PopSkipJump: Decision-Based Attack for Probabilistic Classifiers

Carl-Johann Simon-Gabriel [1] **Noman Ahmed Sheikh** [1] **Andreas Krause** [1]

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 *P*(robabilisticH)*opSkipJump* 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 *p*robabilistic version of H*opSkipJump*, *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.

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

To appear in *Proceedings of the 38^{th} International Conference on Machine Learning*, PMLR 139, 2021.

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 ($\boldsymbol{x}, \boldsymbol{\delta}, \boldsymbol{g}, \ldots$), their coordinates and 1d variables are non-bold italic ($x, \delta, g, \ldots$). Random vectors are bold and upright ($\boldsymbol{\delta}, \mathbf{g}, \boldsymbol{\phi}, \ldots$), their coordinates and 1d random variables are non-bold upright ($\delta, \mathrm{g}, \phi, \ldots$) When it cannot be avoided, we will also use the index-notation (e.g., $\boldsymbol{\delta}_i$) to designate the (i-th) coordinate of vector ($\boldsymbol{\delta}$). $\mathrm{Ber}(p)$ 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) $\boldsymbol{\phi}$ from a set of inputs $\mathbb{X}$ to a set of K classes $\mathbb{K} := \{1, ..., K\}$. Said differently, for any $\boldsymbol{x} \in \mathbb{X}$, $\boldsymbol{\phi}(\boldsymbol{x})$ is a random variable taking values in $\mathbb{K}$.[1] Repeated queries of $\boldsymbol{\phi}$ at $\boldsymbol{x}$ yield i.i.d. copies of $\boldsymbol{\phi}(\boldsymbol{x})$.

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.

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

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 $\boldsymbol{\varphi}(\boldsymbol{x}) \in \mathbb{R}^K$ can be turned into a probabilistic classifier by sampling from its logits instead of returning the usual $\arg\max_k \boldsymbol{\varphi}(\boldsymbol{x})_k$. 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 $\boldsymbol{\varphi}$, which is why we also

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

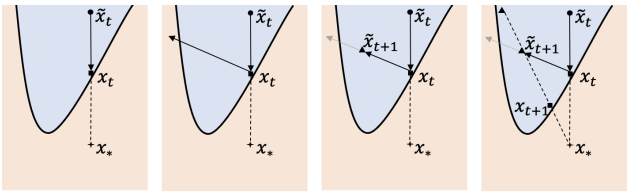

Figure 1: Original HopSkipJump algorithm for deterministic classifiers (Chen et al., 2019). Illustrations correspond to steps (a), (b), (c) and (a) from the text.

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

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

For deterministic classifiers, the expectation is taken only over the distribution of the labeled datapoints $(\boldsymbol{x}, c) \in \mathbb{X} \times \mathbb{K}$. For probabilistic classifiers, it is also taken over the randomness of the output $\phi(\boldsymbol{x})$ (at fixed input $\boldsymbol{x}$). We define adversarial accuracy as $1 - \mathrm{AR}(\mathcal{L}_{0/1})$ 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 $\boldsymbol{x}_*$ 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 $\boldsymbol{x}_*$ with label c and a neural network classifier $\boldsymbol{\varphi}$ which correctly assigns label c to $\boldsymbol{x}_*$, i.e., $c = \arg\max_{k \in \mathbb{K}} \boldsymbol{\varphi}(\boldsymbol{x}_*)_k$, and let $b(\boldsymbol{x}) := \boldsymbol{\varphi}(\boldsymbol{x})_c - \max_{k \in \mathbb{K} \setminus \{c\}} \boldsymbol{\varphi}(\boldsymbol{x})_k$. Then $b(\boldsymbol{x}) > 0$ if $\boldsymbol{\varphi}$ assigns class c to $\boldsymbol{x}$, and $b(\boldsymbol{x}) < 0$ if it does not. Consequently, we call *boundary* the set of points $\boldsymbol{x}$ such that $b(\boldsymbol{x}) = 0$.

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 $\boldsymbol{x}_t$ over iterations t by moving it along the decision boundary to get closer to the attacked image $\boldsymbol{x}_*$. More precisely, each iteration consists of three steps (see Fig. 1):

(a) *binary search* on the line between an adversarial image $\tilde{\boldsymbol{x}}_t$ and the target $\boldsymbol{x}_*$, which yields an adversarial point $\boldsymbol{x}_t$ near the classification border.

(b) *gradient estimation*, which estimates $\boldsymbol{g}(\boldsymbol{x}_t) := \nabla_{\boldsymbol{x}} b(\boldsymbol{x}_t) / \|\nabla_{\boldsymbol{x}} b(\boldsymbol{x}_t)\|_2$, the normal vector to the boundary at $\boldsymbol{x}_t$, as

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

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

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

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 $\theta_t^{\mathrm{det}} = d^{-3/2} \|\boldsymbol{x}_t - \boldsymbol{x}_*\|_2$ for stopping the binary search; (b) the sample size $n_t^{\mathrm{det}} = n_0^{\mathrm{det}} \sqrt{t}$ and sampling radius $\delta_t^{\mathrm{det}} = \theta_t^{\mathrm{det}} \sqrt{d}$ used to estimate the gradient; (c) the step size $\xi_t^{\mathrm{det}} = \|\boldsymbol{x}_t - \boldsymbol{x}_*\|_2 / \sqrt{t}$.

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 $\boldsymbol{x}_t$ can end up being non-adversarial and/or far from the classification border. For another, even if binary search worked, the gradient estimate $\hat{\boldsymbol{g}}$ needs more sample points $\boldsymbol{\delta}^{(i)}$ to reach the same average performance with noise than without.

Overview of PopSkipJump. To solve these issues, our *P*robabilistic H*opSkipJump* attack, *PopSkipJump*, replaces the binary search procedure by *(sequential) Bayesian experimental design, NoisyBinSearch*, that not only yields a point $\boldsymbol{x}_t$, 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 $\hat{\boldsymbol{g}}(\boldsymbol{x}_t)$ with the same expected performance – as measured by $\mathbb{E}[\cos(\hat{\boldsymbol{g}}, \boldsymbol{g})]$ 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 $p_c(\boldsymbol{x})$ of the class c of $\boldsymbol{x}_*$ (the

Algorithm 1 PopSkipJump

Input: attacked point $\boldsymbol{x}_*$; starting point $\tilde{\boldsymbol{x}}_0$ 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 $f =$ RHS of Eq. 3

  # Do noisy bin-search btw. $\tilde{\boldsymbol{x}}_t$ and $\boldsymbol{x}_*$ with target cos $= C_t^{\text{det}}$ used in the stopping criterion
  $\text{posterior}(z, s, \epsilon) \leftarrow \text{NoisyBinSearch}(\tilde{\boldsymbol{x}}_t, \boldsymbol{x}_*, C_t^{\text{det}})$ *# (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*
  $\boldsymbol{x}_t \leftarrow \hat{z}\tilde{\boldsymbol{x}}_t + (1 - \hat{z})\boldsymbol{x}_*$ *# 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 $f^{-1} =$ RHS Eq. 4

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

Algorithm 2 NoisyBinSearch

Input: attacked point $\boldsymbol{x}_*$; $\tilde{\boldsymbol{x}}_t$ from adv. class; probabilistic classifier ϕ with query model $\phi(x) \sim p_c(x)$ from Eq. 2; target cos C_t^{det}; prior $p(z, s, \epsilon)$
for $i = 1$ **to** ∞ **do**
  # Compute acquisition fct for all $x \in [\tilde{\boldsymbol{x}}_t, \boldsymbol{x}_*]$
  $a(x) = I(\phi(x), \text{z}, \text{s}, \epsilon)$ 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 cos $= C_t^{\text{det}}$
  $n_i = \mathbb{E}_{p(z,s,\epsilon)}\left[f^{-1}(z - \hat{z}, s, \epsilon)\right]$, $f^{-1} :=$ 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 $[\boldsymbol{x}_t, \boldsymbol{x}_*]$. More precisely, for $\boldsymbol{x} = (1 - x)\boldsymbol{x}_t + x\boldsymbol{x}_*$ with $x \in [0, 1]$, we assume that

$$p_c(\boldsymbol{x}) \equiv p_c(x) = \epsilon + (1 - 2\epsilon)\, \sigma(s(x - z)) \qquad (2)$$

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 $[\boldsymbol{x}_t, \boldsymbol{x}_*]$ at various iterations t of an attack on two sample images $\boldsymbol{x}_*$. 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 $\phi(\boldsymbol{x}^{(k)})$, update our prior with the posterior distribution of (ϵ, z, s) and iterate over these steps for $k = 0, 1, ...$ 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 $\text{I}(\phi(x) \,\|\, \text{s}, \text{z}, \epsilon)$ 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 $\mathbb{E}_{s,t,\epsilon}[n_t |\, \phi(x)]$ 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 $\boldsymbol{\delta}^{(i)}$ of the near-boundary point $\boldsymbol{x}_t$ on a sphere, we instead sample them from a normal distribution

$\mathcal{N}(0, \beta_t \boldsymbol{I}_d)$ 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, $\|\boldsymbol{\delta}^{(i)}\|_2 \approx \delta_t$).

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

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

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:

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

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 $\sigma(x) = \mathrm{clip}(x + 1/2, 0, 1)$. Then*

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

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

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 $n = n_t^{\mathrm{det}}$ 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 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 $\mathbb{E}_{z,s,\epsilon\sim\mathrm{posterior}}[n]$) using Eq. 4 and stop the binary search when the absolute difference $|n_{new} - n_{old}|$ 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 $\hat{\boldsymbol{g}}$) will be to z. This in turn will reduce the number of queries required for $\hat{\boldsymbol{g}}$ 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, $|n_{new} - n_{old}|$ 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 $\boldsymbol{\delta}^{(i)}$ in $\hat{\boldsymbol{g}}$ (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.

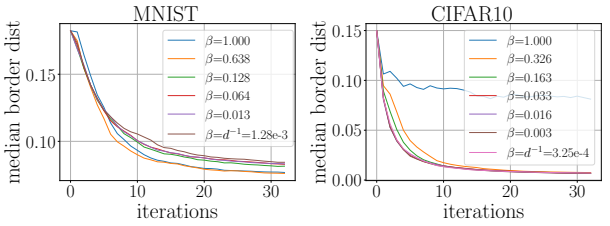

Figure 2: HSJ's performance on deterministic deep net classifiers is largely independent of the effective sampling radius $\delta_t = \beta\|\tilde{\boldsymbol{x}}_t - \boldsymbol{x}_*\|_2$ of the gradient estimator. So we can safely increase β (hence δ_t) by several orders of magnitude, which greatly 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.

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 $\delta_t = \|\tilde{\boldsymbol{x}}_t - \boldsymbol{x}_*\|_2/d$,

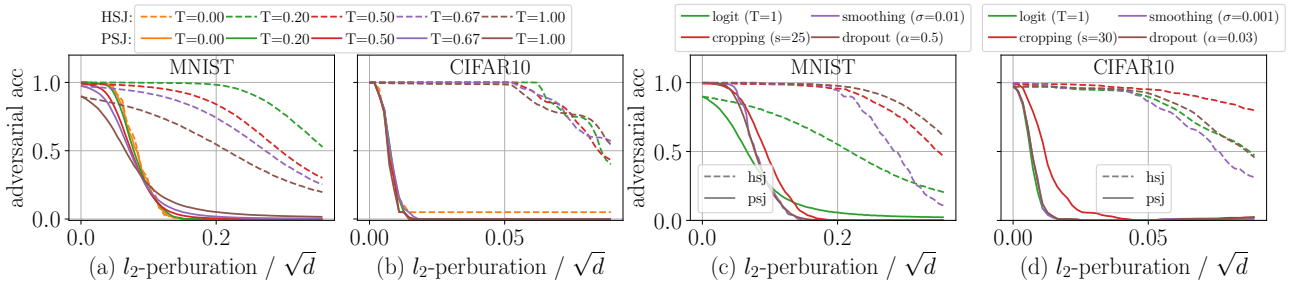

Figure 3: Adversarial accuracy versus attack size η for PSJ and HSJ for a fixed noise model (logit-sampling) and various noise levels (temperatures T) in Figs. (a) &(b); and for various noise models and fixed, high noise levels in Figs. (c) & (d). The curves obtained with PSJ are well below their HSJ 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 $\delta_t = \sqrt{d}\|\tilde{\boldsymbol{x}}_t - \boldsymbol{x}_*\|_2/100$. 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 $\tilde{\boldsymbol{x}}_t$ be on the adversarial side of the border. Therefore, it always tests whether the point $\tilde{\boldsymbol{x}}_t := \boldsymbol{x}_{t-1} + \xi_t \hat{g}(\boldsymbol{x}_{t-1})$ is indeed adversarial. If not, it divides ξ_t by 2 and tests again. Since by design $\boldsymbol{x}_{t-1}$ 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 $[\tilde{\boldsymbol{x}}_t, \boldsymbol{x}_*]$; and on the other, we are less interested in the point $\boldsymbol{x}_t$ and the point value $p_c(\boldsymbol{x}_t)$ than in the global *direction* from $\boldsymbol{x}_*$ to $\boldsymbol{x}_t$. We therefore use ξ_t as is, without geometric progression.

Enlarging bin-search interval $[\tilde{\boldsymbol{x}}_t, \boldsymbol{x}_*]$. 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 $\tilde{\boldsymbol{x}}_t$ tends to be very close to z. We therefore increase the size of the sampling interval, from $[\tilde{\boldsymbol{x}}_t, \boldsymbol{x}_*]$ to $[\boldsymbol{x}_* + 1.5(\tilde{\boldsymbol{x}}_t - \boldsymbol{x}_*), \boldsymbol{x}_*]$, 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.

[2]The 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.

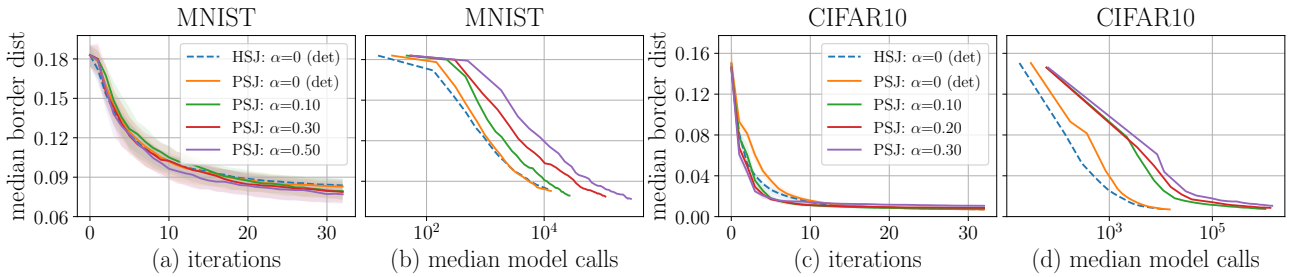

Figure 4: PopSkipJump's performance (lower is better) for a fixed noise model (dropout) and various noise levels (dropout rate α). Performance is shown as a function of the number of algorithm iterations (a & c) and model queries (b & d). Shaded areas depict the 40$^{\text{th}}$ to 60$^{\text{th}}$ percentiles. 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.

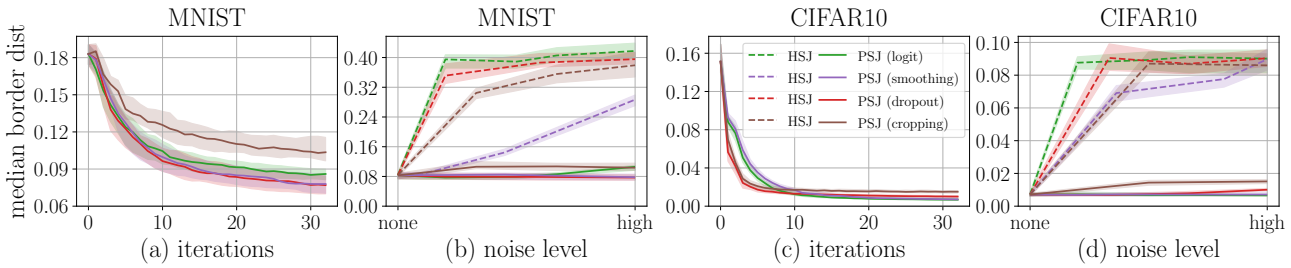

Figure 5: PopSkipJump's performance (lower is better) for various noise models. Plots (a) & (c): Performance as a function of the number of algorithm iterations, when using, for each noise model, the highest noise levels considered in Fig. 4 (see also Fig. 9 in appendix). All curves are very similar, 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 accross all noise levels and noise models.

• *logit sampling*: divide the output logits $\boldsymbol{\varphi}(\boldsymbol{x})$ by a temperature parameter T to get $\boldsymbol{\varphi}_T(\boldsymbol{x}) := \boldsymbol{\varphi}(\boldsymbol{x})/T$ and sample from the new logits $\boldsymbol{\varphi}_T(\boldsymbol{x})$. By changing T we can smoothly interpolate between the deterministic classifier ($T \to 0$) 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. Changing 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 40$^{\text{th}}$ to 60$^{\text{th}}$ percentiles. To simplify the comparison across datasets (cf. Eq. 3 in Simon-Gabriel et al. 2019), we divide all ℓ_2-

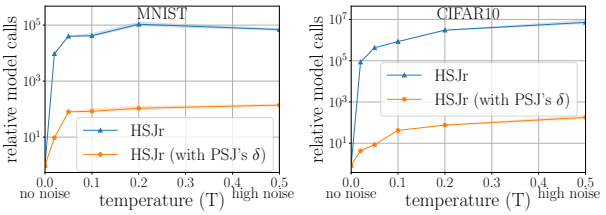

Figure 6: Ratio $N_{\text{HSJr}}/N_{\text{PSJ}}$ as a function of noise level, where N_{HSJr} is the total amount of queries needed by HSJ-with-repeated-queries to match the performance of PSJ with N_{PSJ} queries. HSJr needs several orders of magnitude more queries than PSJ, even when the classifier 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 $\{\boldsymbol{x}\}$ 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 $\boldsymbol{x}$ to the closest boundary point that lies on the line $(\boldsymbol{x}_*, \boldsymbol{x})$, i.e., the closest point $\boldsymbol{x}'$ where the true and adversarial class have same probability. We define the *border-distance of* $\boldsymbol{x}$ *to* $\boldsymbol{x}_*$ as the ℓ_2-distance $\|\boldsymbol{x}' - \boldsymbol{x}_*\|_2$ (re-scaled by $1/\sqrt{d}$). Note that for this evaluation metric, what matters is not so much the output point $\boldsymbol{x}$ than finding an output-*direction* $(\boldsymbol{x}_*, \boldsymbol{x})$ 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 40^{th} and 60^{th} 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 $N_{\text{HJSr}}/N_{\text{PSJr}}$ of total amount of queries. Concretely,

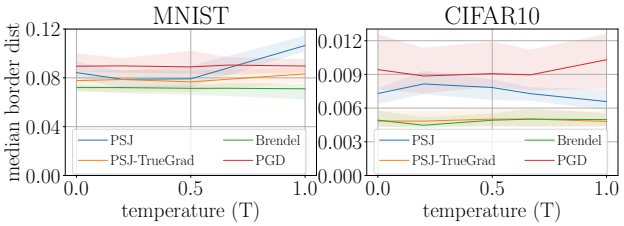

Figure 7: PSJ's adversarial examples are on par with the white-box attacks, even in noisy regimes (high T), and even though the white box attacks use the true gradients of the tempered logits and therefore never face any actual noise.

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 40$^{\text{th}}$, 50$^{\text{th}}$, 60$^{\text{th}}$ 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 Foolbox 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 *p*robabilistic H*opSkipJump* 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., Kümmerer, 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. Hopskipjumpattack: 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.

PopSkipJump: Decision-Based Attack for Probabilistic Classifiers Supplementary Materials

A. Justifications and Proofs for Section 3.4

A.0.1. JUSTIFYING EQ. 3

First, let us extend Eq. 2 and assume that, near the point $\boldsymbol{x}_t$ (where we estimate the gradient), the classification probabilities $p_c(\boldsymbol{x})$ are given (approximately) by a planar sigmoid, meaning:

$$p_c(\boldsymbol{x}) = \epsilon + (1-2\epsilon)\,\sigma(s\langle \boldsymbol{x}-\boldsymbol{z}\,,\,\boldsymbol{g}\rangle_2) = \epsilon + (1-2\epsilon)\,\sigma(s(x-z)) \tag{7}$$

where $s \in (0, +\infty]$, $\boldsymbol{g} = \boldsymbol{g}(\boldsymbol{z})$ is a unit vector in the gradient direction at point $\boldsymbol{z}$ (given by z in Eq. 2), and where x, z are the first coordinates of $\boldsymbol{x}, \boldsymbol{z}$ in an orthonormal basis $\mathcal{B} = (\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_d)$ where $\boldsymbol{e}_1 = \boldsymbol{g}$. In the deterministic case, when $s = \infty$, Eq. 7 amounts to assuming that the boundary is a linear hyperplane $\mathbb{H}$ going through $\boldsymbol{z}$ and orthogonal to $\boldsymbol{g}$. Also, notice that Eq. 7 is independent of the choice of $\boldsymbol{z}$, as long as $\boldsymbol{z}$ is contained in the hyperplane $\mathbb{H}$.

Remark 5 (Link between Eq. 2 and Eq. 7). Equation 7 is consistent with Eq. 2 in the sense that $p_c(\boldsymbol{x})$ will indeed be a sigmoid along any arbitrary line, as in Eq. 2. However, the notations are different: in Eq. 7, s, x, z are coordinates along $\boldsymbol{g}$, whereas in Eq. 2, they are coordinates along the line $[\tilde{\boldsymbol{x}}_t, \boldsymbol{x}_*]$. There is a factor $\cos(\tilde{\boldsymbol{x}}_t - \boldsymbol{x}_*, \boldsymbol{g})$ between the two, which, upon convergence, should converge to 1 (Chen et al., 2019, Thm.1).

Lemma 6. *Let n be a positive integer and $\boldsymbol{\delta}^{(i)} \sim \mathcal{N}(0, \boldsymbol{I}_d)$ for $i = 1, ...n$. Let $\beta > 0$ and $\phi(\boldsymbol{x} + \beta\boldsymbol{\delta}^{(i)}) \sim \mathrm{Ber}(p_c(\boldsymbol{x} + \beta\boldsymbol{\delta}))$ with values in $\{-1, 1\}$ and where p_c given by Eq. 7. Let $\hat{\boldsymbol{g}}(\boldsymbol{x}) := \mathbf{u}/\|\mathbf{u}\|_2$ with $\mathbf{u} := \sum_{i=1}^n \phi(\boldsymbol{x} + \beta\boldsymbol{\delta}^{(i)})\boldsymbol{\delta}^{(i)}$. Then*

$$\cos(\hat{\mathbf{g}}(\boldsymbol{x}), \boldsymbol{g}) = \frac{\xi}{\sqrt{\xi^2 + \chi^2_{d-1}/n}} \quad \textit{where} \quad \begin{cases} \chi^2_{d-1} \sim \textit{chi-square distribution of order } d-1 \\ \xi := \frac{1}{n}\sum_{i=1}^n \mathrm{y}_\epsilon(s(\Delta + \beta\delta^{(i)}))\boldsymbol{\delta}^{(i)} \\ \mathrm{y}_\epsilon(x) \sim \mathrm{Ber}(\epsilon + (1-2\epsilon)\,\sigma(x)) \textit{ and } \Delta := x - z. \end{cases} \tag{8}$$

Moreover, when $(d, n) \to (\infty, \infty)$ with $\frac{d-1}{n}$ converging to a fixed ratio denoted $\frac{\tilde{d}-1}{\tilde{n}}$, then, almost surely,

$$\cos(\hat{\mathbf{g}}(\boldsymbol{x}), \boldsymbol{g}) \xrightarrow{\text{a.s.}} \frac{1}{\sqrt{1 + \frac{\tilde{d}-1}{\tilde{n}\alpha}}} \quad \textit{where} \quad \alpha := \mathop{\mathbb{E}}_{\delta,\mathrm{y}}[\mathrm{y}(s(\Delta + \beta\delta))\delta]\,. \tag{9}$$

Equation 9 says that, if n and d are large enough, we can replace the random quantities ξ and χ^2_{d-1} of Eq. 8 by their expectations, $d-1$ and α respectively, and get the RHS of Eq. 3. Hence, it proves Eq. 3 in the large n and d limit. However, one may wonder whether Eqs. 3 and 9 also hold with good approximations for finite n, d. It is not difficult to estimate the order of magnitude of n and d needed in Eq. 9 by computing the variances of ξ and χ^2_{d-1} and using the central limit theorem. However, since Eq. 3 has an additionnal expectation on the LHS which may quicken the convergence, we will prefer to approximate $\mathbb{E}[\cos(\hat{\mathbf{g}}, \boldsymbol{g})]$ by the average over many random draws from Eq. 8 (Monte-Carlo method) and comparing it with the results obtained by the RHS of Eq. 3. Results are shown in Fig. 8.

Proof. (Lemma 6.) Let us work in an orthonormal basis $\mathcal{B} := (\boldsymbol{e}_1 = \boldsymbol{g}, \boldsymbol{e}_2, ..., \boldsymbol{e}_d)$ of $\mathbb{R}^d$ and possibly drop the index '1' for the first coordinate (as in x for x_1). Since $\cos(\hat{\mathbf{g}}(\boldsymbol{x}), \boldsymbol{g})$ is invariant by orthogonal translations to $\boldsymbol{g}$, i.e., by changes in the coordinates $(x_2, x_3, \ldots, x_d)$ of $\boldsymbol{x}$, let us assume, w.l.o.g., that $\boldsymbol{x} = (x, 0, 0, \ldots, 0)$. Then

$$\begin{aligned} \cos(\hat{\mathbf{g}}(\boldsymbol{x}), \boldsymbol{g}) &= \frac{\langle \hat{\boldsymbol{g}}(\boldsymbol{x})\,,\,\boldsymbol{g}\rangle_2}{\|\hat{\boldsymbol{g}}\|_2\|\boldsymbol{g}\|_2} \overset{(*)}{=} \frac{\sum_{i=1}^n \phi(\boldsymbol{x} + \beta\boldsymbol{\delta}^{(i)})\langle \boldsymbol{\delta}^{(i)}\,,\,\boldsymbol{g}\rangle_2}{\|\sum_{i=1}^n \phi(\boldsymbol{x} + \beta\boldsymbol{\delta}^{(i)})\boldsymbol{\delta}^{(i)}\|_2} \\ &= \frac{\sum_{i=1}^n \phi(\boldsymbol{x} + \beta\boldsymbol{\delta}^{(i)})\delta^{(i)}}{\sqrt{\left(\sum_{i=1}^n \phi(\boldsymbol{x} + \beta\boldsymbol{\delta}^{(i)})\delta^{(i)}\right)^2 + \sum_{j=2}^d \left(\sum_{i=1}^n \phi(\boldsymbol{x} + \beta\boldsymbol{\delta}^{(i)})\delta_j^{(i)}\right)^2}} \end{aligned}$$

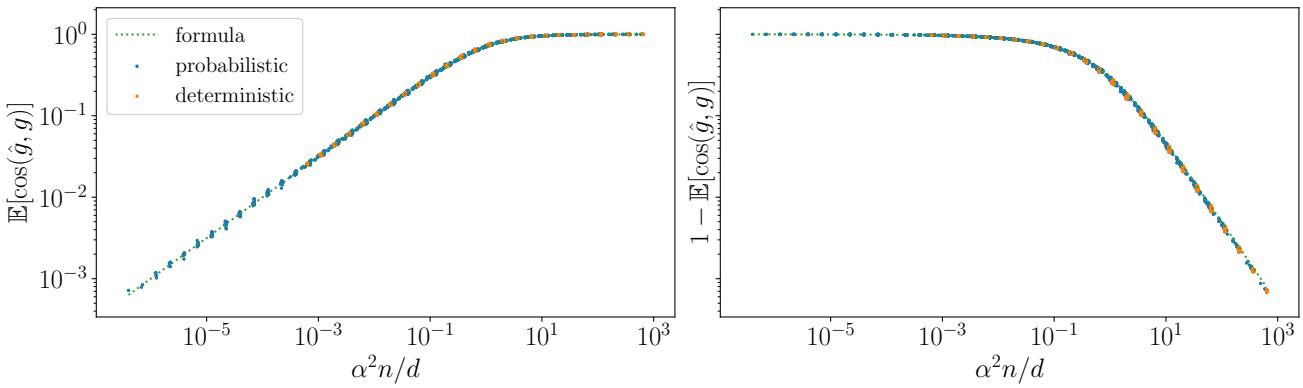

Figure 8: Comparing analytical and numerical approximations of $\mathbb{E}[\cos(\hat{\mathbf{g}}, \boldsymbol{g})]$. Dotted line: $= (1 + \frac{d-1}{n\alpha^2})^{-1/2}$ (eq. 3). Points: $\mathbb{E}[\cos(\hat{\mathbf{g}}, \boldsymbol{g})]$ computed by averaging repeated draws from Eq. 8. Each point represents one such average for a given combination of $(n, d, s) \in \mathbb{N} \times \mathbb{D} \times \mathbb{S} \cup \{\infty\}$, where $\mathbb{N} = \mathbb{D}$ and $\mathbb{S}$ where computed using numpy's `logspace`$(1, 4, \texttt{num} = 13)$ and `logspace`$(-2, 2, \texttt{num} = 17)$ respectively. For all points, we fixed $\beta = 1$ and $x - s = 0$. Points for $s = \infty$ are colored orange.

$$= \frac{\xi_1}{\sqrt{\xi_1^2 + \sum_{j=1}^d \xi_j^2}},$$

where we defined $\xi_j := \frac{1}{n} \sum_{i=1}^n \phi(\boldsymbol{x} + \beta \boldsymbol{\delta}^{(i)}) \delta_j^{(i)}$. With the change of variable $\mathrm{y}(s(\Delta + \beta\delta^{(i)})) = \phi(\boldsymbol{x} + \beta \boldsymbol{\delta}^{(i)})$ and using Eq. 7, we see that $\mathrm{y}(x) \sim \epsilon + (1 - \epsilon)\, \sigma(x)$ and get $\xi_j := \frac{1}{n} \sum_{i=1}^n \mathrm{y}(s(\Delta + \beta\delta^{(i)})) \boldsymbol{\delta}^{(i)}$.

Since, for any i, $\mathrm{y}(s(\Delta + \beta\delta^{(i)}))$ follows a Bernoulli distribution that is independent of $\delta_j^{(i)} \sim \mathcal{N}(0, 1)$ whenever $j \geq 2$, the products $\mathrm{y}(s(\Delta + \beta\delta^{(i)}))\delta_j^{(i)}$ follows again a standard normal distribution. So, for any $j \geq 2$, ξ_j is the mean of n independent Gaussians $\mathcal{N}(0, 1)$, hence $\xi_j \sim \mathcal{N}(0, \sigma^2 = 1/n)$. Since all $(\delta_j^{(i)})_{ij}$ are mutually independent, so are the products $(\mathrm{y}(s(\Delta + \beta\delta^{(i)}))\delta_j^{(i)})_{ij}$, and therefore also all $(\xi_j)_j$. Hence $\chi^2_{d-1} := n \sum_{j=2}^d \xi_j^2$ follows a chi-squared distribution of order $d - 1$, which yields Eq. 8, where $\xi = \xi_1$.

For Eq. 9, note that, by the law of large numbers, almost surely, $\xi_1 \to \mathbb{E}[\xi_1] = \alpha$ as $n \to \infty$, and $\chi^2_{d-1}/d{-}1 \to 1$ as $d \to \infty$, i.e., $\chi^2_{d-1}/n = \frac{d-1}{n} \frac{\chi^2_{d-1}}{(d-1)} \to \frac{\tilde{d}-1}{\tilde{n}}$. We conclude by applying the continuous mapping theorem with the function $(x_1, x_2) \mapsto x_1/\sqrt{x_1^2 + x_2}$ to $(\xi_1, \chi^2_{d-1}/n)$ when $n, d \to \infty$ with $\frac{d-1}{n} \to \frac{\tilde{d}-1}{\tilde{n}}$. □

A.0.2. Proof of Proposition 2

First, the following computations shows that we can recover the case of arbitrary $\beta > 0$ from the case $\beta = 1$.

$$\alpha(\Delta, s, \beta, \epsilon) := \mathbb{E}[\mathrm{y}_\epsilon(s(\Delta + \beta\delta))\delta] = \mathbb{E}[\mathrm{y}_\epsilon(s\beta(\Delta/\beta + \delta))\delta] = \alpha(\Delta/\beta, s\beta, 1, \epsilon)$$

So, from now on, let us assume that $\beta = 1$. Then

$$\begin{aligned}
\mathbb{E}[\mathrm{y}_\epsilon(s(x + \delta))\delta] &= \int_{\delta=-\infty}^{+\infty} \delta \Big(\underbrace{p(\delta = \delta,\, \mathrm{y}_\epsilon(s(\Delta + \delta)) = 1)}_{p(\delta=\delta)(\epsilon + (1-2\epsilon)\, \sigma(s(\delta+\Delta)))} + \underbrace{p(\delta = -\delta,\, \mathrm{y}_\epsilon(s(\Delta + \delta)) = -1)}_{p(\delta=\delta)(1-\epsilon-(1-2\epsilon)\, \sigma(s(-\delta+\Delta)))} \Big)\, \mathrm{d}\delta \\
&= \int_{\delta=-\infty}^{+\infty} \delta \mathcal{N}(\delta; 0, 1) \Big((1 - 2\epsilon)\, \sigma(s(\delta + \Delta)) + 1 - (1 - 2\epsilon)\, \sigma(s(\Delta - \delta)) \Big)\, \mathrm{d}\delta \\
&= (1 - 2\epsilon) \Bigg(\underbrace{\int_{\delta=-\infty}^{+\infty} \delta \mathcal{N}(\delta; 0, 1)\, \sigma(s(\delta + \Delta))\, \mathrm{d}\delta}_{(**)} + \underbrace{\int_{\delta=-\infty}^{+\infty} \delta \mathcal{N}(\delta; 0, 1)\, \sigma(s(\delta - \Delta))\, \mathrm{d}\delta}_{(*)} \Bigg).
\end{aligned}$$

Next we compute $(*)$ and $(**)$.

$$(*) = \int_{-\infty}^{\Delta - 1/2s} 0 \,\mathrm{d}\delta + \underbrace{\int_{\Delta - 1/2s}^{\Delta + 1/2s} \delta \mathcal{N}(\delta; 0, 1)(1/2 + s(\delta - \Delta)) \,\mathrm{d}\delta}_{(a)} + \underbrace{\int_{\Delta + 1/2s}^{+\infty} \delta \mathcal{N}(\delta; 0, 1) \,\mathrm{d}\delta}_{(b)}$$

$$\begin{aligned}
(a) &= \frac{1}{\sqrt{2\pi}} \int_{\Delta - 1/2s}^{\Delta + 1/2s} s(\delta^2 - 1) e^{\delta^2/2} + s e^{\delta^2/2} + (1/2 - s\Delta)\delta e^{-\Delta^2/2} \,\mathrm{d}\delta \\
&= \frac{1}{\sqrt{2\pi}} \left[-s\delta e^{-\delta^2/2} + s\sqrt{2\pi}\Phi(\delta) - (1/2 - s\Delta) e^{-\delta^2/2} \right]_{\Delta - 1/2s}^{\Delta + 1/2s} \\
(b) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{1}{2s} + \Delta)^2} \,,
\end{aligned}$$

where Φ designates the cumulative distribution function of the standard normal. Noticing that $(**)$ is $(*)$ with Δ replaced by $-\Delta$ (in (a) and (b)) and adding everything up, we get

$$\begin{aligned}
\mathbb{E}[\mathrm{y}_\epsilon(s(\Delta + \delta))\delta] &= (1 - 2\epsilon) 2s \left(\Phi(1/2s + \Delta) + \Phi(1/2s - \Delta) - 1 \right) \\
&= (1 - 2\epsilon) 2s \left(\Phi(1/2s + \Delta) + \Phi(\Delta - 1/2s) \right) \\
&= (1 - 2\epsilon) s \left(\mathrm{erf}\left(\frac{\Delta + 1/2s}{\sqrt{2}} \right) - \mathrm{erf}\left(\frac{\Delta - 1/2s}{\sqrt{2}} \right) \right) \,. \qquad (10)
\end{aligned}$$

As for the (deterministic) case $s = \infty$, we can either redo the previous calculations with σ being the step function $\mathbf{1}(x) = 1$ if $x \geq 0$ and 0 otherwise; or we can let $s \to \infty$ in Eq. 10 and use a Taylor development the error-function erf centered on $\Delta/\sqrt{2}$, which gives

$$\mathbb{E}[\mathrm{y}_\epsilon(s(\Delta + \delta))\delta] = (1 - 2\epsilon) s \left(\mathrm{erf}'\left(\frac{\Delta}{\sqrt{2}} \right) \frac{1}{s\sqrt{2}} + \mathrm{erf}'''\left(\frac{\Delta}{\sqrt{2}} \right) \frac{2}{3!(2s\sqrt{2})^3} + O(\frac{1}{s^5}) \right)$$

with $\mathrm{erf}'(x) = \frac{2}{\sqrt{\pi}} e^{-x^2}$ and $\mathrm{erf}'''(x) = \frac{4}{\sqrt{\pi}}(2x^2 - 1)e^{-x^2}$. Hence

$$\mathbb{E}[\mathrm{y}_\epsilon(s(\Delta + \delta))\delta] = (1 - 2\epsilon)\sqrt{\frac{2}{\pi}} e^{-\Delta^2/2} \left(1 + (\Delta^2 - 1)\frac{1}{24s^2} + O\left(\frac{1}{s^4} \right) \right) \;\to\; (1 - 2\epsilon)\sqrt{\frac{2}{\pi}} e^{-\Delta^2/2}.$$

B. Additionnal Plots and Tables

B.1. Extension of Table 1

In Tables 2 and 3 we extend Table 1 on the performance of decision-based attacks in presence of small noise. The extended tables also show the performance of the boundary attack (Brendel et al., 2018) (BA) and of the boundary attack with three repeated queries and majority voting (BA-repeat3). We have now broken the results into two tables: Table 2 shows the performance of the various attacks in terms of border-distance (see definition Section 4); where Table 3 reports the total number of model calls needed. Overall, both tables confirm Table 1: small noise suffices to break traditional decision-based attacks, even with repeated queries. The performance of PSJ stays identical, with only a few more queries in the noisy setting (and far less queries than the repeated queries based attacks).

Table 2: Extension of Table 1, showing the median border-distance achieved by various decision-based attacks at various noise levels. PSJ, HSJ, BA stand for PopSkipJump (our attack), HopSkipJump and boundary attack (Brendel et al., 2018) respectively. HSJ-repeat3 and BA-repeat3 are HSJ and BA where we repeat every query 3 times and apply majority voting. First number reports the median border-distance, as defined in Section 4. The two numbers in bracket are the 40^{th} and 60^{th} percentiles. All numbers were computed on the MNIST test subset of 500 images using the CNN described in Section 4. A noise level $\nu = 5\%$ means that the CNN outputs its argmax label with probability .95, and some other random label with probability .05. Conclusion: all attacks perform similarly in the deterministic case ($\nu = 0$), but small noise ν suffices to break HSJ and BA (with or without repeated queries), but not PSJ.

FLIP	PSJ	HSJ	HSJ-repeat3	BA	BA-repeat3
$\nu = 0\%$	0.60[0.45-0.72]	0.61[0.51-0.72]	0.59[0.48-0.72]	0.46[0.38-0.55]	0.49[0.36-0.56]
$\nu = 5\%$	0.63[0.53-0.76]	2.16[1.76-2.93]	0.75[0.58-1.01]	15.58[13.66-16.94]	15.60[12.94-16.73]
$\nu = 10\%$	0.65[0.55-0.80]	3.67[3.15-4.26]	1.32[0.98-1.61]	15.84[14.35-16.85]	15.90[14.24-16.96]

Table 3: Same as Table 2 but showing the median model calls. In the deterministic setting, PSJ only needs a few more calls than HSJ, and much less calls than BA. In the noisy setting, the number of calls increase only slightly for PSJ, while its performance (Table 2) is often much better than the repeated-query attacks, which need much more calls.

FLIP	PSJ	HSJ	HSJ-repeat3	BA	BA-repeat3
$\nu = 0\%$	14270	12844	38532	27560	82665
$\nu = 5\%$	17856	13333	42009	27561	82716
$\nu = 10\%$	17931	13108	40813	27551	82677

B.2. From PSJ to HSJ for various attacks noise levels

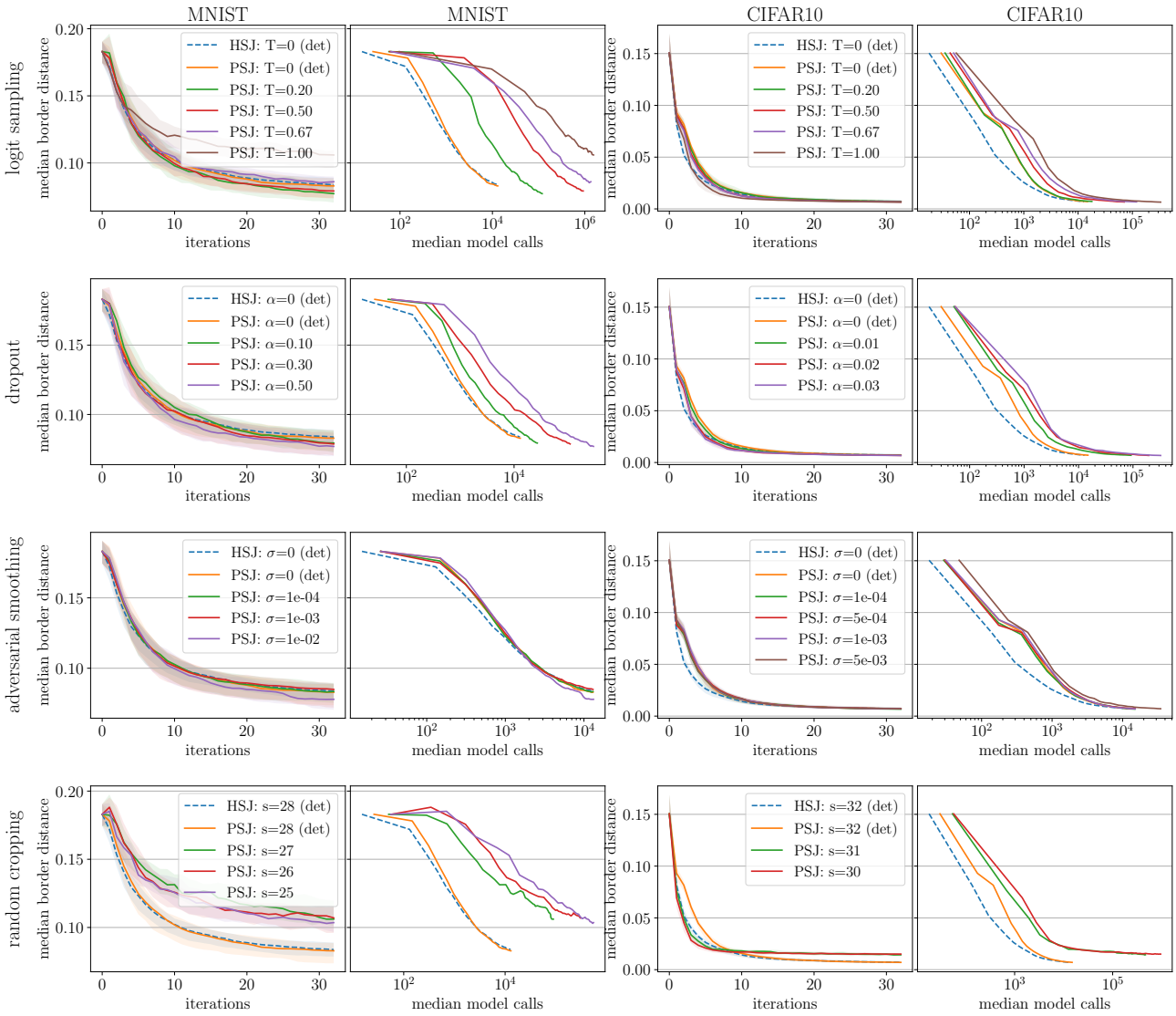

Figure 9: Evolution of PopSkipJump's performance (median border-distance) with the amount of algorithm iterations (columns a & c) and model queries (columns b & d). Each row uses a different noise model (i.e., randomization scheme) applied to a same deterministic base classifier (CNN for MNIST, Densenet for CIFAR10). Each curve uses a different noise level. Columns a & c illustrate property 2.: the *per iteration* performance of PSJ is largely independent of the noise level and noise model, and is on par with HSJ's performance on the deterministic base classifier. Columns b & d illustrate property 1.: 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.

B.3. Output probabilities along bin-search lines are sigmoids

In this section we briefly corroborate our assumption from equation 2 that the probabilities of neural networks along the binary search lines $[\boldsymbol{x}_t, \boldsymbol{x}_*]$ have a sigmoidal shape. We do so by plotting these probabilities in Figs. 10 and 11 on two randomly chosen images – one from MNIST and one from CIFAR10 – at various iterations of the attack. Note that we got similar plots for almost all images that we tested.

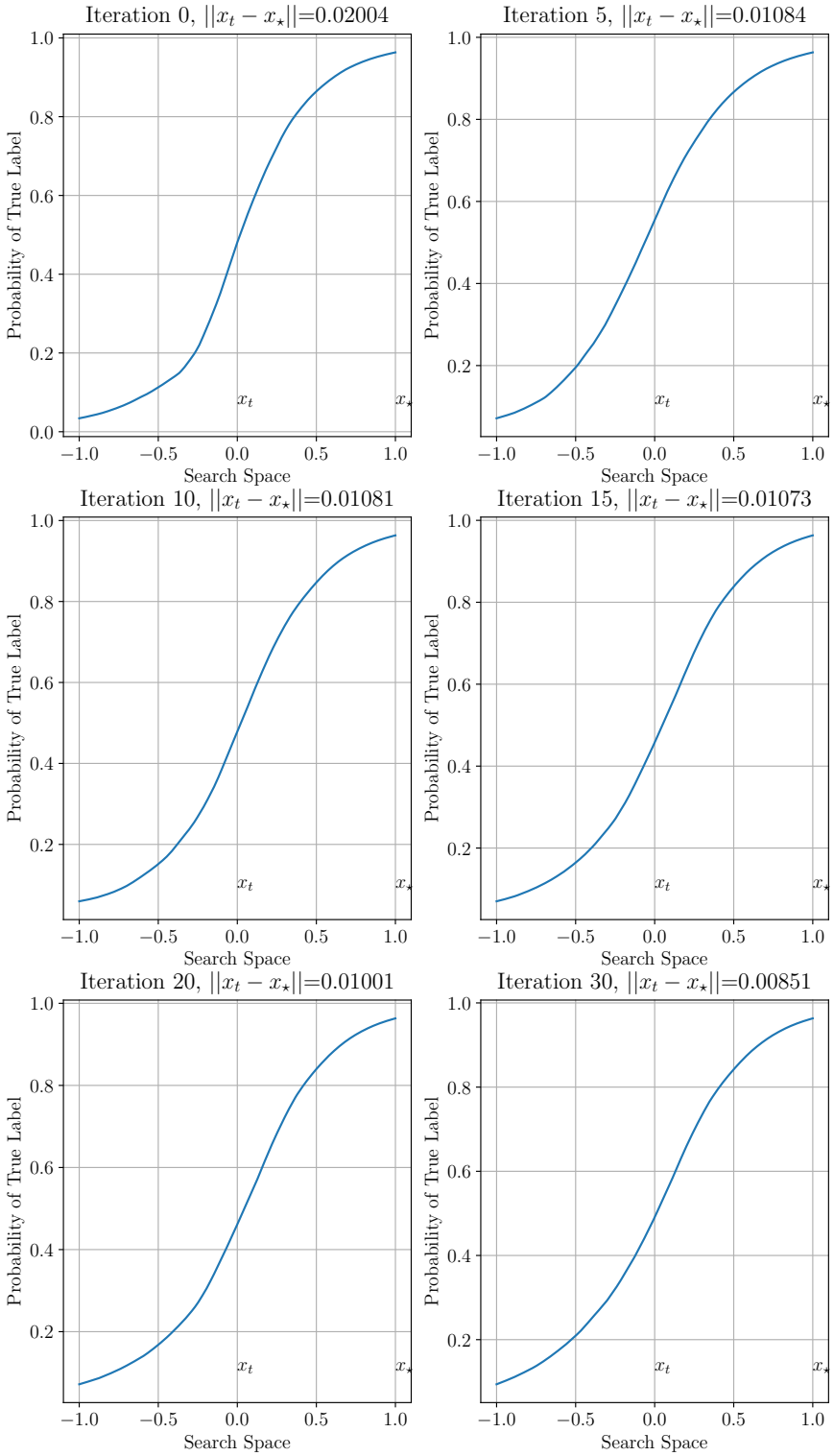

Figure 10: Output probabilities of the CNN along binary search direction $[\tilde{\boldsymbol{x}}_t, \boldsymbol{x}_*]$ for different iterations t of an attack on some fixed MNIST image $\boldsymbol{x}_*$. These plots match our assumption that the model's probabilities along binary search directions are given by a sigmoid, as in Eq. 2. This assumption was satisfied for almost all images $\boldsymbol{x}_*$ that we checked.

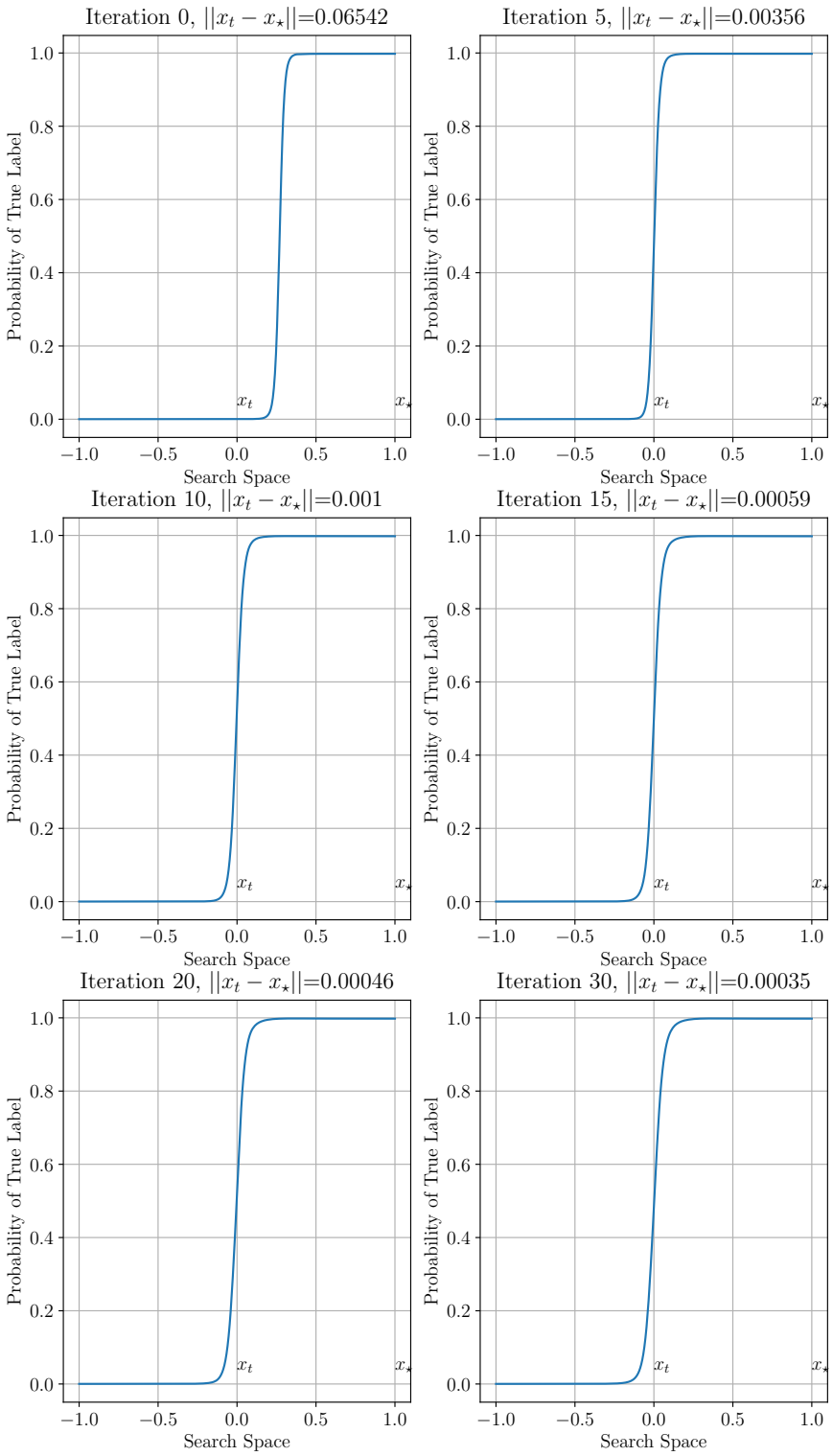

Figure 11: Same as Fig. 10, but for the Densenet used on a CIFAR-10 image $\boldsymbol{x}_*$. There again, the output probabilities sigmoidal, but closer to a step-function than in Fig. 10.

C. How sensitive is HSJ to the number of gradient queries n_t per iteration?

In Section 3.4 we showed how to use the estimated output distribution from the binary search step to compute the number of gradient queries required to get the same estimation quality in the noisy case than in the deterministic one. This makes PSJ's per-iteration performance independent of the noise level and would in particular allow us to optimize the number of gradient queries in the deterministic setting only, and then automatically infer the optimal number of queries for all noisy settings without any additional optimization. One natural question then, however, is how sensitive HSJ is to the number of gradient queries n_t per round t. To get a rough idea, we plot the performance of HSJ in the deterministic setting when multiplying the original, default number of gradient queries $n_t = 100\sqrt{t}$ by a factor $r = 1, 2, ..., 5$. The results are shown in Fig. 12.

Not surprisingly, the performance *per iteration* of HSJ (Fig. a) increases with r, since more queries means a better gradient estimate. However, the performance as a function of the overall number of queries (Fig. b) seems independent of r. This suggests that the overall performance of HSJ – and therefore also of PSJ – is not very sensitive to the actual number of gradient queries per iteration, which in turn suggests that the considerations in Section 3.4 and formulae Eqs. 3 and 4 are not essential to the query efficiency and success of PSJ.

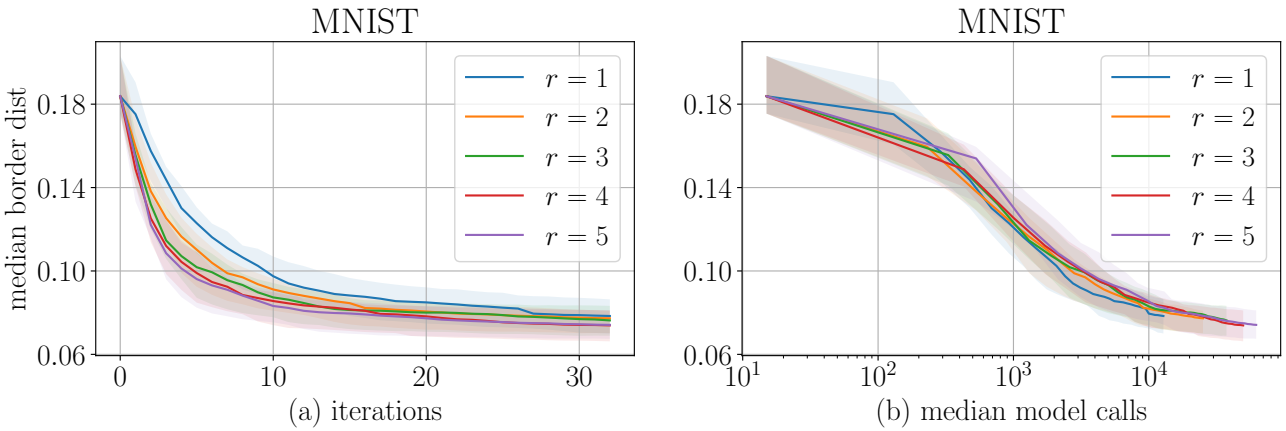

Figure 12: Performance of HSJ (lower is better) on a deterministic classifier when multiplying the default number of gradient queries n_t at every iteration t by a constant factor r. Not surprisingly, the performance as a function of the algorithm's iterations (Fig. a) increases with the query factor r, since more queries means a better gradient estimate. However, the performance as function of the overall number of queries (Fig. b) seems independent of r, which suggests that the considerations in Section 3.4 and Eqs. 3 and 4, however beautiful in theory, are not essential to the query efficiency and success of PSJ.

D. Time, query and computational complexity of PopSkipJump

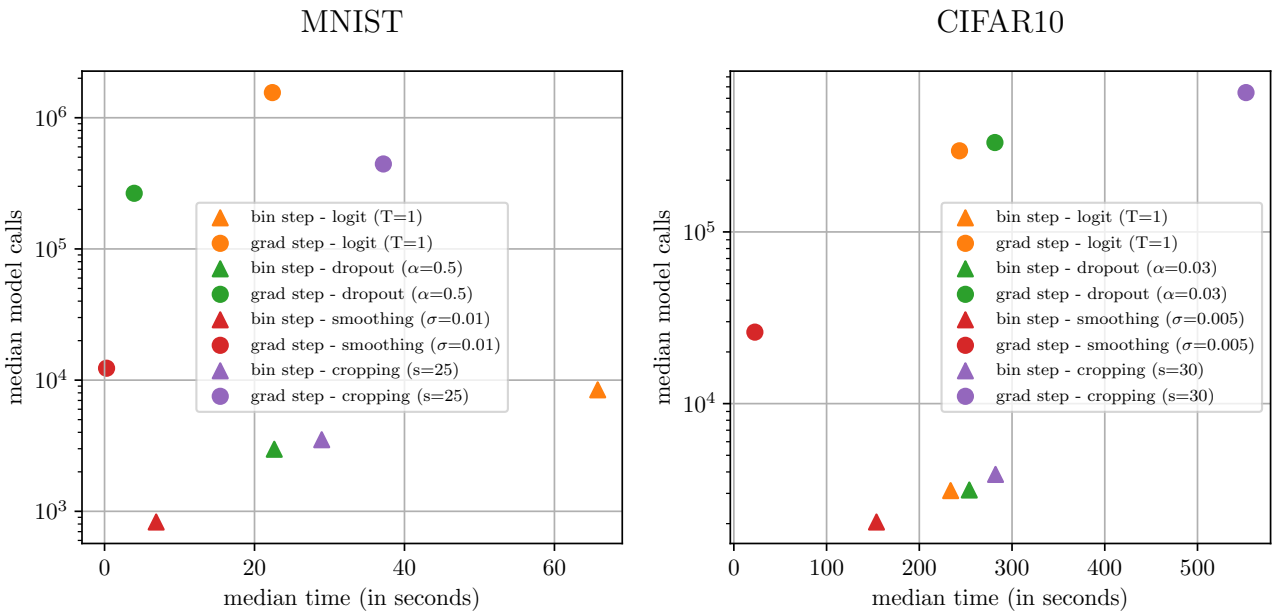

Figure 13: Number of queries versus time spent in different parts of the PSJ algorithm. Gradient estimation needs many ($\geq 100\times$) more queries than noisy bin-search but less time, because its queries are independent, hence batchable and parallelizable, whereas noisy bin-search is sequential and has an expensive information maximization step after each query. Here, gradient estimation queries are computed in batches of 256 queries (on a single GPU). For every noise model, we used the highest noise levels that we considered in this paper (temperature T, dropout rate α, standard deviation σ and cropped size s).

Overview. PopSkipJump has two resource-intensive parts: noisy binary search and the gradient estimation step (points (a) and (b) in Section 3.1 respectively). Gradient estimation typically needs many more queries than binary search ($\geq 100\times$ more). Every query has same time and computational complexity, which increases with the network architecture. But the queries for gradient estimation are all independent and can therefore be batched and parallelized. The overall time for gradient estimation can in principle be driven down arbitrarily with enough cores or GPUs. Noisy bin-search on the other side is sequential by essence and has an expensive mutual information maximization step after every query. In our experiments, this leads to comparable time complexities, as shown in Fig. 13. We will now first discuss in more details the *query* complexities of bin-search and gradient estimation, then the *computational* complexity of the information maximization step, and finish with two tricks to accelerate the noisy bin-search steps.

D.1. Query complexities.

Both for the noisy bin-search and for the gradient estimation, the number of queries depends mainly on the true parameters ϵ and s of the underlying sigmoid (i.e., roughly speaking, on the noisiness of the classifier): increasing the noise ϵ and/or decreasing the shape s (i.e., flattening the sigmoid) tends to increase the expected amount of queries. For gradient estimation, the exact number of queries is computed as described by Algorithm 1 and its formula for n_t (see also Section 3.4). There, for the computation of C_t^{det} (the expected cosine at step t for a deterministic classifier), we used $n_t^{det} = 100\sqrt{t}$ (as in the original HSJ algorithm), and $\theta_t^{det} = 0.010$ and $\beta = 0.280$ for MNIST, and $\theta_t^{det} = 0.003$ and $\beta = 0.185$ for CIFAR10. (As explained in the caption of Fig. 2, since we use a larger β than in the original HSJ algorithm, we also use a larger "bin-size" θ_t^{det}.) As for noisy bin-search, the number of queries is driven by the number of queries needed to determine the posterior distribution of the sigmoid parameters with sufficient precision to meet the stopping criterium from Section 3.4. More noise ϵ and flatter sigmoid shapes (i.e., smaller values of s) both decrease the expected amount of information on the sigmoid's center z carried by every query, which increases the expected total number of queries needed.

D.2. Computational complexity.

As discussed above, the computational complexity of every query – be it for bin-search or for gradient estimation – is mainly driven by the network architecture and its size. The complexity of the mutual information maximization, however, depends on the discretizations used to model the prior/posterior probabilities of the sigmoid parameters z, ϵ and s. More precisely, we represent our priors/posteriors over z, ϵ and s by constraining them to intervals I_z, I_ϵ, I_s that are discretized into n_z, n_ϵ and n_s bins/points respectively. At every bin-search step, and for n_x values of $x \in I_z$, we compute the mutual information $\mathrm{I}(\phi(x) \,\|\, \mathrm{s}, \mathrm{z}, \epsilon)$ given by:

$$\mathrm{I}(\phi(x) \,\|\, \mathrm{s}, \mathrm{z}, \epsilon) = \sum_{z \in I_z, \epsilon \in I_\epsilon, s \in I_s} p(\phi(x), s, z, \epsilon) \log \frac{p(\phi(x), s, z, \epsilon)}{p(\phi(x)) p(s, z, \epsilon)}$$

where $p(\phi(x), s, z, \epsilon) = p(\phi(x) \,|\, s, z, \epsilon) p(s, z, \epsilon) = p_c(x) p(s, z, \epsilon)$ (see Eq. 2), $p(\phi(x))$ is given by marginalizing out s, z, ϵ in $p(\phi(x), s, z, \epsilon)$ and where $p(s, z, \epsilon)$ is the current prior/posterior. Hence, $\mathrm{I}(\phi(x) \,\|\, \mathrm{s}, \mathrm{z}, \epsilon)$ is a sum of $n_z n_\epsilon n_s$ terms with same complexity each, so it costs $O(n_z n_\epsilon n_s)$. Since we repeat this computation for every location $x \in I_z$, the overall computation of the mutual information acquisition function has complexity

$$\text{complexity of mutual info maximization} = O(n_x n_z n_\epsilon n_s). \tag{11}$$

In practice, we chose $I_\epsilon = \{0.9, 1\}$ (i.e., $n_\epsilon = 2$), and $\log_{10}(I_s) = [1, 3]$ with $n_s = 31$, and

- for MNIST, $I_z = I_x = [0, 1]$ with $n_z = n_x = 101$;
- for CIFAR10, $I_z = I_x = [0, 1]$ with $n_z = n_x = 301$.

The values of n_z were chosen so that the step-size $\tilde{\theta}$ would roughly respect the ratio $\beta/\tilde{\theta} = \delta_t/\theta_t = \sqrt{d}$ suggested by Chen et al. (2019) . They could probably be optimized further.

D.3. Accelerating noisy binary search

In this section we study the following two tricks to accelerate the noisy bin-search steps.

(1) *sampling multiple points after each information maximization step*: since one of the bottle-necks of noisy bin-search is that queries cannot be batched, we considered altering the algorithm by querying the classifier multiple times after every mutual information maximization step. These multiple queries could occur either at the same point x – the maximizer of the mutual information –, or uniformly over all points that are within a range of, say, 90% of the maximum. Thereby, one may lose some query efficiency (more queries needed for a same average information gain), but spare a lot of time via query batching. Since when writing this paper, we were primarily thinking of applications were query efficiency mattered most, we did not include this acceleration trick in our experiments. However, in many other applications, using a bit more queries to save wall-clock time of computation can be the better option.

(2) *reducing the range of priors*: we noticed that the parameters of the sigmoid found by the noisy bin-search procedure become increasingly similar from one PSJ iteration to other. So, instead of re-initializing the priors uniformly over the same intervals I_z and $\log_{10}(I_s)$ at the beginning of every bin-search procedure, we used intervals $\tilde{I}_z$ and $\log_{10}(\tilde{I}_s)$ that were centered on the output $\hat{z}$ and $\log_{10} \hat{s}$ of the previous iteration and whose length we decreased at every iteration. We chose this length to be a fraction $1/k$ of the length of the original intervals I_z and $\log_{10}(I_s)$, with $k = 1 \ldots 10$ for iterations 1 to 10, and $k = 10$ for iterations ≥ 10.

We tested these acceleration tricks on 20 images with results shown in Figs. 14 to 17. Figure 14 confirms that both tricks accelerate the bin-search steps and can be combined for further acceleration. Figure 15 confirms that trick (1), despite decreasing wall-clock time, increases the amount of bin-search queries, and that trick (2) decreases it (with tighter priors we need less queries to determine the parameters up to a given precision). Figures 16 and 17 show that, despite accelerating the attack (column c and Fig. 14), the tricks do not significantly affect PSJ's output quality (column a) and number of median model queries (column c).

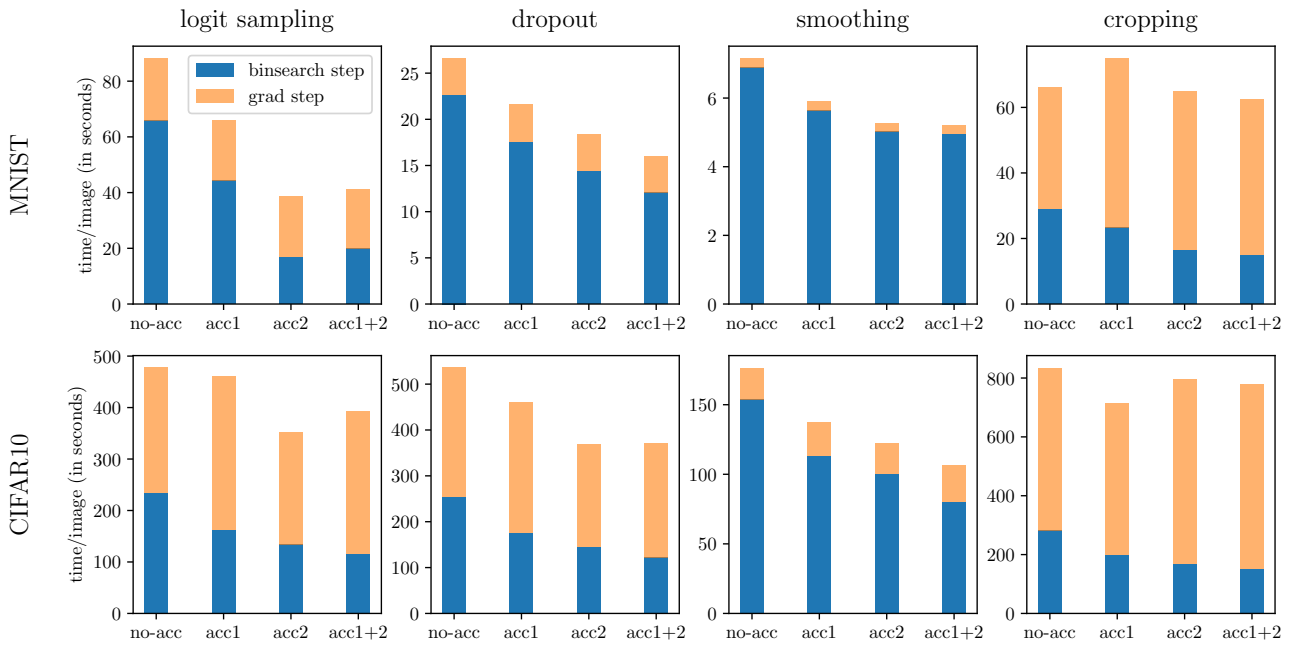

Figure 14: Median time (over 20 images) spent for bin-search and gradient estimation with and without acceleration tricks. As expected, the acceleration tricks (1) and (2) do help accelerating the binary search step. Note that the acceleration tricks should not affect the gradient estimation step. For the multiple sampling trick (1), we made 5 queries per information maximization step.

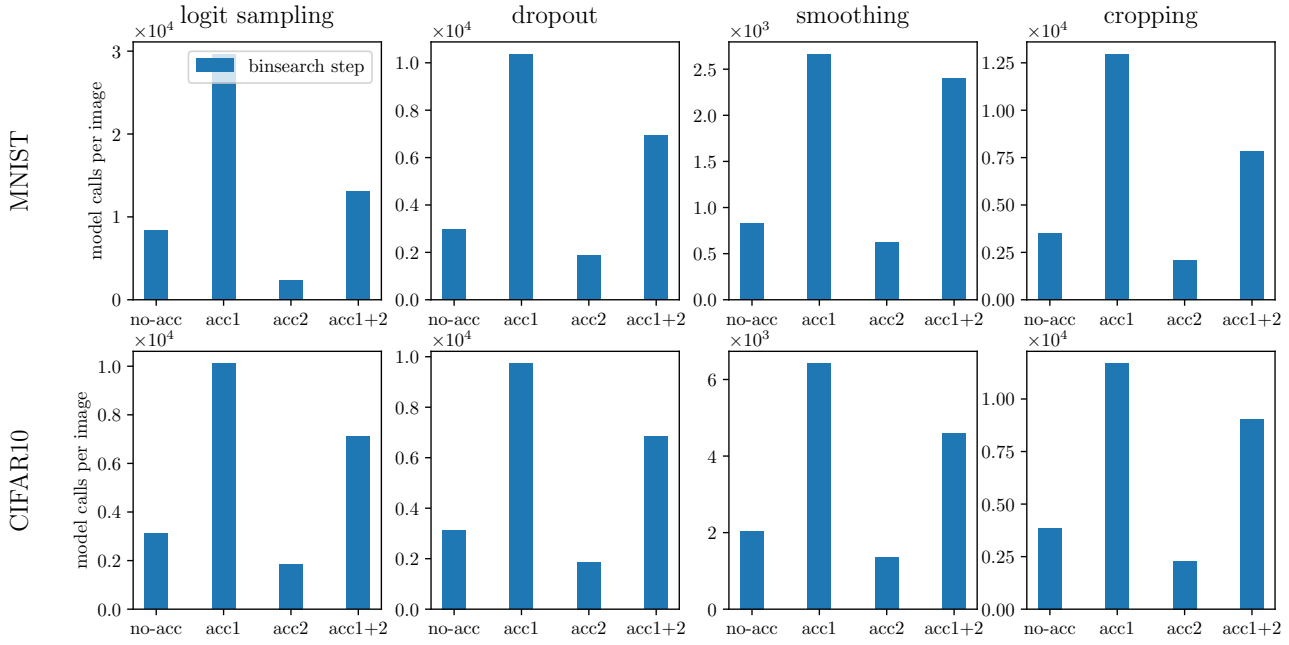

Figure 15: Median amount (over 20 images) of total bin-search queries with and without acceleration tricks. Acceleration trick (1) (multiple queries after each information maximization) increases the number of queries, since the expected amount of information conveyed by each query is reduced. Acceleration trick (2) (tightening the priors on the sigmoid's parameters) reduce the amount of bin-search queries. Note however that the number of noisy bin-search queries is 2 orders of magnitudes smaller than the number of queries for gradient estimations. So the variations observed here have almost affect the overall amount of queries per attack. For the multiple sampling trick (1), we made 5 queries per information maximization step.

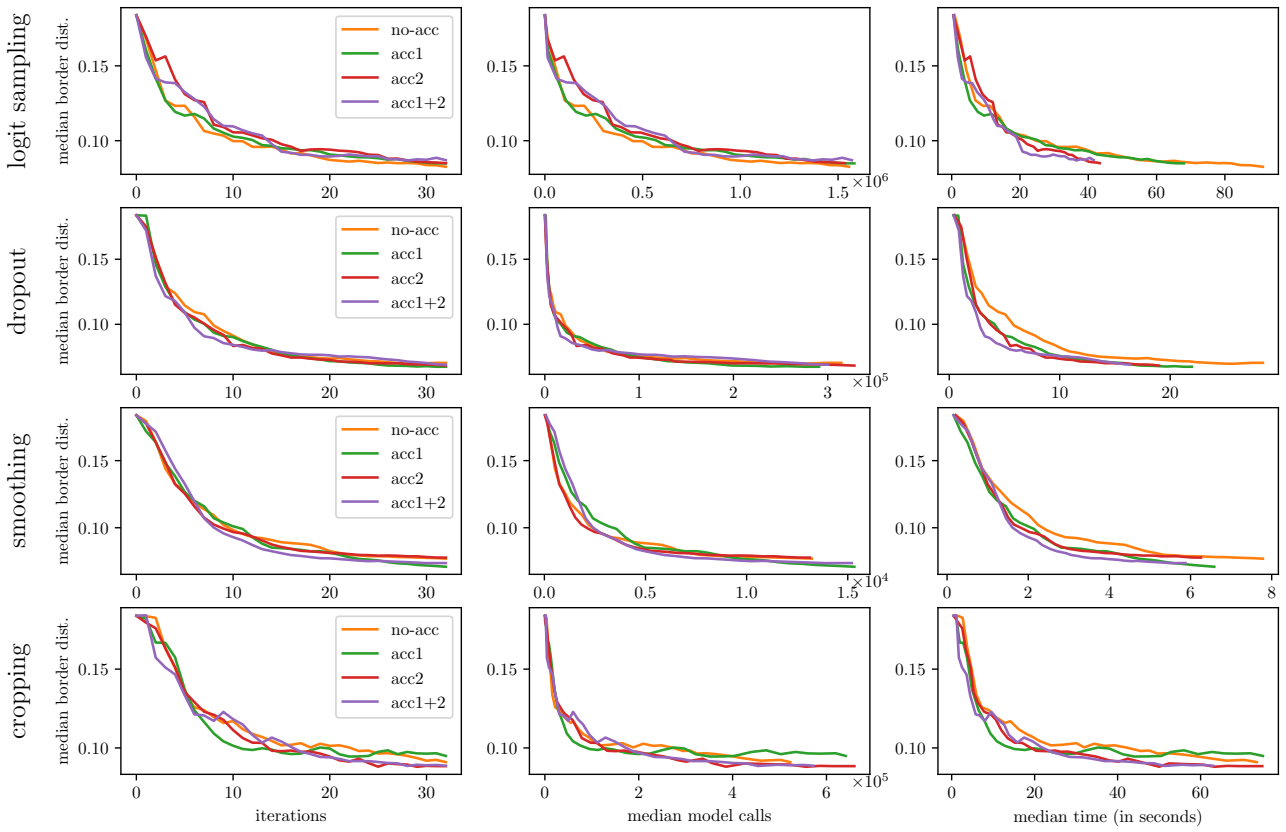

Figure 16: **MNIST**: Effects of acceleration tricks. Column (a) shows that acceleration has no significant effect on the algorithm's output (i.e., on the median border distance after every iteration). Column (b) shows that, as expected, the algorithm runs faster (in wall-clock time), with a similar output performance. Column (c) shows that the median number of model calls is not significantly affected by the acceleration.

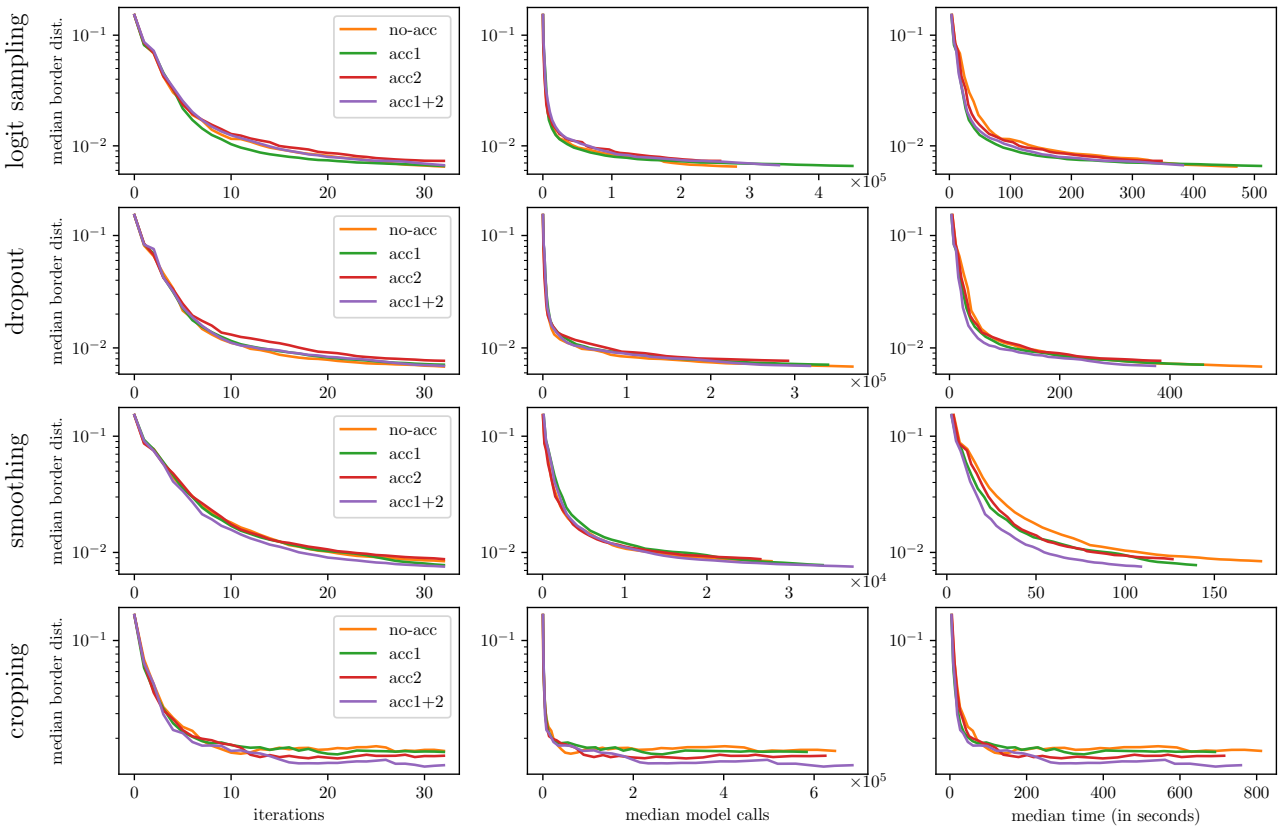

Figure 17: **CIFAR10**: Same comments as for Fig. 16.