# Neural Adversarial Attacks with Random Noises

Hatem Hajri, Manon Cesaire, Lucas Schott, Sylvain Lamprier, Patrick Gallinari

International Journal on Artificial Intelligence Tools


# Neural Adversarial Attacks with Random Noises

Hatem Hajri, Manon Césaire, Lucas Schott
*Institute of Research and Technology SystemX*
*Palaiseau, France*
*first name.second name@irt-systemx.fr*

Sylvain Lamprier, Patrick Gallinari
*Sorbonne University*
*Paris, France*
*first name.second name@isir.upmc.fr*

In this paper, we present an approach which relies on the use of random noises to generate adversarial examples of deep neural network classifiers. We argue that existing deterministic attacks, which perform by sequentially applying maximal perturbations on selected components of the input, fail at reaching accurate adversarial examples on real-world large scale datasets. By exploiting a simple Taylor expansion of the expected output probability under the noise perturbation, we introduce noise-based sparse (or $L_0$) targeted and untargeted attacks. Our proposed method, called Voting Folded Gaussian Attack (VFGA), achieves significantly better $L_0$ scores than state-of-the-art $L_0$ attacks (such as SparseFool and Sparse-RS) while being faster on both CIFAR-10 and ImageNet. Moreover, we show that VFGA is also applicable as an $L_\infty$ attack and outperforms the state-of-the-art projected gradient attack (PGD) method.

*Keywords*: deep neural network classifiers, adversarial attacks, adversarial examples.

## 1. Introduction

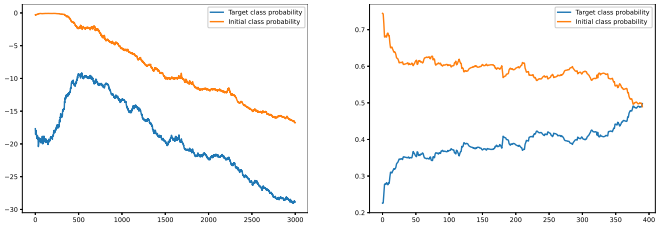
The study of adversarial examples in machine learning has been pushed by the desire to boost the performance of deep neural networks and their use in critical applications. Most of the works related to adversarial examples have been around three categories of attacks according to the minimised distance between original and adversarial samples: $L_2$ (squared error) [28,21,4], $L_\infty$ (max-norm) [11,19,15] and $L_0$ (or sparse) [22,4] attacks (minimising the number of modified components).

Given a neural network classifier (NNC) $F : \mathbb{R}^n \to \mathbb{R}^p$, the predicted label for an input $x$ is $\text{label}(x) = argmax_k F_k(x)$, where $F_1, \cdots, F_p$ are the class probabilities of $F$. We recall that an adversarial example to $x$ is an item $x^*$ such that $\text{label}(x^*) \neq \text{label}(x)$ (untargeted attack), or such that $\text{label}(x^*) = c$, with $c \neq \text{label}(x)$ a specific class (targeted attack). In this paper, we mostly discuss $L_0$ (or sparse) attacks and $L_\infty$ ones.

Sparse perturbations can be encountered in many situations and have been motivated in many works [4,22,20,2,1,5,8,7,10]. For instance, they could correspond to some

raindrops on traffic signs that are sufficient to fool an autonomous driver [20]. Understanding these special alterations is important to mitigate their effects and take a step forward trusting neural networks in real-life.

In this work, we aim at presenting a general probabilistic approach to generate $L_0$ attacks that rely on random noises; called throughout the paper stochastic sparse adversarial attacks (SSAA). We argue that existing deterministic attacks named XSMA (JSMA [22], WJSMA, TJSMA [5]), which perform by sequentially applying maximal perturbations on selected components of the input, fail at reaching accurate adversarial examples on real-world large scale datasets.




(a) Targeted XSMA (base-10 log scale)

(b) Targeted VFGA10

Fig. 1: Plots of the initial and targeted class probabilities for a one pixel version of XSMA on the left failing to converge along more than 3,000 iterations and our VFGA10 method converging efficiently in **less than 400 iterations** on the right.

Figure 1 (left) illustrates this failure on the ImageNet dataset [25] for a one-component version of the targeted XSMA which does not succeed to affect the initial probability of the input on the Inception-v3 network [27]. On the other hand, working with more than one component at a time, while more accurate, does not scale at all on datasets as ImageNet [4]. An alternative would be to repeatedly apply very small perturbations on components, but this would be at the cost of efficiency. These issues, well-known for the XSMA, are completely solved by SSAA through sampling from noise near the most salient component while keeping the same general iterative process followed by the XSMA. Experimental results on large scale datasets, as depicted on the same example as the failure case of XSMA in Figure 1 (on the right), show that our SSAA approaches (denoted VFGA10) succeed at efficiently producing accurate attacks in most cases.

The rest of the paper is organised as follows:

In Section 2, we introduce our best methods of SSAA called scalable SSAA. These attacks are gradient-based and scale efficiently to large datasets.

In Sections 3 and 4, we experiment scalable SSAA on deep NNC on CIFAR-10 [14] and ImageNet [25] and compare their performances with the-state-of-the-art methods SparseFool [20], GreedyFool [10], Brendel & Bethge $L_0$ attack (B&B) [3] and

Sparse-RS [7]. Experimental results show that our method called VFGA outperforms SparseFool while being faster. Furthermore, VFGA achieves better $L_0$ scores than Sparse-RS in most cases and in all cases on ImageNet; when both attacks are fully successful. Moreover, VFGA is significantly less complex than B&B and GreedyFool and achieves competitive results in many cases.

In Section 5, we introduce more examples of SSAA. These attacks are Hessian-based and so not scalable to ImageNet. However, they can be used as efficient methods on small datasets. We illustrate their usefulness through some experiments on MNIST [17].

In Section 6, we give two theoretical results on the expected logits of a ReLU-NNC under a one component perturbation by a Gaussian or Folded Gaussian noise. These results shed some light on the interpretation of the effect of a single pixel perturbation by Gaussian or folded Gaussian noise on the decision of the neural network.

In Section 7, we introduce $L_\infty$ versions of SSAA and compare the $L_\infty$ version of VFGA with the well-known projected gradient descent attack (PGD) baseline [19] on CIFAR-10. Results show the superiority of VFGA over PGD.

Finally, a nice property of SSAA is that they can be combined together in a single attack. While this combination is done only through VFGA (which combines two noises) in the present paper, we expect that the combination of more noises can significantly boost the performance of SSAA in both the $L_0$ and $L_\infty$ version. This point will be discussed in the last Section 8.

## 2. Scalable stochastic sparse adversarial attacks (SSSAA)

In what follows, we introduce SSSAA by sampling from particular random noises on selected components of the input. To simplify, we mainly discuss targeted attacks and then deduce untargeted ones by applying slight modifications. The aim herein is to iteratively identify the best component to perturb and the best move for this component until the target label becomes the most probable for the NNC.

First, consider a Gaussian noise $X_\theta \sim \mathcal{N}(0,\theta)$ and denote by $(e_1,\cdots,e_n)$ the basis of $\mathbb{R}^n$. Any $c$-targeted probability expectation of the perturbed input $x+X_\theta e_i$ can be expanded as follows:

$$\mathbb{E}[F_c(x+X_\theta e_i)] = F_c(x) + \theta(\mathcal{G}_i F_c)(x) + ... \tag{1}$$

where $\mathcal{G}_i F_c = \frac{1}{2}\frac{\partial^2 F_c}{\partial x_i^2}$ is the infinitesimal generator of $X_\theta$ seen as a diffusion.

When taking the folded Gaussian noise $X_\theta \sim |\mathcal{N}(0,\theta)|$, this expansion becomes:

$$\mathbb{E}[F_c(x+X_\theta e_i)] = F_c(x) + \sqrt{\frac{2\theta}{\pi}}\frac{\partial F_c}{\partial x_i}(x) + \theta(\mathcal{G}_i F_c)(x) + ... \tag{2}$$

For any given noise $Y_\theta$, we build our reasoning upon a heuristic which is to look for the input feature $i$ that maximises $\mathbb{E}[F_c(x+Y_\theta e_i)]$. The assumption behind this

heuristic is that searching for the best expectation will allow to discover the best moves according to the distribution of the noise $Y_\theta$.

As a first remark, notice that the Gaussian noise $X_\theta \sim \mathcal{N}(0, \theta)$ is not a good candidate since it requires computing second derivatives of the NNC and moreover the approximations $\mathbb{E}[F_c(x + X_\theta e_i)] \approx F_c(x) + \theta(\mathcal{G}_i F_c)(x)$ and $Var[F_c(x + X_\theta e_i)] \approx \theta \left(\frac{\partial F_c}{\partial x_i}\right)^2 (x)$ are of the same order as $\theta$, indicating that variance should be taken into account in selecting the best components to perturb. For these reasons, we do not discuss the Gaussian noise in this section and postpone its study to Section 5.

On the other hand, considering the folded Gaussian noise, the approximation $\mathbb{E}[F_c(x + X_\theta e_i)] \approx F_c(x) + \sqrt{\frac{2\theta}{\pi}} \frac{\partial F_c}{\partial x_i}(x)$ induces a negligible variance (only terms of $\theta^p$ with $p \geq 1$) in front of the expectation, at least when $|\theta| < 1$ and allows us to only focus on the expected probability of the perturbed input. Note also that the approximation of the expected probability only contains first derivatives w.r.t. to the input component which is a practical advantage of the folded over the pure Gaussian noise.

Taking a folded noise has another important advantage for bounded inputs. Please note that, without loss of generality, we consider inputs bounded in $[0, 1]$ in this paper, as well as the adversarial samples which share the same support domain. In the following, we propose to automatically tune the variance parameter $\theta$ of $X_\theta$ according to the distance of the input $x_i$ to these bounds. Please note that, for a given component $i$, $x_i \neq 0.5$, the possible amplitude of move is not the same in both directions. Considering a Gaussian noise, since symmetric, would be problematic for this $\theta$ tuning. Rather, considering two folded Gaussian noises for each component, one positive (only for component increase) and one negative (only for component decrease) allows better fitted selections.

Next, we present a one-sided, only increasing perturbations, stochastic attack based on the folded Gaussian noise called FGA. Then, we introduce, in the same way another attack that relies on the uniform noise called UA. Finally, we deduce a simple both-sides attack, that considers the best choice between the increasing and decreasing FGA at each selected component, called Voting Folded Gaussian Attack (VFGA).

### 2.1. *Folded Gaussian Attack (FGA)*

For our one-side targeted attack FGA, the most relevant input feature to perturb is thus selected by the rule $i = \underset{j}{\text{argmax}} \sqrt{\theta_j} \frac{\partial F_c}{\partial x_j}$, considering a folded Gaussian noise $|\mathcal{N}(0, \theta_i)|$.

**Choosing the variance $\theta_i$.** Since FGA only considers positive perturbations of the input, fixing the variance $\theta_i$ must consider the upper-bound of the input domain. A quite natural choice could be either $\theta_i = 1 - x_i$ (variance $= 1 - x_i$) or $\sqrt{\theta_i} = 1 - x_i$

(standard deviation $= 1 - x_i$). We choose $\sqrt{\theta_i} = 1 - x_i$ to ensure that a generated perturbation $x_i + \mathcal{N}_i$ to $x_i$ has probability $2/3$ to be inside the interval $[x_i, 1]$ (before clipping to $[0, 1]$) which is a more motivated choice. Our experimental results (not reported in this paper) show that this choice gives slightly more effective attacks than the second one.

After selecting the input feature $i$, our proposal is to simulate $N_S$ samples from $|\mathcal{N}(0, \theta_i)|$ to find an accurate move towards a close adversarial sample. The complete process is depicted in Algorithm 1 introducing the increasing FGA (and the decreasing FGA by analogy).

**Algorithm 1** (Increasing) Folded Gaussian Attack (FGA)

**Inputs:** $x$: input to the NNC of label $l$, $c \neq l$: targeted class, $N_S$: number of samples to generate at each iteration, `maxIter`: maximum number of iterations.

**Output:** $\tilde{x}$: adversarial sample to $x$.

**Initilialise** $\tilde{x} \leftarrow x$, $\Gamma \leftarrow \{1, \cdots, \dim(x)\} \setminus \{i : \tilde{x}_i = 1\}$, `iter` $\leftarrow 0$.

**while** $\Gamma \neq \emptyset$, *label*$(\tilde{x}) \neq c$ *and* `iter` $<$ `maxIter` **do**

  $i_0 = \underset{i \in \Gamma}{\operatorname{argmax}} \, (1 - \tilde{x}_i) \, \dfrac{\partial F_c}{\partial x_i}(\tilde{x})$.

  Generate samples $\left(S^h\right)_{1 \leq h \leq N_S}$ from $|\mathcal{N}(0, \theta_{i_0})|$ where $\sqrt{\theta_{i_0}} := 1 - \tilde{x}_{i_0}$.

  **for** $h \in [\![1, N_S]\!]$ **do**

    Define the input $\tilde{y}^h$ by

$$\begin{cases} \tilde{y}^h_j \leftarrow \mathrm{Clip}_{[0,1]}\left(\tilde{x}_j + S^h\right) & \textbf{if } j = i_0 \\ \tilde{y}^h_j = \tilde{x}_j & \textbf{otherwise.} \end{cases}$$

  Batch compute $F_c\left(\tilde{y}^h \, ; \, h \in [\![1, N_S]\!]\right)$.

  $\tilde{x} \leftarrow \underset{\tilde{y}^h}{\operatorname{argmax}} \, F_c\left(\tilde{y}^h\right)$, $\Gamma \leftarrow \Gamma \setminus \{i_0\}$

  `iter` $\leftarrow$ `iter` $+ 1$.

**return** $\tilde{x}$

**Choosing $N_S$.** The main hyperparameter in Algorithm 1 is the number $N_S$. Given the way it is defined, one can expect that increasing $N_S$ will increase, up to saturation, the effectiveness of the attacks. This may, however, slow down their speeds. Thanks to batch computing, with sufficient memory, Step 8 can be performed at the cost of $N_S = 1$ and (reasonably) augmenting $N_S$ can make Algorithm 1 converge faster as less iterations would be needed. In most of our experiments, we fix this number to $N_S = 10$ but also address some comparisons with $N_S = 20, 100$. We refer to the analysis of the experimental results for more discussions related to this point. Finally, we notice that batch computing used here does not often require a parallel

computing effort by the user as this option is available in standard libraries.

The previous attack only applies perturbations that increase the input. Lowering the input features intensities can be as effective as increasing them. Following the same analogy, we introduce the decreasing FGA attack by taking $\sqrt{\theta_i} = x_i$ rather than $\sqrt{\theta_i} = 1-x_i$ and replacing $|\mathcal{N}(0,\theta_i)|$ with $-|\mathcal{N}(0,\theta_i)|$ in the previous algorithm. Note that FGA and XSMA are one sided attacks but XSMA apply predefined maximal perturbations and FGA explores in real time best perturbations to apply.

### 2.2. *Uniform Attack (UA)*

The UA can be introduced in the same way as FGA by sampling from the uniform noise. It follows the lines of Algorithm 1 (in its untargeted form), but Step 3 is replaced with $i_0 = \underset{i\in\Gamma}{\operatorname{argmin}} (1 - x_i) \dfrac{\partial F_c}{\partial x_i}(x)$ and sampling in Step 4 is done from $\mathcal{U}([0,\theta_i]),\ \theta_i = 1 - x_i$.

### 2.3. *Voting attacks (VA)*

In this section, we propose a two-sided attack, which both considers $\mathbb{E}[F_c(x + |X_{\theta_i^+}|\, e_i)]$ and $\mathbb{E}[F_c(x - |X_{\theta_i^-}|\, e_i)]$ for each feature, with $X_\theta \sim \mathcal{N}(0,\theta)$, $\sqrt{\theta_i^+} = 1 - x_i$ and $\sqrt{\theta_i^-} = x_i$. This method applies increasing and decreasing FGA at each iteration and chooses the most effective moves in both directions and is called voting FGA (VFGA) and is presented in Algorithm 2. Note that, in the same way, by considering the uniform noise it is possible to introduce more examples of voting attacks.

### 2.4. *Untargeted attacks*

The main focus for these attacks is to decrease the class probability of the input until a new class label is found. Only few modifications are required to deduce the untargeted versions of the previous Algorithms: by assuming $c$ is the true label of $x$ and replacing argmax with argmin in Steps 3 and 8 of Algorithm 1 and making similar slight changes in Algorithm 2.

## 3. Experiments on untargeted SSSAA

In this section, we present experiments to highlight the benefits of our untargeted SSSAA. First, we aim to showcase the relevance of FGA in comparison with UA. Second, we aim to compare our methods and more specifically VFGA with relevant state-of-the-art approaches. The codes of the present and next section are available in PyTorch at `https://github.com/hhajri/stochastic-sparse-adv-attacks`.

**Algorithm 2** Voting Folded Gaussian Attack (VFGA)

**Inputs:** as Algorithm 1.

**Output:** as Algorithm 1.

**Initilialise** $\tilde{x} \leftarrow x$, $\Gamma \leftarrow \{1, \cdots, \dim(x)\} \setminus \{i : \tilde{x}_i = 1\}$, $\texttt{iter} \leftarrow 0$.

**while** $\Gamma \neq \emptyset$, *label*$(\tilde{x}) \neq c$ *and* $\texttt{iter} < \texttt{maxIter}$ **do**

$i^+ = \underset{i \in \Gamma}{\operatorname{argmax}} (1 - \tilde{x}_i) \dfrac{\partial F_c}{\partial x_i}(\tilde{x})$, $i^- = \underset{i \in \Gamma}{\operatorname{argmin}} \tilde{x}_i \dfrac{\partial F_c}{\partial x_i}(\tilde{x})$.

Generate samples $\left(S^{+,\,h}\right)_{1 \leq h \leq N_S}$ from $|\mathcal{N}(0,\, \theta_i^+)|$ where $\sqrt{\theta_i^+} := 1 - \tilde{x}_{i^+}$.

Generate samples $\left(S^{-,\,h}\right)_{1 \leq h \leq N_S}$ from $-|\mathcal{N}(0,\, \theta_i^-)|$ where $\sqrt{\theta_i^-} := \tilde{x}_{i^-}$.

**for** $h \in [\![1,\, N_S]\!]$ **do**

Define the input $\tilde{y}^{\,h}$ by

$$\begin{cases} \tilde{y}_j^{\pm,\,h} \leftarrow \mathrm{Clip}_{[0,1]}\left(\tilde{x}_{i^\pm} + S^{\pm,\,h}\right) & \textbf{if } j = i^\pm \\ \tilde{y}_j^{\pm,\,h} = \tilde{x}_j & \textbf{otherwise.} \end{cases}$$

Batch compute $F_c\left(\tilde{y}^{\,\pm,\,h}\,;\, h \in [\![1,\, N_S]\!]\right)$ ($2N_S$ propagations).

$\tilde{x} \leftarrow \underset{\tilde{y}^{\,\pm,\,h}}{\operatorname{argmax}} F_c\left(\tilde{y}^{\,\pm,\,h}\right)$, $\Gamma \leftarrow \Gamma \setminus \{i_0\}$ with $i_0 = i^+$ or $i^-$ according to the best move; $\texttt{iter} \leftarrow \texttt{iter} + 1$.

**return** $\tilde{x}$

In the experiments, we consider two popular computer vision datasets illustrating small and high dimensional data: CIFAR-10 [14] ($32 \times 32 \times 3$ images divided into 10 classes) and ImageNet [25] (ILSVRC2012 dataset containing $299 \times 299 \times 3$ images divided into 1,000 classes). The used neural network classifiers are described in the upcoming paragraphs.

The state-of-the-art attacks considered for comparison in this section are:

**SparseFool [20].** This method is fast and scalable. At each iteration, it applies DeepFool [21] to estimate the minimal adversarial perturbation thanks to a linearization of a classifier. Then, it estimates the boundary point and the normal vector of the decision boundary and finally updates the input features with a linear solver.

**Brendel and Bethge $L_0$ attack (B&B)** [3]. This adversarial attack is gradient-based. It follows the boundary between the space of adversarial and non-adversarial images to find the minimum distance to the clean image. It is powerful and more efficient (but also slower and more complex) than many gradient-based approaches such as SparseFool.

**GreedyFool** [10]**.** This attack is an improvement of SparseFool. It is however more complex than the later as it needs to carefully train a distortion map which is a generative adversarial network GAN [12]. We remark (based on one experiment on ImageNet) that it is less efficient (but also faster and less complex) than B&B.

**Sparse-RS** [7]**.** This attack is fast and achieves high success rate on ImageNet outperforming many white-box attacks such as $\text{PGD}_0$ [8]. It requires fixing the maximum number of pixels to modify which is then fully exploited. In order to generate adversarial examples with minimal $L_0$ perturbations by Sparse-RS, one needs to run this method for several budgets before selecting an optimal budget.

**A notable difference with Sparse-RS.** It should be mentioned that our attacks and Sparse-RS follow different strategies. Indeed, the budget $k$ for Sparse-RS is fixed in the pixel space. For instance, on CIFAR-10 this can go up to $32 \times 32$, and once $k$ is fixed the number of modified pixels in the input space, for Sparse-RS, is near $3 \times k$. Our attacks compute perturbations directly in the input space. All attacks are however $L_0$ in the usual definition and they are compared according to the most commonly used metric which is, up to our knowledge, the $L_0$ distance in the input space.

All the previous attacks are experimented using the original implementations by the authors and following the recommended hyperparameters.

To compare between the different methods, we rely on the following scores: success rate (SR), Mean and Median number of changed pixels, complexity based on the number of model propagation [9] (MP). We prefer MP over the running time per image since it is not dependent on the software used when executing the codes.

**More approaches.** In this paper, we propose fast methods, and thus we only focus on comparisons with similar fast approaches like SparseFool and Sparse-RS. The B&B, although not fast, has been selected as a highly efficient benchmark attack. We omit comparison with Carlini&Wagner $L_0$ [4] and we believe the results would be similar to our comparison with B&B. Also, we omit comparison with CornerSearch [8] because it is less effective than Sparse-RS based on the work [7] and also needs a large computational cost on ImageNet (see results in [8]).

### 3.1. *On CIFAR-10.*

On this dataset, we use the ResNet18 [13] and VGG-19 [26] models. After training with PyTorch, these networks reached 95.55% and 93.87% accuracies respectively. For our attacks UA, FGA and VFGA, the hyperparameter $N_S$ is fixed to $N_S = 10$

(the obtained attacks are denoted UA10, FGA10 and VFGA10). The effect of augmenting $N_S$ is analysed later on in this section. We notice that, otherwise stated, `maxIter` is put to its maximal value. The state-of-the-art approaches outlined before, except GreedyFool, are tested and compared with our methods on the correctly predicted samples among the $10,000$ CIFAR-10 test images. We refrained from comparing with GreedyFool because of the need to train the distortion map network on CIFAR-10 not provided in the code of [10] (this network has been made available for the ImageNet dataset and comparison on this dataset is considered in the next section). For Sparse-RS, several budgets of pixels $k$ (the number of pixels to modify) have been experimented and the smallest budget giving 100% has been selected. On CIFAR-10, we choose the optimal $k$, i.e $k' = k - 1$ does not give full success of the attack. Our intention is to show that under the condition of full success for all attacks (when possible) our VFGA method is overall more advantageous.

| Attacks | SR | Mean | Median | MP |
|---|---|---|---|---|
| ResNet18 | | | | |
| **B&B** | 100 | 8.33 | 8.0 | 1927 |
| **SparseFool** | 99.31 | 36.48 | 9.0 | 520 |
| **Sparse-RS** ($k = 10$) | 100 | 29.79 | 30 | **GS** + 49 |
| **UA10** | 100 | 30.94 | 20.0 | 363 |
| **FGA10** | 100 | 29.70 | 20.0 | 134 |
| **VFGA10** | 100 | **17.03** | **11.0** | **99** |
| VGG-19 | | | | |
| **B&B** | 100 | 5.30 | 6.0 | 1483 |
| **SparseFool** | 97.98 | 67.71 | 8.0 | 686 |
| **Sparse-RS** ($k = 7$) | 100 | 20.82 | 21 | **GS** + 55 |
| **UA10** | 100 | 22.16 | 11.0 | 281 |
| **FGA10** | 100 | 19.67 | 11.0 | 103 |
| **VFGA10** | 100 | **11.40** | **7.0** | **80** |

Table 1: Results on the correctly predicted samples among the $10,000$ test images of CIFAR-10. SR is the success rate of the attack, Mean, Median are the average and median number of modified pixels on successful samples and MP is the number of model propagations. **GS** is a greed-search to find optimal values of $k$ giving full success that took **several hours**. The highlighted results of our VFGA in comparison with the fast methods SparseFool and Sparse-RS are in bold.

**Comments.** The previous results show that the folded Gaussian noise is more

advantageous in attacking than the uniform noise and that combining two folded distributions is useful not only for the SR, Mean and Median but also for the model propagation score. Concerning the comparison with the state-of-the-art methods: VFGA has less advantageous Mean and Median than B&B (near 2 times greater for Mean and the gap is reduced for Median). Nevertheless, it is up to $\frac{1}{20}$ less complex based on the MP score. Second, all of our methods and more particularly VFGA significantly outperform SparseFool. Regarding the comparison with Sparse-RS, we remark that VFGA has notable Mean and Median advantages (up to 3 times fewer) and is also less complex given the number of experiments carried for Sparse-RS to achieve full success (with minimal Mean and Median). Notice also the difficulty to find a good $k$ with full success for Sparse-RS as the optimal value depends on each sample and high values impact the overall performance of this attack. An advantage for us is that this parameter is set automatically and is optimal for each sample.

**A comparison between Sparse-RS and VFGA for different distortions.** As stressed before, we only focus on performances under the condition of full success which is usually reported to summarise the contribution of new methods. If we relax this condition, we remark that for small budgets $k$ when both VFGA and Sparse-RS are not fully successful, Sparse-RS outperforms VFGA in SR but VFGA obtains better Mean and Median which are always near $k$ for Sparse-RS. Starting from a $k$ which approaches full success, VFGA becomes more advantageous in SR, Mean and Median.

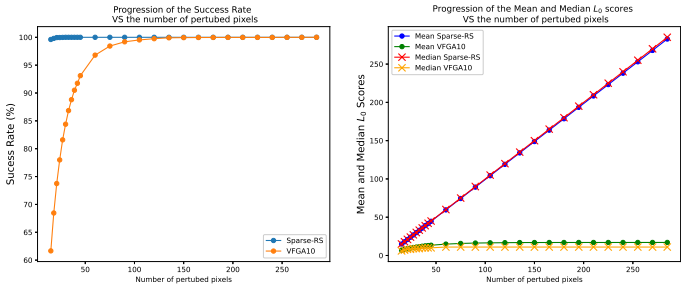



Fig. 2: Results over the number of perturbed pixels for untargeted attacks (**VFGA10 and Sparse-RS**) on CIFAR-10 for the ResNet-18 model. On the left, the success rate SR and on the right, the Mean and Median $L_0$ scores.

We draw in Figure 2 the SR (on the left), and Mean and Median $L_0$ scores (on the right) versus the number of perturbed pixels for VFGA10 and Sparse-RS. When only few pixels are perturbed, Sparse-RS performs better than VFGA10 in SR. Once the number of perturbed pixels exceeds 105, the two SR become very competitive (near and then equal to 100%). The Mean and Median $L_0$ scores for VFGA10 are, however, always better than those by Sparse-RS regardless the num-

ber of pixels which have been modified.

For Sparse-RS, Median is a linear function of the number of perturbed pixels. For the budget of 285 modified pixels, Mean is 282.78. For VFGA10, this budget is the worst $L_0$ score for the 10,000 test images of CIFAR-10 and with this value, Mean is 17.03 which is about 16 times fewer than Sparse-RS. In short, even if for a small number of disturbed pixels, VFGA10 is not able to reach 100% in SR, its Mean and Median are still better than those of Sparse-RS.

**Augmenting $N_S$.** In what follows, we investigate the impact of augmenting $N_S$ on the performances of our attacks by testing UA20, FGA20 and VFGA20, which correspond to $N_S = 20$, on the same data as Table 1.

| Attacks | SR | Mean | Median | MP |
|---|---|---|---|---|
| ResNet18 | | | | |
| **UA20** | 100 | 30.93 | 20.0 | 684 |
| **FGA20** | 100 | 30.23 | 19.0 | 193 |
| **VFGA20** | 100 | 16.76 | 11.0 | 131 |
| VGG-19 | | | | |
| **UA20** | 100 | 20.60 | 10.0 | 457 |
| **FGA20** | 100 | 19.71 | 10.0 | 134 |
| **VFGA20** | 100 | 11.22 | 7.0 | 113 |

Table 2: Comparison between our attacks for $N_S = 20$.

We remark that SR, Mean and Median are slightly improved but based on the MP score, the attacks become more complex. This illustrates the fact that increasing so much $N_S$ may not significantly improve the attacks but on the other hand it may slow down them. Also, we observe that despite augmenting $N_S$ from 10 to 20, the uniform attack cannot beat FGA10. This is quite remarkable since for the uniform distribution the $N_S$ generated samples are different and fall inside the domain of the input features while for the folded Gaussian distribution, due to clipping, several samples are likely to be clipped at the minimal and maximal bounds. Augmenting $N_S$ also increases more quickly MP for the uniform noise.

### 3.2. *On ImageNet.*

In this section, we test the ability of the previously tested attacks and additionally GreedyFool to generate adversarial examples at large scale by considering models on ImageNet. Two pre-trained networks provided by PyTorch are considered for testing: Inception-v3 [27] and VGG-16 [26] whose accuracies are respectively 77.45%

and 71.59%. Inputs are of size 299×299×3 and 224×224×3 for the first and second model respectively.

Again, we consider B&B as a benchmark of a highly successful attack. GreedyFool requires training a GAN network on ImageNet but once carefully done it is highly successful. We recover the GAN model from the available code of [10] which we complement by adding the computation of the MP score for this attack. For Sparse-RS, we again fix our objective to compare with this attack when 100% SR is achieved. This requires launching several experiments for different values of $k$ on the whole considered set of images in order to obtain a **near-optimal** budget value. By this, we mean a value $k$ giving 100% SR ; there exists $k' < k$ and the performances of VFGA in Mean and Median are better than those obtained by Sparse-RS with budget $k'$. This implies in particular that VFGA gives better results than Sparse-RS when tested with the optimal value of $k$. To get an idea of the difference between our results and those by Sparse-RS, we always report the results for $k'$ and $k$ by Sparse-RS (in this section and next one).

The obtained results for the different attacks are reported in Table 3 and commented after.

**Comments.** First, B&B achieves the best SR, Mean and Median scores. GreedyFool comes after but with the cost of training a GAN model on ImageNet. The complexity comparison between these two attacks is difficult to address and we only claim that both methods are significantly more complex than our approach (GreedyFool is complex to reproduce on new datasets). Despite this fact, we observe that on VGG-16 VFGA has a gap of Mean and Median of less than 12 pixels which is relatively small. Among the methods shown in the previous table, our attacks and SparseFool are the fastest under the full success condition and when minimising at the same time Mean and Median. Our attacks, obtain, however, overall better performances than SparseFool according to all metrics. Specifically, VFGA always significantly outperforms SparseFool. Moreover, despite the fact that we select near-optimal values of $k$ for Sparse-RS, VFGA is still more advantageous regarding Mean and Median and also faster if the complexity of finding $k$ is added. Finally, we notice that after finding a good $k$ giving full success for Sparse-RS, this attack can not generate relevant adversarial examples with minimal $L_0$ distance 1, while due to the flexibility of our attack, several such examples can be generated. This is a further advantage of our attack.

## 4. Experiments on targeted SSSAA

Targeted attacks are more challenging than untargeted ones. The objective of this section is to compare (targeted) VFGA, our selected method, with (targeted) Sparse-RS as a fast attack outperforming several state-of-the-art methods [7]. We

| Attacks | SR | Mean | Median | MP |
|---|---|---|---|---|
| Inception-v3 | | | | |
| **B&B** | 100 | 43.96 | 37.0 | 5602 |
| **GreedyFool** | 100 | 86.09 | 79.0 | **GAN** + 617 |
| **SparseFool** | 100 | 348.16 | 167.5 | 2531 |
| **Sparse-RS** ($k' = 90$) | 99.62 | 267.13 | 270.0 | **GS** + 341 |
| **Sparse-RS** ($k = 100$) | 100 | 297.12 | 300.0 | **GS** + 358 |
| **UA10** | 100 | 335.19 | 101.0 | 3042 |
| **FGA10** | 100 | 323.27 | 102.0 | 744 |
| **VFGA10** | 100 | **198.25** | **64.0** | **1133** |
| VGG-16 | | | | |
| **B&B** | 100 | 39.24 | 25.0 | 3416 |
| **GreedyFool** | 100 | 66.18 | 31.0 | **GAN** + 589 |
| **SparseFool** | 100 | 216.21 | 164.0 | 1460 |
| **Sparse-RS** ($k' = 60$) | 99.78 | 179.01 | 180.0 | **GS** + 240 |
| **Sparse-RS** ($k = 70$) | 100 | 204.59 | 210.0 | **GS** + 246 |
| **UA10** | 100 | 150.04 | 85.0 | 2122 |
| **FGA10** | 100 | 140.15 | 82.0 | 986 |
| **VFGA10** | 100 | **77.85** | **43.0** | **709** |

Table 3: Results on the firstly $6,000$ correctly predicted validation images of ImageNet. **GS** is a greed-search to find near-optimal values of $k$ that took **several days**. **GAN** is a generative network trained on ImageNet. Our results in comparison with the fast methods SparseFool and Sparse-RS when fully successful are highlighted in bold.

recall that SparseFool is not efficient as a targeted attack. We do not report results by FGA and UA but claim that FGA is still more relevant than UA and only omit to address a similar comparison as before. We do not report results by B&B and GreedyFool as targeted attacks because of the need of the distortion map on CIFAR-10 for GreedyFool, the non ability to reproduce B&B in the targeted mode and moreover since, we consider that these approaches are complex to reproduce on new datasets. Thus, we only focus on the comparison with Sparse-RS and defend our approach as an efficient fast method. Our main conclusion in this paragraph is that, for ImageNet which is more challenging, VFGA is still more relevant than Sparse-RS regarding the same previous metrics when both attacks are fully successful and despite the fact that a near-optimal value of $k$ is selected for Sparse-RS. On CIFAR-10, we conclude on the basis of our experiments that Sparse-RS is more

advantageous in Mean and Median.

We consider the same network models as before. To simplify the experimentation, we do not consider all possible target labels but, for each test dataset, we generate a list of random labels which were fixed once for all. Each input image is then attacked to have one desired label. For Sparse-RS, we again select a near-optimal $k$ in all experiments. This task took several hours on CIFAR-10 and several days on ImageNet. Tables 4 and 5 show the results obtained on CIFAR-10 and ImageNet (in Table 4 VFGA100 is simply VFGA with $N_S = 100$).

| Attacks | SR | Mean | Median | MP |
|---|---|---|---|---|
| ResNet18 | | | | |
| **Sparse-RS** ($k = 30$) | 100 | **89.43** | **90.0** | **GS** + 1678 |
| **VFGA10** | 100 | 641.49 | 174.0 | **13427** |
| **VFGA100** | 100 | 154.43 | 105.0 | **20619** |
| VGG-19 | | | | |
| **Sparse-RS** ($k = 25$) | 100 | **73.17** | **75.0** | **GS** + 1123 |
| **VFGA10** | 100 | 551.27 | 150.0 | **12213** |
| **VFGA100** | 100 | 174.93 | 97.0 | **21417** |

Table 4: Results on the correctly-predicted test images of CIFAR-10. **GS** is a greed-search to find near-optimal values of $k$ that took **several hours**. The results for $k'$ are not reported since Sparse-RS is here more advantageous in full success.

First, we notice that Sparse-RS obtains better Mean and Median than VFGA10. Increasing $N_S$ from 10 to 100 improves considerably VFGA but our results are still less better than Sparse-RS. Given the time needed to find the near-optimal values $k$, we claim that our attacks are still overall much faster than Sparse-RS to obtain full success with optimal Mean and Median.

Our interpretation of Table 5 is overall similar to that of Table 3. When attacking the Inception-v3 model, which is more challenging, VFGA10 outperforms Sparse-RS regarding all scores. When attacking VGG-16, Sparse-RS only takes a slight advantage of Mean. As for untargeted attacks, a notable advantage of our methods is the flexibility of the number of modifiable pixels allowing us to generate adversarial examples with minimum $L_0$ distance while being 100% successful on all samples.

| Attacks | SR | Mean | Median | MP |
|---|---|---|---|---|
| Inception-v3 | | | | |
| **Sparse-RS** ($k' = 950$) | 99.87 | 2671.19 | 2850.0 | **GS** + 6591 |
| **Sparse-RS** ($k = 1000$) | 100 | 2898.72 | 3000.0 | **GS** + 6899 |
| **VFGA10** | 100 | **2148.53** | **1843.45** | **21616** |
| VGG-16 | | | | |
| **Sparse-RS** ($k' = 450$) | 99.91 | 1278.45 | 1350.0 | **GS** + 6963 |
| **Sparse-RS** ($k = 500$) | 100 | **1398.56** | 1500.0 | **GS** + 7003 |
| **VFGA10** | 100 | 1436.02 | **1057.38** | **14223** |

Table 5: Results obtained on the 5,000 firstly correctly-predicted validation images of ImageNet. **GS** is a greed-search to find near-optimal values of $k$ that took **several days**.

## 5. Less scalable SSAA

In this section, we focus on sparse targeted and untargeted perturbations by a distribution of the form $\mathcal{N}(a\theta, b^2\theta) = a\theta + b\mathcal{N}(0, \theta)$ where $\theta$ is a parameter to be tuned as before using the bounds of the input domain and $a, b$ are scaling parameters which determine the contribution of the deterministic component $\theta$ and the pure Gaussian one $\mathcal{N}(0, \theta)$. The resulting attack is denoted GA(a,b). Note that the multidimensional Gaussian noise has been used to generate effective adversarial $L_2$ and $L_\infty$ attacks in [18]. Our attack can be thought of as a sparse version of [18] since it follows a similar methodology based on the minimisation of the expected probability.

If $X_\theta^{a,b} \sim \mathcal{N}(a\theta, b^2\theta)$, then

$$\mathbb{E}[F_c(x + X_\theta^{a,b} e_i)] = F_c(x) + \theta(\mathcal{A}^{a,b} F_c)_{c,i}(x) + o(\theta) \tag{3}$$

where $(\mathcal{A}^{a,b} F_c)_{c,i}(x) := a\frac{\partial p_c}{\partial x_i}(x) + \frac{b^2}{2}\frac{\partial^2 F_c}{\partial^2 x_i}(x)$.

Before explaining the minor changes to introduce G(a,b), let us remark that our GA requires computing the diagonal of the Hessian of $F_c$ with respect to the input coordinates which is costly on large datasets. For this reason, we make the following assumption under which these calculations become more feasible (at least on small datasets):

$$\textit{All hidden layers activation functions of the NNC are ReLU} \tag{4}$$

Note that state-of-the-art NNC make use of ReLU and so (4) is not a limitation as such. Under (4), the Hessian diagonal of the NNC is a simple function of the NNC Jacobian as shown here.

**Fast Hessians diagonals computations for ReLU activations.** Call $L_i$ the logits of the NNC. Since each $L_t$ is the composition of affine and ReLU functions which both have null second derivatives, by a simple recurrence argument $\frac{\partial^2 L_t(x)}{\partial x_i} = 0$. Using this fact and deriving twice the identity $F_t(x) \times \sum_{j=1}^{K} e^{G_j(x)} = e^{G_t(x)}$, we deduce the main relations:

- $\frac{\partial F_t}{\partial x_i} = F_t(\frac{\partial G_t}{\partial x_i} - \sum_j F_j \frac{\partial G_j}{\partial x_i})$
- $\frac{\partial^2 F_t}{\partial^2 x_i} = \frac{\partial F_t}{\partial x_i}(\frac{\partial G_t}{\partial x_i} - \sum_j F_j \frac{\partial G_j}{\partial x_i}) - F_t \times \sum_j \frac{\partial F_j}{\partial x_i}\frac{\partial G_j}{\partial x_i}$

Experimentally, this implementation gives a very accurate approximation of the Hessian diagonal calculated with PyTorch or Tensorflow.

The (targeted) attack G(a,b) follows Algorithm 1; where Step 3 should be replaced with $i_0 = \underset{i \in \Gamma}{\text{argmax}}\, \mathcal{A}^{a,b} F_c(\tilde{x})$. For Step 4, sampling is done from $\mathcal{N}(a\theta, b^2\theta_{i_0})$ where $\theta_{i_0}$ is to be set by the user. Next, we only experiment G(0,1) (the purely Gaussian case) and G(1,1) (the mixed balanced deterministic-Gaussian case). The parameter $\theta_{i_0}$ for these attacks is fixed as follows:

- $\theta_{i_0} = \max(\tilde{x}_{i_0}, 1 - \tilde{x}_{i_0})$ for G(0,1).
- $\theta_{i_0} = \frac{1-\tilde{x}_{i_0}}{2}$ for G(1,1).

As a complement, note that the choice $(a, b) = (1, 0)$ is the one which has led to the XSMA attacks.

In the rest of this section, we compare the untargeted GA(0,1) and GA(1,1) with UA and FGA in attacking LeNet-5 model on MNIST [16]. We use the same model proposed in [5] which has 99.98% accuracy on the training dataset and 99.49% accuracy on the test set. We report in Table 7 the results on 1, 000 images.

| Attacks | SR | Mean | Median |
|---|---|---|---|
| **UA** | 100% | 19.53 | 13.0 |
| **FGA** | 100% | 17.69 | 13.0 |
| **GA(0,1)** | 100% | 22.45 | 16.0 |
| **GA(1,1)** | 100% | 17.68 | 12.0 |

Table 6: Results obtained on 1, 000 images of the MNIST test set.

The results of this table can be ranked as follows (a<b means b outperforms a) GA(0,1)<UA<GA(1,1)<FGA. First, they confirm the overall superiority of FGA over UA yet on the MNIST dataset. GA(0,1) has overall less better results than the other attacks which is somewhat expected since it has been remarked in [22] that, on MNIST, increasing the input features works better than decreasing them and GA has less chance to increase the input features than the other attacks.

While the previous results could be explained by the nature of the MNIST dataset, since increasing the pixel values is known to give better results [22], we do not expect similar behaviours on other datasets where there is no privileged direction for the attacker. Our main claim is that, independently of each attack performance, combining all of them (in the same way as VFGA) and possibly varying the intensity $\theta$ can lead to more efficient $L_0$ attacks. Indeed, experimentally, we observe many successful adversarial examples which modify a single or very few pixels by each method of Table 7 whereas, at the same time, the other methods fail in doing so. Two illustrations are given through Figure 3.

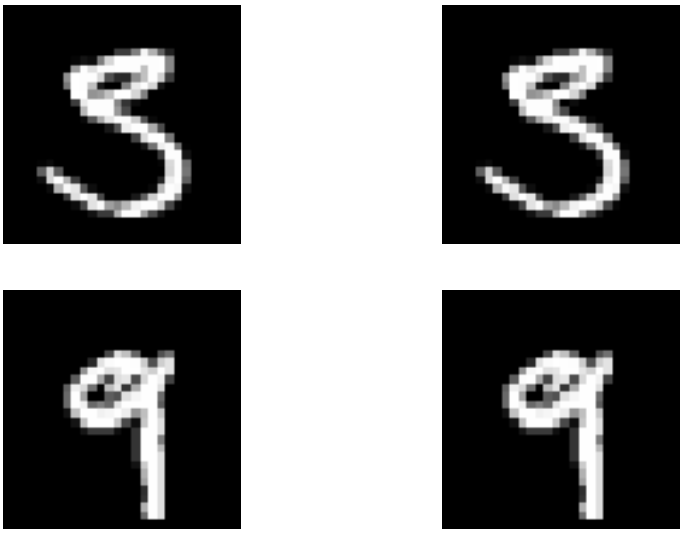


Fig. 3: Top: left-right: an input of label 3 and its perturbation by GA(0,1) (interpreted as 5) by modifying one pixel. The $L_0$ scores for FGA, GA(1,1) and UA are respectively: 2, 5, 2. Down: left-right: an input of label 9 and its perturbation by GA(1,1) (interpreted as 7) while modifying one pixel. The $L_0$ scores for GA(0,1), FGA and UA are respectively: 20, 6, 8.

## 6. A fine analysis of one-pixel perturbations by the Gaussian and folded Gaussian noise for ReLU-networks

In this section, we want to interpret the behaviour of a ReLU NNC under one pixel perturbation by a Gaussian or Folded Gaussian distribution. We focus on the output logits $(G_i)_i$ as they fully determine the decision of the NNC.

The main result is the next Theorem which gives a decomposition of the expected logit $\mathbb{E}[G_c(x + X_\theta e_i)]$, where $X_\theta \sim \mathcal{N}(0, \theta)$ or $X_\theta \sim |\mathcal{N}(0, \theta)|$.

**Theorem 6.1.** Denote by $(G_i)_{1\leq i\leq p}$ the output logits of a ReLU-neural network classifier. Then for every $c \in [1, p]$, we have

(i) $\mathbb{E}[G_c(x + \mathcal{N}(0, \theta)e_i)] = G_c(x) + \frac{1}{2}\sum_{j\in\mathcal{D}} \Gamma_\theta^j(\mathrm{Gr}_{j+} - \mathrm{Gr}_{j-})$
(ii) $\mathbb{E}[G_c(x + |\mathcal{N}(0, \theta)|e_i)] = G_c(x) + \Gamma_\theta \frac{\partial G_c}{\partial x_i}(x + 0^+ e_i) + \frac{1}{2}\sum_{j\in\mathcal{D}^+} \Gamma_\theta^{j,+}(\mathrm{Gr}_{j+} - \mathrm{Gr}_{j-})$

where $\mathrm{Gr}_u = \partial_u G_c(x + ue_i)$ and $\mathcal{D}$ is the set of discontinuities (or jumps) of Gr, $\Gamma_\theta^j = \mathbb{E}[|\mathcal{N}(0, \theta) - j|] - |j|$; $\mathcal{D}^+$ is the set of jumps of $\mathrm{Gr}_u, u > 0$ in $[0, \theta]$, $\Gamma_\theta = \sqrt{\frac{2\theta}{\pi}}$ and $\Gamma_\theta^{j,+} = \mathbb{E}[||\mathcal{N}(0, \theta)| - |j||] - |j|$, $\frac{\partial G_c}{\partial x_i}(x + 0^+ e_i) = \lim_{h\to 0+} \frac{\partial G_c}{\partial x_i}(x + he_i)$.

**Proof.** Let us give the main steps to prove (i). By applying Itô-Tanaka formula (see [23]) for the function $g(u) = G_c(x + ue_i)$ (which can be written as the difference of two convex functions and so satisfies the assumption of the formula), we have

$$G_c(x + W_\theta e_i) = G_c(x) + M_\theta + \frac{1}{2}\int L_\theta^a T_g^{''}(da)$$

where $M_\theta$ is a martingale; in particular $\mathbb{E}[M_\theta] = 0$; $L_\theta^a$ is the local time of $W$ at level $a$ and time $\theta$ and $T_g''$ is the second derivative in the sense of distributions of $g$. Next, we use the decomposition of distributions:

$$(T_g)'' = T_{g''} + \sum_{a\in\mathcal{D}'_g} (g'(a+) - g'(a-))\delta_a$$

where $\mathcal{D}'_g$ is the set of discontinuities of $g'$. Note that $T_{g''} = 0$ since $g$ is a piecewise constant function. Thus, with the notation of the theorem, we have

$$\mathbb{E}[G_c(x + W_\theta e_i)] = G_c(x) + \frac{1}{2}\sum_{j\in\mathcal{D}} \Gamma_\theta^j(\mathrm{Gr}_{j+} - \mathrm{Gr}_{j-})$$

where $\Gamma_\theta^j = \mathbb{E}[L_\theta^j]$ which also coincides with the expression given in the theorem. (ii) can be proved in a similar way. □

Let us give some remarks regarding the previous theorem:

- In the particular case of one hidden layer, Theorem 6.1 (i) allows to retrieve the expression found in Theorem 1 in [2]. In this case, the set $\mathcal{D}$ is reduced to one point and there is only one jump of Gr. The expression given here is much more interpretable than [2] and general for any ReLu-based network. Note also that the number of points contained in $\mathcal{D}$ is bounded by the number of hidden layers.

- The constants $\Gamma_\theta^j, \Gamma_\theta, \Gamma_\theta^{j,+}$ in Theorem 6.1 are easy and fast to approximate with high precision. Thus the main complexity in the expansions given above is the computation of the jumps of Gr. Once these jumps are computed efficiently, they can be used once for all to approximate all $\mathbb{E}[G_c(x + X_\theta e_i)]$, $X_\theta \sim \mathcal{N}(0, \theta)$ or $|\mathcal{N}(0, \theta)|$ for every possible $\theta$. In short, we claim that Theorem 1 reduces the computation of $\mathbb{E}[G_c(x + X_\theta e_i)]$ for all $\theta$ to the computation of the jumps of the trajectory of the gradient of $G_c(x)$ with respect to the $i$-th coordinate.

In some situations, when simple Gaussian sparse attacks or also our attacks such as GA(0,1) and FGA are successful in one iteration, the user or designer of the NNC would be interested in understanding such a failure. The main tool given in this section suggests to compare the quantities given in front of the expectations in Theorem 6.1 for different $c$, which as seen before, depends on the jumps of the gradient paths of the logits according to the chosen component.

## 7. $L_\infty$ bounded attacks

In this section, we focus on $L_\infty$ attacks. This category seems to be more studied in the literature than $L_0$ and $L_2$ ones. One reason is that $L_\infty$ attacks have reached the-state-of-the-art performances in Adversarial Training [19]. Indeed, [19] introduced a strong $L_\infty$ baseline, the PGD attack, to evaluate NNC and adversarially train them. Several improvements of PGD have been proposed but they all either follow the main idea of PGD or combine many PGD varieties at once (see [6]).

Our attacks presented previously can be applied as $L_\infty$ ones by slight modifications. Algorithm 1 below introduces the $L_\infty$ $\varepsilon$-bounded version of VFGA where the main idea is to sample perturbations from $\mathcal{N}(0, \varepsilon)$.
Notice that $N_S$ is the only hyperparameter for the attack. Its interpretation is almost the same as before. Our main claim is that, despite the iterative scheme of VFGA which makes the attack slow in comparison with PGD, this attack can outperform PGD. We propose an experiment to support this by attacking the previous CIFAR-10 Resnet model on 1000 correctly predicted input test images. The applied attacks to this model are VFGA, PGD and PGD with random restart denoted PGD-RR [19] for two values of $\varepsilon = 4/255, 2/255$. The results are given in Table 7.

We remark that for $\varepsilon = 4/255$, the performances are very competitive between our attack and the two PGD varieties. However, for for $\varepsilon = 4/255$ the results are significantly better for VFGA.

## 8. Conclusion

This paper introduced noise-based attacks to generate sparse adversarial samples to inputs of deep neural network classifiers. A first advantage of our methods is that they work as both untargeted and targeted attacks. Moreover, they are very simple to put in place and require fixing only one parameter whose interpretation

**Algorithm 3** $L_\infty$ VFGA with intensity $\varepsilon$

**Inputs:** as Algorithm 1.

**Output:** as Algorithm 1.

**Initilialise** $\tilde{x} \leftarrow x$, $\Gamma \leftarrow \{1, \cdots, \dim(x)\} \setminus \{i : \tilde{x}_i = 1\}$, $\texttt{iter} \leftarrow 0$.

**while** $\Gamma \neq \emptyset$, *label*$(\tilde{x}) \neq c$ *and* $\texttt{iter} < \texttt{maxIter}$ **do**

$i^+ = \underset{i \in \Gamma}{\operatorname{argmax}} \dfrac{\partial F_c}{\partial x_i}(\tilde{x})$, $i^- = \underset{i \in \Gamma}{\operatorname{argmin}} \dfrac{\partial F_c}{\partial x_i}(\tilde{x})$.

Generate samples $\left(S^{+,\,h}\right)_{1 \leq h \leq N_S}$ from $\text{Clip}_{[0,\varepsilon]}(|\mathcal{N}(0,\,\varepsilon)|)$;

Generate samples $\left(S^{-,\,h}\right)_{1 \leq h \leq N_S}$ from $-\,\text{Clip}_{[0,\varepsilon]}(|\mathcal{N}(0,\,\varepsilon)|)$;

**for** $h \in [\![1,\, N_S]\!]$ **do**

Define the input $\tilde{y}^{\,h}$ by

$$\begin{cases} \tilde{y}_j^{\pm,\,h} \leftarrow \text{Clip}_{[0,1]}\left(\tilde{x}_{i^\pm} + S^{\pm,\,h}\right) & \textbf{if } j = i^\pm \\ \tilde{y}_j^{\pm,\,h} = \tilde{x}_j & \textbf{otherwise.} \end{cases}$$

Batch compute $F_c\left(\tilde{y}^{\,\pm,\,h}\,;\, h \in [\![1,\, N_S]\!]\right)$ $(2N_S$ propagations).

$\tilde{x} \leftarrow \underset{\tilde{y}^{\,\pm,\,h}}{\operatorname{argmax}} F_c\left(\tilde{y}^{\,\pm,\,h}\right)$, $\Gamma \leftarrow \Gamma \setminus \{i_0\}$ with $i_0 = i^+$ or $i^-$ according to the best move; $\texttt{iter} \leftarrow \texttt{iter} + 1$.

**return** $\tilde{x}$

| Attacks | $\varepsilon$ | SR |
|---|---|---|
| **PGD** | 2/255 | 63.5% |
| **PGD-RR** | 2/255 | 67.5% |
| **VFGA** | 2/255 | **86.4%** |
| **PGD** | 4/255 | 97.6% |
| **PGD-RR** | 4/255 | **98.4%** |
| **VFGA** | 4/255 | 97.9% |

Table 7: Results obtained on the first 1000 correctly classified images of the CIFAR-10 test set.

is intuitive (the bigger the best up to saturation in performance). Our attacks are faster to apply on new models and datasets than existing approaches (SparseFool, GreedyFool) while ensuring full success. They are much less complex than the-state-of-the-art method B&B relying on the model propagation score (near $\frac{1}{20}$ on CIFAR-10 and $\frac{1}{5}$ on ImageNet) and achieve competitive results in some cases. In comparison with Sparse-RS, our attacks are flexible allowing to find an optimal budget of pixels for each input image and achieve full success with minimal $L_0$

scores. They are applicable as $L_\infty$ attacks and can outperform the PGD baselines. We believe that our results can be improved in a crucial way by combining different kind of noises as explained in the paper.

**Acknowledgements.** This research work has been carried out in the framework of IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the scope of the French Program Investissements d'Avenir. This work is a part of the project EPI project (EPI for "Evaluation des Performances de l'Intelligence artificielle"). The project is supervised by IRT systemX and its partners which are Apsys, Expleo France, Naval Group and Stellantis. Also, we would like to thank Jérôme Rony for useful discussions and for his short implementation of our attack in [24].

## References

1. Modar Alfadly, Adel Bibi, Emilio Botero, Salman Alsubaihi, and Bernard Ghanem. Network Moments: Extensions and Sparse-Smooth Attacks. *arXiv e-prints*, page arXiv:2006.11776, June 2020.
2. Adel Bibi, Modar Alfadly, and Bernard Ghanem. Analytic expressions for probabilistic moments of pl-dnn with gaussian input. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
3. Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge. Accurate, reliable and fast robustness evaluation, 2019.
4. N Carlini and D Wagner. Towards evaluating the robustness of neural networks. *CoRR*, 1608.04644v2, 2017.
5. Théo Combey, António Loison, Maxime Faucher, and Hatem Hajri. Probabilistic jacobian-based saliency maps attacks. *Machine Learning and Knowledge Extraction*, 2(4):558–578, 2020.
6. Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *CoRR*, abs/2010.09670, 2020.
7. Francesco Croce, Maksym Andriushchenko, Naman D. Singh, Nicolas Flammarion, and Matthias Hein. Sparse-rs: a versatile framework for query-efficient sparse black-box adversarial attacks. 2020.
8. Francesco Croce and Matthias Hein. Sparse and imperceivable adversarial attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.
9. Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020.
10. Xiaoyi Dong, Dongdong Chen, Jianmin Bao, Chuan Qin, Lu Yuan, Weiming Zhang, Nenghai Yu, and Dong Chen. Greedyfool: Distortion-aware sparse adversarial attack, 2020.
11. I J Goodfellow, J Shlens, and C Szegedy. Explaining and harnessing adversarial examples. *ICLR*, 1412.6572v3, 2015.
12. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
13. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for

image recognition, 2015.
14. Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).
15. A Kurabin, I J Goodfellow, and S Bengio. Adversarial examples in the physical world. *ICLR*, 1607.02533v4, 2017.
16. Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.
17. Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
18. Yandong Li, Lijun Li, Liqiang Wang, Tong Zhang, and Boqing Gong. Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks. In *ICML*, 2019.
19. A Madry, A Makelov, L Schmidt, D Tsipras, and A Vladu. Towards deep learning models resistant to adversarial attacks. 1706.06083v3, 2017.
20. Apostolos Modas, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Sparsefool: a few pixels make a big difference, 2019.
21. Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks, 2016.
22. N Papernot, P McDaniel, S Jha, M Fredrikson, Z Berkay Celik, , and A Swami. The limitations of deep learning in adversarial settings. *IEEE*, 1511.07528v1, 2015.
23. Daniel Revuz and Marc Yor. Continuous martingales and brownian motion. 3rd ed. 1999.
24. Jérôme Rony and Ismail Ben Ayed. Adversarial Library.
25. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
26. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.
27. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.
28. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.