical errors in the various operations can accumulate during the evaluation [16].

A good way to get a handle on these errors is to use interval arithmetic [1, 5]. One solution is to compute an interval containing the given value instead of the floating-point number. The method is well suited for neural networks and also for their evaluation since the operations that can be performed on real numbers can be easily extended to intervals. In this case, the inputs of the network are intervals. One of the advantages of the method is that it increases the running time only minimally compared to floating point evaluation. Since it is reliable and robust in class, this method is also used in the evaluation phase.

At the last ICAI conference, the first results of an interval-based verification algorithm [3]. In this approach, a simple natural interval expansion was used to compute inclusion functions. This mod was able to include the values of the output neurons in the input interval. This proves the correct result for each point of the input.

The network shown in Figure 1 is evaluated according to the rules of naive interval arithmetic. The ReLU activation function is contained in some neurons of the network. The value of the output neuron x_5 lies in the interval $[0; 5]$. It can be observed that the upper limit of 5 would occur only if the neuron x_3 had the value 5 and the neuron x_4 had the value 6. It can be seen that under the given input conditions, these values can never occur simultaneously. As a result, the upper bound of x_5 was never sharp, leading to an overestimation in the evaluation. The main reason for the overestimation is the dependency problem, which can become quite strong and unmanageably wide in the results as the number of layers increases.

The goal of our work is to implement and compare numerically correct systems that handle the dependency problem but do not drastically increase the runtime.

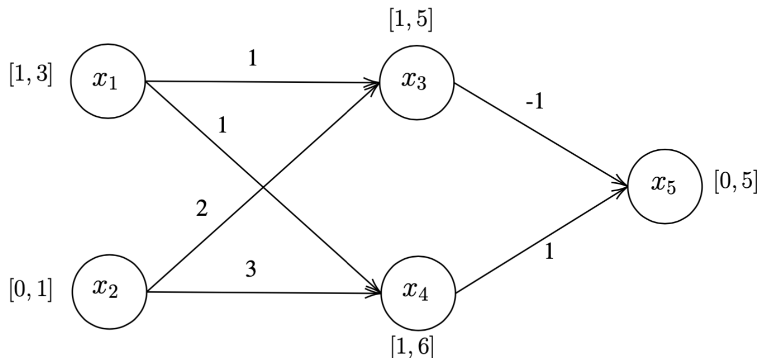

Figure 1. Naive interval arithmetic network evaluation.

3. Methods

In this work, three techniques were presented, each based on the use of a 1-1 function for the set of values of neurons on the given input set. The first technique is symbolic propagation [12], where we search for the best-fitting function and then handle the nonlinearity with new variables. In the second case, 2 separate functions were held for each neuron and calculated with them. In the third case, the well-known linear affine expression was used.

3.1. Symbolic function propagation

In our system, the overestimates resulting from the naive evaluation have been effectively handled. The method has a ReLU activation function that is fully feedforward and was used to evaluate networks with connected layers. The main feature of the system is that the evaluation captures dependency relations rather than the values of the current layer. In addition to a neuron of the input layer x_1 and x_2, the function $x_3 = x_1 + 2x_2$ is recorded for the neuron x_3 (Figure 1). When the network is evaluated, the corresponding function is determined for all neurons, and the result of its evaluation in intervals is the set of values of the neuron.

The transformations between layers can be easily extended to the functional representation of neurons. The result of the product of a given function x and a constant w is the function xw, which is obtained by multiplying all coefficients of the function x by the constant w. For the sum of the functions, we calculate a function formed by the sum of the coefficients. Since we examined ReLU networks in the evaluation, the activation function had to be extended to functions as well. To calculate the transformation, we used the method in the RefineZono article [8].

In the operation, we can distinguish 3 cases. If the lower bound of the interval is greater than 0, then ReLU is return with an original function, since there is no cut. If the upper bound is less than or equal to zero, then ReLU nullifies all coefficients of the input function, ensuring that the output is reduced to 0. On the other hand, if the input function contains the value 0, the symbolic function $\hat{y} = \hat{x} + x_{\text{new}}$ is calculated from the input interval and the calculated output (see Figure 2).

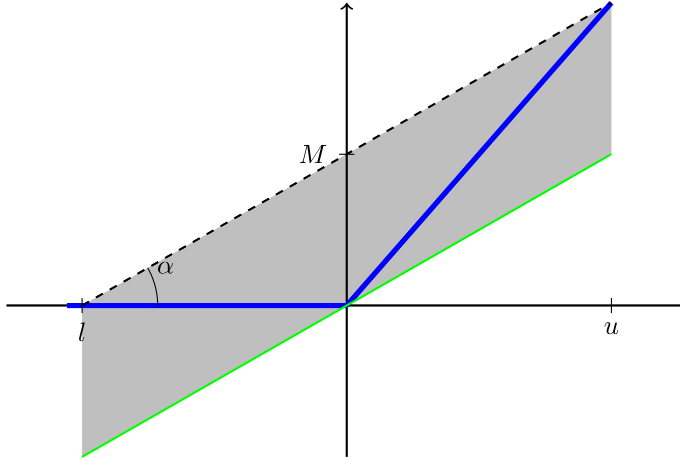

Figure 2. ReLU inclusion with Symbolic propagation.

The first step in determining y is the green line through 0 defined in Figure 2. To do this, we first calculate the slope of the line, which is arbitrarily $[l; u]$ in the case of an interval:

$$\lambda = \frac{u}{u - l}.$$

Then the coefficients of the input function x were multiplied by λ, and the result is a line passing through zero, which is the lower bound of inclusion. To determine the upper bound, the function $x + M$ must be calculated. For the result of the inclusion to be a function, a new variable is introduced. The output can be easily handled in the form of the function, only the coefficients need to be stored. Also, by introducing the new variables, we can handle a larger dependency. The value interval of the new variable can be calculated with the following formula:

$$x_{\text{new}} = [0, \lambda|l|].$$

The result of the transformation will be a new function $\hat{y} = \lambda\hat{x} + x_{\text{new}}$.

3.2. ReLU inclusion with 2 functions

The major advantage of the inclusion shown in Figure 2 is that it can be computed quickly, and the newly introduced variable makes it easier to handle dependency relationships in the computation. However, a disadvantage is that the inclusion

does not guarantee that the 0 lower bounds will be overestimated in the calculation of the lower bound. To deal with this, we also investigated an evaluation in which we maintain separate lower and upper inclusion functions for each neuron. The investigated to determine the value set of a neuron, both functions must be evaluated. The lower bound of the lower inclusion function gives the lower bound of the neuron. For the upper bound, we take the upper bound of the inclusion function.

In this case, we also had to handle the ReLU transformation. ReLU's input, in this case, was 2 functions that were evaluated to get the boundaries. If the lower bound is at least 0, the transformation leaves both functions unchanged. If the upper bound was less than 0, then the coefficients of the boundary function were set to zero by zero to obtain the interval $[0; 0]$. In the case where the interval encompasses zero, the inclusion shown in Figure 2 is calculated. By setting the coefficients of the lower bound function to zero, the lower bound 0 can be ensured so that no overestimation occurs here. Determining the upper bound for the method is shown in Figure 3 and proceeds similarly. First, we determine the value by which we multiply the above coefficients of the function λ, then we add the value of the shift M. Then we get the transformation:

$$\hat{y}_{\text{lower}} = 0,$$
$$\hat{y}_{\text{upper}} = \hat{x}\lambda + M.$$

The advantage of inclusion is that a zero lower bound can be guaranteed for output. However, the disadvantage is that a new variable was not introduced and use two separate limit functions, so the dependence information registered between neurons is reduced and the computing demand increases.

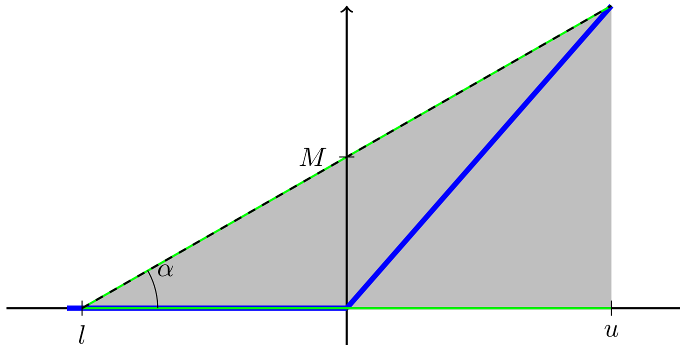

Figure 3. Layer transformation for separate boundary functions.

3.3. Affine representation of neurons

To further manage dependency information, we also explored a method in which input intervals were recorded in the affine form [7]. Then, any input x was treated

in the following form:

$$\hat{x} = x_0 + x_1\epsilon_1 \cdots x_n\epsilon_n.$$

The coefficients x_i are floating point values, and the ϵ_i are independent in the $[-1;1]$ interval is given. The x_0 value is used as the mean, and the ϵ_i are to be referred to as noise symbols.

The conversion between the affine and interval forms is easy to write. A x result interval in the case of an affine expression can be given in the following form: x_0 indicates the center of the interval and the radius can be calculated from the absolute sum of the corresponding coefficients.

$$x = [x_0 - r, x_0 + r], \text{ where } r = \sum_{i=1}^{n} |x_i|.$$

For an interval $x = [a;b]$ the corresponding affine form x can be calculated with the following relation: $\hat{x} = x_0 + x_k\epsilon_k$, where $x_0 = \frac{a-b}{2}$, $x_k = \frac{b-a}{2}$ and ϵ_k is a new variable.

The layer transformations can be easily extended in this case as well. The result of the product of an affine expression x and the constant α is an affine form where the coefficients of x have been multiplied by the value α. The sum of an affine expression and a constant value is also an affine form where the mean x_0 is increased by the specified number.

An extension of the ReLU transformation similar to the method shown in Figure 3. In the first step, we calculate the slope by which we multiply the coefficients of the input function. On the other hand, when introducing the new variables, so that the values remain in the range $[-1;1]$, we perform the following transformation:

$$[0, \lambda|l|] = \epsilon_k \left[0, \frac{\lambda|l|}{2}\right] + \frac{\lambda|l|}{2}, \text{ where } \epsilon_k = [-1;1].$$

Two methods have been used to handle numerical errors. One solution is the coefficients of the affine expression x are stored with interval inclusion and in the evaluation, and the transformations were calculated according to the rule of interval arithmetic. The other solution is to expand our expression with a new z_k k-error term during each operation. Here z_k is the upper bound of the absolute error in the coefficients of the affine expression and k is a completely new symbol.

4. Results

During the evaluation, our goal was to make as diverse as possible examples. Our methods were tested on different sizes, types, and quality networks. For this reason, the networks were examined on those published in the ERAN [9] system. 6 networks from the set, on which we also studied the runtime, output widths, and robustness. The parameters of each network are listed in Table 1. All trained networks have a ReLU activation function.

Table 1. Parameters of the ERAN networks.

Layer number	Layer width	Training method
3	50	regular
3	100	regular
4	1024	regular
6	500	regular
6	500	PGD (0.1) robust
6	500	PGD (0.3) robust

4.1. Results for 0.02 wide input intervals

The average total output width of the symbolic propagation (Table 3) was 0.41. For the symbolic system (Table 2), this value is 210, which is about a 510 times improvement over the naive method. The most significant reduction in width is the PGD (0.3) teaching network, where the improvement was nearly 3000 times. Overall, symbolic propagation produced the narrowest results in this case as well. Using the affine method (Table 4) produced an average improvement of about 108 times compared to the naive method. However, for the network with the most neurons (4X1024), we obtained outputs that were about 7 times wider than those of the symbolic propagation next to it. The results of the solution with separate constraint functions are visible in Table 5. Compared to the naive method, the average improvement was about 10 times.

Table 2. Naive interval arithmetic.

Network	Training	ϵ		Mean time(s)	
		$\epsilon = 0.02$	$\epsilon = 0.1$	CPU	GPU
3×50	regular	4.70	24.31	0.0006	0.0005
3×100	regular	9.62	53.41	0.001	0.0005
4×1024	regular	1088.51	5385.14	0.058	0.00089
6×500	regular	35.15	315.58	0.042	0.0012
6×500	PGD (0.1) robust	96.23	444.36	0.027	0.0012
6×500	PGD (0.3) robust	29.60	170.57	0.059	0.0012

4.2. Results for 0.1 wide input intervals

With wider input bounds, we obtain almost 578 times narrower intervals with symbolic propagation, than with the naive method. With the affine method, this quotient reduces to about 6. The average width with separate limit functions is 420, which turns out to be about 2.5 times narrower than with the naive method.

Table 3. Symbolic function propagation.

Network	Training	ϵ		Mean time(s)	
		$\epsilon = 0.02$	$\epsilon = 0.1$	CPU	GPU
3 × 50	regular	0.62	2.5	0.0038	0.0021
3 × 100	regular	0.56	2.41	0.006	0.0021
4 × 1024	regular	1.20	4.85	0.092	0.0046
6 × 500	regular	0.08	0.41	0.076	0.0046
6 × 500	PGD (0.1) robust	0.02	0.10	0.058	0.0047
6 × 500	PGD (0.3) robust	0.01	0.11	0.094	0.0046

Table 4. Affine propagation.

Network	Training	ϵ		Mean time(s)	
		$\epsilon = 0.02$	$\epsilon = 0.1$	CPU	GPU
3 × 50	regular	0.89	14.14	0.0045	0.0016
3 × 100	regular	1.06	28.48	0.0059	0.0016
4 × 1024	regular	9.35	843.41	0.093	0.0035
6 × 500	regular	0.19	131.77	0.061	0.0039
6 × 500	PGD (0.1) robust	0.06	15.67	0.056	0.0040
6 × 500	PGD (0.3) robust	0.05	13.39	0.062	0.0037

Table 5. Separated propagation.

Network	Training	ϵ		Mean time(s)	
		$\epsilon = 0.02$	$\epsilon = 0.1$	CPU	GPU
3 × 50	regular	1.15	12.83	0.179	0.0023
3 × 100	regular	1.85	26.91	0.325	0.0024
4 × 1024	regular	123.84	2260.3	11.10	0.0064
6 × 500	regular	2.83	101.89	25.21	0.0061
6 × 500	PGD (0.1) robust	3.67	82.27	5.99	0.0061
6 × 500	PGD (0.3) robust	1.36	36.38	50.91	0.006

4.3. Effect of robust training

For 6-layer networks with 500 neurons, we compared how robust training affects the average output width. During the test, we calculated the robust of the average output widths of the simple trained network and the robustly trained network, which represents the degree of improvement compared to the simple training method. Table 6 shows the symbolic propagation (S_ IA), the affine method (S_ AFF), and the results of the evaluation under separate boundary functions (S_ SEP). Robust teaching had a positive effect on the increase in initial width in almost all cases. The reason is that the networks trained in this way are prepared for a given

input change in their environment.

Table 6. Roboust teaching effect.

Methods	Training	ϵ	Mean improvement
S_IA	PGD(0.1)	0.02	3.46
		0.1	3.88
	PGD(0.3)	0.02	9.40
		0.1	10.95
S_AFF	PGD(0.1)	0.02	3.27
		0.1	10.80
	PGD(0.3)	0.02	16.10
		0.1	290.90
S_AFF	PGD(0.1)	0.02	0.74
		0.1	1.23
	PGD(0.3)	0.02	13.59
		0.1	3.49

5. Conclusion

In this work, we demonstrate the effectiveness of different techniques for multiple neural networks. The CPU/GPU time of the algorithms was shown on self-trained networks of different sizes and on ERAN networks. We will also separately explain how the computation time evolves in the case of networks trained with other robust techniques. We hope that these methods will be more effective in training than the interval method used robust training [14].

Acknowledgements. Support by the the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National.

References

[1] G. Alefeld, G. Mayer: *Interval analysis: theory and applications*, Journal of Computational and Applied Mathematics 121.1 (2000), pp. 421–464, ISSN: 0377-0427, DOI: 10.1016/S0377-0427(00)00342-3, URL: https://www.sciencedirect.com/science/article/pii/S0377042700003423.

[2] R. Bunel, J. Lu, I. Turkaslan, P. H. S. Torr, P. Kohli, M. P. Kumar: *Branch and Bound for Piecewise Linear Neural Network Verification*, J. Mach. Learn. Res. 21 (2019), 42:1–42:39.

[3] T. Csendes, N. Balogh, B. Bánhelyi, D. Zombori, R. Tóth, I. Megyeri: *Adversarial Example Free Zones for Specific Inputs and Neural Networks*, in: International Conference on Applied Informatics, 2020.

[4] G. Kovásznai, K. Gajdár, N. Narodytska: *Portfolio solver for verifying binarized neural networks*, Annales Mathematicae et Informaticae 53 (2021), Cited by: 0; All Open Access, Bronze Open Access, Green Open Access, pp. 183–200, doi: 10.33039/ami.2021.03.007.

[5] R. Moore, R. Kearfott, M. Cloud: *Introduction To Interval Analysis*, Cambridge Uni Press (CUP), 2009.

[6] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, T. Walsh: *Verifying Properties of Binarized Deep Neural Networks*, Proceedings of the AAAI Conference on Artificial Intelligence 32 (Sept. 2017), doi: 10.1609/aaai.v32i1.12206.

[7] E. Remm, M. Goze: *Affine structures on abelian Lie groups*, Linear Algebra and its Applications 360 (2003), pp. 215–230, issn: 0024-3795, doi: 10.1016/S0024-3795(02)00452-4, url: https://www.sciencedirect.com/science/article/pii/S0024379502004524.

[8] G. Singh, T. Gehr, M. Püschel, M. T. Vechev: *Boosting Robustness Certification of Neural Networks*, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019, url: https://openreview.net/forum?id=HJgeEh09KQ.

[9] G. Singh, T. Gehr, M. Püschel, M. T. Vechev: *Boosting Robustness Certification of Neural Networks*, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019, url: https://openreview.net/forum?id=HJgeEh09KQ.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, R. Fergus: *Intriguing properties of neural networks*, in: 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, ed. by Y. Bengio, Y. LeCun, 2014, url: http://arxiv.org/abs/1312.6199.

[11] V. Tjeng, K. Y. Xiao, R. Tedrake: *Evaluating Robustness of Neural Networks with Mixed Integer Programming*, in: International Conference on Learning Representations, 2017.

[12] S. Wang, K. Pei, J. Whitehouse, J. Yang, S. Jana: *Formal Security Analysis of Neural Networks Using Symbolic Intervals*, in: Proceedings of the 27th USENIX Conference on Security Symposium, SEC'18, Baltimore, MD, USA: USENIX Association, 2018, pp. 1599–1614, isbn: 9781931971461.

[13] S. Wang, K. Pei, J. Whitehouse, J. Yang, S. S. Jana: *Efficient Formal Safety Analysis of Neural Networks*, in: Neural Information Processing Systems, 2018.

[14] K. Xiao, V. Tjeng, N. Shafiullah, A. Mądry: *Training for faster adversarial robustness verification via inducing Relu stability*, in: International Conference on Learning Representations, May 2019.

[15] X. Xie, K. Kersting, D. Neider: *Neuro-Symbolic Verification of Deep Neural Networks*, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, ed. by L. D. Raedt, Main Track, International Joint Conferences on Artificial Intelligence Organization, July 2022, pp. 3622–3628, doi: 10.24963/ijcai.2022/503.

[16] D. Zombori, B. Bánhelyi, T. Csendes, I. Megyeri, M. Jelasity: *Fooling a Complete Neural Network Verifier*, in: International Conference on Learning Representations, 2021.