Dynamic and Scalable Deep Neural Network Verification Algorithm

Mohamed Ibn Khedher[1], Hatem Ibn-Khedher[2] and Makhlouf Hadji[2]
[1]*IRT - SystemX, 8 Avenue de la Vauve, 91120 Palaiseau, France*
[2]*Université de Paris, Lipade, F-75006 Paris, France*

Keywords: Feed-forward Neural Network, Neural Network Verification, Big-M Optimization, Robustness.

Abstract: Deep neural networks have widely used for dealing with complex real-world problems. However, a major concern in applying them to safety-critical systems is the great difficulty in providing formal guarantees about their behavior. Verifying its behavior means study the evolution of its outputs depending on the variation of its inputs. This verification is crucial in an uncertain environment where neural network inputs are noisy. In this paper, we propose an efficient technique for verifying feed-forward neural networks properties. In order to quantify the behavior of the proposed algorithm, we introduce different neural network scenarios to highlight the robustness according to predefined metrics and constraints. The proposed technique is based on the linearization of the non-convex Rectified Linear Unit (ReLU) activation function using the Big-M optimization approach. Moreover, we contribute by an iterative process to find the largest input range verifying (and then defining) the neural network proprieties of neural networks.

1 INTRODUCTION

Deep Neural networks have been widely used in many applications(Jmila et al., 2019) , such as image classification (Khedher et al., 2018), telecommunications, robot navigation and control of autonomous systems (Bunel et al., 2018).
Currently, despite the huge effort in deep neural networks deployment in real time applications, the overall configuration requires an intelligent tuning process across the input/output bounds that can secure and verify the final constraints.

Taken the example of real-time autonomous system, Deep Neural Networks (DNN) can be used for several tasks :

- The perception of a vehicle: detection and recognition of obstacles such as pedestrians, traffic signs, road markings, etc.
- Driver status monitoring: eye direction, head angle, heart and respiratory rates, etc.
- Decision-making depending on the environment of a vehicle: lane change, speed and steering angle calculation, etc.

It is worth mentioning here that in most application fields, the predicted decision of neural networks (i.e., detecting the presence/absence of a pedestrian, changing or keeping the lane, etc.) has a serious impact on the safety of the driver, the safety of passengers and other users of the road. For instance, detecting the absence of the pedestrian while he is actually present can cause a serious accident.

Despite the power of Deep Neural Networks that can process large inputs, analyse big data, and recommend some tuning or decisions, the slight perturbation in the input space can lead to a bad decision and worst use cases (Bunel et al., 2018). These poor decisions are generally caused by the disruption of the environment, hence the need for a share of metrics allowing the evaluation of the Neural Network to keep security proprieties faced the uncertainty of the environment.

The study of the security of a DNN consists in verifying the capacity of the Neural Network to take the same decision for all similar data, despite the attacks they can undergo. An attack is defined as any noise that can disrupt the neural network. The checker (the system verifying the security of the DNN) takes an input, a data and an attack, and delivers an information in the form of "*secure data faced the attack*" or "*non-secure data faced the attack*". Therefore, a Neural Network Verification (NNV) approach is needed to provide robustness metrics to neural networks.

The principle of Neural Network Verification is to find out, from the input data, all possible data resulting from the attack (noisy data), and verify that the

Khedher, M., Ibn-Khedher, H. and Hadji, M.
Dynamic and Scalable Deep Neural Network Verification Algorithm.
DOI: 10.5220/0010323811221130
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 1122-1130
ISBN: 978-989-758-484-8

Neural Network properties remain valid for the noisy data. However, since the input space comes with very large size, it is not feasible (in acceptable times) to check all possible inputs. Even networks that perform well on a large sample of inputs may not correctly generalize to new situations and may be vulnerable to adversarial attacks.

The NNV task faced several challenges related to:

- How can we fix the uncertainty interval of the input data, i.e in other words in which range of input data, we should have to verify the properties of Neural Network ?
- How can we mathematically formulate a physical attack?
- How can we translate the NNV problem to a easy mathematical formulation that can be solved using existing libraries ?

In this paper, we are interested in the third challenge about the mathematical formulation of the NNV task. Mathematically, a Neural Network represents the function that maps inputs to outputs through a sequence of layers. At each layer, the input to that layer undergoes a linear transformation followed by a simple nonlinear transformation before being passed to the next layer. These nonlinear transformations are often called activation functions, and a common example is the rectified linear unit (ReLU), which transforms the input by setting any negative values to zero.

Our contribution in NNV formulation, consists in proposing a new scalable and adaptive algorithm, which is based on Linear Programming (LP) technique that converges to optimal solutions in negligible times. Hence, our exact (i.e. that converge always to the optimum) approach based on LP formulation, is simplifying the non-linearities caused by ReLu function, by encoding them using binary variables. Then, the LP formulation can investigate all of the feasible solutions to find a counter example (if any) in order to verify the input/output constraints described in next sections.

The rest of the paper is organized as follows. In section 2, the principles of feed-forward neural networks are presented and backgrounds of the NNV task are described. In the section 3, a state of the art of approaches proposed to cope with the NNV task is presented. The structure of our approach is described in section 4. Section 5 includes the experimental results and section 6 concludes the paper.

2 DEEP NEURAL NETWORK VERIFICATION BACKGROUNDS

2.1 Deep Neural Networks

A Deep Neural Network is a extension of neural network with several hidden layers. It consists of three typical types of layers: *i*) an input layer, *ii*) some hidden layers of neuron computations and *iii*) an output layer. Each neuron is a simple processing element that responds to the weighted inputs it received from other neurons.

For a given neuron i ($i = 1, \ldots, n$), its action depends on its activation function provided by:

$$y_i = f\left(\sum_{j=1}^{n} w_{ij} * x_j + b_i\right) \tag{1}$$

where x_j is the j^{th} input of the i^{th} neuron, w_{ij} is the weight of the arc from the j^{th} input to the i^{th} neuron, b_i is the bias of the i^{th} neuron, y_i is the output of the i^{th} neuron and $f(.)$ is the activation function. The activation function is, mostly, a nonlinear function describing the reaction of i^{th} neuron with inputs.

Among the main DNN instances, we quote Feed Forward Neural Network (FFNN), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN). In our paper, we focus on the Feed-Forward Neural Network.

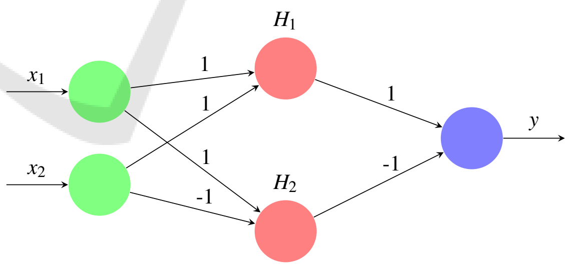

Figure 1: Example of neural network.

Fig.1 illustrates a simple example of feed-forward with only one hidden layer. Hence, the NNV problem in this figure, consists in answering some questions such as: Is there an input vector $\vec{x} = (x_1, x_2) \in [-2,2] \times [-2,2]$ where $y < -5$?

2.2 General Neural Networks Verification Problem

Given a deep neural network $N : x \rightarrow y$, a set of properties P covering the inputs and a set Q covering the outputs, the NNV problem is to answer the following question: Is there an input x resulting an output $y = N(x)$, verifying P and failing Q ?

In the rest of our paper, we consider:

- $P(x)$ represents the constraints fixed on inputs (for example, input should belongs a predefined range).
- $Q(x)$ represents constraints fixed on outputs.

For sake of clarity, we take the example of Fig.1 to illustrate P and Q that can be written as follows:

- $P(\boldsymbol{x}) = (x_1 > -2) \wedge (x_1 < 2) \wedge (x_2 > -2) \wedge (x_2 < 2)$
- $Q(y) = (y > -5)$

3 RELATED WORK

The robustness of decision-making functions in uncertain environment is an important research domain. In the automotive context, the perturbation of environment can be caused by the failure of perception sensors. So to ensure operational safety and road safety, it is crucial to assure the robustness of these systems faced sensor uncertainty. Recently, several research studies have demonstrated the sensitivity of the neural network against certain attacks. In this section, we present a state of the art of approaches proposed to verify a Neural Network, i.e evaluate the robustness of Neural Network in uncertain environment.

The proposed approaches can be grouped according to the formulation of the problem. We focus on the three following formulations: feasibility problem formulation, reachability problem formulation and optimization problem formulation.

3.1 Feasibility Problem Formulation

The principle of this approach (formulation) consists in converting the Neural Network to a feasibility problem for the existence of an counter-example. The algorithm, in this case, returns an information in the following form: "*a counter-example is found*" or "*no counter-example is found*".

The verification process includes a set of mini-tasks:

1. Convert the neural network to a set of equations using the relationship between neurons of successive network layers.
2. Add an infeasibility constraint (example:$\neg Q(\bar{y})$, where $\neg$ is the logical negation operator).
3. Search of a counter-example:
   - If a counter-example is found , the initial problem is called non-feasible.
   - If no counter-example is found, the initial problem is called feasible.

As described in 2.1, a Neural Network includes activation functions which are generally non-linear (e.g., $\tanh(x)$, $\text{Sigmoïde}(x) = \frac{1}{1+e^{-x}}$ and $\text{ReLU}(x) = \max(x, 0)$). This non linearity makes the complexity of the problem NP-Hard. In this case, the contribution of different articles/papers in the literature consists in proposing a solution to linearize activation functions (see Table 1).

In (Katz et al., 2017), authors propose to adapt the simplex algorithm, a standard algorithm for solving linear programming, to non-linear activation functions, specifically to support the ReLU function (ReLU for " Rectified Linear Unit"). The algorithm is called Reluplex, i.e. ReLU with the simplex method. Reluplex uses the simplex algorithm to search a feasible activation pattern that leads to an in-feasible output. The principle of Reluplex is to solve a system of equation from an initial assignment. At each iteration, the algorithms attempts to correct certain constraints violation. In fact, from one iteration to another, variable assignment can violate constraints.

In (Bunel et al., 2018), the authors propose PLANET (for "a Piece-wise LineAr feed-forward NEural network verification Tool"). It consists first in replacing the non-linear functions of the Neural Network by a set of linear equations. Then, the algorithm tries to find a solution for the resulting system of equations.

3.2 Reachability Problem Formulation

The reachability problem can be formulated as follows: given a computational system with a set of allowed rules, decide whether a certain state of a system is reachable from a given initial state of the system. The process of verification includes three main steps:

1. Compute the input set X defined as all possible inputs that system can takes as input.
2. Compute the output reachable set Y defined as: $Y = \{y |\ y = N(x), x \in X)\}$, where N is the Neural Network function.

Table 1: Summary of research work related to NNV.

Reference	Principle	Year
	Feasibility formulation	
(Katz et al., 2017)	Using the simplex algorithm	2017
(Ehlers, 2017)	Linearization of the nonlinear functions	2017
(Bunel et al., 2018)	Using the heuristic Branch & Bound	2018
	Reachability formulation	
(Xiang et al., 2018)	Approximation of the reachability set by a sensitivity study	2018
(Gehr et al., 2018)	Over-approximation of the reachability set using the abstract interpretation	2018
(Xiang et al., 2018)	Exact calculation of all reachability	2018
	Approximation formulation	
(Lomuscio and Maganti, 2017)	Resolution of a linear program by Gurobi algorithm	2017
(Tjeng et al., 2019)	Search of the maximum disturbance to distort Neural Network	2019
(Dvijotham et al., 2018)	Approximation using Lagrangian relaxations	2018
(Wong and Kolter, 2018)	Approximation using convex relaxation	2018
(Raghunathan et al., 2018)	Convex optimization by the positive semi-definite method	2018

3. Check if Y satisfies the constraints Q (constraints fixed on outputs).

Moreover, the determination of the reachable set Y can be taxonomized into two main categories:

1. Exact (optimal) approaches: to model inputs, no assumption is considered. This type of approach look for the exact set of reachable output.
2. Approximate (near-optimal) approaches: to model inputs, assumptions are considered. The input set is generally over approximated using standard geometric shapes. In fact, the reachable set is not the exact output set but a over approximation.

In (Xiang et al., 2018), only neural network with ReLU activation function can be considered. To determine the exact reachable set, the authors assume that the input and output are represented by the union set of polytopes. Moreover, any over-approximation is applied. Hence, the number of polytopes grows exponentially with each layer.

In (Gehr et al., 2018), the authors propose ”Abstract Interpretation for Artificial Intelligence (AI2)” technique. It consists in over-approximating inputs using geometric shapes namely zonohedron. Then, each Neural Network layer is converted to a equivalent abstract layer. Finally, the input shape evolves through the layers of the network. The, output shape is called the reachable set.

3.3 Optimization Problem Formulation

The optimization problem consists in converting the Neural Network to a set of conjunction or dis-junction of linear properties. Given the example of a ”Fully-connected” layer, it can be represented by a chain of conjunctions as follows. As notation, the variables x_i represents the output vector of layer i. The relationship between successive layers (x_{i-1} and x_i) can be encoded by the constraint C_i as follows:

$$C_i \equiv \{x_i^j = w_i^j * x_{i-1} + b_i^j\}$$

where w_j^i is the j^{th} column of w_i, w is the weight matrix and b is the bias matrix. After encoding all layers, the Neural Network is represented by the junction of all conjunctions. Several solutions have been proposed to solve the previously obtained system, and can be grouped into two groups: primal problem based and dual problem based.

Regarding primal-based formulation, this type of approaches consists in solving directly system equations using linear programming. In (Lomuscio and Maganti, 2017) authors convert the Neural Network to a set of constraints. Then the algorithm ***Gurobi*** is applied to find a solution. In (Tjeng et al., 2019), authors seek the maximum perturbation to miss-classify the Neural Network using the Mixed-integer linear programming (MILP).

Regarding dual-based approaches, the idea is to use relaxations (approximations) of linear equations to solve the optimization problem. In (Dvijotham et al., 2018), authors propose the use of Lagrangian relaxation to approximate bounds of neuron values. In (Wong and Kolter, 2018), to estimate bounds, authors use convex relaxations. The authors of (Raghunathan et al., 2018) propose the use of positive semi-definite optimization to approximate bounds.

4 THE PROPOSED APPROACH

In this section we describe the proposed adaptive and scalable neural network verification algorithm. Table 2 defines the NNV parameters used to formulate the problem of Figure 1. We consider a neural network as an input. It is encoded as series of data inputs ranging from lower to upper bounds. Moreover, neural network nodes have activation functions applied at the output of an artificial neural node in order to transform the incoming flow into another domain. It is worth mentioning here that the considered activation function is the *ReLU* function. For sake of clarity, *ReLU* function is given by:

$$ReLU(X) = \max\{X;0\} \tag{2}$$

where X represents the incoming flow at that artificial node.

ReLU is a non-linear activation function. Therefore, we propose to consider the *bigM* technique as an automated encoder that linearizes the hidden constraint. It is a mixed integer linear programming transformation that exactly transforms non-linear constraints into linear inequalities. More details on *bigM* are given in the sequel.

4.1 Generalized Big-M Approach in Deep Neural Networks

To solve the NNV problem, we propose a general mathematical formulation to handle with various deep neural networks. In other words, we consider different neural networks sizes with large data input, several hidden layers, and outputs representing the decision. In Fig. 2, and for sake of clarity, we depict a typical example to be considered in our modelling.

Therefore, we consider a neural network with m layers noted by $\{L_1, L_2, \ldots, L_m\}$. In each layer, we consider n neurons represented by nodes in Fig.2. There exists an arc (i, j) between each neuron i in a layer L_s and j in a different layer L_t $(s \neq t)$. This arc is weighted by w_{ij} as depicted by Fig.2. Moreover, for each neuron j (node in the graph of Fig.2), we consider two main variables $a_{in}(j)$ and $a_{out}(j)$ given by the following:

1. if $j \in L_1$, hence we have the two following inputs:
   - $a_{in}(j) = x_j$
   - $a_{out}(j) = x_j$
2. if $j \in L_k$ where $2 \leqslant k \leqslant m-1$, hence we have:
   - $a_{in}(j) = \sum_{i\in\Gamma^-(j)} w_{ij} a_{out}(i)$
   - $a_{out}(j) = \max\{a_{in}(j);0\}$
3. if $j \in L_m$ (last layer in our neural network):
   - $a_{in}(j) = \sum_{i\in\Gamma^-(j)} w_{ij} a_{out}(i)$
   - $a_{out}(j) = \max\{a_{in}(j);0\}$: not concerned in our scenarios.

where $\Gamma^-(j)$ indicates the set of predecessor nodes of j in the considered neural network.

According to the previous modelling, we propose in the following the final mathematical model that will serve to identify if our system has at least one solution or not:

$$\begin{array}{l}
\max Z = Constant \\
S.T.: \\
\forall j \in L_1: \\
a_{in}(j) = x_j \\
a_{out}(j) = x_j \\
\forall j \in L_k (2 \leqslant k \leqslant m-1): \\
a_{in}(j) = \sum_{i\in\Gamma^-(j)} w_{ij} a_{out}(i) \\
a_{out}(j) = \max\{a_{in}(j);0\} \\
\forall j \in L_m: \\
a_{in}(j) = \sum_{i\in\Gamma^-(j)} w_{ij} a_{out}(i) \\
\\
a_{in}(j) = \sum_{i\in\Gamma^-(j)} w_{ij} a_{out}(i) \leqslant \beta \\
\\
a_0 \leqslant x_j \leqslant b_0
\end{array} \tag{3}$$

where a_0 and b_0 are real inputs ($a_0, b_0 \in \mathbb{R}$) and $\beta \in \mathbb{R}$ is a parameter known before the optimization process.

Verifying the existence of any violation of the mathematical formulation (3) is equivalent to solve this system of non linear inequalities and some linear equalities and hence verifying if certain constraints are violated by others or not.

In system (3), inequalities using to determine the maximum between 0 and $a_{in}(j)$ (for a given neuron j) are non-linear and necessitate to be linearized to facilitate solving the model in negligible times.

In the sequel, we propose a linearization approach based on Big-M technique to totally eliminate non-linear equalities in the mathematical formulation (3). We consider, for instance, the following non-linear equality (for a given j):

$$a_{out}(j) = \max\{a_{in}(j);0\} \tag{4}$$

We introduce a new binary variable $\theta \in \{0,1\}$ to discuss the different cases that can be resulted from (4). In fact, we consider:

$$a_{out}(j) = \begin{cases} a_{in}(j), \text{ if } a_{in}(j) > 0 \\ 0, \; else \end{cases}$$

Hence, we propose:

$$a_{out}(j) \geqslant (\theta - 1)M + a_{in}(j) \tag{5}$$

$$a_{out}(j) \leqslant (1-\theta)M + a_{in}(j) \tag{6}$$

$$a_{out}(j) \leqslant \theta \times M \tag{7}$$

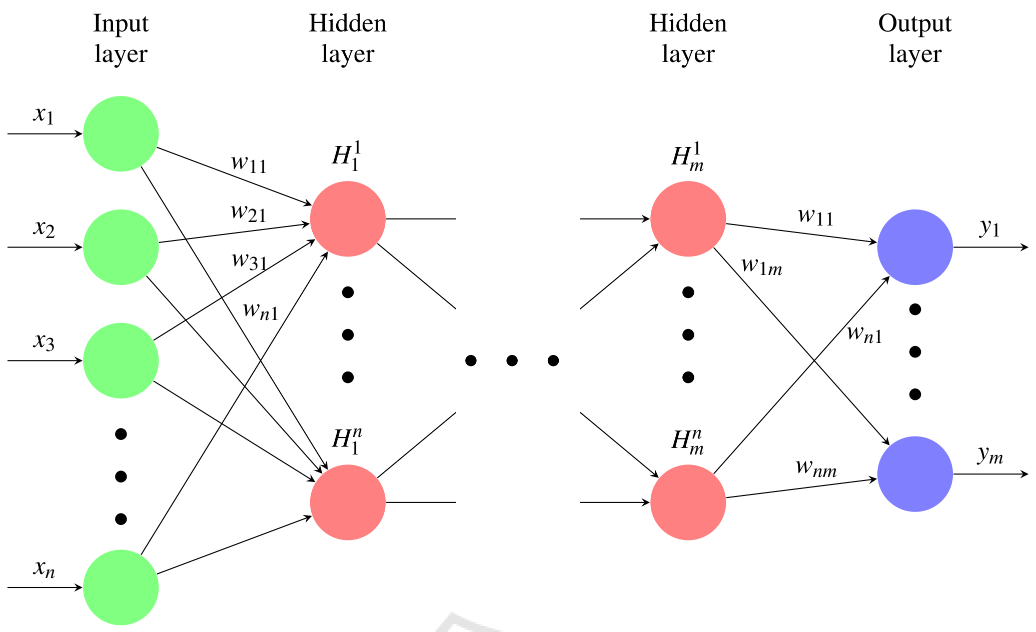

Figure 2: Example of neural network.

Table 2: Neural Network Verification Problem Notation.

Parameters	***Definition***
x_1	input 1 in L_1
x_2	input 2 in L_1
$a_{in}(H_1)$	input to the H_1
$a_{out}(H_1)$	non linear function of a_{in} (output of H_1)
$a_{in}(H_2)$	input to the H_2
$a_{out}(H_2)$	non linear function of a_{in} (output of H_2)
β	a parameter with a value of -5

$$a_{in}(j) \geqslant (\theta - 1) \times M \tag{8}$$
$$a_{in}(j) \leqslant \theta \times M \tag{9}$$
$$\theta \in \{0,1\} \tag{10}$$

Using the formulations (5) to (9), one can verify, according to the values of θ, that we can attend the same results than those of equation (4).

For sake of clarity, we propose to discuss deeply the example of Figure 1, and hence we apply the result of the proposed and generalized mathematical formulation (3) on this example (see Figure 1).

4.2 BigM Approach: Application on Example of Figure 1

We consider the example of Fig.1, the Neural Network consists of three layers L_1, L_2, L_3, with one node in the L_3 (output layer). Table 2 is summarizing the different parameters of the example of Fig.1.
The Big-M approach for the example of Fig.1 can be formulated as follows (according to system (3)):

$$\max\ Z = Constant \tag{11}$$

$Subject\ to$

$$a_{in}(H_1) = x_1 + x_2 \tag{12}$$
$$a_{in}(H_2) = x_1 - x_2 \tag{13}$$
$$a_{out}(H_1) = \max(a_{in}(H_1), 0) \tag{14}$$
$$a_{out}(H_2) = \max(a_{in}(H_2), 0) \tag{15}$$
$$a_{out}(H_1) - a_{out}(H_2) \leqslant -5 \tag{16}$$
$$-2 \leqslant x_1 \leqslant 2 \tag{17}$$
$$-2 \leqslant x_2 \leqslant 2 \tag{18}$$

This formulation (from (12) to (18)) is non linear as it contains at least two non-linear *ReLU* activation functions or equations given by (14) and (15). Therefore, we propose to linearize this model to obtain a solution if exists. Hence, equation (14) will be formulated as follows.

$$\begin{aligned}
&a_{out}(H_1) \geqslant (\theta_1 - 1)M + (x_1 + x_2)\\
&a_{out}(H_1) \leqslant (1 - \theta_1)M + (x_1 + x_2)\\
&a_{out}(H_1) \leqslant \theta_1 \times M\\
&x_1 + x_2 \geqslant (\theta_1 - 1) \times M\\
&x_1 + x_2 \leqslant \theta_1 \times M\\
&\theta_1 \in \{0,1\}
\end{aligned} \tag{19}$$

Similarly, equation (15) is linearized as the following

$$\begin{aligned}
&a_{out}(H_2) \geqslant (\theta_2 - 1)M + (x_1 - x_2)\\
&a_{out}(H_2) \leqslant (1 - \theta_2)M + (x_1 - x_2)\\
&a_{out}(H_2) \leqslant \theta_2 \times M\\
&x_1 - x_2 \geqslant (\theta_2 - 1) \times M\\
&x_1 - x_2 \leqslant \theta_2 \times M\\
&\theta_2 \in \{0,1\}
\end{aligned} \tag{20}$$

Proposition 4.1. *The mathematical formulation concerning equations (inequalities) from (12) to (18) has* ***no solution*** *in the defined domain.*

Proof. We start the proof using the following mathematical formulation (after the proposed linearization), in which we substitute a_{in}, and b_{in} by $x_1 + x_2$ and $x_1 - x_2$ respectively. We obtain:

$$\begin{aligned}
&\max Z = Constant\\
&S.T.:\\
&a_{out}(H_1) \geqslant (\theta_1 - 1)M + (x_1 + x_2)\\
&a_{out}(H_1 \leqslant \theta_1 \times M\\
&x_1 + x_2 \geqslant (\theta_1 - 1) \times M\\
&x_1 + x_2 \leqslant \theta_1 \times M\\
&a_{out}(H_2) \geqslant (\theta_2 - 1)M + (x_1 - x_2)\\
&a_{out}(H_2) \leqslant \theta_2 \times M\\
&x_1 - x_2 \geqslant (\theta_2 - 1) \times M\\
&x_1 - x_2 \leqslant \theta_2 \times M\\
&a_{out}(H_1) - a_{out}(H_2) \leqslant -5\\
&-2 \leqslant x_1 \leqslant 2\\
&-2 \leqslant x_2 \leqslant 2
\end{aligned} \tag{21}$$

We suppose that $\theta_1 = \theta_2$. If we consider the difference between the **first** inequality of the model (21) and the **fifth** inequality of the same model, then we obtain:

$$a_{out}(H_1) - a_{out}(H_2) \geqslant 2x_2 \tag{22}$$

At the same time, we also considered:

$$a_{out}(H_1) - a_{out}(H_2) \leqslant -5 \tag{23}$$

Hence, the addition (22)+(23) will lead to:

$$x_2 \leqslant \frac{-5}{2} \tag{24}$$

The result (24) contradicts the domain of x_2 provided by (18). Hence, there is no solution for this system in the respective domains of x_1 and x_2. □

5 PERFORMANCE EVALUATION

5.1 Neural Network Configuration Setting

The final system of equations (3) is implemented using IBM CPLEX optimization tool and considering a constant objective function to optimize. We have considered a feed-forward neural network architecture. We show in Table 3 hereafter the used simulation parameters.

Table 3: Neural Network Configuration Setting.

Simulation Parameters	***Values***
m	from 50 to 100 which models most of the neural networks (Carlini and Wagner, 2018)
n	50 and 100 neurons
Type	Fully connected
M or $Big-M$	infinity
θ	Binary decision variable (0 or 1)
$x_j\ j \in \{1,\ldots,n\}$	normalized and scaled data inputs
y_j	classes or predicted values. We specify single and multi output neurons

5.2 Impact of n and m on Convergence Time

In this section, we investigate the impact of varying the input size n in the verification optimization instances. Hence, we address two scenarios: *i*) a first scenario of 50 input neurons and *ii*) a second scenario of 100 input neurons.

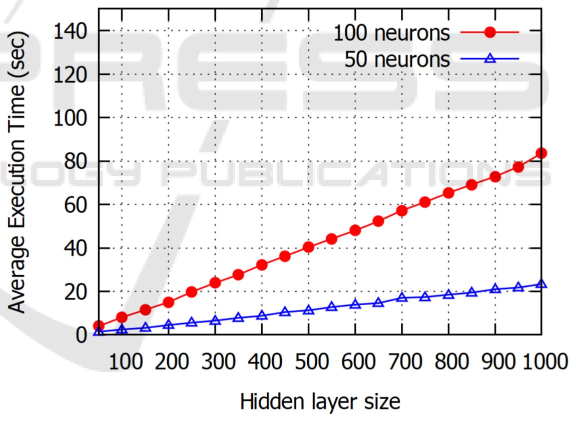

Figure 3: Execution time of different input sizes.

Figure 3 depicts the necessary convergence time of our approach for a hidden layer size (m) in the $[20, 1000]$ interval. We considered the two scenarios of $n = 50$ and $n = 100$ neurons to illustrate the negligible necessary execution time which approximates 80 seconds in the worst case ($m = 1000$, and $n = 100$) to reach the optimal solution. This is due to the efficiency of our complete mathematical formulation to converge rapidly to optimal solutions even for large problem instances. Hence, we show the feasibility of our Big-M based verification algorithm for the considered scenarios and illustrate clearly the scalability of this method. Indeed, our mathematical formulation is linear and its resolution is based on a

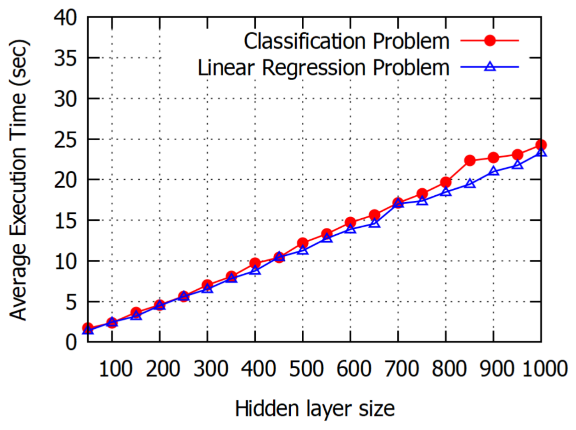

Figure 4: Execution time of linear regression and classification problems.

simple relaxation of the ReLU equalities leading to deeply simplifying the initial considered problem.

In the following, and to highlight the efficiency of our approach, we consider two verification problems based on linear regression and classification.

5.3 Linear Regression and Classification Problems Verification

We consider linear regression and classification models (the set of problem constraints) in order to evaluate the time complexity of the proposed Big-M based verification algorithm. We set in both problems 50 neuron processors as the input size (it represents a typical configuration parameter).

We run 100 times the two scenarios using our Big-M based approach and take the average value of the necessary execution time (in seconds) to obtain the optimal solutions. Figure 4 depicts the same evolution of the convergence time for the two problems when the number of hidden layers is in the $[100, 800]$ interval. For hidden layers with size between 800 and 900 neurons, we can observe a slight difference in favor of the linear regression problem. Indeed, this small difference is due to considering more binary variables in the formulation of the classification problem, which necessitates more time to converge to the optimal solution. For more than 900 hidden layer neurons, the average execution time for the classification problem converges to the average necessary time of the linear regression problem, which confirms the efficiency and the scalability of our proposed Big-M-based approach to the optimal solution even for large instances (the worst case in Figure 4 converges in less than 25 seconds).

5.4 Single-output and Multi-output Verification Scenarios

To extend the illustration and the efficiency of our approach, we propose to consider two new scenarios addressing a neural network with a single output, and another scenario with multiple output. The verification of these scenarios using our approach consists in modifying the two last inequalities of the mathematical formulation (3).

- **Single Verification Scenario.** In this scenario, our neural network outputs a single value. The verification of the model is realized using a single output constraint (last constraint of (3) adapted to this scenario). It is formulated as follows:

$$y = a_{in}(o) = \sum_{i \in \Gamma^-(o)} w_{i,o} a_{out}(i) \qquad (25)$$

  Where o is the single output neuron at the output layer.

- **Multi-output Verification Scenario.** In this scenario, the considered neural network outputs a multi-output. The verification problem consists of multi-output constraints. Each output constraint may represent a verification problem instance. It is formulated as follows:

$$\forall j \in L_m, y(j) = a_{in}(j) = \sum_{i \in \Gamma^-(o_j)} w_{i,o_j} a_{out}(i) \qquad (26)$$

  Where o_j is the j^{th} output neuron at the output layer.

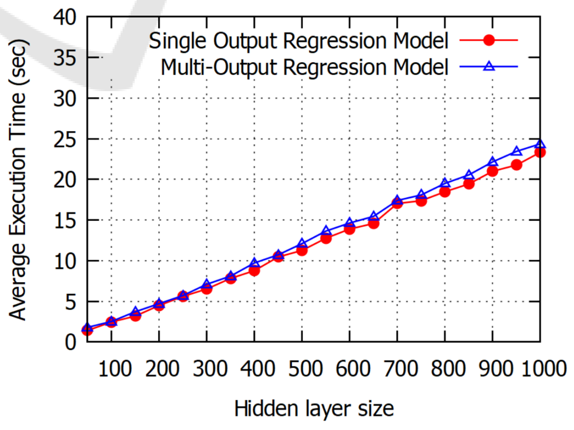

Figure 5: Execution time of single and multi-output problem instances.

Figure 5 depicts the behavior of the proposed Big-M optimisation problem according to the above scenarios. It shows that adding extra output constraints requires a slight and negligible (less than 3 seconds in the worst case) augmentation of execution time which proves the feasibility of the algorithm in complex scenarios.

6 CONCLUSION AND PERSPECTIVES

In this paper, we examined the safety of Neural Network against input perturbations i.e in an uncertain environment. Our challenge was to verify neural network output according to input range and provide a formal guarantees about its behavior. Hence, our contribution to the formulation of the verification problem is based on linear programming technique. We proposed an exact mathematical formulation and then eliminated the non-linearities by encoding them with the help of binary variables. In the numerical evaluation, different scenarios are discussed, and results show that our approach is feasible, in terms of convergence time, and scalable even for large neural networks.

Our approach considered only neural network with ReLU activation functions. In future work, we plan to extend our study to other activation functions, such as *Tanh*, Sigmoïde, etc. Moreover, we plan to validate our proposed approach on real use cases such as image classification, self driving, etc.

REFERENCES

Bunel, R., Turkaslan, I., Torr, P. H., Kohli, P., and Kumar, M. P. (2018). A unified view of piecewise linear neural network verification. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 4795–4804, USA. Curran Associates Inc.

Carlini, N. and Wagner, D. (2018). Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7.

Dvijotham, K., Stanforth, R., Gowal, S., Mann, T. A., and Kohli, P. (2018). A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 550–559.

Ehlers, R. (2017). Formal verification of piece-wise linear feed-forward neural networks. *CoRR*, abs/1705.01320.

Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and Vechev, M. (2018). Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*.

Jmila, H., Khedher, M. I., Blanc, G., and El-Yacoubi, M. A. (2019). Siamese network based feature learning for improved intrusion detection. In Gedeon, T., Wong, K. W., and Lee, M., editors, *Neural Information Processing - 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part I*, volume 11953 of *Lecture Notes in Computer Science*, pages 377–389. Springer.

Katz, G., Barrett, C. W., Dill, D. L., Julian, K., and Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 97–117.

Khedher, M. I., Jmila, H., and Yacoubi, M. A. E. (2018). Fusion of interest point/image based descriptors for efficient person re-identification. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.

Lomuscio, A. and Maganti, L. (2017). An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, abs/1706.07351.

Raghunathan, A., Steinhardt, J., and Liang, P. (2018). Certified defenses against adversarial examples. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Tjeng, V., Xiao, K. Y., and Tedrake, R. (2019). Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Wong, E. and Kolter, J. Z. (2018). Provable defenses against adversarial examples via the convex outer adversarial polytope. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5283–5292. PMLR.

Xiang, W., Tran, H., and Johnson, T. T. (2018). Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783.

Xiang, W., Tran, H.-D., and Johnson, T. T. (2018). Reachable set computation and safety verification for neural networks with relu activations. *In Submission*.