
PopSkipJump: Decision-Based Attack for Probabilistic Classifiers

Carl-Johann Simon-Gabriel¹ Noman Ahmed Sheikh¹ Andreas Krause¹

Abstract

Most current classifiers are vulnerable to adversarial examples, small input perturbations that change the classification output. Many existing attack algorithms cover various settings, from white-box to black-box classifiers, but typically assume that the answers are *deterministic* and often fail when they are not. We therefore propose a new adversarial decision-based attack specifically designed for classifiers with *probabilistic* outputs. It is based on the HopSkipJump attack by Chen et al. (2019), a strong and query efficient decision-based attack originally designed for deterministic classifiers. Our *ProbabilisticHopSkipJump* attack adapts its amount of queries to maintain HopSkipJump’s original output quality across various noise levels, while converging to its query efficiency as the noise level decreases. We test our attack on various noise models, including state-of-the-art off-the-shelf randomized defenses, and show that they offer almost no extra robustness to decision-based attacks. Code is available at <https://github.com/cjsg/PopSkipJump>.

1. Introduction

Over the past decade, many state-of-the-art neural network classifiers turned out to be vulnerable to adversarial examples: small, targeted input perturbations that manipulate the classification output. The many existing attack algorithms to create adversarial inputs cover a wide range of settings: from white- to black-box algorithms (a.k.a. decision-based) over various gray-box and transfer-based settings, targeted and untargeted attacks for various kinds of data (images, text, speech, graphs); etc. Decision-based attacks are arguably among the most general attacks, because they try not to rely on any classifier specific information, except final decisions. Said differently, they can attack anything that can be queried often enough; in principle, even humans. Surprisingly however, despite this generality, they typically cannot deal with

noisy or probabilistic classification outputs – a quite natural and common setting in the real world. There could indeed be many reasons for this noise: occasional measurement errors, classifiers’ uncertainty about the answer, queries coming from different/changing classifiers, or intentional variations from a randomized adversarial defense. Yet, as shown by Table 1 and Appendix B.1, randomly changing the output label of only 1 out of 20 queries when applying SOTA decision-based attacks suffices to make them fail.

One way to deal with probabilistic outcomes would be to apply majority voting on repeated queries. In practice, however, some input regions may need more queries than others, either because they are noisier, or because they are more important for the success of the attack. So a naive implementation where every point gets queried equally often would require unnecessarily many queries, which is often not acceptable in real-world applications.

Contributions We therefore propose to adapt Chen et al. (2019)’s *HopSkipJump* (HSJ) algorithm, a query efficient, decision-based, iterative attack for *deterministic* classifiers, to make it work with noisy, probabilistic outputs. We take a model-based, Bayesian approach that, at every iteration, evaluates the local noise level (or probabilities) and uses it to optimally adapt the number of queries to match HSJ’s original performance. The result is a *probabilistic* version of *HopSkipJump*, *PopSkipJump* (PSJ), that

1. outperforms majority voting on repeated queries;
2. efficiently adapts its amount of queries to maintain HSJ’s original output quality at every iteration over increasing noise levels;
3. gracefully converges to HSJ’s initial query efficiency when answers become increasingly deterministic;
4. works with various noise models.

In particular, we test our attack on several recent state-of-the-art off-the-shelf randomized defenses, which all rely on some form of deterministic base model. PSJ achieves the same performance as HSJ on the original base models, showing that these defense strategies offer no extra robustness to decision-based attacks. Finally, most parameters of HSJ have a direct counterpart in PSJ. So by optimizing the parameters of HSJ or PSJ in the deterministic setting we get automatic improvements in the various stochastic settings.

¹ETH Zürich. Correspondence to: CJSG <cjsg@ethz.ch>.

Related Literature. White-box attacks are the one extreme case where the attacker has *full knowledge* of the classifiers’ architecture and weights. They typically use gradient information to find directions of high sensitivity to changes of the input, e.g., FGSM (Goodfellow et al., 2015), PGD (Madry et al., 2018), Carlini&Wagner attack (Carlini & Wagner, 2017) and DeepFool (Moosavi-Dezfooli et al., 2016). At the other extreme, decision-based attacks only assume access to the classifiers’ *decisions*, i.e., the top-1 class assignments. State-of-the-art decision-based attacks include the *boundary attack* by Brendel et al. (2018), *HopSkipJump* by Chen et al. (2019) and *qFool* by Liu et al. (2019). In between these two extremes, many intermediate, gray-box settings have been considered in the literature. When the architecture is at least roughly known but not the weights, one can use transfer attacks, which compute adversarial examples on a similar, substitute network in the hope that they will also fool (‘transfer to’) the targeted net. Ilyas et al. (2018) consider cases where the attacker knows the top- k output logits/probabilities, or the top- k output ranks. However, all these attacks were originally designed for deterministic classifiers, and typically break when answers are just slightly noisy. This has led several authors to propose alleged “defenses” using one or another form of randomization, such as neural dropout, adversarial smoothing, random cropping and resizing, etc. (details in Section 4). Shortly later however, Athalye et al. (2018) managed to circumvent many of these randomized defenses *in the white-box setting*, by adapting existing white-box attacks to cope with noise. Cardelli et al. (2019b;a) studied *white-box* attacks and certifications for Gaussian process classifiers and Bayesian nets. To date however, and to the best of our knowledge, there has been no such attempt in the decision-based setting. In particular, we do not know of any SOTA *decision-based* attack that can handle noisy or randomized outputs. Our paper closes this gap: we provide a decision-based attack that achieves the same performance on randomized defenses than SOTA decision-based attacks on the undefended nets.

Notations Vectors are bold italic ($\mathbf{x}, \boldsymbol{\delta}, \mathbf{g}, \dots$), their coordinates and 1d variables are non-bold italic (x, δ, g, \dots). Random vectors are bold and upright ($\boldsymbol{\delta}, \mathbf{g}, \boldsymbol{\phi}, \dots$), their coordinates and 1d random variables are non-bold upright (δ, g, ϕ, \dots) When it cannot be avoided, we will also use the index-notation (e.g., δ_i) to designate the (i -th) coordinate of vector ($\boldsymbol{\delta}$). $\text{Ber}(p)$ denotes the Bernoulli distribution *with values in* $\{-1, 1\}$, returning 1 with probability p .

2. Probabilistic Classifiers

Definition. We define a probabilistic classifier as a random function (or stochastic process) ϕ from a set of inputs \mathbb{X} to a set of K classes $\mathbb{K} := \{1, \dots, K\}$. Said differently, for

FLIP	PSJ	HSJ	HSJ x3
$\nu = 0\%$	0.006 (1x)	0.006 (0.90x)	0.005 (2.70x)
$\nu = 5\%$	0.006 (1x)	0.022 (0.74x)	0.007 (2.35x)
$\nu = 10\%$	0.006 (1x)	0.036 (0.73x)	0.013 (2.27x)

Table 1: Median size of adversarial perturbation (“border-distance”, see Section 4) and relative number of model calls (in brackets) for different attacks (columns) and different noise levels (rows) on MNIST. All attacks perform similarly on deterministic classifiers. Randomly changing ν percent of outputs however suffices to break HopSkipJump (HSJ), even with majority voting on 3 repeated queries, whereas PopSkipJump (PSJ, our method) remains unaffected.

any $\mathbf{x} \in \mathbb{X}$, $\phi(\mathbf{x})$ is a random variable taking values in \mathbb{K} .¹ Repeated queries of ϕ at \mathbf{x} yield i.i.d. copies of $\phi(\mathbf{x})$.

Remark 1. For every probabilistic classifier ϕ , there exists a function of logits $\mathbf{x} \mapsto \boldsymbol{\varphi}(\mathbf{x}) := (\varphi(\mathbf{x})_1, \dots, \varphi(\mathbf{x})_K) \in (\mathbb{R} \cup \{\pm\infty\})^K$ which (after a softmax) defines the distribution of $\phi(\mathbf{x})$ at every point \mathbf{x} . Conversely, every such logit function defines a unique probabilistic classifier. So a probabilistic classifier is nothing but an arbitrary function φ with values in $(\mathbb{R} \cup \{\pm\infty\})^K$ for which querying at \mathbf{x} means returning a random draw $\phi(\mathbf{x})$ from the distribution defined by the logits $\boldsymbol{\varphi}(\mathbf{x})$. In this paper, φ is a priori unknown. But it can of course be recovered with arbitrary precision at any point \mathbf{x} by repeatedly querying ϕ at \mathbf{x} .

Examples. Any deterministic classifier can be turned into a probabilistic classifier with noise level ν by swapping the original output with probability ν for another label, chosen uniformly at random among the remaining classes. This could, e.g., model a noisy communication channel between the classifier and the attacker, noisy human answers, or answers that get drawn at random from a set of different classifiers. Sometimes, noise is also injected intentionally into the classifier as a form of defense, e.g., by adding Gaussian noise to the inputs as in adversarial smoothing (Cohen et al., 2019), or via dropout of neural weights (Cardelli et al., 2019a; Feinman et al., 2017), random cropping, resizing and/or compression of the inputs (Guo et al., 2018). All these defenses yield random outputs, i.e., probabilistic classifiers. Finally, any neural network with logit outputs $\boldsymbol{\varphi}(\mathbf{x}) \in \mathbb{R}^K$ can be turned into a probabilistic classifier by sampling from its logits instead of returning the usual $\arg \max_k \varphi(\mathbf{x})_k$. Even though perhaps not common in practice, such randomized selection is indeed used, e.g., in softmax-exploration in RL. Moreover, as Remark 1 shows, sampling from logits can in principle model *all* the previous examples, if the neural network is given sufficient capacity to model arbitrary logit functions φ , which is why we also

¹Note however that, starting from Section 3, we will convene that ϕ takes its values in $\{-1, 1\}$.

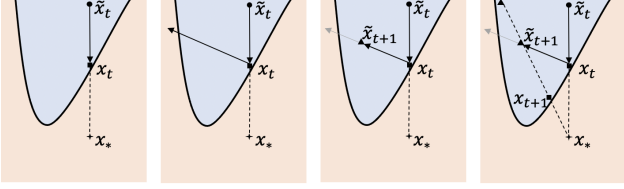


Figure 1: Original HopSkipJump algorithm for deterministic classifiers (Chen et al., 2019). Illustrations correspond to steps (a), (b), (c) and (a) from the text.

consider this setting in our experiments (Section 4).

Adversarial risk and accuracy. For a given norm $\|\cdot\|$, loss \mathcal{L} and perturbation size $\eta > 0$, we generalize the notion of ϵ -adversarial (or robust) risk AR (Madry et al., 2018) from the deterministic to the probabilistic setting as

$$\text{AR}(\|\cdot\|, \mathcal{L}, \eta, \phi) := \mathbb{E}_{(\mathbf{x}, c)} \mathbb{E}_{\phi(\mathbf{x})} \left[\max_{\|\delta\| \leq \eta} \mathcal{L}(\phi(\mathbf{x} + \delta), c) \right].$$

For deterministic classifiers, the expectation is taken only over the distribution of the labeled datapoints $(\mathbf{x}, c) \in \mathbb{X} \times \mathbb{K}$. For probabilistic classifiers, it is also taken over the randomness of the output $\phi(\mathbf{x})$ (at fixed input \mathbf{x}). We define adversarial accuracy as $1 - \text{AR}(\mathcal{L}_{0/1})$ where $\mathcal{L}_{0/1}$ denotes the 0-1 loss. In this work, $\|\cdot\|$ will always be the ℓ_2 -norm $\|\cdot\|_2$.

3. The PopSkipJump Algorithm

From now on, let us fix the attacked image \mathbf{x}_* with true label c , and assume that the classifier ϕ returns values in $\{-1, 1\}$ with 1 meaning class c and -1 not class c .

3.1. HopSkipJump algorithm for deterministic outputs

Given an image \mathbf{x}_* with label c and a neural network classifier φ which correctly assigns label c to \mathbf{x}_* , i.e., $c = \arg \max_{k \in \mathbb{K}} \varphi(\mathbf{x}_*)_k$, and let $b(\mathbf{x}) := \varphi(\mathbf{x})_c - \max_{k \in \mathbb{K} \setminus \{c\}} \varphi(\mathbf{x})_k$. Then $b(\mathbf{x}) > 0$ if φ assigns class c to \mathbf{x} , and $b(\mathbf{x}) < 0$ if it does not. Consequently, we call *boundary* the set of points \mathbf{x} such that $b(\mathbf{x}) = 0$.

Similar to the boundary attack (Brendel et al., 2018) or qFool (Liu et al., 2019), HopSkipJump (HSJ) is a decision based attack that gradually improves an adversarial proposal \mathbf{x}_t over iterations t by moving it along the decision boundary to get closer to the attacked image \mathbf{x}_* . More precisely, each iteration consists of three steps (see Fig. 1):

- (a) *binary search* on the line between an adversarial image $\tilde{\mathbf{x}}_t$ and the target \mathbf{x}_* , which yields an adversarial point \mathbf{x}_t near the classification border.
- (b) *gradient estimation*, which estimates $\mathbf{g}(\mathbf{x}_t) := \nabla_{\mathbf{x}} b(\mathbf{x}_t) / \|\nabla_{\mathbf{x}} b(\mathbf{x}_t)\|_2$, the normal vector to the

boundary at \mathbf{x}_t , as

$$\hat{\mathbf{g}}(\mathbf{x}_t) := \frac{\mathbf{u}}{\|\mathbf{u}\|_2} \text{ with } \mathbf{u} := \sum_{i=1}^{n_t} \phi(\mathbf{x}_t + \delta^{(i)}) \delta^{(i)} \quad (1)$$

where the $\delta^{(i)}$ are uniform i.i.d. samples from a centered sphere with radius δ_t . We will often simply refer to \mathbf{g} as “the gradient” and to $\hat{\mathbf{g}}$ as “the gradient estimate”.

- (c) *gradient step*, a step of size ξ_t in the direction of the gradient estimate $\hat{\mathbf{g}}(\mathbf{x}_t)$, yielding $\tilde{\mathbf{x}}_{t+1} := \mathbf{x}_t + \xi_t \hat{\mathbf{g}}(\mathbf{x}_t)$.

Chen et al. (2019) provide various convergence results to justify their approach and fix the size of the main parameters, which are (a) the minimal bin size $\theta_t^{\text{det}} = d^{-3/2} \|\mathbf{x}_t - \mathbf{x}_*\|_2$ for stopping the binary search; (b) the sample size $n_t^{\text{det}} = n_0^{\text{det}} \sqrt{t}$ and sampling radius $\delta_t^{\text{det}} = \theta_t^{\text{det}} \sqrt{d}$ used to estimate the gradient; (c) the step size $\xi_t^{\text{det}} = \|\mathbf{x}_t - \mathbf{x}_*\|_2 / \sqrt{t}$.

3.2. From HopSkipJump to PopSkipJump

While HSJ is very effective on deterministic classifiers, small noise on the answers suffices to break the attack: see Table 1. This is because, for one reason, binary search is very noise sensitive: one wrong answer of the classifier during the binary search and \mathbf{x}_t can end up being non-adversarial and/or far from the classification border. For another, even if binary search worked, the gradient estimate $\hat{\mathbf{g}}$ needs more sample points $\delta^{(i)}$ to reach the same average performance with noise than without.

Overview of PopSkipJump. To solve these issues, our Probabilistic HopSkipJump attack, *PopSkipJump*, replaces the binary search procedure by (*sequential*) *Bayesian experimental design*, *NoisyBinSearch*, that not only yields a point \mathbf{x}_t , but also evaluates the noise level around that point. We then use this evaluation of the noise to compute analytically (eq. 4) how many sample points n_t we need to get a gradient estimate $\hat{\mathbf{g}}(\mathbf{x}_t)$ with the same expected performance – as measured by $\mathbb{E}[\cos(\hat{\mathbf{g}}, \mathbf{g})]$ using Eq. 3 – than that of the same estimator with n_t^{det} points on a deterministic classifiers. Interestingly, when the noise level decreases, our noisy binary search procedure recovers usual binary search, and n_t decreases to n_t^{det} . PopSkipJump can therefore automatically adapt to the noise level and recover the original HopSkipJump algorithm if the classifier is deterministic. We now explain in more detail the two parts of our algorithm, noisy binary search and sample size estimation.

3.3. Noisy bin-search via Bayesian experimental design

Sigmoid assumption. Our noisy binary search procedure assumes that the probability $p_c(\mathbf{x})$ of the class c of \mathbf{x}_* (the

Algorithm 1 PopSkipJump

Input: attacked point \mathbf{x}_* ; starting point $\tilde{\mathbf{x}}_0$ from adversarial class; probabilistic classifier ϕ ; input dim d ; HSJ parameters: sampling sizes n_t^{det} , sampling radii δ_t^{det} , min bin-sizes θ_t^{det} and gradient step sizes ξ_t^{det} .

for $t = 1$ **to end do**

Compute expected cosine C_t^{det} of the HSJ gradient estimate when classifier is deterministic
 $C_t^{\text{det}} \leftarrow f(s = \infty, \epsilon = 0, n = n_t^{\text{det}}, \Delta = \theta_t^{\text{det}})$ where $f = \text{RHS of Eq. 3}$

Do noisy bin-search btw. $\tilde{\mathbf{x}}_t$ and \mathbf{x}_* with target $\text{cos} = C_t^{\text{det}}$ used in the stopping criterion
 posterior(z, s, ϵ) \leftarrow NoisyBinSearch($\tilde{\mathbf{x}}_t, \mathbf{x}_*, C_t^{\text{det}}$) # (z, s, ϵ) = sigmoid parameters
 ($\hat{z}, \hat{s}, \hat{\epsilon}$) \leftarrow $\mathbb{E}_{\text{posterior}}[(z, s, \epsilon)]$ # Compute mean a posteriori of sigmoid parameters
 $\mathbf{x}_t \leftarrow \hat{z}\tilde{\mathbf{x}}_t + (1 - \hat{z})\mathbf{x}_*$ # Move to sigmoid center (“border”)

Compute query size n_t for grad estimate on prob classifier to match HSJ’s performance on det classifier
 $n_t = \mathbb{E}_{z \sim \text{post}}[f^{-1}(\Delta = z - \hat{z}, s = \hat{s}, \epsilon = \hat{\epsilon}, C = C_t^{\text{det}})]$ where $f^{-1} = \text{RHS Eq. 4}$

$\hat{\mathbf{g}}(\mathbf{x}_t) = \text{RHS of Eq. 1}$ # Estimate gradient \mathbf{g} at point \mathbf{x}_t
 $\tilde{\mathbf{x}}_{t+1} \leftarrow \mathbf{x}_t + \xi_t \hat{\mathbf{g}}(\mathbf{x}_t)$ # Make step in the estimated gradient direction
 $\tilde{\mathbf{x}}_{t+1} \leftarrow \mathbf{x}_* + 1.5(\tilde{\mathbf{x}}_{t+1} - \mathbf{x}_*)$ # Enlarge obtained interval $[\mathbf{x}_{t+1}, \mathbf{x}_*]$ to improve next bin-search

end for and return $\mathbf{x}_t, \hat{z}, \hat{s}, \hat{\epsilon}$

Algorithm 2 NoisyBinSearch

Input: attacked point \mathbf{x}_* ; $\tilde{\mathbf{x}}_t$ from adv. class; probabilistic classifier ϕ with query model $\phi(x) \sim p_c(x)$ from Eq. 2; target $\text{cos} C_t^{\text{det}}$; prior $p(z, s, \epsilon)$

for $i = 1$ **to** ∞ **do**

Compute acquisition fct for all $x \in [\tilde{\mathbf{x}}_t, \mathbf{x}_*]$
 $a(x) = I(\phi(x), z, s, \epsilon)$ with $p(z, s, \epsilon) \& \phi(x) \sim p_c(x)$

Sample at argmax of acquisition function
 $\phi(\hat{x}) \sim p_c(\hat{x})$ where $\hat{x} = \arg \max_x a(x)$

Update prior and estimates with posterior
 $p(z, s, \epsilon) \leftarrow p(z, s, \epsilon | \phi(\hat{x}))$
 $\hat{z}, \hat{s}, \hat{\epsilon} \leftarrow \mathbb{E}_{p(z, s, \epsilon)}[p(z, s, \epsilon)]$

Use current posterior to compute query size for the next grad estimate to reach an expected $\text{cos} = C_t^{\text{det}}$
 $n_i = \mathbb{E}_{p(z, s, \epsilon)}[f^{-1}(z - \hat{z}, s, \epsilon)], f^{-1} := \text{RHS of Eq. 4}$

Stop if k bin-search queries spare $\leq k$ grad queries
if $|n_i - n_{i-k}| \leq k$: **break**

end for

Output: posterior $p(z, s, \epsilon)$, posterior means $(\hat{z}, \hat{s}, \hat{\epsilon})$.

attacked image) has a sigmoidal shape along the line segment $[\mathbf{x}_t, \mathbf{x}_*]$. More precisely, for $\mathbf{x} = (1 - x)\mathbf{x}_t + x\mathbf{x}_*$ with $x \in [0, 1]$, we assume that

$$p_c(\mathbf{x}) \equiv p_c(x) = \epsilon + (1 - 2\epsilon) \sigma(s(x - z)) \quad (2)$$

where ϵ models an overall noise level, and where $\sigma(x) := 1/(1 + e^{-4x})$ is the usual sigmoid, rescaled to get a slope = 1 in its center z when the inverse scale parameter s is equal to 1. This assumption is particularly well-suited for the examples discussed in Section 2, such as a probabilistic classifier whose answers are sampled from a final logit layer. This assumption is also confirmed by Appendix B.3, where we plot the output probabilities along the bin-search line $[\mathbf{x}_t, \mathbf{x}_*]$ at various iterations t of an attack on two sample images \mathbf{x}_* . Note that when $s = \infty$, we recover the deterministic case,

with or without noise on top of the deterministic output, depending on ϵ .

Bayesian experimental design. The noisy binary search procedure follows the standard paradigm of Bayesian experimental design. We put a (joint) prior $p^{(k)}$ on ϵ, z and s , query the classifier at a point $x^{(k)} \in [0, 1]$ to get a random label $\phi(x^{(k)})$, update our prior with the posterior distribution of (ϵ, z, s) and iterate over these steps for $k = 0, 1, \dots$ until convergence. The stopping criterion will be discussed in Section 3.4. We choose $x^{(k)}$ by maximizing a so-called *acquisition* function $\text{acq}(x|p^{(k)})$, which evaluates how “informative” it would be to query the classifier at point x given our current prior $p^{(k)}$ on its parameters. We tested two standard acquisition functions: (i) mutual information $I(\phi(x) || s, z, \epsilon)$ between the random answer $\phi(x)$ to a query at x and the parameters s, z, ϵ ; (ii) an expected improvement approach, where we choose x to minimize the expected sample size $\mathbb{E}_{s, z, \epsilon}[n_t | \phi(x)]$ that will be required for the next gradient estimation and where n_t is computed using Eq. 4 below. Mutual information worked best, which is why we keep it as default. We can then use the final prior/posterior to get an estimate $(\hat{z}, \hat{s}, \hat{\epsilon})$ of the true parameters (z, s, ϵ) , for example with the maximum or the mean a posteriori. We compute all involved quantities by discretizing the parameter space of (x, z, s, ϵ) and start with uniform priors. See details in Section 4.

3.4. Sample size for the gradient estimate

From sphere to normal distribution. Although the original gradient estimate in the HopSkipJump attack samples the perturbations $\delta^{(i)}$ of the near-boundary point \mathbf{x}_t on a sphere, we instead sample them from a normal distribution

$\mathcal{N}(0, \beta_t \mathbf{I}_d)$ with diagonal standard deviation $\beta_t := \delta_t/d$. This will simplify our analytical derivations and makes no difference in practice, since, in high dimensions d , this normal distribution is almost a uniform over the sphere with radius δ_t (in particular, $\|\delta^{(i)}\|_2 \approx \delta_t$).

Sample size n_t . We use the previous hypothesis to approximate the expected cosine $\mathbb{E}[\cos(\hat{\mathbf{g}}, \mathbf{g})]$ between the gradient $\mathbf{g}(\mathbf{x})$ at point \mathbf{x} and its estimate $\hat{\mathbf{g}}(\mathbf{x})$ for a sample size n as follows (justification in Appendix A.0.1).

$$\mathbb{E}[\cos(\hat{\mathbf{g}}, \mathbf{g})] \approx \frac{1}{\sqrt{1 + \frac{d-1}{n\alpha^2}}} \quad \text{where} \quad \begin{cases} \alpha(\Delta, s, \beta, \epsilon) := \mathbb{E}_{\delta, y_\epsilon} [y_\epsilon (s(\Delta + \beta\delta))\delta] \\ \delta \sim \mathcal{N}(0, 1), y_\epsilon(x) \sim \text{Ber}(\epsilon + (1-2\epsilon)\sigma(x)), \end{cases} \quad (3)$$

where we defined the displacement $\Delta := x - z$ between the gradient sampling center x and the sigmoid center z and where Ber denotes the Bernoulli distribution *with values in* $\{-1, 1\}$. This equation is easily inverted to get the sample size n as a function of the expected cosine C :

$$n \approx \frac{C^2}{\alpha^2(\Delta, s, \beta, \epsilon)} \frac{d-1}{1-C^2}, \quad C := \mathbb{E}[\cos(\hat{\mathbf{g}}, \mathbf{g})]. \quad (4)$$

Finally, the following result shows how to compute α when replacing the usual sigmoid by its close approximation, a clipped linear function (proof in Appendix A.0.2).

Proposition 2. *Assume that, in Eq. 3, σ is the clipped linear function $\sigma(x) = \text{clip}(x + 1/2, 0, 1)$. Then*

$$\alpha(\Delta, s, \beta, \epsilon) = \begin{cases} (1-2\epsilon)s\beta \cdot \\ \left(\text{erf}\left(\frac{\Delta+1/2s}{\beta\sqrt{2}}\right) - \text{erf}\left(\frac{\Delta-1/2s}{\beta\sqrt{2}}\right) \right) \end{cases} \quad (5)$$

$$\alpha(\Delta, \infty, \beta, \epsilon) = (1-2\epsilon)\sqrt{\frac{2}{\pi}} e^{-(\beta\Delta)^2/2}. \quad (6)$$

We now explain how to use these four formulae, together with the estimates $(\hat{z}, \hat{s}, \hat{\epsilon})$ from the binary search procedure, to evaluate the sample size n_t that we need to get the same expected cosine value than with n_t^{det} points sampled from a deterministic classifier. First we set $s = \infty, \epsilon = 0$, and $n = n_t^{\text{det}}$ in Eq. 3 and compute the expected cosine C_t^{det} on a deterministic classifier with n_t^{det} sample points; then we apply Eq. 4 with our estimates $(\hat{z}, \hat{s}, \hat{\epsilon})$ and use the obtained value n_t . (Alternatively, instead of using the point estimate $(\hat{z}, \hat{s}, \hat{\epsilon})$, we could also use Eq. 4 to compute the $\mathbb{E}_{z,s,\epsilon}[n_t]$ using the full posterior over (t, s, ϵ) .)

Stopping criterion for bin-search. We leverage Eq. 4 to design a stopping criterion for the (noisy) bin-search procedure that minimizes the overall amount of queries in PSJ. We use it to stop the binary search when one additional query there spares, on average, less than one query in the gradient

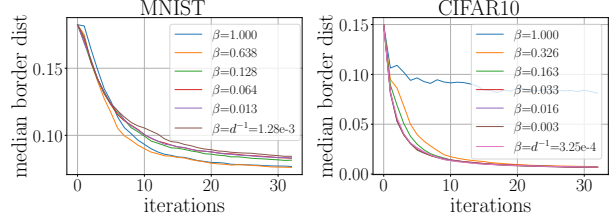


Figure 2: HSJ’s performance on deterministic deep net classifiers is largely independent of the effective sampling radius $\delta_t = \beta\|\tilde{\mathbf{x}}_t - \mathbf{x}_*\|_2$ of the gradient estimator. So we can safely increase β (hence δ_t) by several orders of magnitude, which greatly enhance PSJ’s query efficiency in the noisy setting (see Section 3.5) without affecting PSJ/HSJ’s output quality in the deterministic setting. Here we let β range from HSJ’s original choice $1/d$ to 1, and adjusted the minimal bin-size θ_t to preserve the ratio $\delta_t/\theta_t = \sqrt{d}$, as in Chen et al. (2019), IV.C.b. and eq. 15.

estimation procedure. Concretely, every k queries (typically, $k = 10$), we use our current bin-search estimate of (z, s, ϵ) (or the full posterior) to compute n (or $\mathbb{E}_{z,s,\epsilon \sim \text{posterior}}[n]$) using Eq. 4 and stop the binary search when the absolute difference $|n_{\text{new}} - n_{\text{old}}|$ between the new and old result is $\leq k$. The idea is that, the better we estimate the center z of the sigmoid, the closer x (center of $\hat{\mathbf{g}}$) will be to z . This in turn will reduce the number of queries required for $\hat{\mathbf{g}}$ to reach a certain expected cosine. (To see that, notice for example that Eq. 4 decreases with $|\Delta|$.) Since a query tends to yield more information about the position of z at the beginning of the bin-search procedure than later on, $|n_{\text{new}} - n_{\text{old}}|$ tends to decrease with the amount of bin-search queries. The order of magnitude of $|\Delta|$ when meeting the stopping criterion depends on the shape parameter s and noise level ϵ of the underlying sigmoid. For a deterministic classifier ($s = \infty, \epsilon = 0$), it must be at least of the order of β , the standard deviation of the samples $\delta^{(i)}$ in $\hat{\mathbf{g}}$ (see eq. 1): otherwise, all points $x + \delta^{(i)}$ would belong to the same class and yield no information about the border. (See also IV.C.a. in Chen et al. 2019.) But if $\beta \ll 1/s$, the characteristic size of the linear part of the sigmoid, then $|\Delta| \geq \beta$ is acceptable, as long as it is $\leq 1/s$. Our stopping criterion provides a natural and systematic way to trade off these considerations.

3.5. PopSkipJump versus HopSkipJump

Here we discuss additional small differences between HSJ and PSJ, besides the obvious ones that we already mentioned – binary search, its stopping criterion, and the sample size for the gradient estimate.

Gradients: variance reduction and size of δ_t . The authors of HSJ propose a procedure to slightly reduce the variance of the gradient estimate (Sec. III.C.c), which we do not use here. Moreover, they use $\delta_t = \|\tilde{\mathbf{x}}_t - \mathbf{x}_*\|_2/d$,

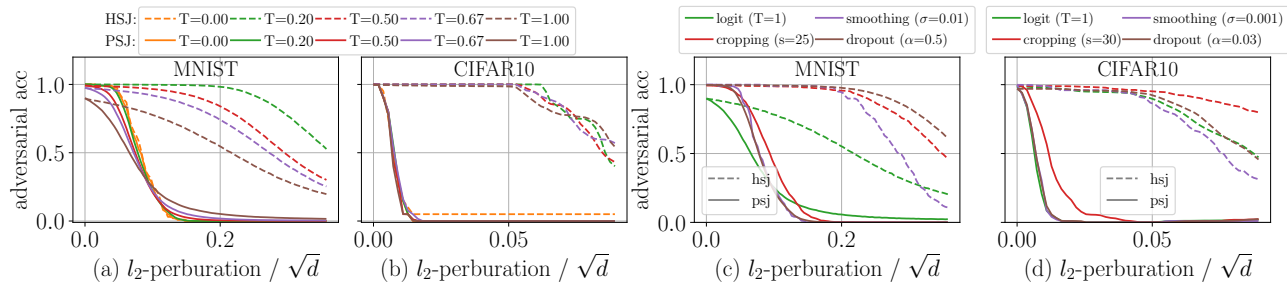


Figure 3: Adversarial accuracy versus attack size η for PSJ and HSJ for a fixed noise model (logit-sampling) and various noise levels (temperatures T) in Figs. (a) & (b); and for various noise models and fixed, high noise levels in Figs. (c) & (d). The curves obtained with PSJ are well below their HSJ counterparts in all noisy settings and both curves coincide in the deterministic case (Figs. a & b, $T=0$). This illustrates the clear superiority of PSJ over HSJ. Note that, even though all classifiers use a same underlying base classifier, their PSJ curves do not coincide (even when $\eta = 0$, i.e., for usual accuracy). This confirms that adversarial accuracy is ill-suited for comparisons between different noise levels and that the median border distance should be preferred, as in Figs. 4 and 5. See paragraph “Adversarial accuracy” and Remark 4.

whereas we use $\delta_t = \sqrt{d}\|\tilde{x}_t - x_*\|_2/100$. The reason is that, whenever $s < \infty$, smaller δ_t yield noisier answers which increases the queries needed both in the gradient estimation and in the bin-search. Given that the logits of deep nets typically have shape parameters $s \approx 1$, HSJ’s original choice would require prohibitively many samples. To reduce noise, the larger the radius δ_t , the better. In practice however, the size of δ_t is limited by the curvature of the border and by the validity range of the sigmoidal assumption (Eq. 2). To trade of these considerations, we evaluated the empirical performance of HSJ (on deterministic classifiers) with various choices of δ_t and chose one of the largest for which results did not differ significantly from the original ones. See Fig. 2. A more theoretically grounded approach that would evaluate the curvature is left for future work.

No geometric progression on ξ_t . For HSJ, it is crucial that \tilde{x}_t be on the adversarial side of the border. Therefore, it always tests whether the point $\tilde{x}_t := x_{t-1} + \xi_t \hat{g}(x_{t-1})$ is indeed adversarial. If not, it divides ξ_t by 2 and tests again. Since by design x_{t-1} is adversarial, this “geometric progression” procedure is bound to converge. In the probabilistic case, however, testing if a point is adversarial can be expensive, and is not needed since, on the one hand, the noisy bin-search procedure can estimate the sigmoid’s parameters even if z is outside of $[\tilde{x}_t, x_*]$; and on the other, we are less interested in the point \tilde{x}_t and the point value $p_c(\tilde{x}_t)$ than in the global *direction* from x_* to \tilde{x}_t . We therefore use ξ_t as is, without geometric progression.

Enlarging bin-search interval $[\tilde{x}_t, x_*]$. It is easier for the noisy bin-search procedure to estimate the sigmoid parameters if it can sample from both sides of the sigmoid center z . In practice however, we noticed that after a few iterations t , the point \tilde{x}_t tends to be very close to z . We therefore increase the size of the sampling interval, from $[\tilde{x}_t, x_*]$ to

$[x_* + 1.5(\tilde{x}_t - x_*), x_*]$, which performed much better.

4. Experiments

The goal of our experiments is to verify points 1. to 4. from the introduction. That is, we want to show that, contrary to the existing decision-based attacks, the performance of PSJ is largely independent of the strength and type of randomness considered, i.e., of both the *noise level* and the *noise model*. At every iteration, PSJ adjusts its amount of queries to keep HSJ’s original output quality, and is almost as query efficient as HSJ on near-deterministic classifiers. To show all this, we apply PSJ (and other attacks) to a deterministic *base classifier* whose outputs we randomize by injecting an adjustable amount of randomness. We test various randomization methods, i.e., noise models, described below, including several randomized defenses proposed at the ICLR’18 and ICML’19 conferences. Figures 4 and 5 summarize our main results.

Remark 3. Although we do compare PSJ to SOTA decision-based attacks, with or without repeated queries, *we do not compare PSJ to any decision-based attack specifically designed for probabilistic classifiers because, to the best of our knowledge, there is not any.*² There are however some *white-box* attacks (e.g., Athalye et al., 2018; Cardelli et al., 2019b;a) that can deal with some specific noise models considered in this paper (see below).

Noise models and randomized defenses. For a given deterministic classifier φ , we consider the following randomization schemes.

²The decision-based attack by Ilyas et al. (2018) for deterministic classifiers may still work to some extent with randomized outputs, but it is less effective than HSJ on deterministic classifiers (Chen et al., 2019). Since we will show that, despite the noise, PSJ stays on par with HSJ, there is no need for further comparisons.

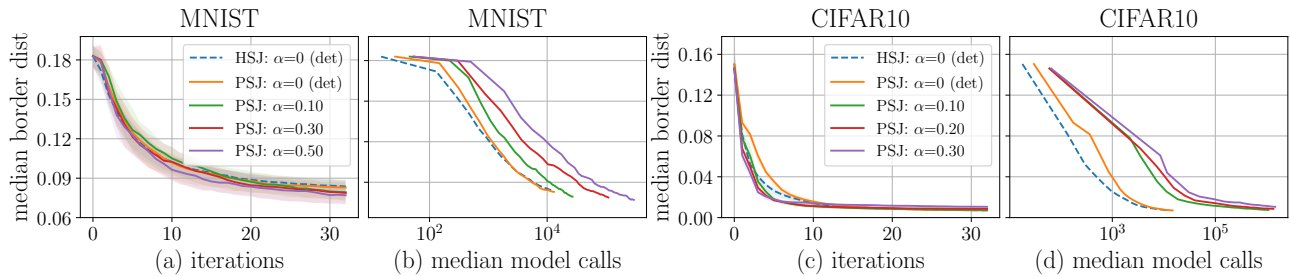


Figure 4: PopSkipJump’s performance (lower is better) for a fixed noise model (dropout) and various noise levels (dropout rate α). Performance is shown as a function of the number of algorithm iterations (a & c) and model queries (b & d). Shaded areas depict the 40th to 60th percentiles. Plots (a) & (c) illustrate property 2.: the *per iteration* performance of PSJ is largely independent of the noise level (here, the dropout rate) and is on par with the performance achieved by HSJ on the deterministic base classifier. Plots (b) & (d) illustrate property 1.: when the noise level (dropout rate) decreases and the classifier becomes increasingly deterministic, the PSJ curves converge to the limiting HSJ curve, i.e., the *per query* performance of PSJ converges to that of HSJ. See Fig. 9 in appendix for similar curves, but with other noise models.

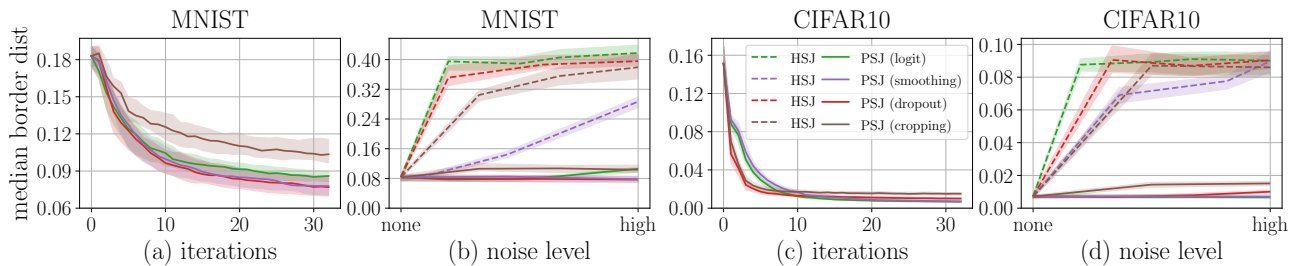


Figure 5: PopSkipJump’s performance (lower is better) for various noise models. Plots (a) & (c): Performance as a function of the number of algorithm iterations, when using, for each noise model, the highest noise levels considered in Fig. 4 (see also Fig. 9 in appendix). All curves are very similar, showing that PSJ is largely invariant to the specific type of randomness used, even at high noise levels. Plots (b) & (d): Performance after 32 algorithm iterations (right-most part of curves in a & c) of PSJ and HSJ as a function of the noise level. While small noise levels suffice to break HSJ (large border-distances at end of attack), PSJ’s performance stays almost constant across all noise levels and noise models.

- *logit sampling*: divide the output logits $\varphi(\mathbf{x})$ by a temperature parameter T to get $\varphi_T(\mathbf{x}) := \varphi(\mathbf{x})/T$ and sample from the new logits $\varphi_T(\mathbf{x})$. By changing T we can smoothly interpolate between the deterministic classifier ($T \rightarrow 0$) and sampling from the original logits ($T = 1$).
- *dropout*: apply dropout with a uniform dropout rate $\alpha \in [0, 1]$ (Srivastava et al., 2014). Taking $\alpha = 0$ yields the deterministic base classifier; increasing α increases the randomness. Dropout and its variants have been proposed as adversarial defenses, e.g., in Cardelli et al. (2019a); Feinman et al. (2017). Note that a network with dropout can be interpreted as a form of Bayesian neural net (Gal & Ghahramani, 2016). As such, sampling from it can be understood as sampling from an ensemble of nets.
- *adversarial smoothing*: add centered Gaussian noise with standard deviation σ to every input before passing it to the classifier. Taking $\sigma = 0$ yields the original base classifier. Cohen et al. (2019) proposed majority voting over several such queries as an off-the-shelf adversarial robustification.
- *random cropping & resizing*: randomly crop and resize every input image before passing it to the classifier. Chang-

ing the cropping size allows to interpolate between the deterministic setting (no cropping) and more noise. This method and a variant were proposed by Guo et al. (2018) and Xie et al. (2018) as adversarial defenses.

We ran all experiments on the MNIST (LeCun et al., 1998) and CIFAR10 (Krizhevsky, 2009) image datasets. Since, at high noise levels, the attack may need a million queries, it could take a minute per attack on a GeForce GTX 1080 for MNIST and a few minutes for CIFAR10 (larger net; see Appendix D for a time and complexity analysis and acceleration tricks.) We therefore restricted all experiments to a same random subset of 500 images of the MNIST and CIFAR10 test sets respectively, where we kept only images that were labeled correctly with probability $\geq .75$ when using the cropping noise model with $s = 22$. On CIFAR10 we use a DenseNet-121 and on MNIST a CNN with architecture ‘conv2d(1, 10, 5), conv2d(10, 20, 5), dropout2d, linear(320, 50), linear(50,10)’. In all plots, shaded areas mark the 40th to 60th percentiles. To simplify the comparison across datasets (cf. Eq. 3 in Simon-Gabriel et al. 2019), we divide all ℓ_2 -

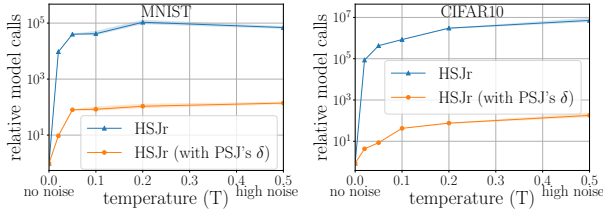


Figure 6: Ratio $N_{\text{HSJr}}/N_{\text{PSJ}}$ as a function of noise level, where N_{HSJr} is the total amount of queries needed by HSJ-with-repeated-queries to match the performance of PSJ with N_{PSJ} queries. HSJr needs several orders of magnitude more queries than PSJ, even when the classifier becomes increasingly deterministic, and even when using PSJ’s larger sampling radius δ for the gradient estimator (see paragraph “PSJ outperforms HSJr”).

distances by \sqrt{d} , where the input dimension d is 27×27 for MNIST and $3 \times 32 \times 32$ for CIFAR-10. Code is available at <https://github.com/cjsg/PopSkipJump>.

Adversarial accuracy (AA). Fig. 3 plots adversarial accuracy as a function of the attack size η for various noise levels and a fixed noise model using logit sampling (Figs. a & b), and for various noise models at fixed, high noise levels (Figs. c & d). The accuracy curves obtained with PSJ are well below their HSJ counterparts in all noisy settings and both curves coincide in the deterministic case (Figs. a & b, $T=0$). This illustrates the clear superiority of PSJ over HSJ. However, despite its standard use in the deterministic setting and its straight-forward generalization to probabilistic classifiers, AA is ill-suited for comparing performances between different noise models and noise levels. An easy way to see this is to notice that the AA curves do not even coincide at $\eta = 0$, even though the value at that point is just standard accuracy and does not depend on the attack algorithm. Instead, we will now introduce the (median) *border distance*, which generalizes the usual “median ℓ_2 -distance of adversarial examples” to the probabilistic setting, and which is better suited for comparisons across noise models and noise levels.

Remark 4. A deeper reason why AA is ill-suited for the comparison between noise levels is the following. From any probabilistic classifier one can define the deterministic classifier obtained by returning, at every point, the majority vote over an infinite amount of repeated queries at that point. This deterministic classifier is “canonically” associated to the probabilistic one in the sense that it defines the same classification boundaries. Naturally, any metric that compares an attack’s performance across various noise levels should be invariant to this canonical transformation. Concretely, it means that a set of adversarial candidates $\{x\}$ should get the same score on a probabilistic classifier than on its deterministic counterpart. The border distance defined below satisfies this property; AA does not.

Performance metric: border distance. In the deterministic case, the border distance is essentially the ℓ_2 -distance of the proposed adversarial examples to the original image. In the probabilistic case, however, an attack like PSJ may return points that are close to the boundary, but actually lie on the wrong side, because the underlying (typically unknown) logit of the true class is only marginally greater than the logit of the adversarial one. So, to ensure that we only measure distances to true adversarials and *for the purpose of evaluation only*, we will assume white-box access to the true underlying logits and then project all outputs x to the closest boundary point that lies on the line (x_*, x) , i.e., the closest point x' where the true and adversarial class have same probability. We define the *border-distance of x to x_** as the ℓ_2 -distance $\|x' - x_*\|_2$ (re-scaled by $1/\sqrt{d}$). Note that for this evaluation metric, what matters is not so much the output point x than finding an output-*direction* (x_*, x) of steep(est) descent for the underlying output probabilities.

PSJ is invariant to noise level and noise model. Figure 4 fixes the noise model (dropout) and compares PSJ’s performance at various noise levels (dropout rate α). (Similar curves for the other noise models can be found in appendix, Fig. 9.) Figure 5 instead studies PSJ’s performance on various noise models. More precisely, Fig. 4 shows the median border-distance at various noise levels (dropout rates α) as a function of PSJ iterations (a & c) and as a function of the median number of model queries obtained after each iteration (b & d). Shaded areas show the 40th and 60th percentiles of border-distances. Figs. (a) & (c) illustrate point 2. from the introduction: the *per iteration* performance of PSJ is largely independent of the noise level and on par with HSJ’s performance on the deterministic base classifier. This suggests that PSJ adapts its amount of queries *optimally* to the noise level: just enough to match HSJ’s deterministic performance, and not more. Figs. (b) & (d) illustrate point 3.: when the noise level decreases and the classifier becomes increasingly deterministic, the *per query* performance of PSJ converges to that of HSJ, in the sense that the PSJ curves become more and more similar to the limiting HSJ curve. Note that the log-scale of the x-axis can amplify small, irrelevant difference at the very beginning of the attack. Figure 5 show that the performance of PSJ is largely invariant to the different noise models considered here. Figs. 5 (b) & (d) also confirm that, contrary to HSJ that fails even with small noise, PSJ is largely invariant to changing noise levels and noise models.

PSJ outperforms HSJ-with-repeated-queries. Let HSJ- r be the HSJ attack with majority voting on r repeated queries at every point. Figure 6 studies how many total queries HSJ- r requires to match the performance of PSJ at various noise levels with logit-sampling. It reports the ratio $N_{\text{HSJr}}/N_{\text{PSJr}}$ of total amount of queries. Concretely,

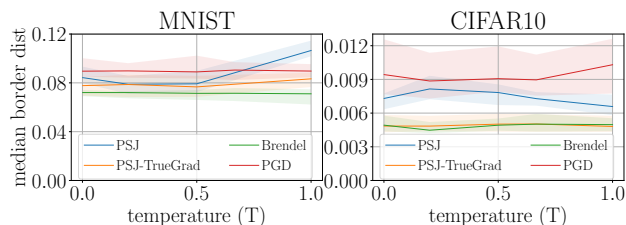


Figure 7: PSJ’s adversarial examples are on par with the white-box attacks, even in noisy regimes (high T), and even though the white box attacks use the true gradients of the tempered logits and therefore never face any actual noise.

for logit sampling with a given temperature T (the noise level), we first compute the median border-distance D_{PSJ} of PSJ after 32 iterations and the 40th, 50th, 60th percentiles N_{40}, N_{50}, N_{60} of the total amount of queries used in each attack. We then run HSJ- r for increasingly high values of r , which improves the median border-distance $D_{\text{HSJr}}(r)$ and increases the total number of queries $N(r)$. We stop when $D_{\text{HSJr}} = D_{\text{PSJ}}$ and plot the resulting ratio N/N_{50} (solid line), and $N/N_{40}, N/N_{60}$ (small shaded area around the median line). The result, Fig. 6, confirms property 1.: PSJ is *much* more query efficient than HSJr. At first, we were surprised that, even at very low noise levels, HSJr needed several order of magnitudes more model queries than PSJ. The reason, we found, is that HSJ uses a very small sampling radius ($\beta = 1/d$) for the gradient estimator, which impedes the estimation in the event of noise, as discussed in Section 3.5. We therefore also compare PSJ to a version of HSJr where we replaced the original sampling radius by the same one we used in PSJ (dashed line). The performance of HSJr improved dramatically, even though PSJ remains much more query efficient overall.

Small noise breaks HSJ. To confirm that small noise suffices to break HSJ, we compare the performance of HSJ and PSJ on MNIST, on a deterministic classifier where labels get corrupted (flipped) with probability $\nu \in \{0, .05, .1\}$ (as in Example ϕ_ν of Section 2). Results are reported in Table 1. Corrupting only 1 out of 20 queries ($\nu = 5\%$) suffices to greatly deteriorate HSJ’s performance (i.e., increase the median border-distance) – even with queries repeated 3 times –, while PSJ, with almost the same amount of queries than HSJ, is almost not affected. Appendix B.1 shows similar results when we replace HSJ by the boundary attack (Brendel et al., 2018). This inability of HSJ to deal with noise can also be seen on Figs. 5 (b) & (d) and 6.

PSJ vs white-box attacks. To evaluate how much performance we lose by ignoring information about the network architecture, we compare PSJ to several white-box attacks: to ℓ_2 -PGD with 50 gradient steps (Madry et al., 2018) and to the ℓ_2 -attack by Brendel et al. (2019), using their Fool-box implementations by Rauber et al. (2017; 2020); and

to a homemade “PSJ-TrueGrad” attack, where we replaced every gradient estimate in PSJ by the true gradient. We compare these attacks on 100 MNIST and 50 CIFAR10 test images, using the “logit sampling” noise model at various temperatures T . That way, the true underlying logits and their gradients are known and can be used by the white-box attacks without resorting to any averaging over random samples. This trick is not applicable to other noise models and makes the comparison with PSJ doubly unfair: first because the white-box attacks have access to the model’s gradients; and second, because here they never face any actual noise. Given these burdens, PSJ’s performance shown in Fig. 7 is surprisingly good: it is on par with the white-box attacks.

5. Conclusion

Although recent years have seen the development of several *decision-based* attacks for *deterministic* classifiers, small noise on the classification outputs typically suffices to break them. We therefore re-designed the particularly query-efficient *HopSkipJump* attack to make it work with probabilistic outputs. By modeling and learning the local output probabilities, the resulting *probabilistic HopSkipJump* algorithm, *PopSkipJump*, optimally adapts its queries to match HSJ’s performance at every iteration over increasing noise levels. It is *much* more query-efficient than the off-the-shelf alternative “HSJ (or any other SOTA decision-based attack) with repeated queries and majority voting”, and matches HSJ’s query-efficiency on deterministic and near-deterministic classifiers. We successfully applied PSJ to randomized adversarial defenses proposed at major recent conferences, and showed that they offer almost no extra robustness against decision-based attack as compared to their underlying deterministic base model. Our adaptations and statistical analysis of HSJ could be straightforwardly used to extend another decision-based attack, qFool by Liu et al. (2019), to cope with probabilistic answers. Overall, we hope that our method will help crafting adversarial examples in more real-world settings with intrinsic noise, such as for sets of classifiers or for humans. However, our results also suggest that the feasibility of such attacks will greatly depend on the noise level, since PSJ can require orders of magnitude more queries to achieve the same performance in the probabilistic setting than in the deterministic one.

Acknowledgements

We thank the reviewers for their valuable feedback and our families for their constant support. This project is supported by the Swiss National Science Foundation under NCCR Automation, grant agreement 51NF40 180545. CJSG is funded in part by ETH’s Foundations of Data Science (ETH-FDS).

References

- Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, 2018.
- Brendel, W., Rauber, J., and Bethge, M. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *ICLR*, 2018.
- Brendel, W., Rauber, J., Kümmeler, M., Ustyuzhaninov, I., and Bethge, M. Accurate, reliable and fast robustness evaluation. In *NeurIPS*, 2019.
- Cardelli, L., Kwiatkowska, M., Laurenti, L., Paoletti, N., Patane, A., and Wicker, M. Statistical guarantees for the robustness of Bayesian neural networks. In *International Joint Conference on Artificial Intelligence*, 2019a.
- Cardelli, L., Kwiatkowska, M., Laurenti, L., and Patane, A. Robustness guarantees for Bayesian inference with Gaussian processes. *AAAI Conference on Artificial Intelligence*, 2019b.
- Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. 2017.
- Chen, J., Jordan, M. I., and Wainwright, M. J. Hop-skipjumpattack: A query-efficient decision-based attack. In *IEEE Symposium on Security and Privacy*, 2019.
- Cohen, J., Rosenfeld, E., and Kolter, Z. Certified Adversarial Robustness via Randomized Smoothing. In *ICML*, 2019.
- Feinman, R., Curtin, R. R., Shintre, S., and Gardner, A. B. Detecting adversarial samples from artifacts. In *arXiv:1703.00410*, 2017.
- Gal, Y. and Ghahramani, Z. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.
- Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *ICLR*, 2015.
- Guo, C., Rana, M., Cisse, M., and Maaten, L. v. d. Countering adversarial images using input transformations. In *ICLR*, 2018.
- Ilyas, A., Engstrom, L., Athalye, A., and Lin, J. Black-box adversarial attacks with limited queries and information. In *ICML*, 2018.
- Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.
- Liu, Y., Moosavi-Dezfooli, S.-M., and Frossard, P. A geometry-inspired decision-based attack. In *ICCV*, 2019.
- Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.
- Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. DeepFool: A simple and accurate method to fool deep neural networks. In *CVPR*, 2016.
- Rauber, J., Brendel, W., and Bethge, M. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017.
- Rauber, J., Zimmermann, R., Bethge, M., and Brendel, W. Foolbox native: Fast adversarial attacks to benchmark the robustness of machine learning models in pytorch, tensorflow, and jax. *Journal of Open Source Software*, 5(53):2607, 2020.
- Simon-Gabriel, C.-J., Ollivier, Y., Bottou, L., Schölkopf, B., and Lopez-Paz, D. First-order adversarial vulnerability of neural networks and input dimension. In *ICML*, 2019.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(56):1929–1958, 2014.
- Xie, C., Wang, J., Zhang, Z., Ren, Z., and Yuille, A. Mitigating adversarial effects through randomization. In *ICLR*, 2018.