machine learning & knowledge extraction

MDPI

Article

Probabilistic Jacobian-Based Saliency Maps Attacks

Théo Combey [1], António Loison [1], Maxime Faucher [1] and Hatem Hajri [2,*]

[1] CentraleSupélec, Mathematics and Computer Science Department, 3 Rue Joliot-Curie, 91192 Gif-sur-Yvette, France; theo.combey@student-cs.fr (T.C.); antonio.loison@student-cs.fr (A.L.); maxime.faucher@student-cs.fr (M.F.)

[2] IRT SystemX, 8 Avenue de la Vauve, 91120 Palaiseau, France

* Correspondence: hatem.hajri@irt-systemx.fr

Received: 10 October 2020; Accepted: 5 November 2020; Published: 13 November 2020

Simple Summary: This paper introduces simple, faster and more efficient versions of the known targeted and untargeted Jacobian-based Saliency Map Attacks (JSMA). Despite creating adversarial examples with a higher average L_0 distance than the state-of-the-art Carlini-Wagner attack, the new versions of JSMA have a significant speed advantage over this attack, making them very convenient for L_0 real-time robustness testing of neural network classifiers.

Abstract: Neural network classifiers (NNCs) are known to be vulnerable to malicious adversarial perturbations of inputs including those modifying a small fraction of the input features named sparse or L_0 attacks. Effective and fast L_0 attacks, such as the widely used Jacobian-based Saliency Map Attack (JSMA) are practical to fool NNCs but also to improve their robustness. In this paper, we show that penalising saliency maps of JSMA by the output probabilities and the input features of the NNC leads to more powerful attack algorithms that better take into account each input's characteristics. This leads us to introduce improved versions of JSMA, named Weighted JSMA (WJSMA) and Taylor JSMA (TJSMA), and demonstrate through a variety of white-box and black-box experiments on three different datasets (MNIST, CIFAR-10 and GTSRB), that they are both significantly faster and more efficient than the original targeted and non-targeted versions of JSMA. Experiments also demonstrate, in some cases, very competitive results of our attacks in comparison with the Carlini-Wagner (CW) L_0 attack, while remaining, like JSMA, significantly faster (WJSMA and TJSMA are more than 50 times faster than CW L_0 on CIFAR-10). Therefore, our new attacks provide good trade-offs between JSMA and CW for L_0 real-time adversarial testing on datasets such as the ones previously cited.

Keywords: Jacobian-based Saliency Map; adversarial attacks; deep neural network classifiers; MNIST; CIFAR-10; GTSRB

1. Introduction

Deep learning classifiers are used in a wide variety of situations, such as vision, speech recognition, financial fraud detection, malware detection, autonomous driving, defense, and more.

The ubiquity of deep learning algorithms in many applications, especially those that are critical such as autonomous driving [1,2] or that pertain to security and privacy [3,4] makes their attack particularly useful. Indeed, this allows firstly to identify possible flaws in the intelligent model and secondly to set up a defense strategy to improve its reliability.

In this context, adversarial machine learning has appeared as a new branch that aims to thwart intelligent algorithms. Many techniques called adversarial attacks succeeded in fooling well-known architectures of neural networks, sometimes very astonishingly. Examples of these methods include for instance: Fast Gradient Sign Method [5], Basic Iterative Method [6], Projected Gradient Descent [7], JSMA [8], DeepFool [9], Universal Adversarial Perturbations [10] and CW attacks [11].

More formally, a neural network classifier (NNC) is an algorithm whose goal is to predict through a neural network which class an item x belongs to, among a family of K possible classes. It outputs a vector of probabilities $F(x) = (F_1(x), \cdots, F_K(x))$ where the label of x is deduced by the rule $\text{label}(x) = \text{argmax}_k F_k(x)$.

An adversarial example constructed on the item x, is an item x^* specially crafted to be as close as possible to x (with respect to some distance function), and such that it is classified by the NNC as $\text{label}(x^*) \neq \text{label}(x)$ (Non-Targeted (NT) attack), or even such that $\text{label}(x^*) = L$, with L chosen by the attacker and such that $L \neq \text{label}(x)$ (Targeted attack).

In this paper, we focus on the following class of attacks:

L_0 **(or sparse) adversarial attacks.** They aim at generating adversarial samples while minimising the number of modified components. Sparse perturbations can be found in many real-life situations. As motivated in [12], *sparse perturbations could correspond to some raindrops that reflect the sun on a"STOP" sign, but that are sufficient to fool an autonomous vehicle; or a crop-field with some sparse colorful flowers that force a UAV to spray pesticide on non affected areas.* This reveals very disturbing and astonishing properties of neural networks as it is possible to fool them by modifying few pixels [12,13]. Their study is therefore fundamental to mitigate their effects and take a step forward towards robustness of neural networks. The first proposed example of L_0 attacks is JSMA, a targeted attack [8]. In a computer vision application, JSMA achieved 97% adversarial success rate by modifying on average 4.02% input features per sample. [8] relates this result to the human capacity of visually detecting changes. An important quality of JSMA is that it is easy to understand, set up and it is relatively fast. For instance, relying on its `cleverhans` implementation [14], JSMA is able to generate 9 adversarial samples on the MNIST handwritten digits dataset [15] in only 2 s on a laptop with 2 CPU cores. Speed and also the ability to run adversarial attacks with limited resources are important criteria for real-life deployment of neural networks [16–18]. JSMA obeys these constraints making its use widespread beyond computer vision applications such as in cybersecurity, anomaly detection and intrusion detection [19–21]. Later on, [11] proposes the second example of targeted L_0 attacks known as CW L_0. The same paper shows that unlike JSMA, CW L_0 scales well to large datasets by considering the IMAGENET dataset [22]. CW L_0 is now the state-of-the-art L_0 targeted attack with the lower average L_0 distance. However, it is computationally-expensive, much slower than JSMA on small datasets (a factor of 20 times slower is reported in [11]) and therefore, despite efficiency, it is less convenient for real-time applications.

Let us now briefly explain how the afore-mentioned attacks concretely work:

JSMA. To fool NNCs, this attack relies on the Jacobian matrix of outputs with respect to inputs. By analysing this matrix, one can deduce how the output probabilities behave given a slight modification of an input feature. Consider a NNC N as before and call $Z(x) = (Z_1(x), \cdots, Z_K(x))$ the outputs of the second-to-last layer of N (no longer probabilities, but related to the final output by a softmax layer). To generate an adversarial example from x, JSMA first computes the gradient $\nabla Z(x)$. The next step is to build a saliency map and find the most salient component i that will then be changed:

$$S[x,t][i] = \begin{cases} 0 \quad \text{if } \dfrac{\partial Z_t(x)}{\partial x_i} < 0 \text{ or } \displaystyle\sum_{k \neq t} \frac{\partial Z_k(x)}{\partial x_i} > 0 \\ \dfrac{\partial Z_t(x)}{\partial x_i} \cdot \left| \displaystyle\sum_{k \neq t} \frac{\partial Z_k(x)}{\partial x_i} \right| \quad \text{otherwise.} \end{cases} \tag{1}$$

$\frac{\partial Z_t(x)}{\partial x_i}$ and $\sum_{k \neq t} \frac{\partial Z_k(x)}{\partial x_i}$ in these maps quantify how much $Z_t(x)$ will increase and $\sum_{k \neq t} Z_k(x)$ will decrease, given a modification of the input feature x_i. Counting on the Z_k's instead of the F_k's has been justified in [8] by the extreme variations induced by the softmax layer. Then the algorithm selects the component: $i_{\max} = \text{argmax}_i S[x,t][i]$ and increases $x_{i_{\max}}$ by a default value θ before clipping to the valid domain. The same process is iterated until the class of x is changed or a maximum allowed number of iterations is reached. This version of JSMA will be called one-component JSMA. A second, more effective, variant of JSMA recalled later relies on doubly indexed saliency maps.

CW L_0 attack. This method is obtained as a solution to the optimisation problem (assuming the domain of inputs is $[0,1]^n$):

$$\text{Minimise } ||r||_0 + cf(x+r), \quad x + r \in [0,1]^n$$

where $||r||_0$ is the L_0 distance of the perturbation r added to x. The recommended choice of f is: $f(x) = (\max_{i \neq t} Z_i(x) - Z_t(x))^+$. Since the L_0 distance is not convenient for gradient descent, the authors of [11] solve this problem by making use of their L_2 attack and an algorithm that iteratively eliminates the components without much effect on the output classification. They finally obtain an effective L_0 attack that has a net advantage over JSMA. However, the main drawback of this attack is its high computational cost.

We summarise our main contributions as follows:

- For targeted misclassification, we introduce two variants of JSMA called Weighted JSMA (WJSMA) and Taylor JSMA (TJSMA). WJSMA applies a simple weighting to saliency maps by the output probabilities, and TJSMA does the same, while additionally penalising extremal input features. Both attacks are more efficient than JSMA according to several metrics (such as speed and mean L_0 distance). We present qualitative and quantitative results on MNIST and CIFAR-10 [23] supporting our claims. Moreover, although they are less efficient than CW L_0, our attacks have a major speed advantage over CW L_0 highlighted by measuring the execution time for each attack.
- For non-targeted (NT) misclassification, we improve the known NT variants of JSMA called NT-JSMA and Maximal-JSMA (M-JSMA) [24]. We do this by introducing NT and M versions of WJSMA and TJSMA. Our attacks yield better results than NT-JSMA and M-JSMA. Also, they are as competitive but significantly much faster than NT CW L_0. These claims are illustrated through applications to attack a deep NNC on the GTSRB dataset [25] in the white/black-box modes.
- We provide a deep comparison between WJSMA and TJSMA which is of independent interest. Our study concludes that in the targeted case, TJSMA is better than WJSMA. However, in the NT case, WJSMA is preferred over TJSMA, mainly due to the simplicity of its implementation.
- We provide fast and optimised implementations of the new attacks using TensorFlow and the `cleverhans` library [14] that might help users working in adversarial machine learning (In all our experiments, we use the original implementation of JSMA available in the `cleverhans` library and the original code of CW publicly available. As for the NT versions of JSMA [24], whose implementations are not available, we re-implement these attacks using TensorFlow and `cleverhans`. A link to the codes is provided at the end of Section 5.2)

The rest of the paper is organised as follows. In Section 2, we discuss our main motivations and then introduce WJSMA and TJSMA as targeted attacks. Section 3 is focused on NT and Maximal versions of WJSMA and TJSMA. Section 4 is dedicated to several comparisons between our attacks, JSMA and CW L_0. We discuss attacking, defending with targeted/non-targeted attacks in both the white/black-box setups. Section 5 offers a conclusion and a summary of the main results of the paper. Finally, Appendices A and B are appendices dedicated to supplementary results and materials.

2. Targeted Attacks

The first attacks, presented in this section, are targeted and called Weighted JSMA (WJSMA) and Taylor JSMA (TJSMA). We give a detailed exposition of WJSMA and motivate the main idea leading to its derivation through a simple example. Then, we deduce TJSMA by applying once more and in a slightly different manner the same argument. In addition to the theoretical presentation, a few preliminary figures are given to illustrate a faster convergence of the new attacks in comparison with JSMA.

2.1. Weighted Jacobian-Based Saliency Map Attack (WJSMA)

The main idea here is to penalise gradients associated with small probabilities so as to mitigate their influences in saliency maps. The goal is to obtain more balanced saliency maps than those proposed by JSMA. We first give a concrete example illustrating a concrete limitation of JSMA which motivated this work.

Motivating example. Assume a number of classes $K \geq 4$ and for some input x: $F_1(x) = 0.5$, $F_2(x) = 0.49$, $F_3(x) = 0.01$ and $F_k(x) = 0$ for all $4 \leq k \leq K$. Consider the problem of generating an adversarial sample to x with label $t = 2$. In order to decrease $\sum_{k \neq 2} Z_k(x)$, the first iteration of JSMA relies on the gradients $\nabla Z_k(x), k \neq 2$. Our main observation is that since the probabilities $F_k(x) = 0$ for $4 \leq k \leq K$ are already in their minimal values, taking into account $\nabla Z_k(x)$ for these values of k in the search of $i_{\max}$ is unnecessary. In other words, by only acting on gradients, JSMA does not consider the crucial constraints on probabilities: $F_k(x) \geq 0$. Moreover, instead of relying equally on $\nabla Z_1(x)$ and $\nabla Z_3(x)$, for this example one would "bet" on $\nabla Z_1(x)$ than $\nabla Z_3(x)$ as the possible decrease for $F_1(x)$ is high (up to 0.5) and $F_3(x)$ is relatively small, thus hard to decrease further.

Weighted JSMA (WJSMA). Our first solution to the previous issue is WJSMA. Its principle is to penalise each gradient $\frac{\partial Z_k(x)}{\partial x_i}$, where $k \neq t$, by the probability $F_k(x)$. Besides the intuition of this idea, we will provide a justification of it by a classical log softmax argument. First, we compute:

$$\frac{\partial}{\partial x_i} \log F_t(x) = (1 - F_t(x)) \frac{\partial Z_t}{\partial x_i}(x) - \sum_{k \neq t} F_k(x) \frac{\partial Z_k}{\partial x_i}(x)$$

One way to maximise this derivative with respect to i, is to maximise $A = \frac{\partial Z_t}{\partial x_i}(x)$ and minimise $B = \sum_{k \neq t} F_k(x) \frac{\partial Z_k}{\partial x_i}(x)$ under the constraints $A > 0$ and $B < 0$. These constraints ensure, in particular, that $\frac{\partial F_t}{\partial x_i}(x)$ remains positive, a fact which is not necessarily guaranteed under JSMA constraints. According to this, we introduce one-component weighted saliency maps as follows:

$$S^W[x,t][i] = \begin{cases} 0 \quad \text{if } \frac{\partial Z_t(x)}{\partial x_i} < 0 \text{ or } \sum_{k \neq t} F_k(x) \frac{\partial Z_k(x)}{\partial x_i} > 0 \\ \frac{\partial Z_t(x)}{\partial x_i} \cdot \left| \sum_{k \neq t} F_k(x) \frac{\partial Z_k(x)}{\partial x_i} \right| \quad \text{otherwise.} \end{cases}$$

Based on these maps, we present Algorithm 1, the first version of WJSMA, to generate targeted adversarial samples.

Algorithm 1 Generating adversarial samples by WJSMA: version 1.

Inputs: N: a NNC, Z: second-to-last output of N, x: input to N, t: target label ($t \neq \text{class}(x)$), `maxIter`: maximum number of iterations, $\theta_{\min}, \theta_{\max}$ lower and upper bounds for features values, θ: positive default increase value.
Output: x^*: adversarial sample to x.

$x^* \leftarrow x$
`iter` $\leftarrow 0$
$\Gamma \leftarrow [\![1, |x|]\!] \setminus \{p \in [\![1, |x|]\!] \mid x[p] = \theta_{max}\}$
while $\text{class}(x^*) \neq t$ **and** `iter` $<$ `maxIter` **and** $\Gamma \neq \emptyset$ **do**
  $p_{max} = \text{argmax}_{p \in \Gamma} S^W[x^*, t](p)$
  Modify x^* by $x^*[p_{max}] = \texttt{Clip}_{[\theta_{\min}, \theta_{\max}]}(x^*[p_{max}] + \theta)$ `//Clip` is the clipping function
  Remove $p_{\max}$ from Γ
  `iter` $++$
end while
return x^*

When the output x^* of Algorithm 1 satisfies $\text{class}(x^*) = t$, the attack is considered as successful.

To relax a bit the search of salient components and motivated by a computer vision application, ref. [8] introduces saliency maps indexed by pairs of components. The main argument is that the conditions required in (1) may be too strict for some applications and very few components will verify it. Our doubly indexed versions of these maps are introduced in the same way as follows:

$$S^W[x,t][i,j] = \begin{cases} 0 \quad \text{if} \quad \displaystyle\sum_{a \in \{i,j\}} \frac{\partial Z_t(x)}{\partial x_a} < 0 \quad \text{or} \quad \sum_{k \neq t} F_k(x) \sum_{a \in \{i,j\}} \frac{\partial Z_k(x)}{\partial x_a} > 0 \\ \displaystyle\sum_{a \in \{i,j\}} \frac{\partial Z_t(x)}{\partial x_a} \cdot \left| \sum_{k \neq t} F_k(x) \sum_{a \in \{i,j\}} \frac{\partial Z_k(x)}{\partial x_a} \right| \quad \text{otherwise.} \end{cases} \tag{2}$$

Algorithm 2 presented below relies on $S^W[x,t][i,j]$ to generate targeted adversarial samples and is our second version of WJSMA.

Algorithm 2 Generating adversarial samples by WJSMA: version 2.

Same inputs and output as Algorithm 1.

$x^* \leftarrow x$
`iter` $\leftarrow 0$
$\Gamma \leftarrow \{(p,q), p, q \in [\![1, |x|]\!], x[p] \neq \theta_{max}, x[q] \neq \theta_{max}\}$
while $\text{class}(x^*) \neq t$ **and** `iter` $<$ `maxIter` **and** $\Gamma \neq \emptyset$ **do**
  $(p_{\max}, q_{\max}) = \text{argmax}_{p,q \in \Gamma} S^W[x^*, t](p,q)$
  Modify x^* by $x^*[a] = \texttt{Clip}_{[\theta_{\min}, \theta_{\max}]}(x^*[a] + \theta)$, $a = p_{\max}, q_{\max}$
  Remove $(p_{\max}, q_{\max})$ from Γ
  `iter` $++$
end while
return x^*

In practice and despite the fact that each iteration of Algorithm 2 is more computationally-expensive than each iteration of Algorithm 1, we find that it gives better results. This agrees with the recommendations of [8] on the superiority of two-components versions for JSMA. Finally, we notice that while in the two previous algorithms, the selected components are always augmented by positive default values, decreasing versions can be given following a similar logic.

2.2. Taylor Jacobian-based Saliency Map Attack (TJSMA)

The principle of our second attack, TJSMA, is to additionally penalise the choice of feature components that are close to the maximum value $\theta_{\max}$. Assume i and j have the same WJSMA score $S^W[x,t][i] = S^W[x,t][j]$ and that x_i is very close to $\theta_{\max}$, while x_j is far enough from $\theta_{\max}$. In this case, looking for more impact, TJSMA prefers x_j over x_i. Concretely, we simultaneously maximise $S_1 = \theta_{\max} - x_i$ and $S_2 = \dfrac{\partial}{\partial x_i} \log p_t(x)$ by maximising the product $S = S_1 S_2$. Accordingly, we introduce new saliency maps for one and two-components selection as follows:

$$S^T[x,t][i] = \begin{cases} 0 & \text{if } \alpha_i < 0 \text{ or } \beta_i > 0 \\ \alpha_i |\beta_i| & \text{otherwise.} \end{cases} \tag{3}$$

where

$$\alpha_i = (\theta_{\max} - x_i)\frac{\partial Z_t(x)}{\partial x_i}, \quad \beta_i = (\theta_{\max} - x_i)\sum_{k \neq t} F_k(x)\frac{\partial Z_k(x)}{\partial x_i}$$

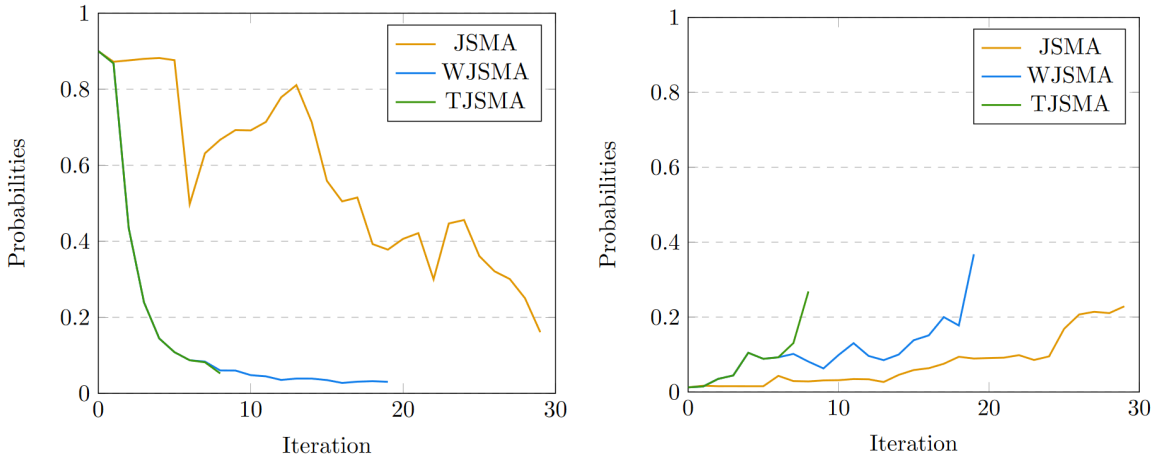
and

$$S^T[x,t][i,j] = \begin{cases} 0 & \text{if } \alpha_{i,j} < 0 \text{ or } \beta_{i,j} > 0 \\ \alpha_{i,j} |\beta_{i,j}| & \text{otherwise.} \end{cases} \tag{4}$$

where

$$\alpha_{i,j} = \sum_{a \in \{i,j\}} (\theta_{\max} - x_a)\frac{\partial Z_t(x)}{\partial x_a}, \quad \beta_{i,j} = \sum_{k \neq t} \sum_{a \in \{i,j\}} F_k(x)(\theta_{\max} - x_a)\frac{\partial Z_k(x)}{\partial x_a}$$

Due to the presence of the Taylor terms $(\theta_{\max} - x_a)\dfrac{\partial Z_k(x)}{\partial x_a}$, we call these maps Taylor saliency maps. We introduce one and two-components TJSMA following exactly Algorithms 1 and 2 and only replacing S^W with S^T.



(**a**) Origin class (**b**) Target class

Figure 1. Evolution of the (**a**) origin class and (**b**) target class probabilities until the target class is reached for JSMA, WJSMA and TJSMA changing the image of a one into a five.

Figure 1a,b offer a concrete illustration of the convergence of JSMA, WJSMA and TJSMA. In particular, we observe that WJSMA and TJSMA decrease/increase the predicted/targeted probability of the original/targeted class much sooner than JSMA. Also, we note that TJSMA behaves like WJSMA until it is able to find a more vulnerable component that makes it converge much faster.

3. Non-Targeted Attacks

NT variants of JSMA have been studied in [24]. In particular, the paper introduces NT-JSMA-F, NT-JSMA-Z based on NT saliency maps whose role is to select the most salient pairs of components to decrease as much as possible the probability of the current class. The notations -F and -Z indicate if the saliency maps either use the F_k's or the Z_k's. Second, [24] proposes maximal JSMA (M-JSMA) as a more flexible attack allowing both increasing/decreasing features and also combining targeted/non-targeted strategies at the same time.

In what follows, we again leverage the idea of penalising saliency maps to give our proper NT JSMA attacks. For a unified presentation, we use the letter X to denote either W (Weighted) or T (Taylor) and the letter Y to denote either Z (logits) or F (probabilities). We notice that while the first version of JSMA uses the logits, variants that rely on the F_k's also demonstrated good performances [11,24]. Thus for a more complete study, we give versions with both Z and F.

By a NT reasoning, similar to the previous section, we define Weighted and Taylor NT saliency maps as follows:

$$S^{X,Y}[x,t][i,j] = \begin{cases} 0 & \text{if } \alpha_{i,j}^{X,Y} > 0 \text{ or } \beta_{i,j}^{X,Y} < 0 \\ |\alpha_{i,j}^{X,Y}|\beta_{i,j}^{X,Y} & \text{otherwise.} \end{cases} \tag{5}$$

where, for $X = W$,

$$\alpha_{i,j}^{W,Y} = \sum_{a\in\{i,j\}} \frac{\partial Y_t(x)}{\partial x_a}, \quad \beta_{i,j}^{W,Y} = \sum_{k\neq t} \sum_{a\in\{i,j\}} F_k(x) \frac{\partial Y_k(x)}{\partial x_a}$$

and, for $X = T$,

$$\alpha_{i,j}^{T,Y} = \sum_{a\in\{i,j\}} (\theta_{\max} - x_a) \frac{\partial Y_t(x)}{\partial x_a}, \quad \beta_{i,j}^{T,Y} = \sum_{k\neq t} \sum_{a\in\{i,j\}} F_k(x)(\theta_{\max} - x_a) \frac{\partial Y_k(x)}{\partial x_a}$$

These maps can be motivated, like in the previous section, by considering the simple one-component case: $i = j$. For example, the role of the penalisation by $F_k(x)$ in $S^{W,Z}$ is to reduce the impact of high gradients $\frac{\partial Z_k(x)}{\partial x_i}$ when the probability $F_k(x)$ is very small (in this case we penalise choosing k as a new target for x).

Relying on saliency maps, Algorithm 3 below presents our improvements of NT-JSMA-Z and NT-JSMA-F. For the sake of simplification, we have employed a unified notation NT-XJSMA-Y. For example, when we use $S^{W,Z}$, the obtained attack is NT-WJSMA-Z. Again, we only write the increasing version.

Algorithm 3 NT-WJSMA and NT-TJSMA attacks.

Inputs: x: input to N with label t, maxIter: maximum number of iterations, $\mathrm{X} \in \{W, T\}$, $\mathrm{Y} \in \{Z, F\}$
Output: x^*: adversarial sample to x.

$x^* \leftarrow x$
`iter` $\leftarrow 0$
$\Gamma \leftarrow \{(p,q), p, q \in [\![\theta_{\max}, |x|]\!], x[p] \neq \theta_{\max}, x[q] \neq \theta_{\max}\}$
while class$(x^*) \neq t$ **and** `iter` $<$ `maxIter` **and** $|\Gamma| \geq 2$ **do**
  $(p_{\max}, q_{\max}) = \mathrm{argmax}_{p,q\in\Gamma} S^{X,Y}[x^*, t](p,q)$
  Modify x^* by $x^*[p_{\max}], x^*[q_{\max}] = \theta_{\max}$
  Remove $(p_{\max}, q_{\max})$ from Γ
  `iter` $++$
end while
return x^*

We now turn to extensions of M-JSMA [24]. This attack modifies the pairs of components achieving the best score among NT-JSMA and all possible targeted (including increasing and decreasing) JSMA. It has a greater capacity to craft adversarial samples but is relatively slower than NT-JSMA. Our extensions of M-JSMA, which we call M-WJSMA and M-TJSMA, are described in Algorithm 4.

Algorithm 4 M-WJSMA and M-TJSMA attack.

Inputs: x: input to N with label t, maxIter: maximum number of iterations, $\mathrm{X} \in \{W, T\}$, $\mathrm{Y} \in \{Z, F\}$
Output: x^*: adversarial sample to x.

$x^* \leftarrow x$
$\texttt{iter} \leftarrow 0$
$\Gamma \leftarrow \{(p,q), p, q \in [\![1, |x|]\!], x[p], x[q] \neq \theta_{\min}, \theta_{\max}\}$
while class$(x^*) \neq t$ **and** $\texttt{iter} < \texttt{maxIter}$ **and** $|\Gamma| \geq 2$ **do**
- Compute all targeted increasing/decreasing saliency maps scores $S^{X,Y}[x,s](p,q), s \neq t$ (Section 2) and all NT increasing/decreasing saliency maps scores $S^{X,Y}[x,t](p,q)$ (5).
- Choose $(p_{\max}, q_{\max})$ achieving the best score and saturate $x^*[p_{\max}], x^*[q_{\max}]$ to $\theta_{\min}$ or $\theta_{\max}$ according to the chosen saliency map.
- Remove $(p_{\max}, q_{\max})$ from Γ
- $\texttt{iter} + +$
end while
return x^*

Note that saliency maps $S^{W,Y}$ for targeted or non-targeted, features-increasing or features-decreasing attacks are exactly the same (one only needs to decide between an argmax or argmin). This is not the case for M-TJSMA since for example $(\theta_{\max} - x_a)$ has to be changed to $x_a - \theta_{\min}$ when decreasing features. As a consequence of this fact, M-WJSMA is less cumbersome to implement than M-TJSMA. Trying to keep the paper and code as simple as possible, we choose M-WJSMA over M-TJSMA and do not include M-TJSMA in our experiments. M-WJSMA already gives us satisfactory results.

4. Experiments

This section is dedicated to a variety of experiments on the proposed attacks and several comparisons with the state-of-the-art methods. The first part focuses on targeted attacks and provides intensive comparisons between JSMA, WJSMA and TJSMA on deep NNCs on MNIST and CIFAR-10 as well as comparisons with CW L_0. In the second part, we show that our approach is still relevant for non-targeted L_0 misclassification in both the white-box and black-box modes. A particular emphasis will be put on the speed of our attacks in comparison with CW L_0.

4.1. Experiments on Targeted Attacks

In the following, we give attack and defense applications illustrating the interest of WJSMA and TJSMA over JSMA. In doing so, we also compare WJSMA and TJSMA and report overall better results for TJSMA despite the fact that for a large part of samples WJSMA outperforms TJSMA. Finally, we provide a comparison with CW L_0 attack and comment on all the obtained results.

The datasets used in this section are:

MNIST [15]. This dataset contains 70,000 28×28 greyscale images in 10 classes, divided into 60,000 training images and 10,000 test images. The possible classes are digits from 0 to 9.

CIFAR-10 [23]. This dataset contains 60,000 $32 \times 32 \times 3$ RGB images. There are 50,000 training images and 10,000 test images. These images are divided into 10 different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6000 images per class.

Figures 2 and 3 display one sample per class, from MNIST and CIFAR-10 respectively.

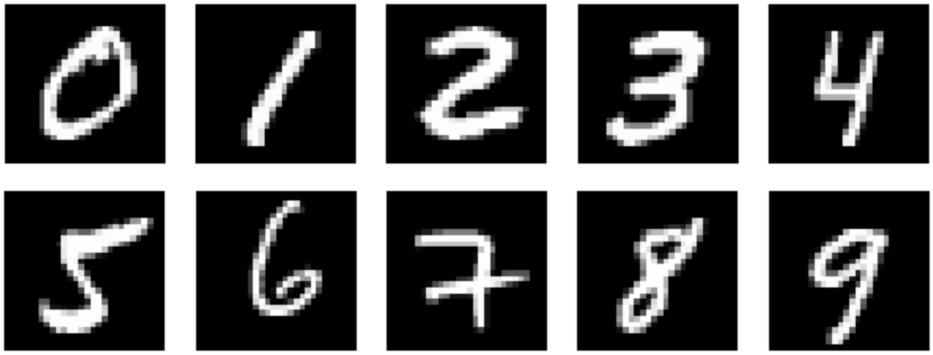


Figure 2. Examples of images from MNIST.

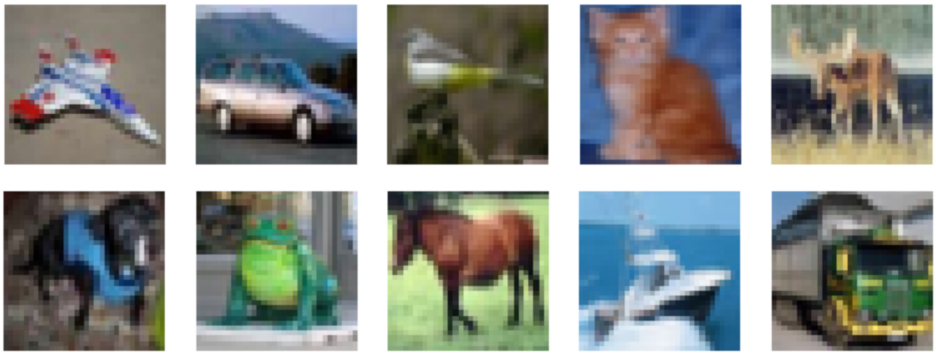


Figure 3. Examples of images from CIFAR-10.

On each dataset, a deep NNC is trained and then confronted to attacks.

NNC on MNIST. Similarly to [8], we consider LeNet-5 model on this dataset [26]. Its full architecture is described in Appendix A. We implement and train this model using a `cleverhans` model that optimises crafting adversarial examples. The number of epochs is fixed to 20, the batch-size to 128, the learning rate to 0.001 and the Adam optimizer is used. Training results in 99.98% accuracy on the training dataset and 99.49% accuracy on the test dataset.

NNC on CIFAR-10. We consider a more complex NNC, trained to reach a good performance on this dataset. Its architecture is inspired by the All Convolutional model proposed in `cleverhans` and is fully described in Appendix A. Likewise, this model is implemented and trained using `cleverhans` for 10 epochs, with a batch size of 128, a learning rate of 0.001 and the Adam optimizer. Training results in a 99.96% accuracy on the training dataset and 83.81% accuracy on the test dataset.

Our first objective is to compare the performances between JSMA, WJSMA and TJSMA on the previous two NNCs. To run JSMA, we use its original implementation, available in `cleverhans`. We have also adapted the code to WJSMA and TJSMA thus obtaining fast implementations of these two attacks.

For testing, we consider all images in MNIST, the first 10,000 training and 10,000 test images of CIFAR-10. Moreover, we only test on the images which are correctly predicted by the neural networks as this makes more sense. In this way, the attacks are applied to the whole training set and the 9949 well-predicted images of the MNIST test images. Similarly, CIFAR-10 adversarial examples are crafted from the well-predicted 9995 images of the first training 10,000 images and the 8381 well-predicted test images.

To compare the three attacks, we rely on the notion of maximum distortion of adversarial samples defined as the ratio of altered components to the total number of components. Following [8], we choose a maximum distortion of $\gamma = 14.5\%$ on the adversarial samples from MNIST, corresponding to $\texttt{maxIter} = \lfloor \frac{784 * \gamma}{2 * 100} \rfloor$. On CIFAR-10, we fix $\gamma = 3.7\%$ in order to have the same maximum number of

iterations for both experiments. This allows a comparison between the attacks in two different settings. Furthermore, for both experiments, we set $\theta = 1$ (note that $\theta_{\min} = 0$, $\theta_{\max} = 1$).

We report the metrics:

1. Success rate: This is the percentage of successful adversarial examples, i.e crafted before reaching the maximal number of iterations `maxIter`,
2. Mean L_0 distance: This is the average number of altered components of the successful adversarial examples,
3. Strict dominance of an attack: Percentage of adversarial examples for which this attack does strictly fewer iterations than the two other ones (As additional results, we give in the Appendix B more statistics on the dominance between any two attacks),
4. Run-time of an attack on a set of samples targeting every possible class.

Results on the metrics 1 and 2 are shown in Table 1 for MNIST and Table 2 for CIFAR-10.

Table 1. Comparison between JSMA, WJSMA and TJSMA on MNIST.

Metric	JSMA	WJSMA	TJSMA
Targeted (Training dataset: Nb of well predicted images = 60,000)			
Success rate	87.68%	**97.14%**	**98.66%**
Mean L_0 distance on successful samples	44.34	**37.86**	**35.22**
Targeted (Test dataset: Nb of well predicted images = 9949))			
Success rate	87.34%	**96.98%**	**98.68%**
Mean L_0 distance on successful samples	44.63	**38.10**	**35.50**

Table 2. Comparison between JSMA, WJSMA and TJSMA on CIFAR-10.

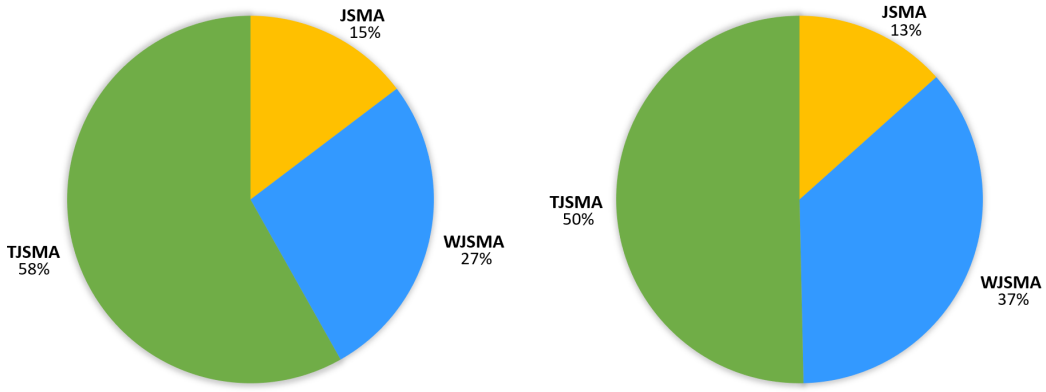
Metric	JSMA	WJSMA	TJSMA
Targeted (Training dataset: Nb of well predicted images = 9995)			
Success rate	86.17	**95.91%**	**97.40%**
Mean L_0 distance on successful samples	47	**38.54**	**36.86**
Targeted (Test dataset: Nb of well predicted images = 8381))			
Success rate	84.91	**94.99%**	**96.96%**
Mean L_0 distance on successful samples	46.13	**38.82**	**37.45**

First, we observe that overall, WJSMA and TJSMA significantly outperform JSMA according to metrics 1 and 2. Here are more comments:

On MNIST. Results in terms of success rate are quite remarkable for WJSMA and TJSMA respectively outperforming JSMA with near 9.46, 10.98 percentage points (pp) on the training set and 9.46, 11.34 pp on the test set. The gain in the average number of altered components exceeds 6 components for WJSMA and 9 components for TJSMA in both experiments.

On CIFAR-10. WJSMA and TJSMA outperform JSMA in success rate by near 9.74, 11.23 pp on the training set and more than 10, 12 pp on the test set. For both training and test sets, we report better mean L_0 distances exceeding 7 features in all cases and up to 10.14 features for TJSMA on the training set.

Dominance of the attacks. Figure 4 illustrates the (strict) dominance of the attacks for the two experiments. In these statistics, we do not count the samples for which TJSMA and WJSMA have the same number of iterations and strictly less than JSMA.



(**a**) MNIST (**b**) CIFAR-10

Figure 4. Distribution of the (strict) dominance of JSMA, WJSMA and TJSMA over the MNIST (**a**) and CIFAR10 (**b**) datasets (training and test sets included).

For both experiments, TJSMA has a notable advantage over WJSMA and JSMA. The benefit of WJSMA over JSMA is also considerable. This shows that, in most cases, WJSMA and TJSMA craft better adversarial examples than JSMA, while being faster. Our results are indeed better when directly comparing WJSMA or TJSMA with JSMA. As additional results, we give in the appendix the statistics for the pairwise dominance between the attacks. As it might be expected, both WJSMA and TJSMA dominate JSMA, moreover TJSMA dominate WJSMA.

4.1.1. Avoid Confusion

It is important to stress that our results do not contradict [8] obtaining 97% success rate on LeNet-5. Indeed, we use a more efficient LeNet-5 model (the one in [8] has 98.93% and 99.41% accuracies on the training and test sets). For completeness, we also generated a second model (with 99.34% and 98.94% accuracies on the training and test sets) and evaluated the three attacks on the first 1000 test MNIST images. We obtain 96.7% success rate for JSMA (very similar to [8]) and more than 99.5% for WJSMA and TJSMA. We preferred to work with the more effective model as this makes the paper shorter and moreover, it values more our approach (giving us more advantage with respect to JSMA).

4.1.2. Run-Time Comparison

In order to have a meaningful speed comparison between the three attacks, we computed the time needed for each attack to successfully craft the first 1000 test MNIST images in the targeted mode. Results are shown in Table 3 and reveal that TJSMA/WJSMA are 1.41/1.28 times faster than JSMA. These performances were measured on a machine equipped with an Intel Xeon 6126 processor and a Nvidia Tesla P100 graphics processor. Note that for WJSMA/TJSMA, the additional computations of one iteration compared to JSMA are negligible (simple multiplications). Thus the difference in speed between the attacks is mainly due to the number of iterations for each attack.

Table 3. Time comparison between JSMA, WJSMA and TJSMA.

Attack	JSMA	WJSMA	TJSMA
Time (second)	3964	**3092**	**2797**

We also notice that in this evaluation, the adversarial samples were crafted one by one. In practice, it is possible to generate samples by batch. In this case, the algorithm stops when all samples are finished. Most of the time, with a batch of large size, the three attacks approximately take the same time to converge. For example, on the same machine as previously and with a batch size equal to 1000, we were able to craft the same amount of samples in about 250 s, for all the attacks.

Defense. The objective now is to train neural networks so that the attacks fail as much as possible. One way of doing this is by adding adversarial samples crafted by JSMA, WJSMA and TJSMA to the training set. This method of training may imply a decrease in the model accuracy but adversarial examples will be more difficult to generate.

We experiment this idea on the MNIST model in every possible configuration. To this end, 2000 adversarial samples per class (20,000 more images in total), with distortion under 14.5%, are added to the original MNIST training set, crafted by either JSMA, WJSMA or TJSMA. Then, three distinct models are trained on these augmented datasets. The models roughly achieve an accuracy of 99.9% on the training set and 99.3% on the test set, showing a slight loss compared to our previous MNIST model. Nevertheless, the obtained neural networks are more robust to the attacks as shown in Table 4. Note that each experiment is made over the well-predicted samples of the test images. For each model and image, nine adversarial examples are generated by the three attacks.

Table 4. Metrics (1) and (2) on JSMA, WJSMA and TJSMA augmented sets.

Metric	JSMA	WJSMA	TJSMA
Model trained over JSMA augmented set (9940 well predicted samples)			
Success rate	77.94%	**84.79%**	**85.08%**
Mean L_0 distance on successful samples	54.48	**52.66**	**52.83**
Model trained over WJSMA augmented set (9936 well predicted samples)			
Success rate	77.61%	**90.05%**	**92.01%**
Mean L_0 distance on successful samples	56.29	**52.72**	**52.18**
Model trained over TJSMA augmented set (9991 well predicted samples)			
Success rate	76.42%	**86.18%**	**87.36%**
Mean L_0 distance on successful samples	54.26	**54.20**	**54.49**

Overall, the attacks are less efficient on each of these models, compared to Table 1. The success rates drop by about 8 pp, whereas the number of iterations is increased by approximately 26%. From the defender's point of view, networks trained against WJSMA and TJSMA give the best performance. The JSMA trained model provides the lowest success rates while the TJSMA trained network is more robust from the L_0 distance point of view. From the attacker's point of view, TJSMA remains the most efficient attack regardless of the augmented used dataset.

4.1.3. Comparison with CW L_0 Attack

Because of the complexity of this attack, comparison on a large number of images as before is very costly. For this reason, we only provide results on the first 100 well-predicted images of CIFAR-10, thus on 1000 adversarial images given in Table 5.

Table 5. Results for L_0 CW on CIFAR-10.

Success rate	Mean L_0 distance	Time
99.89%	24.97	On average more than one hour and a half to generate 9 adversarial samples run one by one on GPU

We report better results of CW in terms of success rate and L_0 distance and a remarkable speed advantage of our attacks. Indeed, generating 9 adversarial samples (one by one) from a CIFAR-10 image by CW took on average near one hour and a half on GPU. The same task took 100 s for our attacks (without batching and 25 s when batching). This makes our attacks at least 54 times faster than CW L_0.

4.2. Experiments on Non-Targeted Attacks

In this part, we test the new NT attacks in the white/black-box modes and compare their performances with NT-JSMA and M-JSMA. In the white-box mode, we also compare with NT CW L_0. For experimentation, we chose the GTSRB dataset [25] widely used to challenge neural networks, especially in autonomous driving environments [27]. We recall that GTSRB contains RGB $32 \times 32 \times 3$ traffic signs images, 86,989 in the training set, and 12,630 in the test set classified into 43 different possible categories. Figure 5 displays some images from this dataset.

Figure 5. Examples of images from GTSRB.

4.2.1. White-Box Experiments

We consider a simplified NNC, described in Appendix A, whose architecture is inspired by Alexnet [28]. After training this model with `cleverhans`, it reaches 99.98% accuracy on the training dataset and 95.83% accuracy on the test set.

We implemented our attacks and those of [24] again with TensorFlow using `cleverhans`. During the first experiment, we run the attacks given in Table 6 on the first 1000 test images after taking a maximum distortion $\gamma = 3.7\%$ similar to CIFAR-10. The obtained results are shown in the same table.

Table 6. Comparison of the performances between the NT attacks over the 1000 first test images of GTSRB.

Attack (U = JSMA)	**Success Rate**	**L_0 Average**	**Time (s)**
NT-U	91.35%	20.66	3604
NT-WU (ours)	**95.31%**	**17.63**	**760**
NT-TU (ours)	**96.87%**	**15.86**	**660**
M-U	98.44%	18.99	7470
M-WU (ours)	**99.37%**	**15.52**	**6302**
CW L_0	97.81%	**13.56**	260,751

To analyse these results, it makes more sense to separately compare the NT-versions (faster), the M-versions (the most effective in success rate but slower) and discuss a global comparison with CW L_0. First, we notice a significant advantage of our NT versions over NT-JSMA according to the three metrics. In particular, NT-TJSMA is up to $5.4\times$ faster than NT-JSMA with nearly 4.8 less modified pixels on average and more than 5 percentage points in success rate. We also notice that NT-TJSMA is the most effective attack among the three NT versions. Its notable benefit over NT-WJSMA is due to the fact that it converges in much less iterations than NT-WJSMA while both attacks need approximately the same time for an iteration. As for the M-versions, our attack M-WJSMA outperforms M-JSMA according to the three reported metrics. Both attacks are however slower than the NT versions. Finally,

we notice that the gap between our attack M-WJSMA and CW L_0 is reduced as we obtain a better success rate, achieve less than two pixels in L_0 average while being more than 40 times faster.

4.2.2. Black-Box Experiments

Here, we consider that black-box attack means the algorithm can use the targeted model as an oracle: when feeding it an image, the oracle returns a single class label. Moreover, we want to make only a limited amount of queries, which in a realistic setup would mean avoiding any suspicious activity, or at least trying not to depend too much on the oracle. To overcome this restriction, we train a substitute NNC using the Jacobian-based Dataset Augmentation (JBDA) [27] method. We then perform white-box attacks on the substitute NNC. If it is good enough, the resulting adversarial images should *transfer* to the oracle, i.e., effectively fool it even though they were designed to fool the substitute. Details about the substitute NNC architecture can be found in Appendix A. The JBDA method allows us to make queries to the oracle only during the training of the substitute network. Black-box attacks are thus closer to real-life setups, as one only needs to access for a short period of time the targeted model before being able to durably fool it. A dangerous application of such techniques is against autonomous driving cars and their sign recognition systems. NT black-box attacks are exactly the kind of threat that can be put in practice with relative ease and still disrupt considerably the car's behaviour. We will illustrate this insecurity through the GTSRB dataset.

Attacks in this paragraph are NT-JSMA and M-JSMA [24], along with our contributions NT-WJSMA, NT-TJSMA and M-WJSMA. We experiment with different distortion rates. Contrary to white-box attacks, in the context of black-box attacks, the distortion rate is not an upper bound on the percentage of pixels that can be modified, but the exact percentage of pixels we want to perturb. This slight difference accounts for the imperfection of the substitute network: even if it mimics quite well the oracle, stopping as soon as the image switches according to the substitute will often not yield good results, so it is necessary to force the algorithm to push a little bit further. As a consequence, to evaluate the performance of the attacks, we will use two metrics: the success and transferability rates, for each distortion value. The first one measures the percentage of attacks that have been successful on the substitute NNC, while the second one measures the same percentage but for the oracle. The obtained results are given in Table 7.

Table 7. Success and transferability rates (SR & TR) in % for the black-box NT attacks on the 1000 to 2000 test samples of GTSRB for a distortion γ varying from 1% to 5%.

γ	Metric	NT-Attack-Z (X = JSMA)					NT-Attack-F (X = JSMA)				
		X	**WX**	**TX**	**M-X**	**M-WX**	**X**	**WX**	**TX**	**M-X**	**M-WX**
1%	SR	65.9	75.4	78.6	37	40.7	75.3	77.1	**80.1**	77	77.4
	TR	43	46.5	48.2	31.8	34.2	48.1	44.8	**50.3**	45.4	43.4
2%	SR	81.2	89.4	93.1	55.4	56.8	88.6	91.3	94.5	94.9	**95.3**
	TR	56.6	59.2	63.4	50.1	49.5	62.1	58	**64.1**	56.5	58.8
3%	SR	88.5	95.5	97.7	68.9	67.4	94.4	95.8	98.6	99	**99.2**
	TR	65.1	67.2	70.2	58.8	59.2	69.3	64.1	**70.7**	64.5	65.3
4%	SR	92.9	97.1	99.2	77.7	74.7	97.4	98.3	99.5	**100**	**100**
	TR	69.2	71	**75.3**	68.5	65.1	73.9	67.5	74.8	69.9	69.5
5%	SR	96	98.7	99.9	83.5	79.5	98.5	99	**100**	**100**	**100**
	TR	71.9	73.6	**77.9**	72.7	68.4	76.3	71.1	77.2	72.3	72.1

We can see in this table that in terms of success rate, the results are compatible with the white-box attack results, meaning that M-WJSMA mostly outperforms NT-TJSMA, which is better than NT-WJSMA which beats NT-JSMA, at least for the F variants. This is somewhat expected, because the attacks as performed on the substitute are merely white-box. However, one can notice that the Z variants of Maximal attacks do not perform well on this substitute network. More interestingly,

concerning the transferability of the attacks, one can notice that for the Z attacks, the same hierarchy NT-TJSMA $>$ NT-WJSMA $>$ NT-JSMA is respected, while for the F attacks, NT-WJSMA is overall inferior to all the other attacks, and NT-TJSMA outperforms NT-JSMA, M-JSMA and M-WJSMA.

Overall, NT-TJSMA is the best attack for black-box non-targeted purposes, but the best variant (F or Z) depends on the distortion rate: NT-TJSMA-Z only beats its counterpart NT-TJSMA-F for $\gamma = 4$ or 5%. Finally, if one variant were to be chosen, it would be NT-TJSMA-F due to its speed and overall best transferability.

5. Comparisons with Non-L_0 Attacks and Conclusion

5.1. Comparison with Non-L_0 Attacks

Previously, we only compared with L_0 attacks as it makes more sense to consider methods that optimise the same metric. In this section, we compare our NT-TJSMA with a well-known non-targeted L_∞ attack which is the Fast Gradient Sign Method (FGSM) [5]. We recall that FGSM attempts to minimise the L_∞ norm. It is a very fast method; significantly faster than NT-TJSMA. To this end, we run both attacks by dropping the assumption on the number of modified input features for our attack and by experimenting with different values of the L_∞ threshold ε for FGSM where the results of the best threshold are kept. Then, we computed the mean L_1 and L_2 errors for each attack as alternative comparison metrics. Table 8 shows the obtained results.

Table 8. Comparison between NT-TJSMA and FGSM.

Metric	MNIST	CIFAR10	GTSRB
Performances of NT-TJSMA			
Success rate	100%	100%	100%
Mean L_1 distance	13.58	13.33	15.01
Mean L_2 distance	3.49	2.93	2.88
Performances of FGSM			
Success rate	93.2% ($\varepsilon = 0.75$)	88.2% ($\varepsilon = 0.05$)	99.5% ($\varepsilon = 0.9$)
Mean L_1 distance	227.21	150.85	1522.31
Mean L_2 distance	12.80	2.74	32.54

As it can be seen, NT-TJSMA is always successful for each model, while FGSM is far from reaching 100% SR. Moreover, NT-TJSMA obtains better L_1 and L_2 scores. Thus our attack outperforms FGSM for the SR and the L_0, L_1 and L_2 metrics, while FGSM has only the L_∞ and speed advantages. Note that for FGSM we considered the best results for different thresholds, while our attack is run one time.

5.2. Conclusions

In this section, we summarise our main findings and also discuss the limitation of our work.

We have introduced WJSMA and TJSMA, new probabilistic adversarial variants of JSMA for targeted and non-targeted misclassification of deep neural network classifiers.

Experiments in the targeted case have demonstrated, after analysing a large amount of images (more than 790,000 images), that our targeted attacks are more efficient and also faster than JSMA. It is important to recall the quite natural derivation of these attacks from a simple and classical log softmax reasoning which has not been noticed before. Our attacks do not beat CW L_0 but have an important speed advantage highlighted in the paper (more than 50 times faster on CIFAR-10). Therefore, for targeted L_0 misclassification, they offer substantial tools to test neural networks in real-time. This fact is supported by our fast implementation provided with the paper.

As a second contribution, we have introduced NT and M variants of WJSMA/TJSMA and have shown that they outperform the previous NT and M versions of JSMA. Through experiments on

GTSRB, we noticed that the gap between our attacks and CW in L_0 average is reduced. Moreover, we obtained better success rates, while remaining at least 40 times faster than CW L_0.

In the NT part of the paper, we did not compare our attacks with the one pixel attack [13]. Indeed, this approach has a high computational cost and we only claim an advantage in speed which is quite evident for us (see also the time evaluation in [12]). Also, we did not provide a comparison with SparseFool [12] an effective NT L_0 attack because of the need to reimplement this attack with TensorFlow. On CIFAR-10, [12] found that crafting an example by SparseFool takes on average 0.34 and 0.69 s on two different neural networks. Our speed performances are very competitive with these values. Indeed, on GTSRB which has many more classes than CIFAR-10, our NT-TJSMA was able to craft an example in near 0.68 s on average (counting only successful images). Thus, regarding SparseFool, we first claim competitive results in speed. Second, our results obtained on LeNet-5 (more than 99.5% on a model similar to [12], see Section 4.1.1) are very close to [12] although we only run the attacks up to a limited maximum number of iterations contrary to [12].

Overall, our results suggest that for adversarial purposes, TJSMA, M-WJSMA and NT-TJSMA should be preferred over the original variants of JSMA, respectively in the case of white-box targeted attacks, white-box non-targeted attacks, and black-box non-targeted attacks. We recall that despite the fact that TJSMA is a more elaborate version of WJSMA, it is hardly compatible with the "Maximal" approach, which in turn proves to be very efficient for non-targeted purposes. For this reason, as we have demonstrated, M-WJSMA is indeed the right choice for this type of attacks. On the other hand, because the "Maximal" approach has not proved to be very efficient on black-box non-targeted attacks, it is the non-targeted version of TJSMA (NT-TJSMA) that is the best in this case.

Finally, we should mention that despite improving JSMA in different ways, like JSMA, our approach is still not scalable to large datasets. This is because of the high computational cost of saliency maps when the dimension of inputs becomes large. Our approach is therefore intended for "small" datasets such as those considered in the paper. Nevertheless, this kind of datasets is very common in real-life applications. Codes are publicly available through the Supplementary Materials.

Supplementary Materials: All our codes are publicly available through the link https://github.com/probabilistic-jsmas/probabilistic-jsmas.

Author Contributions: Conceptualization: T.C., A.L., M.F. and H.H.; Software: T.C., A.L., M.F. and H.H.; Data curation: T.C., A.L. and M.F.; Supervision: H.H.; Writing—original draft: T.C., A.L., M.F. and H.H.; Writing—review and editing: T.C., A.L., M.F. and H.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This work was done in the context of an internship by T.C., A.L. and M.F. supervised by H.H. We thank Gabriel Zeller for his assistance. We are grateful to Wassila Ouerdane and Jean-Philippe Poli at CentraleSupélec for their support. We thank the mesocentre de calcul Fusion, Metz computing center of CentraleSupélec and Stéphane Vialle for providing us effective computing resources. H. Hajri is grateful to Sylvain Lamprier for useful discussions, the scientific direction and the EPI project (Évaluation des Performances de systèmes de décision à base d'Intelligence Artificielle) at IRT SystemX for their support.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

NNC	Neural Network Classifier
JSMA	Jacobian-based Saliency Map Attack
MJSMA	Maximal Jacobian-based Saliency Map Attack
WJSMA	Weighted Jacobian-based Saliency Map Attack
TJSMA	Taylor Jacobian-based Saliency Map Attack
NT	Non-Targeted
M	Maximal
CW	Carlini-Wagner

Appendix A. Architectures of the Deep NNCs

Table A1. Architecture of the used NNC on MNIST (LeNet-5 inspired)

Layer	Parameters
Input Layer	size: (28×28)
Conv2D	kernel size: (5×5), 20 kernels, no stride
ReLu	
MaxPooling2D	kernel size: (2×2), stride: (2×2)
Conv2D	kernel size: (5×5), 50 kernels, no stride
ReLu	
MaxPooling2D	kernel size: (2×2), stride: (2×2)
Flatten	
Dense	size: 500
ReLu	
Dense	size: number of classes (10 for MNIST)
Softmax	

Table A2. Architecture of the used NNC on CIFAR-10

Layer	Parameters
Input Layer	size: (32×32)
Conv2D	kernel size: (3×3), 64 kernels,no stride
ReLu	
Conv2D	kernel size: (3×3), 128 kernels, no stride
ReLu	
MaxPooling2D	kernel size: (2×2), stride: (2×2)
Conv2D	kernel size: (3×3), 128 kernels, no stride
ReLu	
Conv2D	kernel size: (3×3), 256 kernels, no stride
ReLu	
MaxPooling2D	kernel size: (2×2), stride: (2×2)
Conv2D	kernel size: (3×3), 256 kernels, no stride
ReLu	
Conv2D	kernel size: (3×3), 512 kernels, no stride
ReLu	
MaxPooling2D	kernel size: (2×2), stride: (2×2)
Conv2D	kernel size: (3×3), 10 kernels, no stride
GlobalAveragePooling	kernel size: (2×2), stride: (2×2)
Softmax	

Table A3. Architecture of the used NNC on GTSRB (AlexNet inspired)

Layer	**Parameters**
Input Layer	size: (32 × 32)
Conv2D	kernel size: (5 × 5), 64 kernels,no stride
ReLu	
MaxPooling2D	kernel size: (3 × 3), stride: (2 × 2)
Conv2D	kernel size: (5 × 5), 64 kernels,no stride
ReLu	
MaxPooling2D	kernel size: (3 × 3), stride: (2 × 2)
Flatten	
Dense	size: 384
ReLu	
Dense	size: 192
ReLu	
Dense	size: 43
Softmax	

Table A4. Architecture of the used substitute NNC on GTSRB

Layer	**Parameters**
Input Layer	size: (32 × 32 × 3)
Conv2D	kernel size: (3 × 3), 16 kernels,no stride
ReLu	
MaxPooling2D	kernel size: (2 × 2), stride: (2 × 2)
Conv2D	kernel size: (3 × 3), 32 kernels,no stride
ReLu	
MaxPooling2D	kernel size: (2 × 2), stride: (2 × 2)
Conv2D	kernel size: (3 × 3), 64 kernels,no stride
ReLu	
Flatten	
Dense	size: 43
Softmax	

Appendix B. Pairwise Dominance

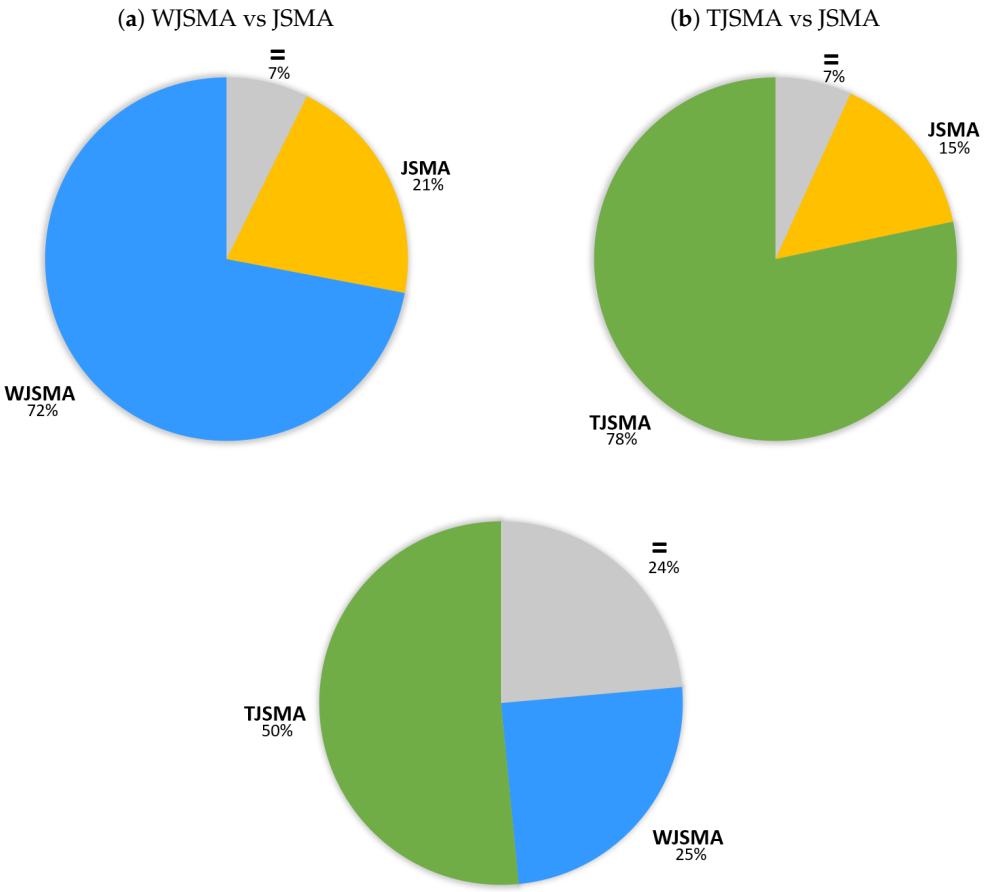


Figure A1. Pairwise dominance on MNIST comparing WJSMA with JSMA (**a**), TJSMA with JSMA (**b**) and TJSMA with WJSMA (**c**). On these charts, "=" corresponds to samples with the same number of iterations by the attacks including when both attacks fail.

Further analysis of the results obtained on MNIST reveals that, even for examples where JSMA is better than WJSMA or TJSMA, on average, less than 10 more components are changed by WJSMA or TJSMA, whereas JSMA changes more than 17 more components on average when it is dominated by WJSMA or TJSMA. A similar gap can be noticed on CIFAR-10.

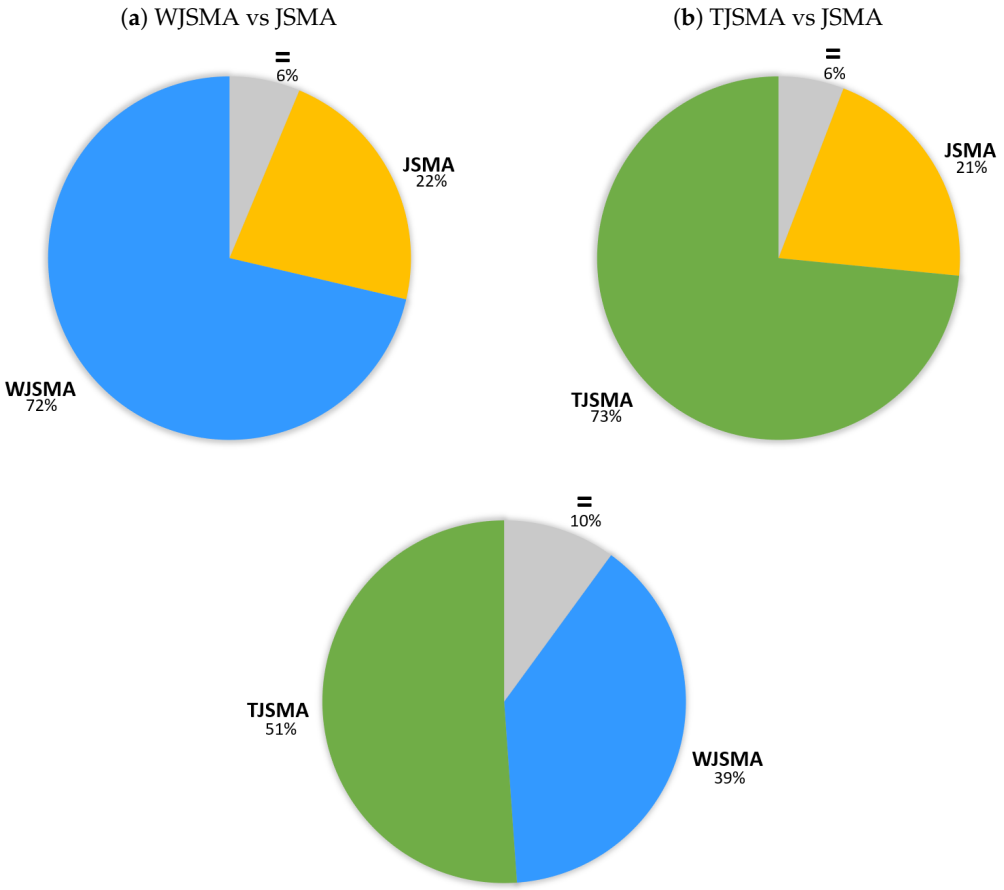


Figure A2. Pairwise dominance on CIFAR-10 comparing WJSMA with JSMA (**a**), TJSMA with JSMA (**b**) and TJSMA with WJSMA (**c**). "=" has the same significance as before.

References

1. Eykholt, K.; Evtimov, I.; Fernandes, E.; Li, B.; Rahmati, A.; Xiao, C.; Prakash, A.; Kohno, T.; Song, D. Robust Physical-World Attacks on Deep Learning Visual Classification. In Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, 18–22 June 2018; pp. 1625–1634.
2. Sitawarin, C.; Bhagoji, A.N.; Mosenia, A.; Chiang, M.; Mittal, P. DARTS: Deceiving Autonomous Cars with Toxic Signs. *arXiv* **2018**, arXiv:1802.06430.
3. Papernot, N.; Song, S.; Mironov, I.; Raghunathan, A.; Talwar, K.; Erlingsson, Ú. Scalable Private Learning with PATE. *arXiv* **2018**, arXiv:1802.08908.
4. Song, L.; Shokri, R.; Mittal, P. Privacy Risks of Securing Machine Learning Models against Adversarial Examples. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, 11–15 November 2019; pp. 241–257. [CrossRef]
5. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. *arXiv* **2015**, arXiv:1412.6572.
6. Kurabin, A.; Goodfellow, I.J.; Bengio, S. Adversarial examples in the physical world. *arXiv* **2017**, arXiv:1607.02533.
7. Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; Vladu, A. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv* **2017**, arXiv:1607.02533.
8. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Berkay Celik, Z.; Swami, A. The limitations of deep learning in adversarial settings. *arXiv* **2015**, arXiv:1511.07528.

9. Moosavi-Dezfooli, S.M.; Fawzi, A.; Frossard, P. Deepfool: A simple and accurate method to fool deep neural networks. *arXiv* **2015**, arXiv:1511.04599.
10. Moosavi-Dezfooli, S.; Fawzi, A.; Fawzi, O.; Frossard, P. Universal Adversarial Perturbations. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, 21–26 July 2017; pp. 86–94. [CrossRef]
11. Carlini, N.; Wagner, D. Towards evaluating the robustness of neural networks. *arXiv* **2017**, arXiv:1608.04644.
12. Modas, A.; Moosavi-Dezfooli, S.; Frossard, P. SparseFool: A Few Pixels Make a Big Difference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, 16–20 June 2019; pp. 9087–9096. [CrossRef]
13. Su, J.; Vargas, D.V.; Sakurai, K. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Trans. Evol. Comput.* **2019**, *23*, 828–841. [CrossRef]
14. Papernot, N.; Faghri, F.; Carlini, N.; Goodfellow, I.J.; Feinman, R.; Kurakin, A.; Xie, C.; Sharma, Y.; Brown, T.H.; Roy, A.; et al. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv* **2016**, arXiv:1610.00768.
15. LeCun, Y.; Cortes, C. MNIST Handwritten Digit Database. 2010. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 10 October 2020).
16. Erba, A.; Taormina, R.; Galelli, S.; Pogliani, M.; Carminati, M.; Zanero, S.; Tippenhauer, N.O. Real-time Evasion Attacks with Physical Constraints on Deep Learning-based Anomaly Detectors in Industrial Control Systems. *arXiv* **2019**, arXiv:1907.07487.
17. Gong, Y.; Li, B.; Poellabauer, C.; Shi, Y. Real-Time Adversarial Attacks. *arXiv* **2019**, arXiv:1905.13399.
18. Lin, J.; Dzeparoska, K.; Zhang, S.Q.; Leon-Garcia, A.; Papernot, N. On the Robustness of Cooperative Multi-Agent Reinforcement Learning. *arXiv* **2020**, arXiv:2003.03722.
19. Parisi, A. *Hands-On Artificial Intelligence for Cybersecurity*; Packt Publishing: Birmingham, UK, 2019.
20. Chio, C.; Freeman, D. *Machine Learning and Security*; Oreilly: Newton, MA, USA, 2018.
21. Sethi, K.; Edupuganti, S.; Kumar, R.; Bera, P.; Madhav, Y. A context-aware robust intrusion detection system: A reinforcement learning-based approach. *Int. J. Inf. Secur.* **2019**, *19*, 657–678. [CrossRef]
22. Deng, J.; Dong, W.; Socher, R.; Li, L.; Li, K.; Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255.
23. Krizhevsky, A.; Nair, V.; Hinton, G. CIFAR-10 (Canadian Institute for Advanced Research). Available online: http://www.cs.toronto.edu/~kriz/cifar.html (accessed on 10 October 2020).
24. Wiyatno, R.; Xu, A. Maximal Jacobian-based Saliency Map Attack. *arXiv* **2018**, arXiv:1808.07945.
25. Stallkamp, J.; Schlipsing, M.; Salmen, J.; Igel, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Netw.* **2012**, *32*, 323–332. [CrossRef]
26. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
27. Papernot, N.; McDaniel, P.D.; Goodfellow, I.J.; Jha, S.; Celik, Z.B.; Swami, A. Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples. *arXiv* **2016**, arXiv:1602.02697.
28. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. Acm* **2017**, *60*, 84–90. [CrossRef]