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) seems to be everywhere nowadays both in research and in the 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 DNNs are not completely understood there are certain issues that could arise, for example a DNN might be faulty, i.e the DNN may behave differently then expected, and it require fixing. Designing and training a DNN require certain expertise, time and processing power, so we might be interested in finding a small fix that will amend a specific problem without changing to much of the original DNN. Fixing a DNN can be very useful for other applications, such as changing the purpose of a pre-trained DNN. Another application of changing an existing DNN that we focused on is the removal of watermarks from the DNN. Due to the increasing demand for a specifically designed DNN, 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 DNN's. 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.

Guy: make this
general. people n
make minimal char
DNNs for multipl
sons. Verificatic
help address this
(Not just watermar

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.

Formal verification of DNN's is a new and promising field. We propose a novel methodology to use verification to measure and verify the robustness of a certain watermarking techniques. This method may me applicable to other verification problems that are not necessarily watermark related such as network correction, meaning we're given a network and a counter example for some desired property of the network. We're interested to find a "Fix" such that the counter example will no longer exist.

The rest of this paper is organized as follows. In Section **??** we provide the necessary background on DNN's, watermarking, and DNN verification. Next, in Section **??** we introduce our technique for casting the watermark resilience problem into a verification problem. Section **??** describes our implementation and evaluation of the approach on several watermarked DNN's for image recognition. We discuss related work in Section **??**, and conclude in Section **??**.

Guy: Add text
our approach: we
mented in bla bla, v
on benchmarks fro
and obtained amaz
sults summarized
bla.

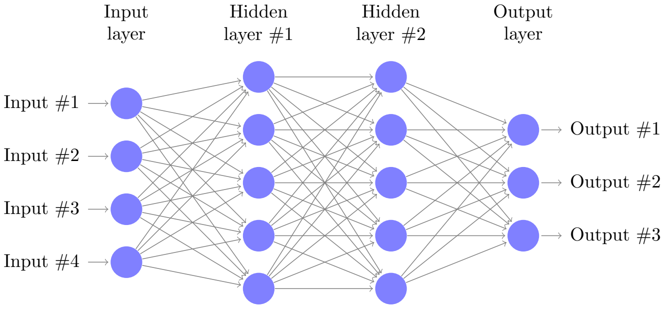


Figure 1: Example of a simple fully connected Deep Neural Network with an input layer of size 4 two hidden layers of size 5 and an output of size 3.

2 Background

2.1 Neural Networks

A neural network is a function that receive an input vector and returns an output vector. There are many types a architectures of neural networks.

To keep things simple we'll only describe the way a feed forward fully connected network calculate an output from a given input. A feed forward fully connected network is constructed from a bunch of layers $l_1, \cdots, l_k$, l_1 is the input layer and l_k is the output later, and an activation function (we're interested in the ReLU activation function). Each layer has a certain number of nodes, l_1 l_k number of nodes is determine by the size of the input and output. Between two adjacent layers l_i l_{i+1} the values of nodes in l_{i+1} is calculated by multiplying the values of l_i by a weights matrix L_i of size $n \times m$ where n and m are the number of nodes in l_{i+1} and l_i respectively. After applying the weights matrix we add a bias vector and apply the activation function, in our case the ReLU activation function, the resulting vector is of size n and it holds the values of the l_{i+1} nodes.

for a single value x

$$ReLU\left(x\right) = max\left(0, x\right)$$

and for a vector $\overrightarrow{v}$

$$ReLU\left(\overrightarrow{v}\right)_i = max\left(0, v_i\right)$$

So basically a feed forward fully connected network with k layers is defined by a set of $k-1$ matrices $L_1, \cdots, L_{k-1}$ and $k-1$ biases vectors $B_1, \cdot, B_{k-1}$ and for a given input x an output vector y is produced like that:

$$y = B_{k-1} + L_{k-1} \cdot ReLU\left(\cdots B_3 + L_3 \cdot ReLU\left(B_2 + L_2 \cdot ReLU\left(B_1 + L_1 \cdot x\right)\right)\cdots\right) \tag{1}$$

For the context of this work we're only dealing with classification neural networks. For these kind of networks the output layer size usually correspond to the number of possible classifications, and each entry in the output vector correspond to the score of a specific classification.

2.2 Verifying Neural Networks

DNN verification answers the following question: Given a neural network N and two predicates P, Q such that P encodes the constraint on the input to the network x and Q encodes the constraint on the output y. We ask if there exist some input x_0 and as a result the output of the network $y_0 = N\left(x_0\right)$, such that $P\left(x_0\right) = True$ and $Q\left(y_0\right) = True$. In this work we're verifying properties of the network a bit differently. Instead of searching an input that answer certain constraints, the input is fixed and we're allowing changes the the network. So the verification question looks like this: Given a neural network N and two predicates P, Q such that P encodes the constraint on the changes to the network ε and Q encodes the constraint on the output y. We ask if there exist ε and as a result the new network $N' = N + \varepsilon$ and new output $y_0 = N'\left(x_0\right)$, such that $P\left(\varepsilon_0\right) = True$ and $Q\left(y_0\right) = True$. [?, ?]

3 Changes to the network as minimization problem

Looking at a trained neural network we're interested to find the minimal change to the network such that the prediction of a specific input will change . We'll look at changes only to the last layer of the network.

Given a trained network N 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 mark as ε is a matrix with the same dimension as L, such that $\varepsilon_{i,j}$ is the change to the last layer matrix entry $L_{i,j}$.

3.1 Defining the problem for single input

For a specific input x we're interested in the input to the last layer, i.e the vector that will be multiplied by the last matrix L, 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:

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

And the changed network prediction:

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

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

$$\left\|\varepsilon\right\|_\infty = max_{i,j} \left\{\left|\varepsilon_{i,j}\right|\right\}. \tag{2}$$

And

$$\left\|\varepsilon\right\|_1 = \sum_{i,j} \left|\varepsilon_{i,j}\right|. \tag{3}$$

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

$$\begin{aligned}
&Minimize: && M\\
&Subject\ to: && \forall i,j \quad -M \le \varepsilon_{i,j} \le M\\
& && y' = (L+\varepsilon)v\\
& && y'_{d_x} \le y'_{d'_x}\\
&\text{Variables are the entries in } \varepsilon, y' \text{ and } M
\end{aligned} \tag{4}$$

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

$$\begin{aligned}
&Minimize: && M\\
&Subject\ to: && \forall i,j \quad -M \le \sum_{i,j} |\varepsilon_{i,j}| \le M\\
& && y' = (L+\varepsilon)v\\
& && y'_{d_x} \le y'_{d'_x}\\
&\text{Variables are the entries in } \varepsilon, y' \text{ and } M
\end{aligned} \tag{5}$$

3.2 Defining the problem for multiple 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

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

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

$$\forall 1 \le j \le k \quad d'_j \ne d_j$$

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

$$\begin{aligned}
&Minimize: && M\\
&Subject\ to: && \forall i,j \quad -M \le \varepsilon_{i,j} \le M\\
& && \forall j \quad y'_j = (L+\varepsilon)v_j\\
& && \forall j \quad \left(y'_j\right)_{d_j} \le \left(y'_j\right)_{d'_j}\\
&\text{Variables are the entries in } \varepsilon, y'_1, \cdots, y'_k \text{ and } M
\end{aligned} \tag{6}$$

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

$$\begin{aligned}
&Minimize: && M\\
&Subject\ to: && \forall i,j \quad -M \le \sum_{i,j} |\varepsilon_{i,j}| \le M\\
& && \forall j \quad y'_j = (L+\varepsilon)v_j\\
& && \forall j \quad \left(y'_j\right)_{d_j} \le \left(y'_j\right)_{d'_j}\\
&\text{Variables are the entries in } \varepsilon, y'_1, \cdots, y'_k \text{ and } M
\end{aligned} \tag{7}$$

4 Methods

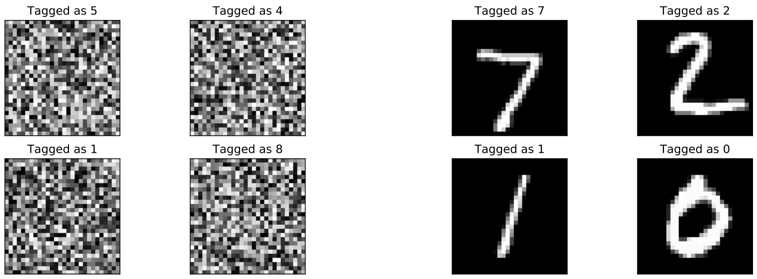
As seen in the previous section we have in our hands a minimization problem. One minimization is according to ℓ_∞ norm and the other is according to ℓ_a norm. In this section we'll show how to convert different problems to the type of minimization problem that was described in the previous section **??**

Guy: In this secti plain how the stuf section 3 can be a to watermarks an work corrections

4.1 Watermarked neural networks

A watermarked neural network is a neural network that was trained with a set of inputs and their desired outputs such that the network will still function properly on the task it meant to do, and the output of the network on the set of inputs is as desired (The desired output can be irrelevant to the network task) [**?**] Given a trained network N and a set of K inputs $\{x_1, \cdots, x_K\}$

As seen in the previous section we have in our hands a minimization problem. One minimization is according to ℓ_∞ norm and the other is according to ℓ_a norm. For the ℓ_∞ norm when choosing new predictions all the constraint of the minimization problems (**??**) (**??**) are linear. There for we choose to solve the ℓ_∞ problem using a linear programming solver. On the other hand ℓ_1 norm minimization problems (**??**) (**??**) 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.



(a) Water marks images.

(b) MNIST images.

Figure 2: Input images examples.

5 Evaluation

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 **??**.

5.1 Removing a single watermark

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

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

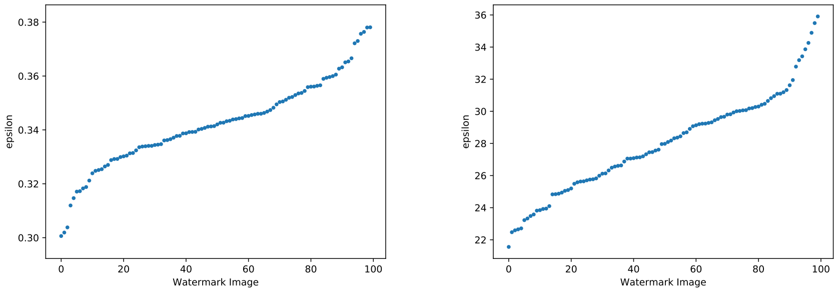
d_w is the original tagging of w.

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

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 **??**. 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



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

Figure 3: 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

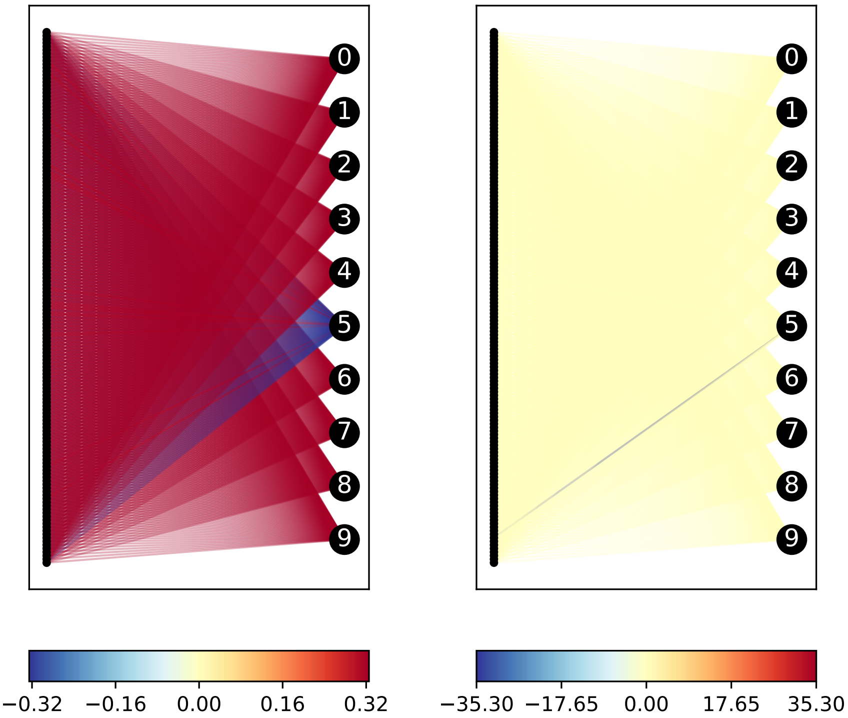
measured the resulting accuracy of each change. As it turns out we get widely different results from both removal methods (norms) when measuring the accuracy of the changed network. As seen in this table **??** the ℓ_1 gives better accuracy result on average but can have a bad accuracy result, compered 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.

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_∞ (**??**) 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 (**??**) 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 **??**

The result of removing a single watermark shows us that the minimal change according to ℓ_∞ is changing the network in a broad way and according to the ℓ_1 norm the change is local, but then not all watermarks are "equal", some are harder to remove and some are easy (as seen in the accuracy test **??**).

5.2 Removing Multiple watermark

As described above the removal of watermarks can be generalized to multiple watermarks (**??**) (**??**). We tested the same MNIST digits classifying network that is signed with 100 watermark.



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

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

Figure 4: Examples of the change to the last layer. Positive change is colored red and negative change is colored blue. There are 150 nodes on the left.

While running the problems on multiple watermarks the linear problem that we solved using the Gurobi solver had no issues with scalability and we where able to find the change that fits to as many watermarks as we want, on the other hand the non-linear problem had some issues with scalability so we where able to find the change that fits to up to 5 watermarks.

dd text about net-
orrection
analyze the tables
ext, explain the in-
ng findings

6 Related Work

7 Conclusion and Future Work

inish these two sec-
In related work,
out DNN verifica-
bout network cor-
, and about water-
3 paragraphs, one
h topic. For dnn
tion stuff you can
t other recent pa-
uval, yizhak, etc).

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.

Number of watermarks	Avrg change	Min change	Max change	Avrg acc	Min acc	Max acc
0	0	0	0	0.97	0.97	0.97
1	0.34	0.3	0.38	0.87	0.86	0.88
2	0.43	0.3	1.83	0.79	0.76	0.88
3	0.53	0.33	1.79	0.71	0.66	0.88
4	0.68	0.33	1.79	0.64	0.57	0.88
5	0.79	0.33	1.87	0.59	0.47	0.8
6	0.89	0.35	1.83	0.53	0.38	0.78
7	1.05	0.34	1.91	0.48	0.28	0.78
25	1.86	1.45	2.09	$9.49 \cdot 10^{-2}$	$2.7 \cdot 10^{-3}$	0.41
50	2.05	1.9	2.15	$4.89 \cdot 10^{-2}$	$2.3 \cdot 10^{-3}$	0.2
75	2.13	2.03	2.17	$5.95 \cdot 10^{-2}$	$2.8 \cdot 10^{-3}$	0.18
100	2.18	2.18	2.18	$5.11 \cdot 10^{-2}$	$5.11 \cdot 10^{-2}$	$5.11 \cdot 10^{-2}$

(a) Change and accuracy when solving for minimal ℓ_∞ change.

Number of watermarks	Avrg change	Min change	Max change	Avrg acc	Min acc	Max acc
0	0	0	0	0.97	0.97	0.97
1	27.95	21.56	35.91	0.94	0.56	0.97
2	51.12	22.65	145.75	0.9	0.56	0.97
3	73.52	25.83	172.67	0.86	0.53	0.97
4	98.39	25.83	205.22	0.86	0.53	0.97

(b) Change and accuracy when solving for minimal ℓ_1 change.

Table 2: Minimal changes and Accuracy for multiple watermarks

[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.