Computing Adversarial Attacks: A Comparison of Logic-Based Methods

ECE653 Project Proposal

Joseph Scott
joseph.scott@uwaterloo.ca

Vineel Nagisetty
vineel.nagisetty@uwaterloo.ca

Laura Graves
laura.graves@uwaterloo.ca

1 Abstract

For our project, we will do a comparison of different TAV tools for generating adversarial examples for machine learning models. More specifically, we will compare the approaches of fuzzing, symbolic execution via a greedy reduction to SAT, and CDCL(NN). Although gradient-based defenses are not as impermeable as previously thought [ACW18], they still present problems for attackers. On the other hand, non-gradient based attacks such as fuzzing or evolutionary algorithms can circumvent gradient defenses. In this project, we will analyze three different non-gradient based approaches and evaluate the efficacy of each. We will attempt *fuzzing* of the input examples, using a fuzzing software platform. We will also show a formal-logic based attack, where we convert the neural network model into a formal logic form (such as SAT or SMT) and use state of the art logic solvers to attempt to find adversarial examples in the input space. Finally, we will use CDCL(NN) to find adversarial examples through an iterative process that begins with a set of clauses that define the problem and constraints and then queries a neural network (as an oracle) and forms and adds new clauses based on the result of the query.

2 Problem Statement & Proposed Approach

Adversarial examples are specifically designed examples that are semantically different from the classification provided by a neural network. For example, if we have a picture of a goat that we have modified so that our neural network classifies it as a sombrero, it is an adversarial example. These examples not only present a problem for neural network model holders concerned with model classification accuracy, but also highlight the shortcomings of neural networks to semantically understand the images they're given.

An important distinction to make is that adversarial examples aren't created, they are found. Neural networks nearly always have input regions that contain adversarial examples, and for some classes of problems adversarial examples have been proven to be unavoidable [SHS+18]. Instead of trying to create perfect models that don't contain adversarial examples, common defenses focus on making finding these examples difficult, often through gradient obfuscation. Most common and efficient attacks find adversarial examples through a gradient descent technique, and obfuscating the gradient is an effective defense against this class of attacks.

Another class of attacks has been presented that do not rely on gradient descent. These include attacks using evolutionary processes, logic-solver based attacks, and other attacks that seek to efficiently search the region around a given example for adversarial examples. Our project will analyze three different approaches in this class. First, we will use a fuzzing software platform to fuzz training examples to try to find an adversarial example in the region around those examples. Second, we'll develop a tool to convert a neural network model into a logical form and use state-of-the-art logic solvers to attempt to find adversarial examples using the logic solver. For example, we can define the neural network model M as a formula $f(M)$, we can define a formula A that asserts that training example x and modified training example x' are similar (within a bound), and we can define a formula B that states that M classifies x and x' differently $(argmaxM(x)! = argmaxM(x')$. If there exists an x' such that $f(M) \wedge A \wedge B$ is true, then x' is an adversarial example against M.

Finally, CDCL(NN) will begin with a SAT formula f with clauses representing the problem definition and any constraints on the system (such that the modified example x' must be similar to the original example x by some metric). An iterative process will then begin wherein we generate an example x' that satisfies f, and query the target neural network model M to find $M(x')$. Based on the result of $M(x')$, we can either conclude that x' is an adversarial example (if $argmax M(x) \neq argmax M(x')$ or we can derive a clause from the result of $M(x')$ to add to f, limiting the search space and broadening the information about the problem contained in f. By repeating this process until we find an adversarial example or discount the entire search space, we can either show an adversarial example (the resulting x') or show that none exist that satisfy our constraints (through the SAT formula).

3 Framework & Benchmarks

4 Plan of Action / Feasability

The three of us bring a significant amount of domain knowledge to the table. Laura and Vineel both have extensive experience with neural networks, including a robust knowledge of attack and defenses against neural networks. Joe brings an extremely robust knowledge of logic solvers and symbolic execution to the table, and between us we believe we will be able to do a full implementation and test of these systems within the time frame.

For our first step, we will develop platforms for each of the attacks. For the fuzzer, it will be relatively straightforward to adapt a fuzzing software platform to search the space around an example for adversarial examples. For the symbolic execution method, the primary hurdle is developing a tool to turn neural network models into logical formulas. We've developed an extremely rudimentary tool that turns neural networks into single-static-assignment C programs which we can then analyze, but we would prefer to be able to directly transfer a neural network model into a formula. This task may prove to be exceedingly difficult, but we will attempt it. Finally, we will first attempt a CDCL(NN) implementation that simply uses a standard solver along with a neural network model outside the solver. The implementation will maintain the current SAT formula and query the solver to get a satisfiable solution (a modified x' that satisfies the constraints and current clauses). The system will then query the neural network with x' and return a prediction vector, which our implementation will use to extract a meaningful clause to append to the SAT formula. Due to the excellent off-the-shelf platforms for both solvers and neural networks, we believe this goal is achievable during the time frame.

Once the platforms have been implemented for each attack, we can compare the effectiveness of each of them against different classes of models. For example, we can test against undefended networks, networks defended with a variety of defense methods (gradient-based or not), and test against different sizes of networks and different activation functions.

5 Proposed Demo

Our software will be presented as a software tool where you can select a trained neural network model, select an input example, and use the different techniques to attempt to find adversarial examples using each technique. This will give users the ability to see the performance of each of these in real-time, and change the model and input example to gain some understanding of what techniques are effective against different platforms as well as how these techniques scale with different neural network sizes.

Additionally, we will present statistics from our findings, such as the average time to find an adversarial example using each technique against a suite of models. Using this, we can compare our techniques and present an analysis of the effectiveness of each.

References

[ACW18] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[SHS+18] Ali Shafahi, W Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? *arXiv preprint arXiv:1809.02104*, 2018.