Verifying the Resilience of Neural Network Watermarking

Ben Goldberger[1], Yossi Adi[2], Joseph Keshet[2], and Guy Katz[1]

[1] The Hebrew University of Jerusalem, Israel
{jjgold, guykatz}@cs.huji.ac.il
[2] Bar Ilan University, Israel
{a, b}@biu.ac.il

1 Introduction

Deep Neural Networks (DNN) are a nowadays phenomena in research and in Industry; They're used for large verity of applications, and achieving state of the art result in many fields (Computer vision, speech recognition, AI, and many more). DNNs flexibility and diversity are pushing the limits on what is possible for a computer to solve efficiently. As a result from their empiric success DNNs are now changing the way software is being designed, and broaden the role of Machine Learning trained functions in applications.

Because of the success of DNNs and because designing and training a DNN require certain expertise, time and processing power, the demand for a specifically designed DNN is growing, And Machine Learning as a Service (MLaaS) is now a thing. There are many machine learning services appearing in many forms, from data visualization and cloud computing to frameworks and semi-trained DNNs. This create some issues regarding the rights and ownership of some part of a trained network or design. Because of the relatively simple components of a DNN (Matrices, vectors and simple functions) it's quite easy to copy or use without permission.

In order to deal with such issues we need a way to authenticate a DNN. This may sound simple but the authentication needs to be robust, such that it's hard to remove. Here comes the concept of Digital Watermarking, a way of signing some Digital property such that it's hard to remove the signature, and said signature is unique. There are some proposed method of watermarking a DNN, but it's unclear how effective they are, meaning how difficult it is to remove the watermark from the DNN and what is the effect of removing the watermark.

DNN verification is a new and promising field. We propose a novel methodology to use verification to measure and verify the robustness of watermarking techniques. Main uses of our approach: verify watermarked networks, assess efficiency of watermarking schemes.

The rest of this paper is organized as follows. In Section 2 we provide the necessary background on DNNs, watermarking, and DNN verification. Next, in Section 3 we introduce our technique for casting the watermark resilience problem into a verification problem. Section 4 describes our implementation and

evaluation of the approach on several watermarked DNNs for image recognition. We discuss related work in Section 5, and conclude in Section 6.

2 Background

[2,3]

3 Defining the minimal change that will change an input decision

Guy: make this a proper citation
Ben: ok

Given a watermarked trained neural network as described here [1]. We tested what is the minimal change to the network last layer in order to "remove" some watermarks from the network.

3.1 Defining the problem for single input

Given a watermarked network N with a set of K watermarks (A set of inputs to the network) $\{x_1, \cdots, x_K\}$ we'll mark the network last layer matrix L such that L is a $m \times n$ matrix were n is the layer's number of neurons and m is the network output size. The change to the last layer will be a matrix with the same dimension as L we'll mark as ε, such that $\varepsilon_{i,j}$ is the change to the last layer matrix entry $L_{i,j}$. We can describe the network output as For a certain input x we're only interested in the input to the last layer we'll mark the input to the last layer v. v is a $n \times 1$ vector. So the original network output $y = Lv$ and the changed network output is $y' = (L + \varepsilon)v$.

For a single input x we denote the original network prediction:

$$d_x := argmax_{i \in [m]} \{y_i\}$$

And the changed network prediction:

$$d'_x := argmax_{i \in [m]} \{y'_i\}$$

We're interested to find the minimal change to the last layer ε so that the prediction will change i.e. $d_x \neq d'_x$

Well measure the overall change to the layer in two ways

$$\|\varepsilon\|_\infty = max_{i,j} \{|\varepsilon_{i,j}|\}. \tag{1}$$

And

$$\|\varepsilon\|_1 = \sum_{i,j} |\varepsilon_{i,j}|. \tag{2}$$

For the ℓ_∞ norm (1) with a chosen d'_x that is different from d_x the minimization problem looks like that:

$$\begin{aligned} &Minimize: && M \\ &Subject\ to: && \forall i,j \quad -M \leq \varepsilon_{i,j} \leq M \\ & && y' = (L+\varepsilon)v \\ & && y'_{d_x} \leq y'_{d'_x} \end{aligned}$$

Variables are the entries in ε, y' and M

(3)

Similarly for the ℓ_1 norm (2) the minimization problem looks like that:

$$\begin{aligned} &Minimize: && M \\ &Subject\ to: && \forall i,j \quad -M \leq \sum_{i,j} |\varepsilon_{i,j}| \leq M \\ & && y' = (L+\varepsilon)v \\ & && y'_{d_x} \leq y'_{d'_x} \end{aligned}$$

Variables are the entries in ε, y' and M

(4)

3.2 Defining the problem for many inputs

Our definition to a single input minimal change ε to the network last layer L can be extended to more then one input very easily by adding more constraint to the problem.
Given inputs $\{x_1, \cdots, x_k\}$ and their respective values:

$$\begin{aligned} &\{v_1, \cdots, v_k\} && \text{:Inputs to the last layer} \\ &\{d_1, \cdots, d_k\} && \text{:Decisions} \end{aligned}$$

Such that

$$\forall 1 \leq j \leq k \quad d_j = argmax_{i \in [m]} \left\{ (Lv_j)_i \right\}$$

With chosen new desired decisions $\{d'_1, \cdots, d'_k\}$ Such that

$$\forall 1 \leq j \leq k \quad d'_j \neq d_j$$

our minimization problem for the ℓ_∞ norm looks like this:

$$\begin{aligned} &Minimize: && M \\ &Subject\ to: && \forall i,j \quad -M \leq \varepsilon_{i,j} \leq M \\ & && \forall j \quad y'_j = (L+\varepsilon)v_j \\ & && \forall j \quad \left(y'_j\right)_{d_j} \leq \left(y'_j\right)_{d'_j} \end{aligned}$$

Variables are the entries in $\varepsilon, y'_1, \cdots, y'_k$ and M

(5)

Similarly for the ℓ_1 norm the minimization problem looks like that:

$$\begin{aligned} &Minimize: && M \\ &Subject\ to: && \forall i,j \quad -M \leq \sum_{i,j} |\varepsilon_{i,j}| \leq M \\ & && \forall j \quad y'_j = (L+\varepsilon)v_j \\ & && \forall j \quad \left(y'_j\right)_{d_j} \leq \left(y'_j\right)_{d'_j} \end{aligned}$$

Variables are the entries in $\varepsilon, y'_1, \cdots, y'_k$ and M

(6)

Guy: Overall, this looks good. Some stuff will need to be moved to other sections according to the paper layout.

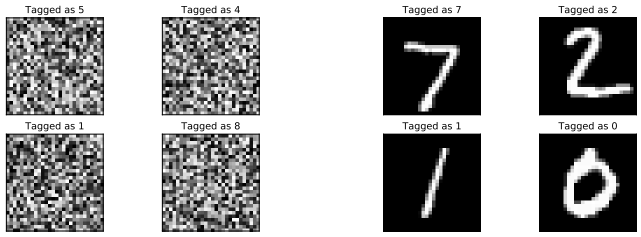
4 Evaluation

For the ℓ_∞ norm when choosing new predictions all the constraint of the minimization problems (3) (5) are linear. There for we choose to solve the ℓ_∞ problem using the Gurobi linear programming solver.
On the other hand ℓ_1 norm minimization problems (4) (6) have non linear constraint so a linear programming solver is not enough. Instead we used Marabou which is a verification tool for neural networks that support piecewise-linear activation functions, Marabou can supply a feasible solution to a set of piecewise-linear constraints so combined with the interval halving method we can find the minimal change for the ℓ_1 case as well.

We tested both approaches on a simple neural network trained on the MNIST data set, the network have one hidden layer with 150 nodes. The network was watermarked with 100 images of noise, examples in Figure 1a.

4.1 Removing a single watermark

The first test we did was to find the minimal change of a single input as defined (3) (4) for every water mark (noise) image. For every decision possible beside the



(a) Water marks images. (b) MNIST images.

Fig. 1: Input images examples.

original decision we found the minimal change and choose the overall minimum. It turns out that the minimal change was always the second best score in the original prediction output. For example an watermark image w with an original output y.

$$d_w = \underset{i \in \{0,\cdots,9\}}{argmax} \{y_i\}$$

d_w is the original tagging of w.

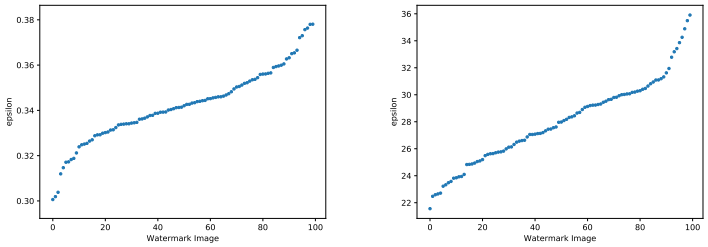
$$d'_w = \underset{i \in \{0,\cdots,9\} \setminus \{d_w\}}{argmax} \{y_i\}$$

d'_w will be the new tagging of w.

So after we apply change to the last layer of the network the new output y' is such that $\underset{i \in \{0,\cdots,9\}}{argmax} \{y'_i\} = d'_w$

For each watermark image have found the minimal ε. This can give us some measure of how difficult it is to remove a single watermark image. And there may be some correlation in the minimal change distribution that is not related to the norm as seen in this figure 2. Beside removing a watermark we're interested in the effect the change we introduced have on the network goal. By applying the change to the original network and evaluating on the MNIST dataset we measured the resulting accuracy of each change. As it turns out we get widely different results from both removal methods (norms) when measuring the accuracy of the changed network. As seen in this table 1 the ℓ_1 gives better accuracy result on average but can have a bad accuracy result, compered to the ℓ_∞ after removing a single watermark that have a consistent result (there minimal accuracy and the maximal accuracy are quite close) but the average is lower then

the ℓ_1 method.



(a) The Minimal $\|\varepsilon\|_\infty$ for every watermark image

(b) The Minimal $\|\varepsilon\|_1$ for every watermark image

Fig. 2: Notice the scale of the graphs, the ℓ_∞ values are much smaller then the ℓ_1 values. As expected

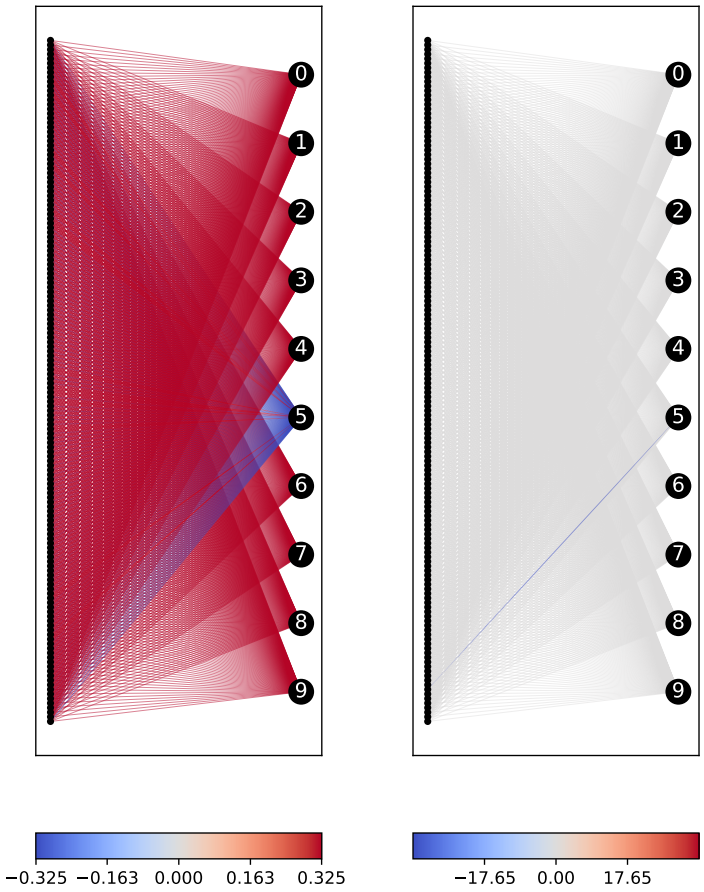
Number of watermarks	Avrg change	Min change	Max change	Avrg acc	Min acc	Max acc	Norm
0	0	0	0	0.97	0.97	0.97	
1	0.34	0.3	0.38	0.87	0.86	0.88	Infinity
1	27.95	21.56	35.91	0.94	0.56	0.97	One

Table 1: Minimal changes and Accuracy

These results begs the question what is the nature of the change and how it differ between the method? Turns out that when minimizing the change according to ℓ_∞ (3) the linear programming solver assign all the entries of ε a small value. This translate to a change to every weight in the last layer which can explain the uniformity of the accuracy test. On th other hand the when minimizing the change according to ℓ_1 (4) the optimal change is to change only a single entry in ε by a seemingly large value. So only one weight in the last layer is changed. See figure 3

4.2 Removing Multiple watermark

Removing a single



(a) The Minimal change according to ℓ_∞ for a single input

(b) The Minimal change according to ℓ_1 for a single input

Fig. 3: Examples of the change to the last layer. Positive change is colored red and negative change is colored blue.

5 Related Work

6 Conclusion and Future Work

References

1. Y. Adi, C. Baum, B. Pinkas, and J. Keshet. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *Proc. 27st USENIX Security Symposium*, 2018.
2. G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.
3. G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, 2019.