A Comparison of Adversarial Attack Methods
ECE653 Final Project

Vineel Nagisetty
vineel.nagisetty@uwaterloo.ca

Laura Graves
laura.graves@uwaterloo.ca

Joseph Scott
joseph.scott@uwaterloo.ca

Abstract

Adversarial examples are specifically crafted inputs that are able to cause a neural network to misclassify them. The challenge in creating these examples is that simple gradient descent is often defeated by defense methods. In this paper we propose and compare a suite of non-gradient-based methods for creating adversarial examples, including methods based on fuzzing, genetic algorithms, and symbolic execution. We find that, while gradient descent outperforms other methods when attacking undefended fully connected models, the non-gradient methods outperform it against convolutional models. This shows promise for finding adversarial examples against defended models as well as highlights the insufficiency of current defense methods, showing the need for greater research into non-gradient based defenses. The code for our implementation and experiments can be found at `https://github.com/vin-nag/checkYourPerturbations`.

1 Introduction

With the developments in machine learning (ML) and artificial neural networks (NN) over the last several years, software systems are increasingly seeing ML techniques such as NNs deployed as integral parts in their core functionality. For example, safety-critical systems, such as driverless cars, make frequent use of NNs. Despite their overall success, NNs are prone to adversarial attacks that trick the NNs into misclassifying inputs with a small amount of noise. For further context, consider Figure 1, where a NN classifies an image of a panda correctly with a confidence of nearly 60%. However, a small perturbation to the original image can result in a blatant misclassification with extremely high confidence.

To this end, there has been an emergence of testing, analysis, and verification (TAV) methods tailored to attacking and making NNs more robust. In this project, we will explore a plethora of preexisting TAV methods to find adversarial examples to neural networks. These methods come four major paradigms, namely: gradient approaches, genetic/evolutionary algorithms, fuzzing solutions, and symbolic execution techniques. In this project, we survey and explore all of these and evaluate their efficiency and efficacy in finding adversarial examples.

Contributions

More specifically, in this project report, we make the following contributions:

1. **A survey of literature to generate adversarial examples** In this report, we survey TAV techniques for finding adversarial examples, such as gradient approaches, genetic/evolutionary algorithms, fuzzing solutions, and symbolic execution techniques. This is done in Sections 2,3.

2. **New fuzzing and genetic/evolutionary approaches to generate adversarial examples** In this project, we propose new TAV methods, specifically fuzzing and genetic/evolutionary methods to generate adversarial examples. The description of these methods are in Section 3.

3. **An empirical analysis of testing, analysis, and verification (TAV) methods for adversarial examples** We perform an empirical evaluation of all considered methods for generating adversarial

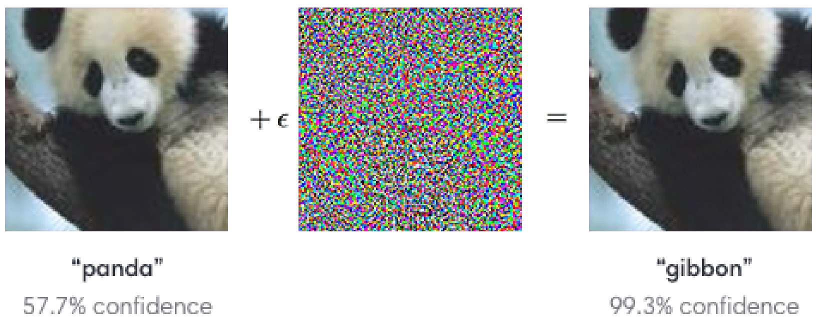

Figure 1: An example of an adversarial attack on a neural network [att]

examples. We find that gradient approaches are significantly more efficient in generating adversarial examples with small perturbations.

The rest of this project report is structured as follows: Section 2 goes over the necessary background of this project, Section 3 describes all TAV methods used to generate adversarial examples, Section 4 presents an empirical evaluation of all considered methods, and Section 5 concludes the project

2 Background and Related Works

Adversarial examples first appeared in literature from Szegedy et al. [SZS+13], where they noticed that there appeared to be examples that are extremely similar to other examples, yet cause a different classification. Soon afterward Goodfellow et al. [GSS14] showed that these examples (deemed *adversarial examples*) can be found easily using gradient ascent. The attack they introduced - the **fast gradient sign method** (FGSM) calculates the gradient of an example x with label y with regard to the loss J of the network with parameters θ, $\Delta_x J(\theta, x, y)$. A step in the direction of that gradient with magnitude ϵ is taken, and the resulting example $x_{adv} = x + \epsilon sign \Delta_x J(\theta, x, y)$ is likely to be adversarial. Note that this is a step in the direction of increased loss.

The authors also proposed the defense technique of *adversarial training*, where a neural network is trained for some iterations, and then a set of adversarial examples are created and added to the set of training examples. Over time, the resulting network becomes robust to adversarial attacks, learning more robust decision boundaries around the training data.

In the years since, many other attacks and defenses have been proposed. One important attack is the Jacobian-based Saliency Map Attack [PMJ+16] which utilize saliency maps (originally designed to help interpret model classifications) to perform efficient attacks with small ℓ_0 bounds. More recently, the Carlini & Wagner Attack [CW17] was designed to find the smallest perturbation that will cause the model classification to change.

One important consideration is that adversarial examples are not an accident of learning - they are inherently a part of deep learning and are in fact inevitable [SHS+18]. In consideration of this, most defense techniques do not attempt to try to avoid the existence of adversarial examples altogether, but simply try to make them as difficult to compute as possible. Many of these defenses are based on obfuscated gradients, but these have been shown to be ineffective [ACW18]. In fact, the property of transferability (that models trained on similar data tend to have similar decision boundaries [PMG16], and thus will make similar predictions on similar inputs) means that black-box attacks where the attacker is given only enough access to input examples and get the resulting prediction vectors are effective. These are done via training a substitute model and creating adversarial examples for that model, and those examples are likely to be adversarial for the black-box model [PMG+17]. In some cases these attacks are effective against models that return only truncated prediction vectors or even models that return only the maximum probability label [OSF19]).

There exist attacks that, unlike the previous attacks, do not use gradient information at all to find adversarial examples. These attacks use methods such as fuzzing ([GJZ+18], [ZDP19]), evolutionary processes ([ASE+18]), logic-solvers ([NKR+18]), and local search ([NK16]). These attacks are more effective against defenses that seek to obfuscate the gradient to defeat gradient-based attacks, such as thermometer encoding [BRRG18]. If the adversary cannot effectively follow the gradient, they can have their attack nullified. Attack algorithms that do not need to follow the gradient have an advantage in such areas.

Multiple researchers have made attempts to encode neural networks in logical forms, so methods such as symbolic execution can be performed and guarantees can be provided. The main one we will consider is the ReluPlex algorithm [KBD+17] which is implemented in the tool Marabou [KHI+19]. ReluPlex reduces neural network verification to a linear programming problem with a permissible non-convex activation function, Relu. With an extended calculus for Relu in a simplex engine, Marabou is able to solve queries on DNNs.

There has been prior work comparing the different adversarial generation methods. Notably, in their work in [MMS+17], Madry et al. created a technique for defending DNNs against adversarial attacks and invited other researchers to attack this resulting robust model. They called this the "MNIST Adversarial Examples Challenge" (since it was using the MNIST dataset). The objective was to find attacks that are effective against this robust model. Researchers were encouraged to create adversarial examples of the entire MNIST test set, where each pixel could at most differ by 0.3. The resulting set was given as input to the robust model and the accuracy was recorded. They evaluated various attacks that were submitted and created a leader board of the top attacks. Note that our work differs from this in the following ways:

1. We use several other attack methods such as fuzzing and evolutionary algorithms for comparison,
2. We set a time limit for generating each adversarial example,
3. We use different kinds of models (such as fully connected and convolutional neural networks), and adversarial defense techniques such as robustness training and thermometer encoding.

Another recent effort is VNN-LIB which stands for Verification of Neural Networks - Library. The organizers of VNN-LIB state that their goals include establishing a "common format for the exchange of DNNs and their properties" and "providing the community with a library of established common benchmarks for VNN tools". They provide different suite of models as benchmarks. However, this work is very recent and no results have been published yet. Still, their philosophy and ideals seem to resonate with ours w.r.t comparing various adversarial example generators on standard benchmarks.

3 Method

We developed multiple methods for generating adversarial examples, ranging from pure fuzzing to gradient-based-fuzzing to a symbolic execution method. Each of the methods attempts to generate an adversarial example either in one pass (such as FGSM) or over multiple attempts. The resulting example must have the following properties to be consider adversarial:

- It must have a different classification on model M than the original: $argmax(M(x)) \neq argmax(M(x_{adv})$
- The modified example must be within an allowable perturbation range ϵ over some ℓ_p norm when compared to the original: $||x_{adv} - x||_p \leq \epsilon$

3.1 Fuzzing

To find these examples we have developed four fuzzing methods which all use randomness. Two are based purely on randomness, while the third uses an explanation AI (xAI) system and the last one works on a combination of randomness and gradient information. First we have *StepFuzz*, which is loosely based on the FGSM algorithm. A fuzz vector z is created with the same dimensionality as the input vector, but where each value is randomly assigned to [-1,0,1]. The fuzzed version of example x is then calculated to be $x_{adv} = x + \epsilon z$, where ϵ is chosen such that the resulting fuzzed image x_{adv} remains within the permitted perturbation range. This can be intuitively viewed as taking a "step" in a random direction.

This is successful because there exist small pockets within the model's decision boundary where other classifications are made, and we can sometimes step into one of those. The probability of this is however fairly low, and we run this algorithm a high number of times to find an example. The speed of this algorithm makes that possible. Our next algorithm is called *LaplaceFuzz*, and it applies fuzz sampled from a Laplacian distribution to the image. A fuzz vector z is sampled from the Laplacian such that $z \in Laplace(\mu, b)$, and then that fuzz vector is used to modify the example such that $x_{adv} = x + z$. This method tends to have a large number of small perturbations and a small number of large perturbations (when compared to StepFuzz), and while it also requires a number of iterations to find an example, it is also very efficient.

Our third algorithm is called *xAIFuzz*, and it utilizes and explanation AI system (specifically LIME [RSG16]). Similar to the previous two methods, a fuzz vector z is generated using a random normal probability distribution. This vector is multiplied by the importance vector i generated by the xAI system, which can loosely be thought of as importance of each pixel to the given prediction. The resulting vector is then added to the original image such that $x_{adv} = x + \epsilon(z * i)$. This method tends to fuzz pixels based on their importance to the overall prediction. A pixel with a higher importance is modified more, and vice versa, which leads to a smarter utilization of the perturbation budget.

Our final fuzzing method is named *VinFuzz*, and it combines random fuzzing with gradient information to result in an efficient and effective fuzzing method. The concept is simple: each feature x^i is given a floor and ceiling value within the perturbation range such that $floorx^i = x^i - \epsilon$ and $ceilx^i = x^i + \epsilon$. From there a new example nx is created where each feature nx^i is randomly selected such that $floorx^i < nx^i < ceilx^i$. In this manner the new example nx is randomly selected from within the perturbation ranges. nx is then passed forward through the target model M, and the sign of the gradient is calculated as $sign\Delta_{nx}J(\theta, nx, y)$. From here, we theorize that if the gradient is negative (and tells us that decreasing the value of nx^i leads to increased loss), then we shouldn't have nx^i go any higher, and we set $ceilx^i = nx^i$. We perform the same process for each feature where the gradient is positive, setting $floorx^i - nx^i$. This updates the floor and ceiling values so that the new ranges are closer to increased loss. We then repeat this process by randomly selecting a new nx within $floorx$ and $ceilx$ ranges, calculating the gradient again, and updating the floor and ceiling values accordingly. This loop is performed iteratively for a number of iterations, and each iteration brings us closer to increased loss and increased chance of nx being adversarial. While this method utilizes the gradient, it does not rely on it alone and as such performs well against both undefended models and models protected by gradient-based defenses.

3.2 Symbolic Execution

We consider three approaches to symbolic execution for the discovery of adversarial examples, namely: The ESBMC C prover [GMM+18], SMT Solvers [BCD+11, DMB08, NP20], and ReluPlex/Marabou [KBD+17, KHI+19]. Regrettably, none of these techniques were efficient nor effective in finding adversarial examples. All code is available on the github link.

ESBMC is a renown C prover with speciality in verifying floating-point programs and support for transcendental functions unlike SMT Solvers and Marabou. We translate the neural network into a simple C program in single-static-assignment (SSA), and then assert if there exists a perturbation to an input (see github for an example). However, in our experimental analysis, we observe ESBMC to be extremely buggy and returned inputs properly classified or the original input.

We further used SMT solvers such as z3 [DMB08], CVC4 [BCD+11], and bitwuzla [NP20] by translating the problem of an adversarial example to an SMT-LIB problem in the logic of QF_FP. However, on MNIST with an input layer of 784 neurons and a single hidden layer of 128 neurons, and an output layer of 10

```
VinFuzz($\theta, x, y, n, \epsilon$)
$\theta$ = model parameters, $x$ = original example, $y$ = true label, $n$ = number of iterations, $\epsilon$ = perturbation range
nx = new Array[x.size]
lb = new Array[x.size]
ub = new Array[x.size]
set the initial lower and upper bound and generate a random example in those ranges
for $i = 0;\ i < x.size;\ i++$ do
    lb[i] = x[i] - $\epsilon$
    ub[i] = x[i] + $\epsilon$
    nx[i] = random(min=lb[i], max=ub[i])
end
run n iterations of the loop for $j = 0;\ j < n;\ j++$ do
    use the sign of the gradient to update the lower and upper bounds and generate a new example
    grad = $sign\Delta_{nx}J(\theta, nx, y)$
    for $i = 0;\ i < x.size;\ i++$ do
        if grad[i] == -1 then
            ub[i] = nx[i]
        if grad[i] == 1 then
            lb[i] = nx[i]
        nx[i] = random(min=lb[i], max=ub[i])
    end
end
return nx
```

neurons, this results in over 150,000 multiplications which is infeasible for modern solvers. Within just a few minutes each solver uses up 30 GB of memory and fails to solve after an hour.

Lastly, we used the Marabou tool which implements the ReluPlex algorithm. Unfortunately this tool while promising is in its infancy and supports a limited frameworks and versions (It does not yet support tensorflow 2.2). Hence, we can not fairly compare it to the other considered techniques. We evaluate Marabou on several ONNX models in the VNN-LIB initiative. Marabou reports 'UNSAT' on all queries to expose an adversarial example, implying none such exists. This is extremely surprising to us, and we are in touch with the authors of the tool for clarification.

4 Evaluation

Our evaluation is implemented using python 3 and the keras framework [C+15] for tensorflow [AAB+15], a popular machine learning platform. Keras provides an easy-to-use framework over the tensorflow platform, and many researchers have implemented their research in keras and tensorflow. In this case, DLFuzz [GJZ+18] and Genetic Algorithm [VN16] are implemented in keras, and the library Cleverhans [PFC+18] implements FGSM, BIM, C&W, Madry, and multiple other attack algorithms in tensorflow.

The methods are evaluated using a python framework that tests each algorithm over a collection of benchmarks. When an adversarial attack is found, the algorithm moves on to the next. If an algorithm is unable to find an adversarial example within a timeout limit, it is stopped and moves on to the next example with a time score of twice the timeout length (consistent with how the Par2 scores are calculated in the SAT community). Algorithms are then compared based on how many examples they are able to find adversarial examples for, as well as the total time taken to find those examples.

4.1 Benchmark

In order to verify the algorithms efficacy on different types of models, we use two model architectures: a fully connected as well as a convolutional neural network - whose details are given in Tables 1 and 2. Further, for both model architectures, we train a regular version and a robust version using adversarial examples. Moreover, we trained another convolutional model (with the same architecture) using thermometer defense [BRRG18] which has been shown to be robust to gradient based attacks. This totals to five different models.

Fully Connected Neural Network (FCNN)		
Layer Name	**Shape**	**Parameters**
Input	(None, 784)	0
Dense	(784, 128)	100,480
Output	(128, 10)	1290
Total Params:		101,770

Table 1: Architecture of FCNN.

Convolutional Neural Network (CNN)		
Layer Name	**Shape**	**Parameters**
Conv2D	(None, 14, 14, 20)	520
MaxPooling2D	(None, 7, 7, 20)	0
Conv2D	(None, 3, 3, 20)	10,020
MaxPooling2D	(None, 1, 1, 20)	0
Dense	(None, 64)	1344
Output	(64, 10)	650
Total Params:		12,534

Table 2: Architecture of CNN.

We created various families (or collections) of benchmarks, each consisting of a set of models, a set of inputs, a time out limit and a similarity distance limit. It is important to note that the smaller the allowable similarity distance, the harder it is for the generators to create adversarial examples. The list of benchmarks are given in Table 3. Note that since the thermometer defense obfuscates gradients, the gradient based models are not able to generate adversarial examples - which is why we only run the non-gradient based generators on the Thermometer benchmark.

Benchmark	**Models**	**Similarity (l2 Distance)**	**Time Limit (sec)**
Main-Similar	Regular and Robust FCNN	5	600
Main	Regular and Robust FCNN	10	600
CNN	Regular and Robust CNN	10	600
Thermometer	Thermometer CNN	10	600

Table 3: Benchmarks created and used in this work.

4.2 Methods

To properly evaluate our methods, we compare our methods against a suite of state-of-the-art adversarial attack methods. Here we list the algorithms tested as well as some of their properties. Note that in most cases, due to the property of transferability, black-box networks can be attacked by creating a substitute model and attacking that model. Due to this, an attack needing white-box access doesn't mean it is necessarily stymied by the attacker having only black-box access.

StepFuzz, **LaplaceFuzz**, **xAIFuzz**, and **VinFuzz** are implemented as written in section 3.1, with randomness sampled using numpy operations **numpy.random.randint()**, **numpy.random.laplace()**, **numpy.random.random()** and **numpy.random.random()**, respectively.

DLFuzz was implemented using their public github implementation, with some minor changes to allow it to be evaluated in comparison with the other methods. First, they limited perturbations based on the relative ℓ_2 norm $\frac{||x_{adv}-x||_2}{||x||_2}$ instead of the standard ℓ_2 norm of the perturbation $||x_{adv} - x||_2$. We modified this to be in line with the perturbation limits the other methods use. Second, they do not clip the features to the allowable feature range, enabling them to call examples adversarial if those examples have features

Algorithm	Author	Gradient-Based	Access Needed
StepFuzz	Nagisetty et al.	No	Black-box
LaplaceFuzz	Nagisetty et al.	No	Black-box
VinFuzz	Nagisetty et al.	Yes	White-box
XAI-Fuzz	Nagisetty et al.	Yes (for XAI System)	White-box
DLFuzz	[GJZ+18]	Yes	White-box
Fast Gradient Sign Method	[GSS14]	Yes	White-box
Basic Iterative Method	[KGB16]	Yes	White-box
C&W Attack	[CW17]	Yes	White-box
Madry Attack	[MMS+17]	Yes	White-box
Genetic Algorithm	[VN16]	No	Black-box

Table 4: Generators Compared in this work.

above the maximum or below the minimum range for that feature. In a domain such as image recognition this may allow for a misclassification, but when the set of features needs to be converted back into an image (with maximum and minimum pixel values) it may no longer be adversarial. Finally, DLFuzz allows you to set the number of perturbation steps taken for each run. We found that while a small number may be effective for easily attacked models, a large number is necessary for protected models. To enable DLFuzz to be both quick against easily attacked models and effective against protected models, we iteratively increase the perturbation depth as we go, returning an adversarial example if one is found and increasing the number of steps allowed for the next attempt if none are found. With these changes we were able to evaluate the efficacy of DLFuzz in comparison to the other methods.

FGSM, **BIM** and **Madry** were implemented with the Cleverhans library which contains versions of each of those algorithms in tensorflow 2. No changes were made to the functionality of each algorithm. The **C&W attack** is not supported by Cleverhans for tensorflow 2, but our implementation uses a fork of Cleverhans found at `https://github.com/Joool/cleverhans/tree/carlini_wagner`.

The **Genetic Algorithm** was implemented using numpy, and closely follows the mechanisms described by Vidnerova and Neruda in [VN16]. The notable difference is that their implementation aimed to create adversarial examples of a specific target label while our attack does not, so that we are consistent with the other generators. In order to achieve that, we modify the fitness function to be $y_{adv}[label] * ||x_{adv} - x||_p$, i.e multiplying the model's prediction on a candidate image for the target label with the similarity between the images. Since our objective is to find individuals that minimize this fitness, an optimal solution will be very similar to the original image while the models prediction on the image will be very low for the target label (i.e lead to misclassification).

4.3 Experimental Setup

The experiment ran the different benchmarks on all the models, using a local hardware running on Ubuntu 16.04 with four Intel i5-4300U (CPU 1.9 GHz) cores and 16 GB RAM. Note that the thermometer benchmark can only be run on generators that do not use gradients so only those were chosen. For a given benchmark, we run all generators on each target model and target instance pair. If the generated instance is verified to be an adversarial example, we record the time and similarity score (w.r.t the target instance). Moreover, we monitor (using separate threads) the time taken for the generator, and if it exceeds the benchmark's time limit, we record that attempt as unsuccessful and note the time taken and similarity score as 2 * the limits of the benchmark (similar to par2 calculation performed in the SAT community). Our implementation of the fuzzers, along with the code for experiments and results can be found at `https://github.com/vin-nag/checkYourPerturbations`.

4.4 Results

The results of the experiments run on the Main and Main-Similar benchmarks is given in Table 5 and Figures 2 and 3 respectively. Overall, Basic Iterative Method (BIM) takes the shortest amount of time to generate adversarial examples on both benchmarks, while DLFuzz fails to generated any adversarial examples at all during the time limit. Three of the fuzzers (Laplace, XAI and Step) show high variance - performing very well on the Main benchmark, but poorly on the Main-Similar benchmark. This variance can be seen in Figures 2 and 3 where the plots for Laplace, Step and (to some extent) XAI differ. By contrast, Madry, FGSM, Carlini Wagner, VinFuzz and Genetic Algorithm show a consistent performance on both benchmarks.

Even when algorithms provide adversarial examples within the similarity limit, we notice a difference in the average similarity of the adversarial examples generated by each algorithm. For instance, gradient based algorithms produce adversarial examples that are highly similar (cumulative similarity distance in range [25, 200]) while non-gradient algorithms do not (cumulative similarity in range [230, 457]). The similarity distance seems consistent among both benchmarks. Note that since the similarity limit is lower in the Main-Similar benchmark, if an algorithm does not succeed to generate an adversarial example, the penalty is lower compared to the Main benchmark i.e the similarity penalty is 10 in Main-Similar as opposed to 20 in Main benchmark. This explains why the similarity score is lower for Genetic Algorithm and VinFuzz in the Main-Similar benchmark - because they have a similar number of instances time out but accrued less penalty on the Main-Similar benchmark.

	Main		**Main-Similar**	
Generator	**Time**	**Sim**	**Time**	**Sim**
BasicIterativeMethod	10.9916	38.3133	14.2705	38.3133
LaplaceFuzz	16.8971	370.0474	41535.0076	395.6257
XAIFuzz	18.8156	236.3747	938.7496	199.0752
StepFuzz	34.4126	350.8247	37421.9183	374.4402
MadryEtAll	68.2202	25.3125	71.9455	25.315
FastGradientSignMethod	58.535	25.2557	80.2188	25.2557
CarliniWagner	9915.0678	197.7778	9821.9288	117.7778
GeneticAlgorithm	23439.0246	420.2257	22371.8497	221.0096
VinFuzz	28050.7742	457.1261	28514.1743	248.1306
DLFuzz	58800	980	58800	490

Table 5: Results on the Main and Main-Similar Benchmark

The results of the experiments run on the Main-CNN and Thermometer benchmarks is given in Table 6 and Figures 4 and 5 respectively. Surprisingly, on the Main-CNN benchmark, the fuzzing algorithms (except for VinFuzz and DLFuzz) vastly outperform the gradient based ones. The top three performing algorithms are Laplace, Step and XAI fuzzers. The performance of Madry and Basic Iterative Methods is the next highest, followed by FGSM, Carlini Wagner, Genetic Algorithm and VinFuzz. The reason why the gradient based methods took so long is due to their poor performance on the robust CNN models - as can be seen in the spike in their plots in Figure 4. DL Fuzz did not produce an adversarial examples within the time limit here as well, and VinFuzz only generated a few successful adversarial examples. Consistent with previous benchmark results, the gradient based algorithms produced the most similar attacks, with Madry, BIM and FGSM being the top three in cumulative similarity distance.

As previously stated, gradient based methods do not work on models defended using our implementation of thermometer encoding, and so they are not run on the Thermometer benchmark. This is because the thermometer encoding layer is not differentiable and so back-propagation does not yield useful information to the previous layers. However, it is surprising how quick the three fuzzers (Laplace, Step and XAI) find adversarial examples. In fact, their performance on Thermometer benchmark is the quickest on any

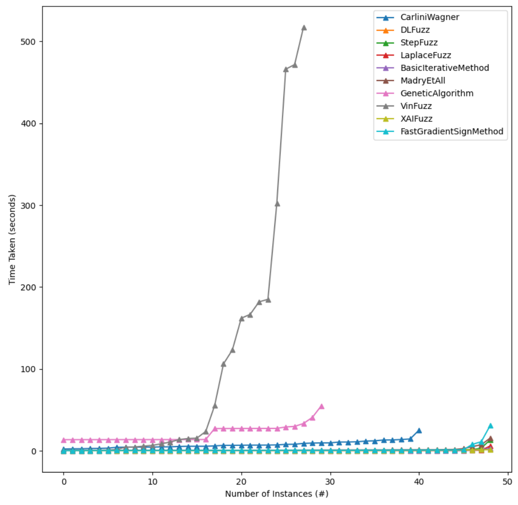

Figure 2: Results on the Main Benchmark.

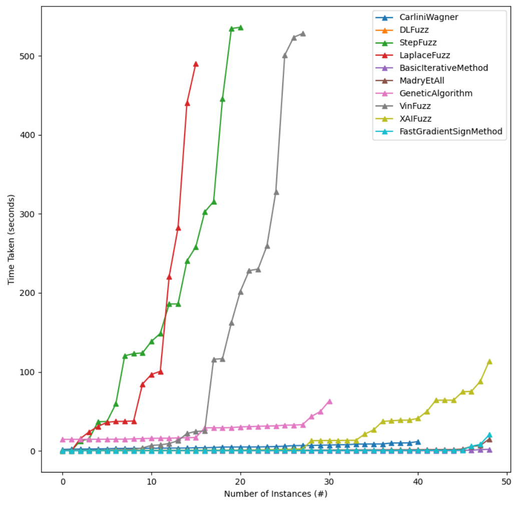

Figure 3: Results on the Main-Similar Benchmark.

benchmark by any algorithm. Genetic Algorithm also took comparatively less time to find adversarial examples. It also produced the most similar adversarial examples.

	Main-CNN		**Thermometer**	
Generator	**Time**	**Sim**	**Time**	**Sim**
LaplaceFuzz	10.6816	319.0187	1.3311	64.082
StepFuzz	15.7379	314.1623	2.0283	63.1283
XAIFuzz	31.5575	207.5959	3.2711	54.3104
GeneticAlgorithm	30773.1413	532.6955	393.3436	26.2774
MadryEtAll	1230.6365	28.747		
BasicIterativeMethod	4623.3858	90.8382		
FastGradientSignMethod	7589.8691	138.8464		
CarliniWagner	31373.3318	534.5619		
VinFuzz	58800.4578	980.4263		
DLFuzz	60000	1000		

Table 6: Results on the Main-CNN and Thermometer Benchmark

4.5 Discussion

Our experiments show several interesting findings. Firstly, the lower the similarity distance limit, the harder it seems to be to find adversarial examples - especially for the fuzzing algorithms. Moreover, different algorithms perform differently depending on model architecture. For example, gradient based algorithms performed very well on fully connected neural networks but poorly on convolutional neural networks. On the other hand, fuzzers performed very well on convolutional models but poorly on fully connected models. On average, algorithms performed worse on robustly trained models, compared to regular models. The notable exception is the model defended using thermometer encoding, where the gradient based algorithms did not work at all while the non-gradient based ones showed a remarkably high performance.

Gradient based algorithms work very well on fully connected neural networks. In these scenarios, they

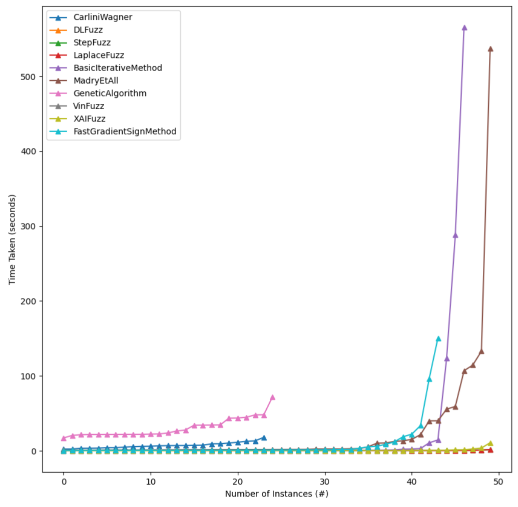

Figure 4: Results on the Main-CNN Benchmark.

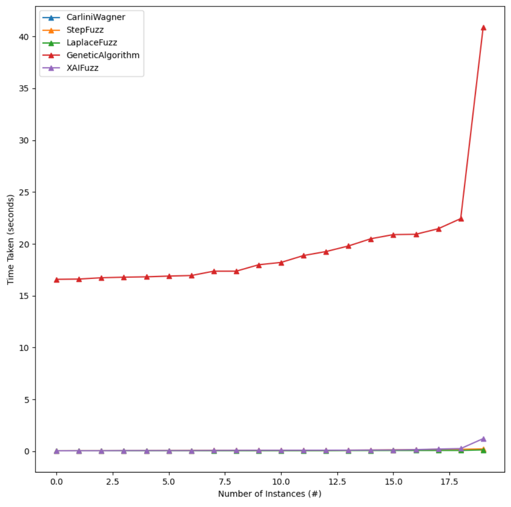

Figure 5: Results on the Thermometer Benchmark.

take the lowest time to generate adversarial examples. They also find adversarial examples with lower similarity distance compared with non-gradient based algorithms - even on defended and convolutional models. Of the gradient based algorithms compared in our experiments, BIM has the best performance while Carlini Wagner has the worst.

On the other hand, non gradient based algorithms, especially fuzzing algorithms, surpass the performance of gradient based ones on convolutional neural networks, especially CNNs that are defended. They also work remarkably well on models defended using thermometer encoding. Of all the fuzzers compared in our experiments, XAI fuzzer is shown to have the best performance while DLFuzz has the worst.

Finally, Genetic Algorithm shows a consistent performance on all benchmarks. Surprisingly, it produces adversarial examples with the lowest similarity distance on the Thermometer benchmark. This shows that there are scenarios where Genetic Algorithm based adversarial example generator can be useful.

5 Conclusion

The results here show that, the more complex a model is and the more difficult it is to gain meaningful information from the gradient, the less useful standard gradient-based attacks become. This trait also holds for robust models that have been adversarially trained. In these scenarios, non-gradient-based attacks become more and more crucial. One significant drawback to these methods is the relatively large perturbations needed by them, because finding small adversarial pockets becomes more difficult with smaller perturbation bounds. This highlights the need for more sophisticated attack methods that don't rely on the gradient, as well as shows the shortcomings of the state of the art attacks that rely heavily on gradients. Future research would do well to focus on logic-based methods for analysis, utilizing logic solvers and symbolic execution tools to be able to efficiently find counterexamples (in this case, adversarial examples) that show where models are vulnerable. The primary drawback is these tools tend to have difficulty scaling, but symbolic analysis methods that are specifically crafted to find adversarial examples could be very promising.

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[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.

[ASE+18] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018.

[att] Attacking machine learning with adversarial examples. `https://openai.com/blog/adversarial-example-research/#:~:text=with`. Accessed: 2020-08-20.

[BCD+11] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *International Conference on Computer Aided Verification*, pages 171–177. Springer, 2011.

[BRRG18] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

[C+15] François Chollet et al. Keras. `https://keras.io`, 2015.

[CW17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[GJZ+18] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. Dlfuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 739–743, 2018.

[GMM+18] Mikhail R Gadelha, Felipe R Monteiro, Jeremy Morse, Lucas C Cordeiro, Bernd Fischer, and Denis A Nicole. Esbmc 5.0: an industrial-strength c model checker. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 888–891, 2018.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[KBD+17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[KHI$^+$19] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[MMS$^+$17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[NK16] Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial perturbations for deep networks. *arXiv preprint arXiv:1612.06299*, 2016.

[NKR$^+$18] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[NP20] Aina Niemetz and Mathias Preiner. Bitwuzla at the smt-comp 2020. *arXiv preprint arXiv:2006.01621*, 2020.

[OSF19] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4954–4963, 2019.

[PFC$^+$18] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[PMG$^+$17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[PMJ$^+$16] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

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

[SZS$^+$13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[VN16] Petra Vidnerová and Roman Neruda. Vulnerability of machine learning models to adversarial examples. In *ITAT*, pages 187–194, 2016.

[ZDP19] Pengcheng Zhang, Qiyin Dai, and Patrizio Pelliccione. Cagfuzz: Coverage-guided adversarial generative fuzzing testing of deep learning systems. *arXiv preprint arXiv:1911.07931*, 2019.