Mathematical Programming Approach for Adversarial Attack Modelling

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

Keywords: Neural Network, Adversarial Attack, Linear Programming (LP).

Abstract: An adversarial attack is defined as the minimal perturbation that change the model decision. Machine learning (ML) models such as Deep Neural Networks (DNNs) are vulnerable to different adversarial examples where malicious perturbed inputs lead to erroneous model outputs. Breaking neural networks with adversarial attack requires an intelligent approach that decides about the maximum allowed margin in which the neural network decision (output) is invariant. In this paper, we propose a new formulation based on linear programming approach modelling adversarial attacks. Our approach considers noised inputs while reaching the optimal perturbation. To assess the performance of our approach, we discuss two main scenarios quantifying the algorithm's decision behavior in terms of total perturbation cost, percentage of perturbed inputs, and other cost factors. Then, the approach is implemented and evaluated under different neural network scales.

1 INTRODUCTION

The past decade has witnessed the great rise of Artificial Intelligence (AI) and especially Deep Learning (DL). The success of deep learning, as Machine Learning (ML) classifier, has drawn great attention in the recent, particularly in computer vision applications(Khedher et al., 2018) , telecommunications (Jmila et al., 2017; Jmila et al., 2019) and control of autonomous systems (Bunel et al., 2018; Rao and Frtunikj, 2018; Kisacanin, 2017). A classifier is an ML model that learns a mapping function between inputs and a set of classes (Khedher et al., 2012; Khedher and El Yacoubi, 2015). For instance, an anomaly detector is a classifier taking as inputs a network traffic features and assigning them to the normal or abnormal class.

Despite the success of DNN deployment in real time applications, it shows a vulnerability to integrity attacks. Such attacks are often instantiated by adversarial examples: legitimate inputs altered by adding small, often imperceptible, perturbations to force a learned classifier to misclassify the resulting adversarial inputs, while remaining correctly classified by a human observer. 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. In fact, knowing the smallest disturbance gives us an idea of the level of robustness of DL in the face of adversary attacks (Cao et al., 2019; Sharif et al., 2016; Carlini and Wagner, 2018).

To illustrate, consider the following images, potentially consumed by an autonomous vehicle (Aung et al., 2017), these images appear to be the same to humans. In fact, our biological classifiers (vision) identify each image as a limit speed sign to 20 km/h. The image on the left is indeed an ordinary image of a limit speed sign. We produced the image on the right by adding a precise perturbation, blurring, that forces a particular DNN to classify it as a yield sign, i.e., as a limit speed sign to *80 km/h* (Aung et al., 2017).

Here, an adversary could potentially use the altered image to cause a car without failsafes to behave dangerously. This attack would require modifying the image used internally by the car through transformations of the physical traffic sign. It is thus conceivable that physical adversarial traffic signs could be gen-

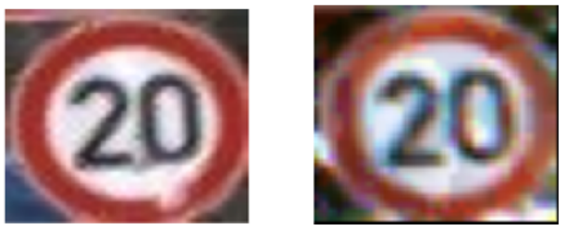

Figure 1: Adversarial attacks. left: initial image, right: Blurring (Aung et al., 2017).

Ibn-Khedher, H., Ibn Khedher, M. and Hadji, M.
Mathematical Programming Approach for Adversarial Attack Modelling.
DOI: 10.5220/0010324203430350
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 343-350
ISBN: 978-989-758-484-8

erated by maliciously modifying the sign itself, e.g., with stickers or paint. 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 the sake of clarity, detecting the absence of the pedestrian while he is actually present can cause a serious accident.

In this paper, we contribute to the search of adversarial attack. Compared to the state of the art, most of proposed approaches follow an iterative procedure where from one iteration to another, we are getting closer to the adversarial example. At the end of procedure, we are not sure to get an adverse example. Contrary, our approach consists in finding the smallest perturbation that changes the decision on DNN, i.e., an adverse example.

Our novel attack strategy is to convert the mapping between outputs and inputs to a linear system of equations. Our approach is based on linear programming (LP) technique and iteration process. Through the first LP optimization technique, we are going to formulate an exact algorithm based on a mathematical model. In the proposed solution, we eliminate the non-linearity by encoding them with the help of binary variables. Then, the iterative process can search a smallest perturbation in order to broke the input/output constraints.

The rest of the paper is organized as follows. In Section 2, the principles of feed-forward neural networks and adversarial attacks are presented. In the Section 3, a state of the art of adversarial attacks methods is discussed. The structure of our proposed approach is described in Section 4. Section 5 includes the experimental results and Section 6 concludes the work.

2 BACKGROUNDS

In this Section, we highlight the main architectures used in DNN and adverse attacks fields.

2.1 Deep Neural Networks

A Deep Neural Network (DNN) consists of a set of several hidden layers, in addition to an input and output layer. Each layer contains a set of neurons. Each of the neurons is connected to those of the neurons of the previous layer. Each neuron is a simple processing element that responds to the weighted inputs it received from other neurons (Shrestha and Mahmood, 2019).

There are several types of DNNs. In this paper, we are focusing on Feed-Forward Neural Network where each neuron in a layer is connected with all the neurons in the previous layer. These connections are not all equal: each connection may have a different strength or weight. The weights on these connections encode the knowledge of a network.
The action of a neuron depends on its activation function, which is described as:

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

where x_j is the j^{th} input of the i^{th} neuron, w_{ij} is the weight from the j^{th} input to the i^{th} neuron, θ_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.

2.2 Adversarial Attacks

An adversarial example is an instance with small perturbation that cause a machine learning model to make a false prediction. There are different ways to find an adversarial example. Most of them rely on minimizing the distance between the adverse example and the original one while making sure that the prediction is wrong. Attacks can be classified into two categories: white-box attacks and black-box attacks (Chakraborty et al., 2018),(Akhtar and Mian, 2018).

- **White-box Attack:** To generate a white-box attack, the attacker should have full access to the architecture and parameters of the classifier (gradient, loss function, etc.)
- **Black-box Attack:** To generate a black-box attack, the attacker does not have complete access to the classifier. In a black-box setting, the classifier parameters are unknown.

For simplicity, we consider X as the set of classifier inputs and Y the set of classifier outputs along K possible classes, $Y = \{1, \dots, K\}$. Finally, we note $C(x)$

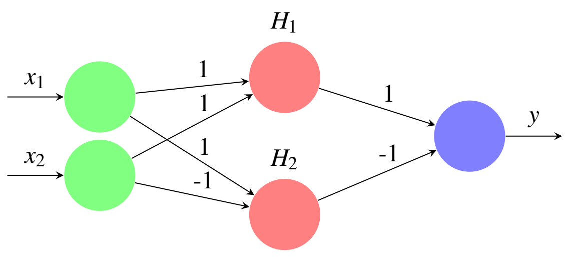

Figure 2: Example of neural network.

the class of x by the neural network $F(\cdot)$. Regardless of the type of attack, it is important to distinct between targeted and non-targeted attack.

- **Untargeted Attack:** an untargeted attack aims to misclassify the benign input by adding adversarial perturbation so that the predicted class of the benign input is changed to some other classes in Y without a specific target class.
- **Targeted Attack:** a targeted attack aims to misclassify the benign input x to a targeted (specific) class $y \in Y$ by adding an adversarial perturbation.

3 RELATED WORK: NEURAL NETWORK ATTACKS FIELD

In this Section, we highlight the relevant work about Neural Network Attacks (NNA) methods.

3.1 Fast Gradient Sign Method (FGSM)

Goodfellow et al. (Goodfellow et al., 2015) have developed a method for generating adverse sample based on the gradient descent method. Each component of the original sample x is modified by adding or subtracting a small perturbation ε. The adverse function ψ is expressed as following:

$$\begin{array}{cccc} \psi & : & X \times Y & \longrightarrow & X \\ & & (x, y) & \longmapsto & -\varepsilon \nabla_x \mathcal{L}(x, y) \end{array}$$

Thus, the loss function of the classifier will decrease when the class of the adverse sample is chosen as y. We then wish to find x' adverse sample of x such as: $\|x' - x\|_p \leqslant \varepsilon$.

It is clear that the previous formulation of FGSM is related to targeted attack case where y corresponds to the target class that we wish to impose on the original input sample. However, this attack can be applied in the case of untargeted attack, by considering the following perturbation:

$$\begin{array}{ccccc} \rho & : & X & \longrightarrow & X \\ & & x & \longmapsto & -\psi(x, C(x)) \end{array}$$

Thus, the original input x is modified in order to increase the loss function $\mathcal{L}$ when the classifier retains the same class $C(x)$. This attack only requires the compute of the loss function gradient, which makes it a very efficient method. On the other hand,vε is a hyper parameter that affect the x' class: if ε is too small, the $\rho(x)$ perturbation may have little impact on the x' class.

3.2 Basic Iterative Method (BIM)

Kurabin et al. (Kurabin et al., 2017) proposed an extension of the FGSM attack by iteratively applying FGSM. At each iteration i, the adverse sample is generated by applying FGSM on the generated sample at the $(i-1)^{th}$ iteration. The BIM attack is generated as the following:

$$\begin{cases} x'_0 = x \\ x'_{i+1} = x'_i + \psi(x'_i, y) \end{cases}$$

where y represents, in the case of a targeted attack, the class of the adverse sample and $y = C(x'_N)$ in the case of an untargeted attack. Moreover, ψ is the same function defined in the case of FGSM attack.

3.3 Projected Gradient Descent (PGD)

The PGD (Madry et al., 2017) attack is also an extension of the FGSM and similar to BIM. It consists in applying FGSM several times. The major difference from BIM is that at each iteration, the generated attack is projected on the ball $\mathcal{B}(x, \varepsilon) = \{z \in X : \|x - z\|_p \leqslant \varepsilon\}$. The adverse sample x' associated with the original one x is then constructed as following:

$$\begin{cases} x'_0 = x \\ x'_{i+1} = \Pi_\varepsilon (x'_i + \psi(x'_i, y)) \end{cases}$$

where Π_ε is the projection on the ball $B(x, \varepsilon)$ and ψ is the perturbation function as defined in FGSM. Likewise, y refers to the target label you wish to reach, in the case of targeted attack, or it refers to $C(x'_N)$ in the case of an untargeted attack.

3.4 Jacobian Saliency Map Attack (JSMA)

This attack differs from the previous ones, since Parpernot et al. (Papernot et al., 2015) did not rely on a gradient descent to build an adverse example but the idea is to disturb a minimal number of pixels, in the case of image input, according to a criterion. Initially proposed for a targeted attack, JSMA consists in controlling the number of pixels of an input image x (or the number of components of an input vector) that should be modified in order to obtain an adverse image associated with a target class y. Iteratively, JSMA consists in modifying pixels until the target class is obtained.

The idea behind is to, on one hand, increase $F_y(x)$ the probability of the target class y and on the other hand, decrease the probabilities of the other classes.

To do this, authors introduced the Saliency Map matrix as following:

$$S(x,y)^+[i] = \begin{cases} 0 \quad \text{si} \; \dfrac{\partial F_y(x)}{\partial x_i} < 0 \; \text{ou} \; \displaystyle\sum_{k \neq y} \dfrac{\partial F_k(x)}{\partial x_i} > 0 \\ \left(\dfrac{\partial F_y(x)}{\partial x_i}\right) \cdot \left| \displaystyle\sum_{k \neq y} \dfrac{\partial F_k(x)}{\partial x_i} \right| \quad \text{otherwise} \end{cases}$$

The Saliency Map is used as criterion to select pixels which should be modified. In fact, the way that Saliency Map is computed, allows to reject pixels that will not increase the probability of the target class y or will not decrease the probabilities of the other classes; for these pixels, the criterion is set to 0.

Indeed, for a pixel x_i, if $\dfrac{\partial F_y(x)}{\partial x_i} < 0$ means F_y will be decreasing by adding a positive term to pixel x_i and thus tend to decrease the probability of the target class y. Similarly, for the pixel x_i, if $\displaystyle\sum_{k \neq y} \dfrac{\partial F_k(x)}{\partial x_i} > 0$ means $\displaystyle\sum_{k \neq y} F_k$ will be increasing by adding a positive term to x_i in x_i and increase the probability of the other classes.

As described below, the proposed criterion acts on a single pixel. Nevertheless, another version of JSMA is proposed by authors and consists in considering a pair of pixels rather than single pixels. At each iteration, pair of pixels ($i_{\max}$ and $j_{\max}$) is found that satisfy :

$$\underset{(p,q)}{\arg\max} \left(\sum_{i=p,q} \frac{\partial F_y(x)}{\partial x_i} \right) \cdot \left| \sum_{i=p,q} \sum_{k \neq y} \frac{\partial F_k(x)}{\partial x_i} \right| \qquad (2)$$

As show above, an adversarial attack requires an enticed algorithm that decides about the maximum allowed perturbation after which the neural network model is no robust. Hereafter we introduce our proposed approaches using linear programming technique.

4 EXACT APPROACH FOR ADVERSARIAL ATTACK USING BIG-M

Most of the approaches consist in converging iteratively towards an adverse example. Obtaining this adverse example is not guaranteed at the end of the application of the algorithm. In fact, it depends on the number of iterations. Compared to the state of the art, our approach consists in determining and in an exact way, after a single iteration the minimum disturbance that disturbs the Neural Network.

The approach takes as input a given neural network. It is encoded as series of data inputs ranging from lower to upper bounds. 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 defined as follows: $ReLU(X) = \max\{X;0\}$, where X represents the incoming flow at that artificial node.

The *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.

After converting the neural network to a linear program, the second stage of our approach it find the small perturbation $\varepsilon \in \mathbb{E}^n$ that change the decision of neural network, where n the dimension of input. Moreover we are interesting in minimizing the number of the perturbed inputs. The approach based on formulating the neural network using linear equation that maps the output classes to the inputs. It is worth mentioning here that introducing perturbation to each inputs may change the decision of the neural network. Therefore, we propose optimal adversarial attack approach using linear programming techniques. Hereafter we describe the problem formulation, decision variables and algorithm constraints.

4.1 Adversarial Attack Problem Formulation

Let 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.

4.2 Decision Variables

We introduce the following decision variables:

- The binary decision variables y indicates if the data input $x_j, j \in N$, is perturbed. It is defined as follows

$$y_i = \begin{cases} 1 & \text{if the data input } x_j \text{ is perturbed} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

- The real variable ε decides about the perturbation value. It indicates the maximum value that does not change the decision (the ground truth in our case). It is defined as follows

$$\varepsilon_j = \begin{cases} > 0 & \text{if } x_j \text{ is perturbed} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

4.3 Algorithm Constraints

According to the previous formulations and inequalities linearization, we summarize in the following the whole constraints of our model:

- First layer constraint: introducing perturbation to the input neurons re-transforms the previous variables of the first layer as follows:

$$\forall j \in L_1 : a_{in}(j) = x_j + \varepsilon_j \tag{5}$$

$$\forall j \in L_1 : a_{out}(j) = x_j + \varepsilon_j \tag{6}$$

- Hidden layer constraint: the data inputs to a layer $m-1$ depends on the output of the previous layer m. It is formulated as follows:

$$\forall j \in L_k (2 \leqslant k \leqslant m-1) : a_{in}(j) = \sum_{i\in\Gamma^-(j)} w_{ij}a_{out}(i) \tag{7}$$

$$\forall j \in L_k (2 \leqslant k \leqslant m-1) : a_{out}(j) = \max\{a_{in}(j);0\} \tag{8}$$

- Output layer constraint: the output flow of the last layer is bounded by β. It is formulated as follows:

$$\forall j \in L_m : a_{in}(j) = \sum_{i\in\Gamma^-(j)} w_{ij}a_{out}(i) \tag{9}$$

$$\forall j \in L_m : a_{in}(j) = \sum_{i\in\Gamma^-(j)} w_{ij}a_{out}(i) \leqslant \beta \tag{10}$$

- Data inputs boundary: It is implied before and after perturbation and formulated as follows

$$a_{0j} \leqslant x_j \leqslant b_{0j} \tag{11}$$

where a_{0j} and b_{0j} represent lower and upper bounds for each neuron $j \in N$

- Perturbation constraint: decisions related to the data to be perturbed and the values of the perturbation are formulated as follows

$$\varepsilon_j \leqslant y_j \times (b_{0j} - x_j) \tag{12}$$

$$\varepsilon_j \geqslant y_j \times (a_{0j} - x_j) \tag{13}$$

- Perturbation boundary: the data values after the perturbation decision should not exceed the data inputs boundary mentioned above:

$$a_{0j} - x_j \leqslant \varepsilon_j \leqslant b_{0j} - x_j \tag{14}$$

- Non negativity of decision variables: While ε is a real variables, y_j is bounded as follows

$$y_j \in \{0,1\} \tag{15}$$

4.4 Objective Function Formulation

Considering different DNN use cases, the main objective functions in our proposed model is to minimize the number of perturbed data inputs while maximizing each perturbation (if it exists). It is formulated as follows:

$$\min Z = \left(\sum_{j\in n} y_j - \sum_{j\in n} \varepsilon_j \right) \tag{16}$$

4.5 Algorithm Constraints Linearization

The previous 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. We consider, for instance, the following non-linear equality (for a given j):

$$a_{out}(j) = \max\{a_{in}(j);0\} \tag{17}$$

We introduce a new binary variable $\theta \in \{0,1\}$ to discuss the different cases that can be resulted from (17). In fact, we consider :

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

Table 1: Neural Network Configuration Setting.

Simulation Parameters	***Values***
m	from 50 to 500 which models most of the neural networks (Carlini and Wagner, 2018)
n	10 and 20 neurons
$Type$	Fully connected feed-forward neural network
M or $Big-M$	infinity
x_j $j \in \{1,\dots,n\}$	normalized and scaled data inputs
Decision Variables	***Definitions***
y_j	Indicates if the data input x_j is perturbed
ε_j	perturbation associated with the input x_j
θ	Binary decision variable used for constraints linearization(0 or 1)

Hence, we propose:

$$\begin{aligned} &a_{out}(j) \geqslant (\theta-1)M + a_{in}(j) \\ &a_{out}(j) \leqslant (1-\theta)M + a_{in}(j) \\ &a_{out}(j) \leqslant \theta \times M \\ &a_{in}(j) \geqslant (\theta-1) \times M \\ &a_{in}(j) \leqslant \theta \times M \\ &\theta \in \{0,1\} \end{aligned} \tag{18}$$

Using the formulation 18, one can verify, according to the values of θ, that we can attend the same results than those of equation (17).

5 PERFORMANCE EVALUATION

5.1 Optimization Scenarios and Key Performance Indicators

We have considered a feed-forward neural network architecture with two neural networks scales (N) and different hidden layers (M). Table 1 summarizes the used simulation parameters in our models.
For the interest of assessing the efficiency of the exact ILP algorithm, we used CPLEX optimization tool [1] for the adversarial attack optimization in small neural network scale.

Further, in order to compare the Big-M based adversarial attack algorithm, we proposed two scenarios: 1) Linear objective function, and 2) Quadratic objective function. The two approaches are defined as follows:

- In linear objective function, the linear distance between perturbed and original inputs is minimized.
- In quadratic objective function, the Euclidean distance (i .e ., the l^2 norm) between perturbed and original inputs is minimized.

[1]https://pypi.org/project/cplex/

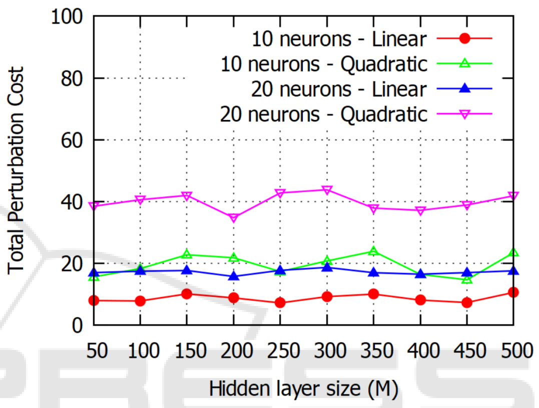

Figure 3: Total Perturbation Cost.

In addition, we propose Total Perturbation Cost (TPC) and Percentage of Perturbed Inputs (PPI) as key metrics to assess the behavior of the proposed algorithm. They can be defined as follows:

$$TPC = \left(\sum_{j\in n} y_j - \sum_{j\in n} \varepsilon_j\right) \tag{19}$$

$$PPI = \frac{\sum_{j\in n} y_j}{|N|} \times 100 \tag{20}$$

5.2 Obtained Results

Since we focused on the total perturbation cost as indicated in the above objective functions, we measure this cost in the two scenarios as shown in Fig. 3. The result shows that the linear approach is slightly efficient in terms of perturbation cost compared to the quadratic objective function.

Fig. 4 illustrates the obtained results in terms of percentage of perturbed inputs using the two mentioned objective functions or metrics. It is clear and with no surprises that the linear approach outperforms the quadratic one, as it necessitates less than

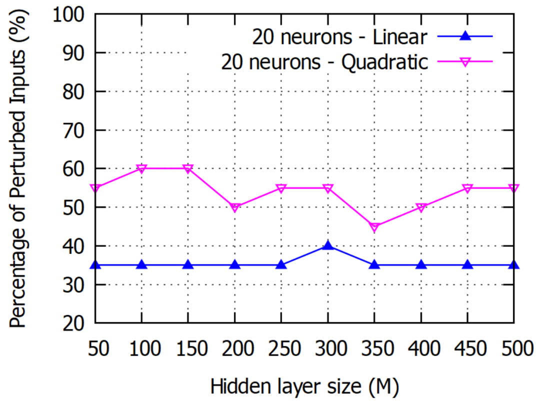

Figure 4: Percentage of Perturbed Inputs (%).

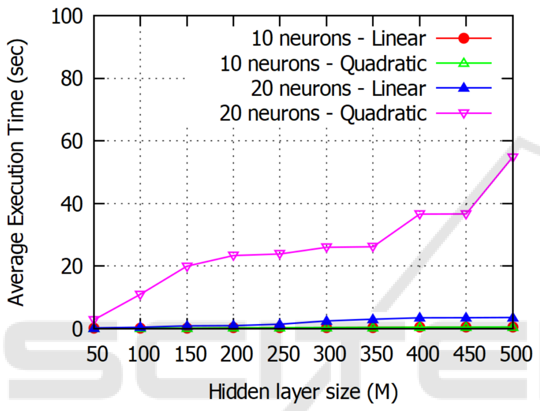

Figure 5: Average Execution Time.

40% of PPI, compared to 60% of PPI when using the quadratic formulation.

The efficiency and feasibility of our adversarial attack algorithm, leveraging an Integer Linear Programming approach, is depicted in Fig. 5. In other words, the average execution time does not exceed 1 minute in the worst scenario (Quadratic objective function with 20 neurons as input). The average time for searching an adversarial image (for instance) is linearly increasing for a hidden layer number ranging from 50 to 500.

6 CONCLUSION

We proposed in this paper a new optimization technique for adversarial attack process. We considered in our optimization the integration of new constraints such as the number of perturbed inputs. Moreover, an optimal optimization algorithm is proposed and evaluated for $|\mathcal{N}|$ and $|\mathcal{M}|$ changes according to predetermined scenarios (linear and quadratic). Performance evaluation is investigated to confirm that $|\mathcal{N}|$ and $|\mathcal{M}|$ have significant impact on the average execution time, TPC, and PPI. Finally, our results show the efficiency of the linear algorithm compared to the quadratic approach.

We considered in this paper only feed-forward neural network. As a future work, we plan to extend our modelling to other deep learning architectures such as Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) . Moreover, we plan to validate our proposed approach on real use cases such as image classification, self-driving, etc.

REFERENCES

Akhtar, N. and Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *CoRR*, abs/1801.00553.

Aung, A. M., Fadila, Y., Gondokaryono, R., and Gonzalez, L. (2017). Building robust deep neural networks for road sign detection. *CoRR*, abs/1712.09327.

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.

Cao, Y., Xiao, C., Cyr, B., Zhou, Y., Park, W., Rampazzi, S., Chen, Q. A., Fu, K., and Mao, Z. M. (2019). Adversarial Sensor Attack on LiDAR-based Perception in Autonomous Driving. In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS'19)*, London, UK.

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.

Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., and Mukhopadhyay, D. (2018). Adversarial attacks and defences: A survey. *CoRR*, abs/1810.00069.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. *ICLR*, 1412.6572v3.

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.

Jmila, H., Khedher, M. I., and El-Yacoubi, M. A. (2017). Estimating VNF resource requirements using machine learning techniques. In Liu, D., Xie, S., Li, Y., Zhao, D., and El-Alfy, E. M., editors, *Neural Information Processing - 24th International Conference, ICONIP 2017, Guangzhou, China, November 14-18, 2017, Proceedings, Part I*, volume 10634 of *Lecture Notes in Computer Science*, pages 883–892. Springer.

Khedher, M. I. and El Yacoubi, M. A. (2015). Local sparse representation based interest point matching for person re-identification. In Arik, S., Huang, T., Lai, W. K., and Liu, Q., editors, *Neural Information Processing*, pages 241–250, Cham. Springer International Publishing.

Khedher, M. I., El-Yacoubi, M. A., and Dorizzi, B. (2012). Probabilistic matching pair selection for surf-based person re-identification. In *2012 BIOSIG - Proceedings of the International Conference of Biometrics Special Interest Group (BIOSIG)*, pages 1–6.

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.

Kisacanin, B. (2017). Deep learning for autonomous vehicles. In *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 142–142.

Kurabin, A., Goodfellow, I. J., and Bengio, S. (2017). Adversarial examples in the physical world. *ICLR*, 1607.02533v4.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Berkay Celik, Z., , and Swami, A. (2015). The limitations of deep learning in adversarial settings. *IEEE*, 1511.07528v1.

Rao, Q. and Frtunikj, J. (2018). Deep learning for self-driving cars: Chances and challenges. In *2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS)*, pages 35–38.

Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. (2016). Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1528–1540, New York, NY, USA. ACM.

Shrestha, A. and Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE Access*, 7:53040–53065.