FUZZING-BASED HARD-LABEL BLACK-BOX ATTACKS AGAINST MACHINE LEARNING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning models are known to be vulnerable to adversarial examples. Based on different levels of knowledge that attackers have about the models, adversarial example generation methods can be categorized into white-box and black-box attacks. We study the most realistic attacks, hard-label black-box attacks, where attackers only have the query access of a model and only the final predicted labels are available. The main limitation of the existing hard-label black-box attacks is that they need a large number of model queries, making them inefficient and even infeasible in practice. Inspired by the very successful fuzz testing approach in traditional software testing and computer security domains, we propose fuzzing-based hard-label black-box attacks against machine learning models. We design an AdvFuzzer to explore multiple paths between a source image and a guidance image, and design a LocalFuzzer to explore the nearby space around a given input for identifying potential adversarial examples. We demonstrate that our fuzzing attacks are feasible and effective in generating successful adversarial examples with significantly reduced number of model queries and L_0 distance. More interestingly, supplied with a successful adversarial example as a seed, LocalFuzzer can immediately generate more successful adversarial examples even with smaller L_2 distance from the source example, indicating that LocalFuzzer itself can be an independent and useful tool to augment many adversarial example generation algorithms.

1 INTRODUCTION

Machine learning models, especially deep neural networks, are demonstrated as being vulnerable to adversarial examples (Biggio et al., 2013; Szegedy et al., 2014). Adversarial examples are generated by adding small perturbations to clean inputs to fool machine learning models to misclassify. In image classification tasks, adversarial examples can be found by many attack methods (Goodfellow et al., 2015; Papernot et al., 2016b; Kurakin et al., 2017; Carlini & Wagner, 2017) with the knowledge of a neural network architecture and its parameters. This type of attacks is considered as *white-box attacks*. While research on white-box attacks significantly helps the community to better understand adversarial examples and deep neural networks, white-box attacks are only applicable to limited real-world scenarios such as on publicly available models or exposed confidential models.

In many real-world scenarios, *black-box attacks* are more realistic, where an attacker only has the query access to a model. Some recent attacks (Narodytska & Kasiviswanathan, 2017; Chen et al., 2017; Hayes & Danezis, 2018; Ilyas et al., 2018) rely on probability vectors (e.g., predicted scores or logits) of a model to generate adversarial examples, and they are referred to as *soft-label black-box attacks*. However, in many more realistic scenarios, only the final predicted labels (e.g., the top-1 class label) of a model are available to the attackers. This category of attacks is referred to as *hard-label black-box attacks*. Three recent attack methods including Boundary attack (Brendel et al., 2017), Label-only attack (Ilyas et al., 2018), and Opt attack (Cheng et al., 2019) fall into this category. However, although they can generate adversarial examples with comparable perturbations to white-box attacks, the main limitation of existing hard-label black-box attacks is that they need a large number of model queries since model information is not available.

From a unique perspective, we propose fuzzing-based hard-label black-box attacks by leveraging the fuzzing approach that is very successful in software testing and computer security domains (My-

ers et al., 2012; Miller et al., 1990; Haller et al., 2013). The generation of adversarial examples can be essentially considered as an optimization problem. In order to find an optimal adversarial example around a clean input, attack algorithms need information such as gradient of the loss function, vectors of classification probability, or hard labels from a model as guidance to walk toward the goal. The adversarial examples then cause the model to misclassify. Interestingly, we consider the following analogy: a machine learning model to be attacked is similar to a target program to be tested for correctness or security bugs. Adversarial examples that cause a model to misclassify are analogous to inputs that trigger a target program to crash. These similarities and the huge success of the fuzzing approach in those traditional domains inspire us to leverage an originally black-box software testing technique, *fuzz testing*, for exploring black-box adversarial example attacks in the adversarial machine learning domain.

The word "fuzz" was first proposed by Miller et al. (1990) to represent random, unexpected, and unstructured data (Takanen et al., 2008). Fuzz testing aims to find program failures by iteratively and randomly generating inputs to test a target program (Klees et al., 2018). It is a very effective approach to identifying correctness or security bugs in traditional software systems (Haller et al., 2013; Appelt et al., 2014; Jeong et al., 2019) as well as development or deployment bugs in machine learning models (Odena et al., 2019; Xie et al., 2018).

In this paper, we propose fuzzing-based attacks against machine learning models in hard-label black-box settings. We take the fuzz testing approach to generate random inputs for exploring the adversarial example space. We design two fuzzers: an adversarial fuzzer (referred to as *AdvFuzzer*) and a local fuzzer (referred to as *LocalFuzzer*). AdvFuzzer explores multiple paths from a clean example to a guidance example. LocalFuzzer explores the nearby space around a given input. Our approach can be applied in both targeted and untargeted settings, aiming to generate adversarial examples using a much smaller number of model queries than existing hard-label black-box attacks. Note that when a successful adversarial example is supplied as the input, LocalFuzzer has the potential to generate a large number of other successful adversarial examples in bulk. This bulk generation can be applied to adversarial examples generated from any attack methods, and potentially refine their "optimized" adversarial examples by reducing the L_2 distance from the source example.

We perform experiments to attack deep neural networks for MNIST and CIFAR-10 datasets to evaluate our fuzzing approach. The experimental results show that our fuzzing attacks are feasible, efficient, and effective. For example, the number of model queries can be reduced by 10-18 folds for MNIST and 2-5 folds for CIFAR-10 in untargeted attacks in comparison between ours and existing hard-label black-box methods. We also evaluate our LocalFuzzer on successful examples generated by Boundary attack, Opt attack, and our fuzzing attacks to validate its usefulness. For example, we achieve 100% success bulk generation rate and 48%-100% success bulk generation rate with lower L_2 for MNIST adversarial examples generated by different methods in untargeted attacks. Our work provides evidence on the feasibility and benefits of fuzzing-based attacks. To the best of our knowledge, this is the first work on exploring fuzz testing in adversarial example attacks.

2 Background and Related Work

Adversarial Examples In this paper, we consider computer vision classification tasks, in which a DNN model f aims to classify an input image $\boldsymbol{x}$ to a class y. We define a clean input image $\boldsymbol{x}$ as *source example* with *source class* y. The attack goal is to generate an adversarial example $\boldsymbol{x'}$ close to source example $\boldsymbol{x}$ such that: (1) $\boldsymbol{x'}$ is misclassified as any class other than the source class y in the ***untargeted attack*** setting, or (2) $\boldsymbol{x'}$ is misclassified as a specific class y_t != y in the ***targeted attack*** setting. We consider that an adversary has the hard-label black-box capability which means the adversary only has the query access to the model f and only final label outputs are available.

White-box attacks Most existing attacks rely on full access to the model architecture and parameters. Example attack algorithms include Fast Gradient Sign Method (Goodfellow et al., 2015), Jacobian-based Saliency Map Approach (Papernot et al., 2016b), Basic Iterative Method (Kurakin et al., 2017), L-BFGS (Szegedy et al., 2014), Carlini & Wagner attack (Carlini & Wagner, 2017), etc. White-box attacks need gradient information of the loss function as guidance to find adversarial examples. However, the white-box scenarios are not very realistic considering the fact that many real-world machine learning models are confidential. Besides, white-box attacks can only be applied to differentiable model architectures like DNNs not to tree-based models.

Black-box attacks In many real-world scenarios, black-box attacks are more realistic where attackers only have the query access to the model and do not have detailed model information. One type of black-box attacks is transferability-based (Papernot et al., 2017), where an adversary trains a substitute model with a substitute dataset and then generates adversarial examples from the substitute model using white-box attacks. Because of the transferability (Szegedy et al., 2014; Goodfellow et al., 2015; Papernot et al., 2016a), adversarial examples generated from the substitute model can potentially fool the targeted model, even if the two models have different architectures. One limitation of this approach is that it needs information about training data. Besides, attacking the substitute model leads to larger perturbation and lower success rate (Chen et al., 2017; Papernot et al., 2016a; 2017).

Soft-label black-box attacks (Chen et al., 2017; Narodytska & Kasiviswanathan, 2017; Ilyas et al., 2018; Zhao et al., 2019) rely on classification probability to generate adversarial examples. Since the classification probability vectors are usually inaccessible, hard-label black-box attacks are considered as more realistic, where only the final predicted labels are available to the attackers. Boundary attack (Brendel et al., 2017) is based on a random walk around the decision boundary using examples drawn from a proposal distribution. Label-only attack (Ilyas et al., 2018) uses discretized score, image robustness of random perturbation, and Monte Carlo approximation to find a proxy for the output probability and then uses an NES gradient estimator to generate adversarial examples in a similar way as soft-label black-box attacks. Opt attack (Cheng et al., 2019) reformulates the problem as a continuous real-valued optimization problem which can be solved by any zeroth-order optimization algorithm. The main limitation of these existing hard-label black-box attacks is that they need a large number of model queries, in part because they follow the traditional (approximated) optimization approach and they all aim to walk closer to a clean example starting from a point that is already adversarial. This limitation makes them inefficient and even infeasible in practice when the allowed number of queries is limited by a model. In contrast, our fuzzing-based hard-label black-box attacks start from a clean example and walk away from it step by step. With careful guidance and leveraging the randomness advantage of the fuzz testing approach, our attacks have the potential to use a much smaller number of queries to generate a successful adversarial example.

3 Approach and Algorithms

3.1 Overview of our Approach

Fuzz testing was first proposed by Miller et al. (1990). The key idea is to use random, unexpected, and unstructured data to find program failures. In recent years, fuzzers such as AFL (Zalewski, 2007) and libFuzzer (Serebryany, 2016) have gained great popularity because of their effectiveness and scalability. A typical fuzzer works by iteratively (1) selecting a seed input from a pool, (2) mutating the chosen seed to generate new inputs, (3) evaluating the newly generated inputs, and (4) recording observations such as program crashes and adding useful inputs into the seed pool. Our fuzzing-based hard-label black-box attacks leverage the basic idea of fuzz testing to explore the adversarial example space. The main challenge of applying fuzz testing to adversarial example generation is on exploring the gigantic search space. Theoretically, a fuzzer is able to explore the entire search space, but this is computationally infeasible. We address this challenge by using a guidance image and certain guidance strategy to improve the attack efficiency, i.e., fewer queries.

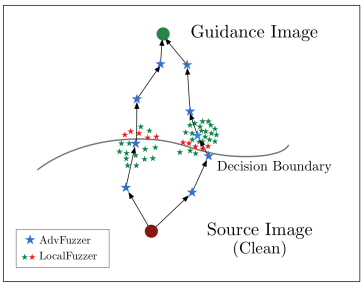

Figure 1: Basic Intuition of Our Attacks. Blue stars represent the main steps AdvFuzzer takes. Green and red stars represent the examples LocalFuzzer generates. Red stars are adversarial examples close to the decision boundary.

Figure 1 depicts the basic intuition of our attacks. We design two fuzzers: an adversarial fuzzer (referred to as *AdvFuzzer*) and a local fuzzer (referred to as *LocalFuzzer*). Starting from a source

image which is clean, AdvFuzzer slowly (thus with less amount of perturbations) walks away from it step by step by randomly selecting a path toward a guidance image. With multiple runs, AdvFuzzer will explore multiple random paths aiming to explore different regions of the adversarial example space between the two images. The guidance image could be a clean image of a specific target class for performing targeted attacks, or any image not of the source (image) class for performing untargeted attacks. LocalFuzzer aims to explore the nearby data points for identifying potential adversarial examples around the current image as AdvFuzzer takes the main steps.

It is important to point out that AdvFuzzer walks from a source image to a guidance image, thus taking a reverse direction from what is taken in existing hard-label black-box attacks including Boundary attack (Brendel et al., 2017), Label-only attack (Ilyas et al., 2018), and Opt attack (Cheng et al., 2019). Combining this strategy with the randomness advantage of the fuzz testing approach, our attacks have the potential to use a much smaller number of queries to more efficiently or practically generate successful adversarial examples. Moreover, LocalFuzzer can indeed be applied to any attack methods including black-box and white-box ones. When LocalFuzzer is supplied with a successful adversarial example as a seed, it can efficiently generate new successful adversarial examples in bulk and among which further optimized (e.g., in terms of reducing the L_2 distance from the source image) ones could even be identified.

3.2 ALGORITHMS

AdvFuzzer The logic for AdvFuzzer is presented in Algorithm 1. It generates an adversarial example img_{adv} as output, and takes a target model f, *isTargeted* parameter, a source image img_s, a guidance image img_g, and attack guidance strategy k as inputs. For every iteration of the while loop, a random main step is taken by selecting a perturbation ϵ based on the attack guidance strategy k (Line 4). We adopt a L_0 strategy which changes a random pixel in the perturbation ϵ image to the corresponding difference pixel value between the guidance image and the current image. Note that other strategies based on L_∞ and L_2 could also be applied but we found L_0 strategy is the most effective. If the current image reaches the attack goal, a medium level LocalFuzzer is applied to explore the nearby space for potentially generating more successful adversarial examples. Otherwise, a small level LocalFuzzer is used to check if some successful adversarial examples could still be found nearby. The while loop ends whenever a successful example is found or the number of steps reaches the maximum number of steps which is the L_0 distance between the source image and the guidance image. It will then fine tune a random successful adversarial example in the adversarial example set S_{adv} by using a walk_back fuzzer. The walk_back fuzzer works in a similar manner as LocalFuzzer as described below.

Algorithm 1 AdvFuzzer: generating an adversarial example using fuzzing

Input: f: a black-box model,
$isTargeted$: *True* for targeted attack and *False* for untargeted attack,
img_s: a source image with class s, *i.e.*, $f(img_s) = s$,
img_g: a guidance image with $f(img_g) = t$ for targeted attack
or $f(img_g) != s$ for untargeted attack,
k: attack guidance, *e.g.*, based on L_0 or L_∞ distance.
Output: img_{adv}: an adversarial example from a successful adversarial examples set S_{adv}.

1: $S_{adv} \leftarrow \emptyset$
2: $img_{cur} \leftarrow img_s$
3: **while** size(S_{adv})== 0 and $num\_steps < MAX\_STEPS$ **do**
4: $\epsilon = SelectPerturbationAlongMainDirection(img_{cur}, img_g, k)$
5: $img_{cur} = img_{cur} + \epsilon$
6: **if** (*isTargeted* and $f(img_{cur})$ == t) or (!*isTargeted* and $f(img_{cur})$!= s) **then**
7: $S_{adv} = S_{adv} \bigcup LocalFuzzer(f, img_{cur}, s, t, isTargeted, medium\_level)$
8: **else**
9: $S_{adv} = S_{adv} \bigcup LocalFuzzer(f, img_{cur}, s, t, isTargeted, small\_level)$
10: num_steps += 1
11: $img_{adv} = walk\_back(f, img_s, S_{adv}[0], s, t, medium\_level, isTargeted)$
12: **return** img_{adv}

LocalFuzzer LocalFuzzer is described in Algorithm 2. Its goal is to generate a set of adversarial examples potentially around an input image img. An adversarial example set S_{adv} and a set S_{all} including all examples are maintained as seed pools. A candidate seed is randomly selected and then mutated. For an input image, *fuzzer_level* controls the total number of mutated images to be generated. In each iteration, a random perturbation based mutation is applied. The mutation function randomly selects one pixel of an image and changes its value to any real number between 0 and 1. The mutated image is then added to the adversarial set S_{adv} if it meets the attack goal. The loop stops once the *fuzzer_level* is reached and the adversarial image set S_{adv} (could be empty) is returned. The difference between a LocalFuzzer and a walk_back fuzzer is the mutation. Instead of changing one random pixel to a random value between 0 and 1, the walk_back fuzzer mutates an image by randomly changing some pixel values of a current image closer or directly to that of the source image.

Algorithm 2 LocalFuzzer: generating a set of adversarial examples using fuzzing locally

Input: f: a black-box model,
img: an input image,
s: source image class,
t: target class,
$isTargeted$: *True* for targeted attack and *False* for untargeted attack,
$fuzzer\_level$: level of local fuzzing.
Output: S_{adv}: a set of successful adversarial examples.

1: $S_{adv} \leftarrow \emptyset$
2: $S_{all} \leftarrow \{img\}$
3: **if** (*isTargeted* and $f(img)$ == t) or (!*isTargeted* and $f(img)$!= s) **then**
4: S_{adv} = $S_{adv} \bigcup \{img\}$
5: **for** i from 1 to $fuzzer\_level$ **do**
6: **if** S_{adv} != $\emptyset$ **then**
7: $img_{rand} = RandomSelect(S_{adv})$
8: **else**
9: $img_{rand} = RandomSelect(S_{all})$
10: img_{mut} = $mutation(img_{rand})$
11: **if** (*isTargeted* and $f(img_{mut})$ == t) or (!*isTargeted* and $f(img_{mut})$!= s) **then**
12: S_{adv} = $S_{adv} \bigcup \{img_{mut}\}$
13: S_{all} = $S_{all} \bigcup \{img_{mut}\}$
14: **return** S_{adv}

Summary of Guidance Strategies. In AdvFuzzer, starting from the source image, each main step is taken by changing one pixel toward the guidance image (Algorithm 1, lines 4-5). This step is guided by the L_0 strategy and is obviously not random. In LocalFuzzer, only one pixel in the seed image is mutated (Algorithm 2, line 10). This step is random on purpose because we want to leverage the randomness advantage of fuzz testing. The walk_back fuzzer mutates an image by randomly changing some pixel values of a current image closer or directly to that of the source image. This step is random on the selection of pixels but guided on value changes. In essence, we considered both certain guidance strategies and some randomness in our overall approach. Lacking either of them would result in poor performance than what we can achieve at this moment.

4 Experimental Results

We now evaluate the feasibility and effectiveness of our fuzzing attacks. We use two standard datasets: MNIST (LeCun & Cortes, 2010) and CIFAR-10 (Krizhevsky, 2009). We compare fuzzing attacks with four attacks including Boundary attack (Brendel et al., 2017), Opt attack (Cheng et al., 2019), and C&W L_0 and L_2 attacks (Carlini & Wagner, 2017). Note that C&W attacks are white-box attacks. All the experiments are performed for both targeted attacks and untargeted attacks. Label-only attack (Ilyas et al., 2018) was evaluated by the authors only on ImageNet and we could not find its code for MNIST and CIFAR-10, so we did not include it in our evaluation.

To have a fair comparison, we adopt the same network architecture for MNIST and CIFAR-10 used in Carlini & Wagner (2017). Note that Brendel et al. (2017) also used the same network architecture, which has two convolution layers followed by a max-pooling layer, two convolution layers, a max-pooling layer, two fully-connected layers, and a softmax layer. Using the same parameters as reported in Carlini & Wagner (2017), we obtained 99.49% and 82.71% test accuracy for MNIST and CIFAR-10, respectively.

4.1 Overall Results

From the test set of each dataset, we randomly selected 10 images for each of the 10 classes. These same 100 images are used as source examples for all attacks. We use white-box C&W L_0 and L_2 attacks[1] as baselines, and use Boundary attack[2] and Opt attack[3] for comparison. We adopt the default parameters for the four attacks from their corresponding original implementations. As for fuzzing attacks, we use three different small LocalFuzzer levels including 100, 300, and 500. We report the average L_0, L_2, and L_∞ distances between a successful adversarial example and a source image, and the average number of queries for successful adversarial examples generated from 100 attack attempts. L_0 measures the number of different pixels between two images. L_2 is the Euclidean distance. L_∞ measures the maximum pixel value. Using the default attack parameters in the corresponding implementations, Opt attack, C&W attacks, and our fuzzing attacks can achieve 100% success rates while Boundary attack achieves between 90% and 100% success rates. Note that due to the randomness of our fuzzing attacks, the distance value and number of query results could vary in a certain range from run to run. So we also report median and standard deviation of distance value and number of query results in Tables 3, 4, 5, and 6 of Appendix A.

Untargeted Attacks An untargeted attack is successful when an adversarial image is classified as any class other than the source class. Although the guidance images in untargeted attacks could be any randomly generated images or legitimate images classified as any class other than the source class, we use the same guidance images in the targeted attacks for consistency. Also, we found that there is not much difference between using a randomly generated guidance image and a legitimate image. The results for untargeted attacks are summarized in Table 1. While Boundary attack and Opt attack can generate successful adversarial examples with smaller L_2 distance compared to C&W attacks, they tend to change most of the pixels in an image since their average L_0 is higher. However, adversarial examples generated from our fuzzing attacks have larger L_2 and L_∞ distances but smaller L_0 distance. The results are comparable to C&W L_0 attack. The first row in Figure 2 shows the successful adversarial examples from the untargeted fuzzing attacks on MNIST and CIFAR-10. The perturbations on CIFAR-10 are largely imperceptible by human eyes while they are more obvious on MNIST. This is mostly due to the uniform dark background of MNIST images (Luo et al., 2018) especially when L_0 is reduced and L_2 is increased. What we want to highlight is that the average number of queries used by our fuzzing attacks is significantly decreased by 10-18 folds and 2-5 folds for MNIST and CIFAR-10, respectively. As shown in Tables 3 and 4 of Appendix A, the median number of queries for fuzzing attacks are all smaller, on both MNIST and CIFAR-10, than those for Opt and Boundary attacks; the standard deviation results on number of queries for fuzzing attacks on CIFAR-10 are higher than those of Boundary and Opt attacks.

We also report average, median, and standard deviation results for untargeted attacks on ImageNet using fuzzing attacks in Table 7 of Appendix A. The small LocalFuzzer level is 10. We use 10 source examples and correspondingly 10 guidance images to perform the experiments. The average L_2 distance for fuzzing attacks is 6.3490, and the average number of queries is 102,652. Note that as reported in Table 1 of Cheng et al. (2019), the average L2 distance for untargeted Opt attacks is between 3.1120 and 6.9796, and the average number of queries is between 71,100 and 237,342; the average L2 distance for untargeted Boundary attacks is between 3.7725 and 5.9791, and the average number of queries is between 123,407 and 260,797.

[1] https://github.com/carlini/nn_robust_attacks

[2] https://github.com/bethgelab/foolbox

[3] https://github.com/LeMinhThong/blackbox-attack

Figure 2: Successful adversarial examples from fuzzing attacks on MNIST and CIFAR-10

Table 1: Average Results for Untargeted Attacks

	MNIST				CIFAR-10			
	Avg L_0	Avg L_2	Avg L_∞	# queries	Avg L_0	Avg L_2	Avg L_∞	# queries
Boundary attack	769	1.1454	0.2657	115,134	3,071	0.1658	0.0182	128,296
Opt attack	784	1.0839	0.3067	67,536	3,072	0.1671	0.0164	56,438
C&W L_0 attack	10	2.5963	0.9519	-	9	0.9988	0.4450	-
C&W L_2 attack	749	1.4653	0.4299	-	3,072	0.1894	0.0226	-
Fuzzing attack 100	19	2.5536	0.9706	6,321	33	1.7838	0.5471	24,045
Fuzzing attack 300	19	2.5041	0.9661	14,697	32	1.7176	0.6593	56,678
Fuzzing attack 500	19	2.4525	0.9673	22,546	28	1.6822	0.6526	88,488

Targeted Attacks We consider *next label targeted attacks* (Brendel et al., 2017; Cheng et al., 2019) where the adversarial goal is for an adversarial example to be misclassified as a target class y_t such that $y_t = (y + 1)$ module 10 where y is the source class. The results for targeted attacks are shown in Table 2. The second row in Figure 2 shows the successful adversarial examples from the targeted fuzzing attacks on MNIST and CIFAR-10. The perturbations for targeted attacks are visually larger than the perturbations for untargeted attacks. Similarly, the fuzzing attack decreases the average L_0 distance while increases the average L_2 and L_∞ distances. The fuzzing attacks also reduce the average number of queries by 8-9 folds and 2-2.5 folds for MNIST and CIFAR-10, respectively. Note that it is harder to find successful adversarial examples using LocalFuzzer levels of 300 and 500 on CIFAR-10 because of the larger feature space compared to MNIST and the smaller adversarial space in the targeted attacks compare to that of the untargeted attacks. As shown in Tables 5 and 6 of Appendix A, the median number of queries for fuzzing attacks are all smaller than those for Opt and Boundary attacks on MNIST, and that is also true for the *fuzzing attack 100* setting on CIFAR-10; the standard deviation results on number of queries for fuzzing attacks on CIFAR-10 are higher than those of Boundary and Opt attacks, but that is not true for the *fuzzing attack 100* setting on MNIST.

Table 2: Average Results for Targeted Attacks

	MNIST				CIFAR-10			
	Avg L_0	Avg L_2	Avg L_∞	# queries	Avg L_0	Avg L_2	Avg L_∞	# queries
Boundary attack	773	1.8393	0.4421	126,433	3,071	0.2368	0.0224	128,657
Opt attack	784	1.9040	0.4525	108,556	3,072	0.6231	0.0461	110,124
C&W L_0 attack	26	3.6744	0.8853	-	19	1.7123	0.6256	-
C&W L_2 attack	742	2.1752	0.5088	-	3,072	0.3491	0.0425	-
Fuzzing attack 100	43	3.3925	0.9750	13,364	57	2.4066	0.6997	50,084
Fuzzing attack 300	42	3.3553	0.9800	35,002	109	3.5439	0.7579	272,274
Fuzzing attack 500	41	3.3690	0.9782	54,811	104	3.4381	0.7652	412,766

Computation time. Computation time results for both untargeted and targeted attacks on MNIST and CIFAR-10 are presented in Table 10 and Table 11 of Appendix A, respectively. The small LocalFuzzer level for fuzzing attacks is 100. The results are calculated based on 10 source examples. In term of untargeted attacks, fuzzing attacks take significantly less total time and generation time to generate successful adversarial examples compared to Boundary and Opt attacks. As for targeted attacks, fuzzing attacks still take less total time and generation time on MNIST, and have comparable results as Boundary and Opt attacks on CIFAR-10.

Overall, the randomness advantages of the fuzzing approach help to reduce the average number of queries and L_0 distance by sacrificing a small amount of L_2 distance. The experimental results show that fuzzing attacks are feasible and effective. However, they potentially can be further strengthened by improving the fuzzing process such as seed selection, main direction selection, step selection, mutation operation selection, etc. It is important to note that minimizing L_2 distance could be the

goal of many adversarial example generation algorithms, but is not necessarily the most important goal of real-world attackers. In other words, attackers' adversarial examples need not be "perfect" (in terms of L_2 distance). Whatever adversarial examples that are reasonably good (e.g., most humans can still recognize as the source class but models will be fooled) can be used by attackers. It is common in many types of real-world attacks that techniques need not be perfect. For example, in phishing attacks, super simple spoofed login webpages are still widely used by attackers.

4.2 Results on Applying LocalFuzzer on Successful Adversarial Examples

We now apply LocalFuzzer on successful adversarial examples generated from Boundary attack, Opt attack, C&W L_0 and L_2 attacks, and our fuzzing attacks to more intensively evaluate the bulk generation capability of LocalFuzzer. We leverage the successful adversarial examples from the experiments in Section 4.1 for both untargeted and targeted attacks. For fuzzing attacks, we use the adversarial examples generated with a LocalFuzzer level of 500 which are shown in the third row of fuzzing attacks in both Table 1 and Table 2. Three experiments with different LocalFuzzer levels of 100, 1,000, and 5,000 are performed.

We report six metrics including M1: success bulk generation rate (i.e., the percent of bulk runs returning successful examples); M2: average number of successful examples generated in a bulk run; M3: success bulk generation rate with lower L_2 (i.e., percent of success bulk runs returning adversarial examples with L_2 lower than that of the seed image); M4: average number of successful examples with lower L_2 in a bulk run; M5: average decreased L_2 of successful examples with lower L_2; M6: average L_2 decreasing rate of successful examples with lower L_2.

The results for untargeted attacks and targeted attacks are presented in Table 8 and Table 9 of Appendix A, respectively. We achieve 100% success bulk generation rate across runs in all experiments, which indicates the great benefits of the bulk generation capability of LocalFuzzer. It is also demonstrated that the "optimized" adversarial examples from Boundary attack and Opt attack can further be refined. LocalFuzzer takes relatively a smaller number of queries (i.e., 100, 1,000, and 5,000 in the experiments) compared with what was taken originally in Boundary attack and Opt attack, but can immediately generate more successful adversarial examples even with smaller L_2 distance from the source example. Meanwhile, the bulk generation of LocalFuzzer based on adversarial examples of white-box including C&W L_0 and L_2 attacks is also effective. These results indicate that LocalFuzzer itself can be an independent and useful tool to augment many adversarial example generation algorithms including both black-box and white-box attacks.

5 Conclusion and Discussion

Inspired by the similarities between attacking a machine learning model and testing the correctness or security bug of a program, we proposed fuzzing-based hard-label black-box attacks to generate adversarial examples. We designed two fuzzers, AdvFuzzer and LocalFuzzer, to explore multiple random paths between a source image and a guidance image, and the nearby space of each step along the way. We evaluated our fuzzing attacks using MNIST and CIFAR-10 datasets, and compared ours with four existing attacks including Boundary attack, Opt attack, and C&W L_0 and L_2 attacks. The experimental results demonstrated that our fuzzing attacks are feasible and effective. Moreover, LocalFuzzer has the bulk successful example generation capability and distance refinement capability on adversarial examples generated from different attack methods. We would recommend LocalFuzzer as an independent and useful tool for augmenting many adversarial example generation algorithms.

Our work provides evidence on adopting fuzz testing in the adversarial example generation domain. Although the randomness advantage of the fuzzing attacks could help reduce the number of model queries, one limitation of our attacks is that they sacrifice the L_2 distance to a small extent. We expect that further improvement of the fuzzing process could be explored to construct more powerful fuzzing-based attacks. For example, potential ways for improvement could be a better guidance strategy in the main direction selection, a refined seed selection process, a refined mutation function, etc. We are working on improving our approach and we hope our fuzzing attacks could inspire more related research in the future.

REFERENCES

Dennis Appelt, Cu Duy Nguyen, Lionel C. Briand, and Nadia Alshahwan. Automated testing for sql injection vulnerabilities: An input mutation approach. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion Attacks against Machine Learning at Test Time. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 387–402, 2013.

Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations (ICLR)*, 2017.

Nicholas Carlini and David A. Wagner. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, 2017.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. pp. 15–26, 11 2017.

Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *International Conference on Learning Representations (ICLR)*, 2019.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations (ICLR)*, 2015.

Istvan Haller, Asia Slowinska, Matthias Neugschwandtner, and Herbert Bos. Dowsing for overflows: A guided fuzzer to find buffer boundary violations. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013.

Jamie Hayes and George Danezis. Learning universal adversarial perturbations with generative models. pp. 43–49, 2018.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

Dae R. Jeong, KyungTae Kim, Basavesh Shivakumar, Byoungyoung Lee, and Insik Shin. Razzer: Finding kernel race bugs through fuzzing. pp. 754–768, 2019.

George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. Evaluating fuzz testing. In *ACM Conference on Computer and Communications Security*, 2018.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial Examples in the Physical World. In *International Conference on Learning Representations (ICLR) Workshop Track*, 2017.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

Bo Luo, Yannan Liu, Lingxiao Wei, and Qiang Xu. Towards imperceptible and robust adversarial example attacks against neural networks. In *AAAI*, 2018.

Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. In *In Proceedings of the Workshop of Parallel and Distributed Debugging*, pp. pages ix–xxi,. Academic Medicine, 1990.

Glenford J. Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 3rd edition, 2012.

Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial perturbations for deep networks. *ArXiv*, abs/1612.06299, 2017.

Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 4901–4911, 2019.

Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *CoRR*, abs/1605.07277, 2016a.

Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The Limitations of Deep Learning in Adversarial Settings. In *IEEE European Symposium on Security and Privacy*, pp. 372–387, 2016b.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks Against Machine Learning. In *Asia Conference on Computer and Communications Security*, pp. 506–519, 2017.

Kostya Serebryany. Libfuzzer: A library for coverage-guided fuzz testing (within llvm). 2016.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2014.

Ari Takanen, Jared DeMott, and Charlie Miller. *Fuzzing for Software Security Testing and Quality Assurance*. Artech House, Inc., 1st edition, 2008.

Xiaofei Xie, Lingfei Ma, Felix Juefei-Xu, Hongxu Chen, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, Jianxiong Yin, and Simon See. Deephunter: Hunting deep neural network defects via coverage-guided fuzzing. abs/1809.01266, 2018.

Michal Zalewski. American fuzzy lop. 2007.

Pu Zhao, Sijia Liu, Pin-Yu Chen, Nghia Nguyên Hoàng, Kaidi Xu, Bhavya Kailkhura, and X Cai Lin. On the design of black-box adversarial examples by leveraging gradient-free optimization and operator splitting method. *ArXiv*, abs/1907.11684, 2019.

A APPENDIX

Table 3: Median Results (corresponding to Table 1) for Untargeted Attacks

	MNIST				CIFAR-10			
	Median L_0	Median L_2	Median L_∞	# queries	Median L_0	Median L_2	Median L_∞	# queries
Boundary attack	770	1.1400	0.2614	114,843	3072	0.1392	0.0157	130,499
Opt attack	784	1.0793	0.2921	62,995	3072	0.1619	0.0150	55,762
C&W L_0 attack	9	2.5594	0.9824	-	6	0.9896	0.4737	-
C&W L_2 attack	750	1.4672	0.4268	-	3072	0.1666	0.0185	-
Fuzzing attack 100	19	2.5079	0.9961	5,902	16	1.1017	0.5782	7,306
Fuzzing attack 300	18	2.4804	0.9922	13,436	20	1.5417	0.6916	18,588
Fuzzing attack 500	18	2.3291	0.9922	20,381	20	1.4422	0.6745	17,614

Table 4: Standard Deviation Results (corresponding to Table 1) for Untargeted Attacks

	MNIST				CIFAR-10			
	SD L_0	SD L_2	SD L_∞	# queries	SD L_0	SD L_2	SD L_∞	# queries
Boundary attack	6.6278	0.3091	0.0730	13,622	2.5784	0.1045	0.0114	5,377
Opt attack	0	0.2897	0.1141	15,081	0.2611	0.1016	0.0121	24,704
C&W L_0 attack	5.5938	0.7662	0.0923	-	7.9998	0.7724	0.2894	-
C&W L_2 attack	16.5430	0.4513	0.1254	-	0.1407	0.1665	0.0220	-
Fuzzing attack 100	8.8428	0.6503	0.0915	3,433	20.4103	0.7776	0.1828	25,550
Fuzzing attack 300	9.6684	0.6617	0.0913	9,613	38.2232	1.0067	0.1669	95,872
Fuzzing attack 500	9.1561	0.6417	0.0907	14,451	32.1710	1.0792	0.1837	167,639

Table 5: Median Results (corresponding to Table 2) for Targeted Attacks

	MNIST				CIFAR-10			
	Median L_0	Median L_2	Median L_∞	# queries	Median L_0	Median L_2	Median L_∞	# queries
Boundary attack	776	1.7285	0.4090	130,896	3072	0.2222	0.0213	129,650
Opt attack	784	1.6717	0.4194	120,291	3072	0.2587	0.0243	128,785
C&W L_0 attack	26	4.0491	0.9975	-	18	1.8128	0.6945	-
C&W L_2 attack	740	2.3436	0.5545	-	3072	0.3594	0.0431	-
Fuzzing attack 100	42	3.1988	0.9961	14,145	46	2.3002	0.7186	29,702
Fuzzing attack 300	35	3.1916	0.9961	35,100	82	3.3722	0.7592	178,573
Fuzzing attack 500	37	3.2281	0.9961	58,106	77	3.3818	0.7922	296,279

Table 6: Standard Deviation Results (corresponding to Table 2) for Targeted Attacks

	MNIST				CIFAR-10			
	SD L_0	SD L_2	SD L_∞	# queries	SD L_0	SD L_2	SD L_∞	# queries
Boundary attack	7.6341	0.6587	0.1605	8,004	1.9573	0.1248	0.0135	2,987
Opt attack	0	0.8567	0.1546	51,251	0.1407	1.2503	0.0693	52,281
C&W L_0 attack	10.1065	0.8887	0.0259	-	11.4445	0.8224	0.2214	-
C&W L_2 attack	16.1253	0.5706	0.1221	-	0.3555	0.2054	0.0252	-
Fuzzing attack 100	20.1423	0.8957	0.0553	5,616	41.8556	1.2002	0.1500	58,629
Fuzzing attack 300	20.2546	0.8563	0.0471	15,716	92.9558	1.8192	0.1150	231,510
Fuzzing attack 500	18.4257	0.8826	0.0508	26,317	103.7606	1.7056	0.1280	370,780

Table 7: Average, Median, and Standard Deviation Results for Untargeted Attacks on ImageNet using Fuzzing attacks. The small LocalFuzzer level is 10. Note that as reported in Table 1 of Cheng et al. (2019), the average L_2 distance for untargeted opt attacks is between 3.1120 and 6.9796; the average L_2 distance for untargeted boundary attacks is between 3.7725 and 5.9791.

	L_0	L_2	L_∞	# queries
Average	311	6.3490	0.8808	102,652
Median	224	5.9794	0.9090	48,507
SD	317	4.0062	0.0873	120,909

Table 8: Bulk Generation of Successful Examples for Untargeted Attacks

	Attack	M1	M2	M3	M4	M5	M6
MNIST	Boundary attack 100	100%	54	48.10%	8	2.98e-3	0.28%
	Boundary attack 1,000	100%	736	58.23%	24	4.07e-3	0.36%
	Boundary attack 5,000	100%	3,851	55.70%	56	4.93e-3	0.48%
	Opt attack 100	100%	63	66.00%	16	2.45e-3	0.23%
	Opt attack 1,000	100%	727	59.00%	50	4.96e-3	0.47%
	Opt attack 5,000	100%	3,888	73.00%	112	5.64e-3	0.56%
	Fuzzing attack 100	100%	92	67.00%	7	1.97e-2	0.90%
	Fuzzing attack 1,000	100%	939	100%	75	3.19e-2	1.46%
	Fuzzing attack 5,000	100%	4,730	100%	425	3.99e-2	1.81%
	C&W L_0 attack 100	100%	92	59.00%	7	5.61e-3	0.21%
	C&W L_0 attack 1,000	100%	950	99.00%	63	8.37e-3	0.34%
	C&W L_0 attack 5,000	100%	4778	99.00%	475	1.11	0.46%
	C&W L_2 attack 100	100%	81	18.00%	1	8.41e-4	0.06%
	C&W L_2 attack 1,000	100%	861	30.00%	2	1.10e-3	0.07%
	C&W L_2 attack 5,000	100%	4534	37.00%	2	1.14e-3	0.08%
CIFAR-10	Boundary attack 100	100%	57	3.26%	3	0.14e-3	0.07%
	Boundary attack 1,000	100%	676	6.52%	4	0.17e-3	0.07%
	Boundary attack 5,000	100%	3,726	9.78%	8	0.29e-3	0.10%
	Opt attack 100	100%	69	2.00%	2	0.15e-3	0.04%
	Opt attack 1,000	100%	757	7.00%	3	0.18e-3	0.06%
	Opt attack 5,000	100%	3,936	10.00%	4	0.12e-3	0.04%
	Fuzzing attack 100	100%	91	24.21%	8	6.34e-3	0.42%
	Fuzzing attack 1,000	100%	943	58.95%	54	8.93e-3	0.56%
	Fuzzing attack 5,000	100%	4,774	80.00%	258	1.23e-2	0.87%
	C&W L_0 attack 100	100%	100	22.00%	7	8.28e-4	0.07%
	C&W L_0 attack 1,000	100%	990	69.00%	30	1.36e-3	0.13%
	C&W L_0 attack 5,000	100%	4953	80.00%	179	1.78e-3	0.18%
	C&W L_2 attack 100	100%	97	53.00%	16	1.12e-4	0.05%
	C&W L_2 attack 1,000	100%	973	62.00%	100	2.13e-4	0.08%
	C&W L_2 attack 5,000	100%	4833	66.00%	419	2.73e-4	0.11%

Table 9: Bulk Generation of Successful Examples For Targeted Attacks

	Attack	M1	M2	M3	M4	M5	M6
MNIST	Boundary attack 100	100%	44	45.68%	11	2.27e-3	0.14%
	Boundary attack 1,000	100%	546	51.85%	44	4.61e-3	0.31%
	Boundary attack 5,000	100%	3,105	45.68%	110	6.21e-3	0.39%
	Opt attack 100	100%	46	53.33%	15	3.71e-3	0.21%
	Opt attack 1,000	100%	598	56.67%	62	6.08e-3	0.34%
	Opt attack 5,000	100%	3,333	67.67%	279	7.11e-3	0.42%
	Fuzzing attack 100	100%	93	82.22%	12	1.32e-2	0.42%
	Fuzzing attack 1,000	100%	946	100.00%	141	2.69e-2	0.89%
	Fuzzing attack 5,000	100%	4,722	98.89%	837	3.54e-2	1.17%
	C&W L_0 attack 100	100%	88	85.00%	11	5.92e-3	0.17%
	C&W L_0 attack 1,000	100%	897	99.00%	133	8.76e-3	0.26%
	C&W L_0 attack 5,000	100%	4532	98.00%	860	1.18e-2	0.36%
	C&W L_2 attack 100	100%	79	35.00%	1	9.08e-4	0.04%
	C&W L_2 attack 1,000	100%	851	49.00%	3	1.39e-3	0.06%
	C&W L_2 attack 5,000	100%	4476	59.00%	6	1.19e-3	0.06%
CIFAR-10	Boundary attack 100	100%	41	6.90%	10	0.21e-3	0.04%
	Boundary attack 1,000	100%	548	10.34%	12	0.20e-3	0.05%
	Boundary attack 5,000	100%	2,947	10.34%	42	0.20e-3	0.05%
	Opt attack 100	100%	54	15.56%	8	0.14e-3	0.02%
	Opt attack 1,000	100%	676	14.44%	11	0.22e-3	0.04%
	Opt attack 5,000	100%	3,084	20.00%	277	0.28e-3	0.05%
	Fuzzing attack 100	100%	96	58.67%	19	0.98e-3	0.04%
	Fuzzing attack 1,000	100%	979	90.67%	149	1.67e-3	0.08%
	Fuzzing attack 5,000	100%	4,920	96.00%	883	2.29e-3	0.09%
	C&W L_0 attack 100	100%	100	41.00%	9	7.87e-4	0.04%
	C&W L_0 attack 1,000	100%	981	92.00%	69	1.36e-3	0.09%
	C&W L_0 attack 5,000	100%	4902	99.00%	397	1.78e-3	0.14%
	C&W L_2 attack 100	100%	97	77.00%	23	1.41e-4	0.04%
	C&W L_2 attack 1,000	100%	971	89.00%	163	2.58e-4	0.07%
	C&W L_2 attack 5,000	100%	4875	90.00%	726	3.52e-4	0.09%

Table 10: Untargeted Attacks: Average, Median, and Standard Deviation Computation Time Results (in seconds). Total time is the time spent for each successful attack attempt. Generation time is the time used for generating images during each attack attempt. Query time is the difference between total time and generation time, not shown in the table.

	Attacks	MNIST		CIFAR-10	
		Total Time	Generation Time	Total Time	Generation Time
Average	Boundary attack	26.3981	15.5242	41.2357	21.3540
	Opt attack	96.2709	16.6981	85.6906	15.9244
	Fuzzing attack 100	7.1731	0.7869	10.7523	1.5613
Median	Boundary attack	25.5868	14.8883	41.1243	21.1549
	Opt attack	94.5755	16.3971	77.6759	14.7546
	Fuzzing attack 100	6.5157	0.7053	7.3152	1.0742
SD	Boundary attack	2.7221	2.0243	0.8511	1.4848
	Opt attack	16.8229	1.9818	25.9558	2.8943
	Fuzzing attack 100	2.3889	0.2903	7.6872	1.1437

Table 11: Targeted Attacks: Average, Median, and Standard Deviation Computation Time Results (in seconds). Total time is the time spent for each successful attack attempt. Generation time is the time used for generating images during each attack attempt. Query time is the difference between total time and generation time, not shown in the table.

	Attacks	MNIST		CIFAR-10	
		Total Time	Generation Time	Total Time	Generation Time
Average	Boundary attack	27.1896	16.1106	38.5594	20.0595
	Opt attack	45.7278	5.0152	145.9843	9.0527
	Fuzzing attack 100	26.0137	2.7582	91.0437	13.7374
Median	Boundary attack	27.5087	16.2304	38.1913	19.7272
	Opt attack	37.3375	4.5640	199.1068	10.7309
	Fuzzing attack 100	25.5663	2.7282	86.0931	12.9244
SD	Boundary attack	1.8790	1.5028	1.4813	1.3669
	Opt attack	21.2563	0.8855	86.8058	2.6389
	Fuzzing attack 100	2.7882	0.3020	74.0702	11.2148