Robustness of Multi-Label Neural Networks

Julian Mour

Advisor: Dana Drachsler Cohen
Technion - Israel Institute of Technology

April 26, 2023

1 Introduction

In the last years, many works have demonstrated that deep neural networks (DNNs) are susceptible to a variety of adversarial example attacks, e.g., [5, 12, 29, 30, 35, 13]. Most adversarial attacks add small changes to the input that fool the network. However, some attacks, known as L_0 attacks, focus on perturbing a very small number of pixels (e.g., one) to an arbitrary value [28, 1, 31]. This type of attack is very effective for concealing adversarial modification in practice, and it can provide geometrical theoretical insight into the input space of a classifier. These attacks are usually local: for each input, they perform a computation to determine the perturbation, which may take a few minutes. Furthermore, these attacks are not robust to input transformation, so each input must be processed differently.

In this thesis, we will design universal L_0 attacks, which rely on programs. The key idea is that program can be specific to input but can be executed much faster and thus will be easier compared to prior attacks. The program will determine which pixels to perturb and to which values. Unlike previous adversarial attack methods, our programs do not employ independent optimization for each image but rather find structures/rules that can be applied to every new input to identify which pixels will be perturbed and how. Our focus is on programs that have two stages. As the first stage, the goal is to limit the search space for a successful attack to a relatively small area in the input image. During the second stage, the focus is primarily on identifying the perturbation (its location and magnitude) within that small area.

Our key idea is to formulate this problem as an optimization problem in which we try to find the optimal program. Two methods have been considered in terms of the initial stage of the program. First is an evaluation search that is trained on a limited set of inputs and given a set of possible program instructions. With each successive iteration, the search seeks to improve the program based on previously generated programs. In the second technique, a deep neural network is used to predict the locations and magnitudes of the perturbed pixels. For the second stage of the program, we developed a small program that iteratively examined all possibilities in a specific area.

In our preliminary research, we implemented our optimization approach on CIFAR-10 dataset on ResNet-18 [7] and evaluated it. We compared our results to the well-known "one-pixel attack method" [28] which is our baseline. Our initial results indicate that our program generated attacks that had almost the same success rate as the baseline, but with much shorter execution times and significantly fewer queries.

In the future, we intend to improve our algorithms to ensure a greater success rate of the attacks and shorter execution times with fewer queries. Additionally, we would like to leverage our programs to provide a better understanding of the network's decision boundaries and the factors that contribute to an attack's success.

2 Problem Definition

In this section, we define the problem we address.

Informally, we are interested in developing an efficient algorithm that given a single image, it returns a robust layer-neighborhood for a target object (target class); A robust sequence of epsilons each representing the perturbation per layer. Each layer is defined to be a set of pixels that share the same distance from a specific non-target object (non-target class) in the image.

Formally, Given a classifier F, an image x of size $n \times m$ containing two objects: c_{target} and $c_{non-target}$ s.t. $F(x) = \{c_{target}, c_{non-target}\}$, we define the set of layers to be:

$$\mathrm{L}_x = \{l_0^x, l_1^x, \ldots, l_r^x\}$$
$$\text{s.t. } \mathrm{l}_d^x = \{(i,j) \in [1,\mathrm{n}] \times [1,m] \mid dist((i,j), c_{non-target}) = d\}$$

We aim to build a program that returns a sequence of epsilons $(\varepsilon_0^x, \varepsilon_1^x, \ldots, \varepsilon_r^x)$ representing a robust perturbations of the pixels in the corresponding layers for the target class. Formally:

$$\varepsilon_x = (\varepsilon_0^x, \varepsilon_1^x, \ldots, \varepsilon_r^x)$$
$$\text{s.t. } \forall x' \in N(x) : c_{target} \in F(x')$$
$$N(x) = \{x' \mid \forall l_d^x \in L_x : \forall (i,j) \in l_d^x : x'_{i,j} \in [x_{i,j} - \varepsilon_d^x, x_{i,j} + \varepsilon_d^x]\}$$

3 Our Approach

In this thesis, we will build a program that given a multi-label classifier and an image, can find a layer-neighborhood of the image around a certain object (non-target class) that is robust for another object (target class) in the image. A straight-forward solution would be to calculate each epsilon individually using a verifier and binary search. Problem is this approach is very expensive in time hence inefficient. Our approach aims to reach similar results as the straight-forward one, but in much less time. In principle, our approach is similar to MaRVeL's [9]. The desired program will have two main components:

- The Verifier - The verifier's main job is to check whether a given layer-neighborhood of an image is robust for the classifier and pass the result to the optimizer. Since this is a multi-label classifier, we are interested only in the target class robustness; The verifier will say that the neighborhood is robust if and only if all images in the neighborhood are classified to the target class. The verifier in our case is a MILP based on the MIPVerify program [33] that translates the robustness problem of a multi-label Neural-Network to a MILP maximization problem and uses the Gurobi Optimizer to solve it. In addition to checking whether the neighborhood is robust or not, the verifier returns the weakest points of the image. These are the points in the checked neighborhood that are least robust and the verifier will pass them to the optimizer as well.

- The Optimizer - The optimizer's job is to update a given robust layer-neighborhood by expanding it more, as we intend to maximize the norm of the neighborhood. In practice, we try to maximize the norm of the neighborhood and the Robustness Level (RL) of the classifier as well. This is done by computing the gradients of both components; The norm's gradient is computed in a straightforward way while the RL's gradient is computed using the weakest points found by the verifier. At last, we calculate the step gradient, which is a combination of both previous gradients, and expand the neighborhood towards the step gradient. If the neighborhood checked by the verifier isn't robust, then we skip the optimization part. Instead, we try to minimally shrink the neighborhood so that its robust. We do that in two different approaches:

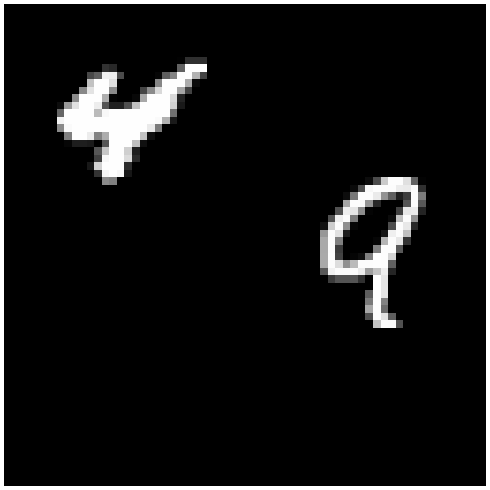

Figure 1: DOUBLE-MNIST sample

  - Fixed weights - We shrink the neighborhood using fixed weights, where we aim to shrink the neighborhood more in layers that are far from the non-target object and less in layers that are close to it.
  - Sensitivity weights - Similar to the fixed weights approach, but we aim to shrink more in layers that are less sensitive to perturbations. We get the sensitivity weights of an image by using the Vanilla Gradient method, introduced by Simonyan et al. [27]

  The new layer-neighborhood will be the input of the verifier in the next iteration.

Like that, we do this in iterations many times until convergence of the layer-neighborhood and the RL. We aim to reach a maximal robust layer-neighborhood at the end of the process.

4 Preliminary Results

We implemented our approach and ran it on part of the DOUBLE-MNIST test dataset, consisting of images from ten classes (the 10 different digits), each image classified to two different classes (contains two different digits). An example of a test dataset sample is shown in Figure 1.

We ran our program on 3 different CNN multi-label DOUBLE-MNIST classifiers. While the three of them solve the same classification problem, they differ in their training process:

- Network with no defense - This network is trained regularly on the original DOUBLE-MNIST training dataset, without additional processing done to the training dataset. We present results of our program ran on this classifier and a single image, as a heatmap representing the epsilons per layer:
- Network with L_0 defense -
- Network with L_{inf} defense -

5 Future Research Objectives

In light of the preliminary work, we aim to further explore our ideas in the following directions:

- Improved algorithm: success rate, execution times and number of queries - As a result of the current implementations and settings of our two approaches, the success rate of the

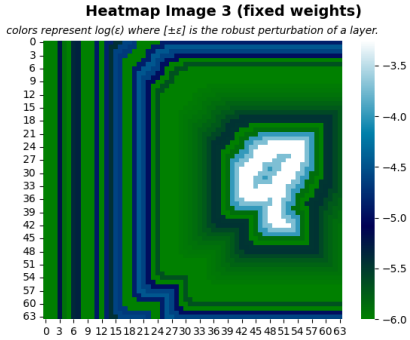

Figure 2. (a): No defense, *FW*

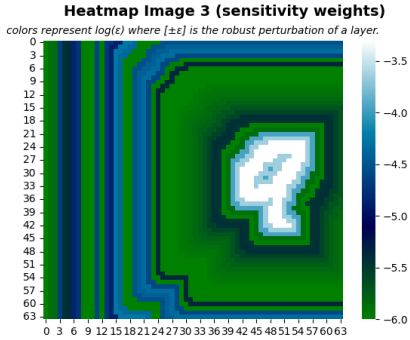

Figure 2. (b): No defense, *SW*

attacks generated by the program is lower than our baseline. Therefore, we will implement different evolution methods, take a different approach to program instruction design, and develop a different architecture (e.g. transformers [32]) for the deep network approach in order to increase success rates. While improving the success rate, we will strive to keep execution times as short as possible with a low number of queries.

- Explainability - One possible advantage of using a universal program to generate adversarial examples is to be able to explain the network's decision boundaries and the reasons that lead to the success of attacks. As an example, by synthesizing a program, we can determine which spatial regions or channels are relevant to the success of the attack. Using the deep learning approach, we are able to use other known tools for an explanation, such as LIME [25].

6 Related Work

Our thesis lies at the intersection of adversarial attacks and program synthesis. We next review some related work.

6.1 Adversarial Attacks

Several aspects can be used to classify adversarial attacks [34]:

- Targeted Vs. Untargeted - A targeted attack [20] attempts to mislead a classifier by classifying input to a *specific class* that differs from the true class of input. In untargeted attacks [5], the goal is to cause the classifier to classify the input to *any class* that is different from the true class.

- Individual Vs. Universal - Individual attacks generate different attacks for each different input, by optimizing the attack based on the input [5, 20, 17]. Universal attacks, on the other hand, generate one attack for all possible inputs. Studies from the last few years have indicated that universal adversarial attacks are capable of generalizing well to unseen data. The first method for universal adversarial attack was based on accumulating perturbations iteratively by composing perturbations for each data point [16]. Other studies have used singular vectors [10] and networks for adversary generation (NAGs) [18].

- White box Vs. Black box - In white box attacks, the architecture, weights, and parameters of the model are known to the adversary. First, white box adversarial attacks were derived through the generation of artificial perturbations using gradient-based optimization [29]. There have been numerous works since then that have demonstrated different approaches to generating white-box adversarial attacks, such as FGSM[5], JSMA [20],

and Deepfool [17]. Attackers using black box methods assume they can only query the model without any further knowledge of it. Several studies have shown that a black-box approach can lead to generating successful attacks [21, 23, 8, 15].

Additionally, we investigate L_0 attacks, which aim to modify a small number of pixels. In previous studies, it has been demonstrated that even a single pixel perturbation can lead to successful attacks[28, 1, 31].

6.2 Program synthesis

Program synthesis is the task of automatically generating a computer program from user requirements. It has been studied for decades and shown success in a variety of fields, such as data wrangling[6], code repair[3], code suggestions[24], probabilistic modeling[19] and superoptimization[14]. Later, constructive techniques have been developed, such as enumerative search[2], programming by examples[6], constraint solving[4], stochastic search[26], neural program synthesis[22] and genetic programming[11]. In our preliminary work, we focused on a variant of genetic search.

References

[1] Janne Alatalo, Joni Korpihalkola, Tuomo Sipola, and Tero Kokkonen. Chromatic and spatial analysis of one-pixel attacks against an image classifier, 2021.

[2] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*, pages 1–8, 2013.

[3] Loris D'Antoni, Roopsha Samanta, and Rishabh Singh. Qlose: Program repair with quantiative objectives. In *27th International Conference on Computer Aided Verification (CAV 2016)*, July 2016.

[4] K. Ghédira and B. Dubuisson. Constraint satisfaction problems: Csp formalisms and techniques. *Constraint Satisfaction Problems: CSP Formalisms and Techniques*, 02 2013.

[5] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *In ICLR*, 2015.

[6] S. Gulwani. *Programming by examples (and its applications in data wrangling)*, pages 137–158. 04 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[8] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *CoRR*, abs/1804.08598, 2018.

[9] Anan Kabaha and Dana Drachsler Cohen. Maximal robust neural network specifications via oracle-guided numerical optimization. *VMCAI*, 2023.

[10] Valentin Khrulkov and Ivan Oseledets. Art of singular vectors and universal adversarial perturbations, 2017.

[11] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[12] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *In ICLR*, 2017.

[13] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *In ICLR*, 2018.

[14] Henry Massalin. Superoptimizer: A look at the smallest program. ASPLOS II, page 122–126, Washington, DC, USA, 1987. IEEE Computer Society Press.

[15] Laurent Meunier, Jamal Atif, and Olivier Teytaud. Yet another but more efficient black-box adversarial attack: tiling and evolution strategies. *CoRR*, abs/1910.02244, 2019.

[16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations, 2017.

[17] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks, 2016.

[18] Konda Reddy Mopuri, Utkarsh Ojha, Utsav Garg, and R. Venkatesh Babu. Nag: Network for adversary generation, 2018.

[19] Aditya Nori, Sherjil Ozair, Sriram Rajamani, and Deepak and Vijaykeerthy. Efficient synthesis of probabilistic programs. In *Programming Language Design and Implementation (PLDI)*. ACM - Association for Computing Machinery, June 2015.

[20] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings, 2015.

[21] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *CoRR*, abs/1602.02697, 2016.

[22] Emilio Parisotto, Abdel rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis, 2016.

[23] Hao Qiu, Leonardo Lucio Custode, and Giovanni Iacca. Black-box adversarial attacks using evolution strategies. *CoRR*, abs/2104.15064, 2021.

[24] Veselin Raychev, Martin Vechev, and Eran Yahav. Code completion with statistical language models. PLDI '14, page 419–428, New York, NY, USA, 2014. Association for Computing Machinery.

[25] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[26] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization, 2012.

[27] Andrea Vedaldi Simonyan, Karen and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps., 2013.

[28] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, Oct 2019.

[29] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

[30] Pedro Tabacof and Eduardo Valle. Exploring the space of adversarial images. *In IJCNN*, 2016.

[31] Danilo Vasconcellos Vargas and Jiawei Su. Understanding the one-pixel attack: Propagation maps and locality analysis, 2019.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[33] Russ Tedrake Vincent Tjeng, Kai Xiao. Evaluating robustness of neural networks with mixed integer programming. *ICLR*, 2019.

[34] Junde Wu and Rao Fu. Universal, transferable and targeted adversarial attacks. 08 2019.

[35] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *In IEEE Trans. Neural Networks Learn. Syst*, 2019.