HopSkipJumpAttack: An Efficient Adversarial Attack

1 Introduction

Many machine learning algorithms have been shown to be susceptible to adversarial examples. For example, image classification neural networks can wrongly classify an image when a small perturbation, unnoticeable to the human eye, is added to the original image which it has previously correctly classified. The goal of an adversarial attack can thus be rephrased as an optimization problem to compute the "smallest" perturbation needed, such that the perturbed example will be misclassified [1].

1.1 Threat Models

State-of-the-art adversarial attacks can be roughly divided into three categories: gradient-based, score-based and decision-based attacks [1]. In **gradient-based attacks**, also denoted as white-box attacks, adversaries are given access to the original underlying target model, and many attacks are often heavily reliant on detailed information including the gradient of the loss with respect to the input. In **score-based attacks**, adversaries are given predicted scores (e.g. class probabilities or logits) of the targeted model, and most attacks also rely on these numerical predictions to estimate the gradient. Lastly, in **decision-based attacks**, adversaries are only given black-box access to the decisions of the victim model.

Check Your Understanding (1) *Which category of attack is the most practical to carry out in real life? Why?*

1.2 HopSkipJumpAttack

In this paper, we focus on a particular decision-based attack called HopSkipJumpAttack [2]. HopSkipJumpAttack is effective against many popular defense mechanisms like defensive distillation, region-based classification and adversarial training, and also on non-differentiable machine learning models like Random Forest and binarization. This attack also requires significantly fewer queries to the black-box models as compared to its counterparts like the Boundary Attack (which is based on iterative rejective sampling). In the following sections, we first introduce the optimization model for decision-based attacks and some metrics used to evaluate its effectiveness, before describing the attack algorithm. We then describe our experiments on attacking the Google Cloud Vision API [3] and discuss our results. We finally conclude by comparing HopSkipJumpAttack to other established decision-based attacks.

2 Background and Framework

In this section, we describe the classification model that we are attempting to attack, and introduce concepts and terminology used in the algorithm.

2.1 Victim Classification Model

We assume that the victim classification model takes in an input image of dimension d, and assigns it probabilities for m possible classes, which must sum up to 1. This can be described by F:

$$F : \mathbb{R}^d \to \Delta_m := \{y \in [0,1]^m : \sum_{c=1}^{m} y_c = 1\}$$

The decision of the model can be written as

$$C(x) = \arg\max_c F(x)_c$$

where x is the original image, and it picks the class with the highest probability given x. We refer to the adversarial perturbed image as x'. We use the ℓ_2-distance in our explanations below, but the algorithm can be generalized to other distance metrics like ℓ_∞, ℓ_0.

2.2 Attack Objectives

There are two possible attack objectives. In *untargeted attacks*, the goal of the adversary is to find an adversarial image such that the victim model misclassifies the image. In *targeted attacks*, the adversary tries to produce an image such that the victim model classifies the image incorrectly as a particular class c^*. Targeted attacks are strictly stronger than untargeted attacks [2].

Untargeted attacks
An untargeted attack has the following objective function:

$$\min_{x'} ||x', x||_2 \text{ subject to } C(x') \neq C(x)$$

Intuitively, this means that we are trying to find the least-perturbed image such that the model misclassifies it.

Targeted attacks
A targeted attack has the following objective function:

$$\min_{x'} ||x', x||_2 \text{ subject to } C(x') = c^*$$

This says that we want the least-perturbed image x' such that the model classifies x' to be our desired class c^*.

2.3 Attack Boundary

We introduce the boundary function S that takes in our perturbed image, and returns a value that represents the success of the attack. We define S as follows:

$$S(x') = \begin{cases} \max\limits_{c \neq C(x)} \{F(x')_c\} - F(x')_{C(x)} & \text{for untargeted attacks} \\ F(x')_{c^*} - \max\limits_{c \neq c^*} \{F(x')_c\} & \text{for targeted attacks} \end{cases}$$

Check Your Understanding (2) *Explain what $S(x')$ does intuitively. What does $S(x') > 0$ and $S(x') < 0$ imply respectively?*

From working through the exercise, we know that our attack succeeds when $S(x') > 0$ and fails when $S(x') < 0$. We can therefore represent a more generalized objective function for the two kinds of attacks as

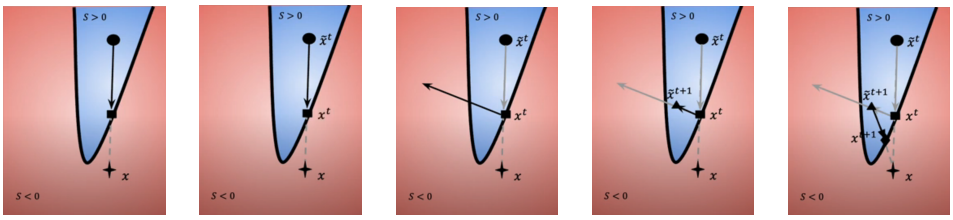
$$\min ||x', x||_2 \text{ subject to } S(x') > 0$$

3 Attack Algorithm

We first provide a brief overview of the general attack plan. We refer to Figure 1 for a simple visualization of the algorithm at each step. We define x as the original image, x_t as the perturbed image at step $t = 1, 2, \cdots$. Further, we have a starting point $\tilde{x}_t$ at each step, and we set $\tilde{x}_0$ to be an image in the target class for the targeted attack, or a misclassified uniform noise for the untargeted attack. Each iteration t can be summarized as follows:

1. Perform boundary search using binary search with x and $\tilde{x}_{t-1}$ to find x_t, as our adversarial example at step t. (Figure 1(b))
2. Estimate the gradient at the boundary, using a batch of random vectors. (Figure 1(c))
3. Find the step size, and use gradients to update x_t to $\tilde{x}_t$ for use in the next iteration. (Figure 1(d))

Note that in Figure 1, the blue area represents the space where $S(x') > 0$ which is when the attack is successful, and the red area represents where $S(x') < 0$ which is when the attack is unsuccessful. We now elaborate on each step of the algorithm.



(a) Initialization (b) Boundary search (c) Estimate gradients (d) Update x_t using gradients (e) New iteration, repeat boundary search

Figure 1: Overview of the Attack Algorithm (Images from [2])

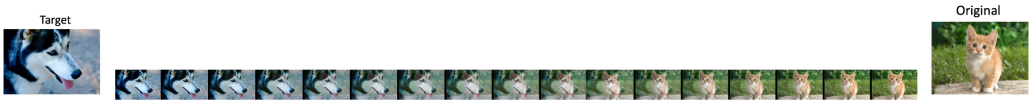


Figure 2: Binary Search for boundary (Images from [2])

3.1 Boundary Binary Search

Firstly, at the start of iteration t, we conduct binary search along the line connecting the x and $\tilde{x}_{t-1}$ to find a point that is sufficiently close to the decision boundary.

Formally, we initialize $\alpha_l = 0$, $\alpha_u = 1$. At each step of the binary search, we set $\alpha_m = \frac{\alpha_l + \alpha_u}{2}$, query the victim model at $\alpha_m x + (1 - \alpha_m)\tilde{x}_{t-1}$, and reset either α_l, α_u accordingly. We stop the binary search when $|\alpha_u - \alpha_l| < \theta$ with threshold $\theta = d^{-\frac{3}{2}}$, where d is the dimension of our image. Setting such θ ensures that the bias of our gradient estimation in the next step is bounded.

Figure 2 visualizes how the binary search of the first iteration will look like for an image classification victim model, where we start off with an image of a cat on the far right as input x, and attempt to fool the model to misclassify it as a dog, in the case of a targeted attack. The point in the blue region $\tilde{x}_{t-1}$ will be the image of the dog on the far left. For each step of the binary search, we query the model with a linear combination of the cat and dog images, until our result is close enough to the decision boundary.

3.2 Gradient Direction Estimation

In the second part of the algorithm, we estimate the gradient direction vector at the boundary. The gradient vector will allow us to arrive at a better adversarial example in the next iteration. However, without knowing the model itself, we can only provide an estimate operating solely based on querying labels in a black-box manner.

At iteration t, we begin by sampling a batch of $B_t = B_0\sqrt{t}$ random unit vectors u_b. We set the size of the perturbation to be $\delta_t = d^{-1}||\tilde{x}_{t-1} - x||_2$ so as to upper bound the bias of our estimation (assuming $\theta = d^{-\frac{3}{2}}$). We further define $\phi(x') := \text{Sign}(S(x'))$, where

$$\text{Sign}(S(x')) = \begin{cases} 1 & \text{if } S_x(x') > 0, \text{ i.e., attack succeeds} \\ -1 & \text{otherwise} \end{cases}$$

We can then find our approximate gradient at the boundary $\widehat{\nabla S}(x_t, \delta_t)$ as follows,

$$\overline{\phi} := \frac{1}{B}\sum_{b=1}^{B} \phi(x_t + \delta_t u_b)$$

$$\widehat{\nabla S}(x_t, \delta_t) := \frac{1}{B-1}\sum_{b=1}^{B} (\phi(x_t + \delta_t u_b) - \overline{\phi})u_b$$

Note that for any boundary point x', the direction of $\widehat{\nabla S}(x', \delta)$ is an asymptotically unbiased estimate as $\delta \to 0^+$. Now, since we only need the direction, we normalize our estimate as

$$v_t(x_t, \delta_t) = \frac{\widehat{\nabla S}(x_t, \delta_t)}{||\widehat{\nabla S}(x_t, \delta_t)||_2}$$

Check Your Understanding (3) *Let* $\widetilde{\nabla S}(x_t, \delta_t) = \frac{1}{B}\sum_{b=1}^{B} \phi(x_t + \delta u_b)u_b$. *Why does* $\widetilde{\nabla S}(x_t, \delta_t)$ *provide a reasonable approximation of the gradient?*

3.3 Step Size Update

Thirdly, at the end of each iteration t, we want to update the size of the step we take in the direction vector v_t determined above, which is normal to the decision boundary. After we determine the step size ξ_t, we prepare $\tilde{x}_t = x_t + \xi_t v_t$ for another binary search in the next iteration.

Now, to find an optimal step size ξ_t for iteration t, we simply perform the following.

1. Set ξ_t to $\frac{||x_t - x||_2}{\sqrt{t}}$.
2. While $\phi_x(x_t + \xi_t v_t) = -1$, set ξ_t to $\frac{\xi_t}{2}$.

By using the step size ξ_t defined above, this algorithm is guaranteed to find a stationary point of the problem due to the following result.

Suppose we use step size $\xi_t = ||x_t - x||_2 t^{-q}$ for some $q \in \left(\frac{1}{2}, 1\right)$, there a exists universal constant c such that for any t,

$$\begin{aligned} 0 \leq 1 - r(x_t, x) &\leq ct^{q-1}, \\ \text{where } r(x_t, x) &= \cos\angle(x_t - x, \nabla S_x(x_t)) \\ &= \frac{\langle x_t - x, \nabla S_x(x_t)\rangle}{||x_t - x||_2 ||\nabla S_x(x_t)||_2} \end{aligned}$$

Check Your Understanding (4) *Let* $x = (2, 1, 3)$, $x_t = (2, 5, 1)$, $\nabla S_x(x_t) = (0, 2, -1)$. *What is the value of* $r(x_t, x)$*? If* $v_t = \nabla S_x(x_t)$*, what is the value of* $|\cos\angle(x_{t+1} - x_t + \nabla S_x(x_t), x_{t+1} - x|$*?*

We are now justified to initially upper bound ξ_t to $\frac{||x_t - x||_2}{\sqrt{t}}$ and keep decreasing it until $\phi_x(x_t + \xi_t v_t) = 1$, so as to guarantee we converge to a stationary point. The requirement on ϕ_x is to ensure that binary search for a point near boundary in the next iteration is possible.

4 Experiments: Attacking Google Cloud Vision AI on Arbitrary Images

In this section, we analyze the HopSkipJumpAttack with some experimental results. Given that the HopSkipJump attack is solely decision-based, we decide to carry out an experiment with existing image labeling solutions. Boasting industry-leading image understanding accuracy, Google Cloud's Vision API provides millions of predefined categories as labels for quick classification. The API is accessed via a project on the Google Cloud Platform. A trial version for demo purposes is available as well.[1]

4.1 Data and Models

For our purposes, all experiments were carried out on a 12-core 2.2 GHz machine, with decision queries over the network. Scripts are available on the public domain[2], making use of `adversarial-robustness-toolbox`[3], a popular Python package for constructing adversarial examples. We focus our attack on three 400×400 images each with 3 color channels fetched from Google Images search, shown in Figure 3.

The selected images are associated with correct labels via the API. The three are initially assigned "Traffic Light", "Cat", and "Wolf" respectively. The classification model from the API is entirely

[1] See `https://cloud.google.com/vision`

[2] See `https://github.com/adbforlife/ml-adversary`

[3] See `https://github.com/Trusted-AI/adversarial-robustness-toolbox`

(a) A photo of a traffic light

(b) A photo of a cat

(c) A photo of a wolf

Figure 3: Original 400×400 images

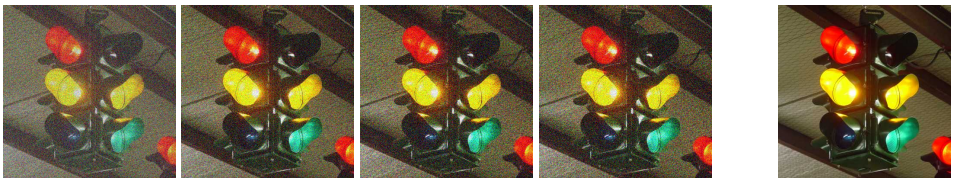


Figure 4: Trajectories with untargeted l_2 attack

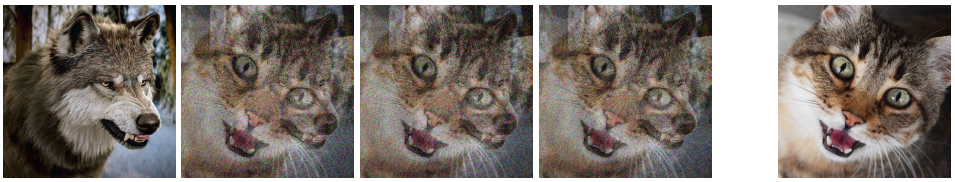


Figure 5: Trajectories with targeted l_2 attack

unknown. As far as we know, the labeling algorithm could range from Convolutional Neural Networks of arbitrary depth to simple decision trees (though unlikely).

For our example in targeted attack, we have three new artificial labels: "0" for "Cat", "1" for "Wolf", and "2" for the situation when neither labels are given by the API. For any image, if both "Cat" and "Wolf" labels are present, we select the one with the higher score to be our new label. On the other hand, we have "0" for "Traffic Light" but "1" for otherwise in the untargeted attack.

4.2 Results

We run the algorithms for untargeted attack and targeted attack separately, each for 50 steps.

For the untargeted attack, we use the traffic light image as the original image and use random noise as a starting point for the algorithm. We attempt to approach an adversarial example which does not have the label "0" with the smallest l_2 distance from the original image. As shown in Figure 4, the four images on the left are produced after 1, 2, 10, and 50 steps, respectively. They all have labels "1" in our new labeling scheme. This demonstrates that Google Cloud is unable to associate them with the "Traffic Light" label, despite their seeming resemblance to traffic lights.

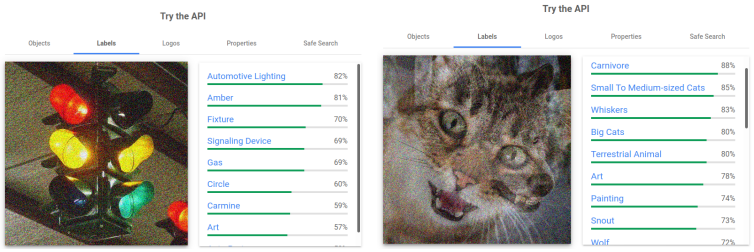
For the targeted attack, we use the cat as the original image but use the wolf as a starting point for the algorithm. We now aim to approach an adversarial example with label "1" in our new labeling scheme. The intermediate steps are shown in Figure 5. The performance of this attack is worse than that of the untargeted one, which meets our expectations.

The intermediate l_2 distances are shown in Table 1, and the API query results on Google's demo website are shown in Figure 6.

Check Your Understanding (5) *Recall the dimension of the images. Assuming element values clipped between 0 and 255 (represented as 8-bit unsigned integers), what is the maximum l_2 distance of two images?*

	$t = 1$	$t = 2$	$t = 10$	$t = 50$
untargeted	30425.33	28533.49	25186.13	23943.96
targeted	67966.53	37060.58	35738.23	34815.02

Table 1: l_2 distances during attack



(a) A query result lacking label "Traffic Light"

(b) A query result whose label "Wolf" is scored higher than label "Cat"

Figure 6: Attack results

5 Related Techniques

We discuss a group of well-known decision-based attacks to help reinforce our understanding of the current attack landscape.

Boundary Attack [4] works by first generating an input that is already adversarial, by applying a sufficiently large perturbation to the source image. The algorithm then seeks to incrementally minimize the perturbation while staying adversarial. This is achieved by finding a random orthogonal step and dynamically adjusting the step size towards the target along the adversarial decision boundary. However, this technique requires a large number of model queries relative to the HopSkipJumpAttack, which makes it impractical in real-world applications, and is only formulated specifically for the ℓ_2-distance. In comparison, the HopSkipJumpAttack can be generalized for other distance metrics, although we only explained the above algorithms assuming the ℓ_2-distance metric.

The Opt Attack [5] takes a different approach by translating the decision problem into a continuous optimization problem. The paper introduces the Randomized Gradient-Free (RGF) method to estimate the gradient, and shows that it takes fewer queries to generate adversarial examples than other state-of-the-art techniques.

The Limited Attack [6] gives the attacker only L queries, and assumes access to the top k labels predicted by a model. It utilizes natural Evolutionary Strategies [7] to perform gradient estimation in a query-efficient manner.

The HopSkipJumpAttack, however, has been able to achieve higher query efficiency over both the Opt Attack and the Limited Attack – it is able to produce adversarial examples with much smaller pertubations than both competing methods given a fixed number of model queries.

6 Conclusion

In this paper, we discuss the concept of adversarial attacks and the three main categories that comprise it. We focus on HopSkipJumpAttack, a particular decision-based attack using a novel algorithm to perform zeroth-order gradient estimation that is query-efficient and competitive in its performance with other state-of-the-art attacks. Please refer to [2] for minor generalizations as well as extended proofs of theorems. We also demonstrate the results of our own experiments on crafting adversarial examples using the HopSkipJumpAttack algorithm against Google Cloud Vision API, which performs to our expectations and credibly demonstrates the threat that this may pose to real-world machine learning systems. Finally, we contrast HopSkipJumpAttack against other decision-based attacks. We conclude optimistically here by remarking that the nascent but highly active subfield of adversarial attack research holds great promise in securing our digital future.

References

[1] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, page 43–58, New York, NY, USA, 2011. Association for Computing Machinery.

[2] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[3] Vision AI | Derive Image Insights via ML | Cloud Vision API. `https://cloud.google.com/vision`.

[4] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models, 2018.

[5] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack:an optimization-based approach, 2018.

[6] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information, 2018.

[7] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *J. Mach. Learn. Res.*, 15(1):949–980, January 2014.

Solutions for Exercises

(1)

Decision-based attacks are the most realistic form of attack to carry out in a real-world context since in most instances an attacker will only have access to a model's output and not its internal decision making process.

(2)

Untargeted attacks: $\max_{c \neq C(x)} \{F(x')_c\}$ returns the probability of the most likely class that is not $C(x)$, the actual correct label. $F(x')_{C(x)}$ is the probability of the right label. Therefore, if first term is larger than the second term, it means that the model will predict the wrong label; whereas if the first term is smaller than the second term, the model continues to predict the correct class.

Targeted attacks: $F(x')_{c^*}$ is the probability of predicting the targeted class that we desire. $\max_{c \neq c^*} \{F(x')_c\}$ is the highest probability of prediction of any of the other classes. So if the first term is larger than the second term, it means that our targeted attack succeeded since it predicts our desired class, while if the first term is less than the second term, the attack fails as some other class will be predicted.

As such, when $S(x') > 0$, our attack is unsuccessful, and when $S(x') < 0$ our attack is unsuccessful.

(3)

Figure 7 shows an example boundary (black line), with the red areas being where $S(x) > 0$ and the blue areas being $S(x) < 0$. After performing binary search, we see that we have x' near the boundary. We now want to estimate the gradient of x', which in the diagram points approximately upwards.

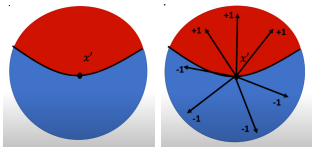


Figure 7: Visualisation of decision boundary and random vectors

Figure 7 shows the result when we add our random vectors u_b to the graph.

The vectors pointing upwards lies in the desired classification region and gives us a sign of 1, which helps to reinforce the gradients in that direction. Similarly, the vectors pointing downwards results in the wrong classification and gives a sign of -1, and so the gradients are reinforced in the opposite direction.

Vectors which are nearly tangential to x' in opposite directions would cancel each other out. Therefore our final estimated gradients will generally be in the right direction upwards.

(4)

For the first part, simply by definition we have

$$\begin{aligned} r(x_t, x) &= \cos \angle (x_t - x, \nabla S_x(x_t)) \\ &= \cos \angle ((0, 4, -2), (0, 2, -1)) \\ &= \cos 0 = 1. \end{aligned}$$

For the second part, we observe that v_t is the direction with which we update x_t to $\tilde{x}_t$. Since v_t is the same as the gradient, which is the same as direction of the difference between x_t and x, we know that $\tilde{x}_t = x_t + \xi_t v_t$, x_t, x are collinear, regardless of value of ξ_t. Now, since binary search preserves collinearity, we know $x_{t+1}, x_t, x, \tilde{x}_t$ are all collinear. Hence, the angle between the two vectors given must be either 0 or π, which gives that the absolute value of the cosine is 1.

(5)

We may compute norm as follows.

$$\max_{x,y} ||x - y||_2 = \sqrt{255^2 \times (400 \times 400 \times 3)} = 177362.00$$