DeepFool: a simple and accurate method to fool deep neural networks

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard
École Polytechnique Fédérale de Lausanne
{seyed.moosavi,alhussein.fawzi,pascal.frossard} at epfl.ch

Abstract

State-of-the-art deep neural networks have achieved impressive results on many image classification tasks. However, these same architectures have been shown to be unstable to small, well sought, perturbations of the images. Despite the importance of this phenomenon, no effective methods have been proposed to accurately compute the robustness of state-of-the-art deep classifiers to such perturbations on large-scale datasets. In this paper, we fill this gap and propose the DeepFool algorithm to efficiently compute perturbations that fool deep networks, and thus reliably quantify the robustness of these classifiers. Extensive experimental results show that our approach outperforms recent methods in the task of computing adversarial perturbations and making classifiers more robust.[1]

1. Introduction

Deep neural networks are powerful learning models that achieve state-of-the-art pattern recognition performance in many research areas such as bioinformatics [1, 16], speech [12, 6], and computer vision [10, 8]. Though deep networks have exhibited very good performance in classification tasks, they have recently been shown to be particularly unstable to *adversarial* perturbations of the data [18]. In fact, very small and often imperceptible perturbations of the data samples are sufficient to fool state-of-the-art classifiers and result in incorrect classification. (e.g., Figure 1). Formally, for a given classifier, we define an adversarial perturbation as the *minimal* perturbation $\boldsymbol{r}$ that is sufficient to change the estimated label $\hat{k}(\boldsymbol{x})$:

$$\Delta(\boldsymbol{x};\hat{k}) := \min_{\boldsymbol{r}} \|\boldsymbol{r}\|_2 \text{ subject to } \hat{k}(\boldsymbol{x}+\boldsymbol{r}) \neq \hat{k}(\boldsymbol{x}), \quad (1)$$

where $\boldsymbol{x}$ is an image and $\hat{k}(\boldsymbol{x})$ is the estimated label. We call $\Delta(\boldsymbol{x};\hat{k})$ the robustness of $\hat{k}$ at point $\boldsymbol{x}$. The robustness of classifier $\hat{k}$ is then defined as

[1]To encourage reproducible research, the code of DeepFool is made available at `http://github.com/lts4/deepfool`

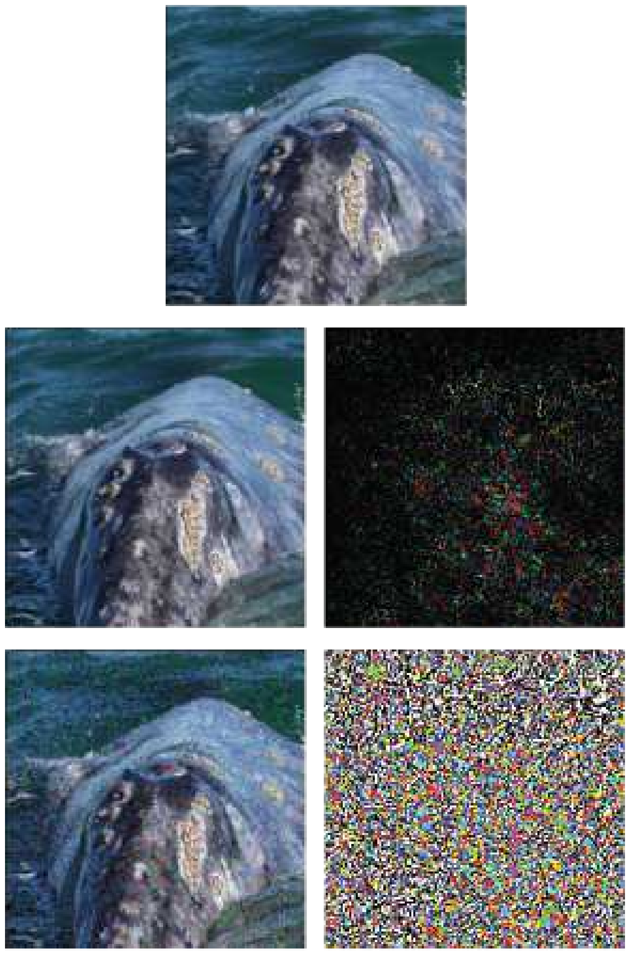


Figure 1: An example of adversarial perturbations. First row: the original image $\boldsymbol{x}$ that is classified as $\hat{k}(\boldsymbol{x})$="whale". Second row: the image $\boldsymbol{x}+\boldsymbol{r}$ classified as $\hat{k}(\boldsymbol{x}+\boldsymbol{r})$="turtle" and the corresponding perturbation $\boldsymbol{r}$ computed by DeepFool. Third row: the image classified as "turtle" and the corresponding perturbation computed by the fast gradient sign method [4]. DeepFool leads to a smaller perturbation.

$$\rho_{\text{adv}}(\hat{k}) = \mathbb{E}_{\boldsymbol{x}} \frac{\Delta(\boldsymbol{x}; \hat{k})}{\|\boldsymbol{x}\|_2}, \qquad (2)$$

where $\mathbb{E}_{\boldsymbol{x}}$ is the expectation over the distribution of data. The study of adversarial perturbations helps us understand what features are used by a classifier. The existence of such examples is seemingly in contradiction with the generalization ability of the learning algorithms. While deep networks achieve state-of-the-art performance in image classification tasks, they are not robust at all to small adversarial perturbations and tend to misclassify minimally perturbed data that looks visually similar to clean samples. Though adversarial attacks are specific to the classifier, it seems that the adversarial perturbations are generalizable across different models [18]. This can actually become a real concern from a security point of view.

An accurate method for finding the adversarial perturbations is thus necessary to study and compare the robustness of different classifiers to adversarial perturbations. It might be the key to a better understanding of the limits of current architectures and to design methods to increase robustness. Despite the importance of the vulnerability of state-of-the-art classifiers to adversarial instability, no well-founded method has been proposed to compute adversarial perturbations and we fill this gap in this paper.

Our main contributions are the following:

- We propose a simple yet accurate method for computing and comparing the robustness of different classifiers to adversarial perturbations.
- We perform an extensive experimental comparison, and show that 1) our method computes adversarial perturbations more reliably and efficiently than existing methods 2) augmenting training data with adversarial examples significantly increases the robustness to adversarial perturbations.
- We show that using imprecise approaches for the computation of adversarial perturbations could lead to *different and sometimes misleading conclusions* about the robustness. Hence, our method provides a better understanding of this intriguing phenomenon and of its influence factors.

We now review some of the relevant work. The phenomenon of adversarial instability was first introduced and studied in [18]. The authors estimated adversarial examples by solving penalized optimization problems and presented an analysis showing that the high complexity of neural networks might be a reason explaining the presence of adversarial examples. Unfortunately, the optimization method employed in [18] is time-consuming and therefore does not scale to large datasets. In [14], the authors showed that convolutional networks are not invariant to some sort of transformations based on the experiments done on Pascal3D+ annotations. Recently, Tsai et al. [19] provided a software to misclassify a given image in a specified class, without necessarily finding the smallest perturbation. Nguyen et al. [13] generated synthetic unrecognizable images, which are classified with high confidence. The authors of [3] also studied a related problem of finding the minimal *geometric* transformation that fools image classifiers, and provided quantitative measure of the robustness of classifiers to geometric transformations. Closer to our work, the authors of [4] introduced the "fast gradient sign" method, which computes the adversarial perturbations for a given classifier very efficiently. Despite its efficiency, this method provides only a coarse approximation of the optimal perturbation vectors. In fact, it performs a unique gradient step, which often leads to sub-optimal solutions. Then in an attempt to build more robust classifiers to adversarial perturbations, [5] introduced a smoothness penalty in the training procedure that allows to boost the robustness of the classifier. Notably, the method in [18] was applied in order to generate adversarial perturbations. We should finally mention that the phenomenon of adversarial instability also led to theoretical work in [2] that studied the problem of adversarial perturbations on some families of classifiers, and provided upper bounds on the robustness of these classifiers. A deeper understanding of the phenomenon of adversarial instability for more complex classifiers is however needed; the method proposed in this work can be seen as a baseline to efficiently and accurately generate adversarial perturbations in order to better understand this phenomenon.

The rest of paper is organized as follows. In Section 2, we introduce an efficient algorithm to find adversarial perturbations in a binary classifier. The extension to the multiclass problem is provided in Section 3. In Section 4, we propose extensive experiments that confirm the accuracy of our method and outline its benefits in building more robust classifiers.

2. DeepFool for binary classifiers

As a multiclass classifier can be viewed as aggregation of binary classifiers, we first propose the algorithm for binary classifiers. That is, we assume here $\hat{k}(\boldsymbol{x}) = \text{sign}(f(\boldsymbol{x}))$, where f is an arbitrary scalar-valued image classification function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. We also denote by $\mathscr{F} \triangleq \{\boldsymbol{x} : f(\boldsymbol{x}) = 0\}$ the level set at zero of f. We begin by analyzing the case where f is an affine classifier $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$, and then derive the general algorithm, which can be applied to any differentiable binary classifier f.

In the case where the classifier f is affine, it can easily

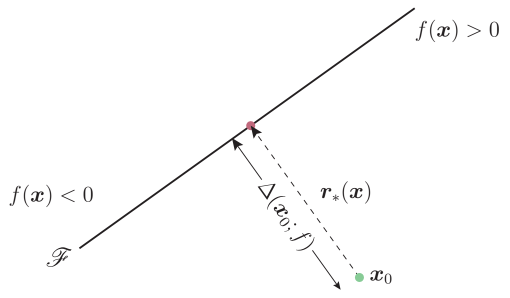


Figure 2: Adversarial examples for a linear binary classifier.

be seen that the robustness of f at point $\boldsymbol{x}_0$, $\Delta(\boldsymbol{x}_0; f)^2$, is equal to the distance from $\boldsymbol{x}_0$ to the separating affine hyperplane $\mathscr{F} = \{\boldsymbol{x} : \boldsymbol{w}^T\boldsymbol{x} + b = 0\}$ (Figure 2). The minimal perturbation to change the classifier's decision corresponds to the orthogonal projection of $\boldsymbol{x}_0$ onto $\mathscr{F}$. It is given by the closed-form formula:

$$\begin{aligned} \boldsymbol{r}_*(\boldsymbol{x}_0) &:= \arg\min \|\boldsymbol{r}\|_2 \\ &\text{subject to } \operatorname{sign}\left(f(\boldsymbol{x}_0 + \boldsymbol{r})\right) \neq \operatorname{sign}(f(\boldsymbol{x}_0)) \\ &= -\frac{f(\boldsymbol{x}_0)}{\|\boldsymbol{w}\|_2^2}\boldsymbol{w}. \end{aligned} \tag{3}$$

Assuming now that f is a general binary differentiable classifier, we adopt an iterative procedure to estimate the robustness $\Delta(\boldsymbol{x}_0; f)$. Specifically, at each iteration, f is linearized around the current point $\boldsymbol{x}_i$ and the minimal perturbation of the linearized classifier is computed as

$$\arg\min_{\boldsymbol{r}_i} \|\boldsymbol{r}_i\|_2 \text{ subject to } f(\boldsymbol{x}_i) + \nabla f(\boldsymbol{x}_i)^T \boldsymbol{r}_i = 0. \tag{4}$$

The perturbation $\boldsymbol{r}_i$ at iteration i of the algorithm is computed using the closed form solution in Eq. (3), and the next iterate $\boldsymbol{x}_{i+1}$ is updated. The algorithm stops when $\boldsymbol{x}_{i+1}$ changes sign of the classifier. The DeepFool algorithm for binary classifiers is summarized in Algorithm 1 and a geometric illustration of the method is shown in Figure 3.

In practice, the above algorithm can often converge to a point on the zero level set $\mathscr{F}$. In order to reach the other side of the classification boundary, the final perturbation vector $\hat{\boldsymbol{r}}$ is multiplied by a constant $1 + \eta$, with $\eta \ll 1$. In our experiments, we have used $\eta = 0.02$.

3. DeepFool for multiclass classifiers

We now extend the DeepFool method to the multiclass case. The most common used scheme for multiclass classifiers is one-vs-all. Hence, we also propose our method based on this classification scheme. In this scheme, the classifier has c outputs where c is the number of classes. Therefore, a classifier can be defined as $f : \mathbb{R}^n \to \mathbb{R}^c$ and the classification is done by the following mapping:

$$\hat{k}(\boldsymbol{x}) = \arg\max_k f_k(\boldsymbol{x}), \tag{5}$$

where $f_k(\boldsymbol{x})$ is the output of $f(\boldsymbol{x})$ that corresponds to the k^{th} class. Similarly to the binary case, we first present the proposed approach for the linear case and then we generalize it to other classifiers.

Algorithm 1 DeepFool for binary classifiers

```
1: input: Image $\boldsymbol{x}$, classifier $f$.
2: output: Perturbation $\hat{\boldsymbol{r}}$.
3: Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
4: while $\operatorname{sign}(f(\boldsymbol{x}_i)) = \operatorname{sign}(f(\boldsymbol{x}_0))$ do
5:     $\boldsymbol{r}_i \leftarrow -\frac{f(\boldsymbol{x}_i)}{\|\nabla f(\boldsymbol{x}_i)\|_2^2}\nabla f(\boldsymbol{x}_i)$,
6:     $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$,
7:     $i \leftarrow i + 1$.
8: end while
9: return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$.
```

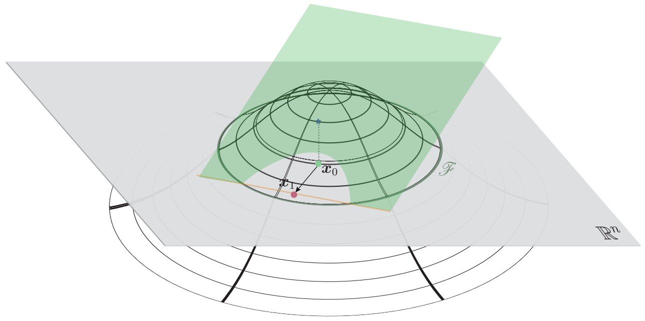


Figure 3: Illustration of Algorithm 1 for $n = 2$. Assume $\boldsymbol{x}_0 \in \mathbb{R}^n$. The green plane is the graph of $\boldsymbol{x} \mapsto f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^T(\boldsymbol{x} - \boldsymbol{x}_0)$, which is tangent to the classifier function (wire-framed graph) $\boldsymbol{x} \mapsto f(\boldsymbol{x})$. The orange line indicates where $f(\boldsymbol{x}_0) + \nabla f(\boldsymbol{x}_0)^T(\boldsymbol{x} - \boldsymbol{x}_0) = 0$. $\boldsymbol{x}_1$ is obtained from $\boldsymbol{x}_0$ by projecting $\boldsymbol{x}_0$ on the orange hyperplane of $\mathbb{R}^n$.

3.1. Affine multiclass classifier

Let $f(\boldsymbol{x})$ be an affine classifier, i.e., $f(\boldsymbol{x}) = \mathbf{W}^\top\boldsymbol{x} + \boldsymbol{b}$ for a given $\mathbf{W}$ and $\boldsymbol{b}$. Since the mapping $\hat{k}$ is the outcome of a one-vs-all classification scheme, the minimal perturbation to fool the classifier can be rewritten as follows

$$\begin{aligned} &\arg\min_{\boldsymbol{r}} \|\boldsymbol{r}\|_2 \\ &\text{s.t. } \exists k : \boldsymbol{w}_k^\top(\boldsymbol{x}_0 + \boldsymbol{r}) + b_k \geq \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}^\top(\boldsymbol{x}_0 + \boldsymbol{r}) + b_{\hat{k}(\boldsymbol{x}_0)}, \end{aligned} \tag{6}$$

[2]From now on, we refer to a classifier either by f or its corresponding discrete mapping $\hat{k}$. Therefore, $\rho_{\text{adv}}(\hat{k}) = \rho_{\text{adv}}(f)$ and $\Delta(\boldsymbol{x}; \hat{k}) = \Delta(\boldsymbol{x}; f)$.

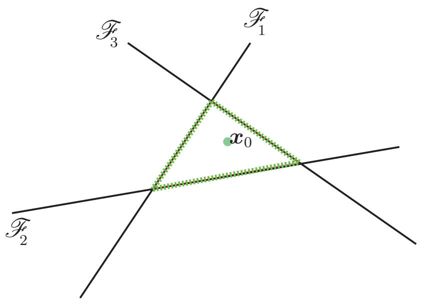


Figure 4: For $\boldsymbol{x}_0$ belonging to class 4, let $\mathscr{F}_k = \{\boldsymbol{x} : f_k(\boldsymbol{x}) - f_4(\boldsymbol{x}) = 0\}$. These hyperplanes are depicted in solid lines and the boundary of P is shown in green dotted line.

where $\boldsymbol{w}_k$ is the k^{th} column of $\mathbf{W}$. Geometrically, the above problem corresponds to the computation of the distance between $\boldsymbol{x}_0$ and the *complement* of the convex polyhedron P,

$$P = \bigcap_{k=1}^{c} \{\boldsymbol{x} : f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}) \geq f_k(\boldsymbol{x})\}, \tag{7}$$

where $\boldsymbol{x}_0$ is located inside P. We denote this distance by $\mathbf{dist}(\boldsymbol{x}_0, P^c)$. The polyhedron P defines the region of the space where f outputs the label $\hat{k}(\boldsymbol{x}_0)$. This setting is depicted in Figure 4. The solution to the problem in Eq. (6) can be computed in closed form as follows. Define $\hat{l}(\boldsymbol{x}_0)$ to be the closest hyperplane of the boundary of P (e.g. $\hat{l}(\boldsymbol{x}_0) = 3$ in Figure 4). Formally, $\hat{l}(\boldsymbol{x}_0)$ can be computed as follows

$$\hat{l}(\boldsymbol{x}_0) = \underset{k \neq \hat{k}(\boldsymbol{x}_0)}{\arg\min} \frac{\left| f_k(\boldsymbol{x}_0) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_0) \right|}{\|\boldsymbol{w}_k - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}\|_2}. \tag{8}$$

The minimum perturbation $\boldsymbol{r}_*(\boldsymbol{x}_0)$ is the vector that projects $\boldsymbol{x}_0$ on the hyperplane indexed by $\hat{l}(\boldsymbol{x}_0)$, i.e.,

$$\boldsymbol{r}_*(\boldsymbol{x}_0) = \frac{\left| f_{\hat{l}(\boldsymbol{x}_0)}(\boldsymbol{x}_0) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_0) \right|}{\|\boldsymbol{w}_{\hat{l}(\boldsymbol{x}_0)} - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}\|_2^2} (\boldsymbol{w}_{\hat{l}(\boldsymbol{x}_0)} - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}). \tag{9}$$

In other words, we find the closest projection of $\boldsymbol{x}_0$ on faces of P.

3.2. General classifier

We now extend the DeepFool algorithm to the general case of multiclass differentiable classifiers. For general non-linear classifiers, the set P in Eq. (7) that describes the region of the space where the classifier outputs label $\hat{k}(\boldsymbol{x}_0)$ is no longer a polyhedron. Following the explained iterative linearization procedure in the binary case, we approximate the set P at iteration i by a polyhedron $\tilde{P}_i$

$$\tilde{P}_i = \bigcap_{k=1}^{c} \Big\{ \boldsymbol{x} : f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i) + \nabla f_k(\boldsymbol{x}_i)^\top \boldsymbol{x} - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)^\top \boldsymbol{x} \leq 0 \Big\}. \tag{10}$$

We then approximate, at iteration i, the distance between $\boldsymbol{x}_i$ and the complement of P, $\mathbf{dist}(\boldsymbol{x}_i, P^c)$, by $\mathbf{dist}(\boldsymbol{x}_i, \tilde{P}_i^c)$. Specifically, at each iteration of the algorithm, the perturbation vector that reaches the boundary of the polyhedron $\tilde{P}_i$ is computed, and the current estimate updated. The method is given in Algorithm 2. It should be noted that the proposed algorithm operates in a greedy way and is not guaranteed to converge to the optimal perturbation in (1). However, we have observed in practice that our algorithm yields very small perturbations which are believed to be good approximations of the minimal perturbation.

It should be noted that the optimization strategy of DeepFool is strongly tied to existing optimization techniques. In the binary case, it can be seen as Newton's iterative algorithm for finding roots of a nonlinear system of equations in the underdetermined case [15]. This algorithm is known as the normal flow method. The convergence analysis of this optimization technique can be found for example in [21]. Our algorithm in the binary case can alternatively be seen as a gradient descent algorithm with an adaptive step size that is automatically chosen at each iteration. The linearization in Algorithm 2 is also similar to a sequential convex programming where the constraints are linearized at each step.

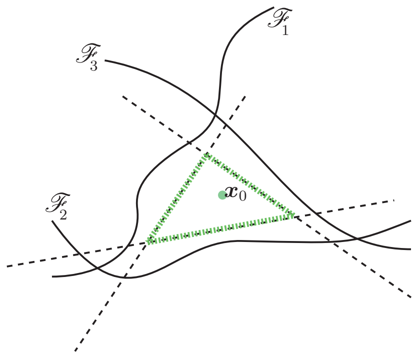


Figure 5: For $\boldsymbol{x}_0$ belonging to class 4, let $\mathscr{F}_k = \{\boldsymbol{x} : f_k(\boldsymbol{x}) - f_4(\boldsymbol{x}) = 0\}$. The linearized zero level sets are shown in dashed lines and the boundary of the polyhedron $\tilde{P}_0$ in green.

3.3. Extension to ℓ_p norm

In this paper, we have measured the perturbations using the ℓ_2 norm. Our framework is however not limited to this choice, and the proposed algorithm can simply be adapted

Algorithm 2 DeepFool: multi-class case

```
1:  input: Image $\boldsymbol{x}$, classifier $f$.
2:  output: Perturbation $\hat{\boldsymbol{r}}$.
3:
4:  Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
5:  while $\hat{k}(\boldsymbol{x}_i) = \hat{k}(\boldsymbol{x}_0)$ do
6:      for $k \neq \hat{k}(\boldsymbol{x}_0)$ do
7:          $\boldsymbol{w}'_k \leftarrow \nabla f_k(\boldsymbol{x}_i) - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
8:          $f'_k \leftarrow f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
9:      end for
10:     $\hat{l} \leftarrow \arg\min_{k \neq \hat{k}(\boldsymbol{x}_0)} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_2}$
11:     $\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_2^2} \boldsymbol{w}'_{\hat{l}}$
12:     $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$
13:     $i \leftarrow i + 1$
14: end while
15: return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$
```

to find minimal adversarial perturbations for any ℓ_p norm ($p \in [1, \infty)$). To do so, the update steps in line 10 and 11 in Algorithm 2 must be respectively substituted by the following updates

$$\hat{l} \leftarrow \underset{k \neq \hat{k}(\boldsymbol{x}_0)}{\arg\min} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_q}, \tag{11}$$

$$\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_q^q} |\boldsymbol{w}'_{\hat{l}}|^{q-1} \odot \operatorname{sign}(\boldsymbol{w}'_{\hat{l}}), \tag{12}$$

where $\odot$ is the pointwise product and $q = \frac{p}{p-1}$.[3] In particular, when $p = \infty$ (i.e., the supremum norm ℓ_∞), these update steps become

$$\hat{l} \leftarrow \underset{k \neq \hat{k}(\boldsymbol{x}_0)}{\arg\min} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_1}, \tag{13}$$

$$\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_1} \operatorname{sign}(\boldsymbol{w}'_{\hat{l}}). \tag{14}$$

4. Experimental results

4.1. Setup

We now test our DeepFool algorithm on deep convolutional neural networks architectures applied to MNIST, CIFAR-10, and ImageNet image classification datasets. We consider the following deep neural network architectures:

- **MNIST:** A two-layer fully connected network, and a two-layer LeNet convoluational neural network architecture [9]. Both networks are trained with SGD with momentum using the MatConvNet [20] package.
- **CIFAR-10:** We trained a three-layer LeNet architecture, as well as a Network In Network (NIN) architecture [11].
- **ILSVRC 2012:** We used CaffeNet [7] and GoogLeNet [17] pre-trained models.

In order to evaluate the robustness to adversarial perturbations of a classifier f, we compute the average robustness $\hat{\rho}_{\text{adv}}(f)$, defined by

$$\hat{\rho}_{\text{adv}}(f) = \frac{1}{|\mathscr{D}|} \sum_{\boldsymbol{x} \in \mathscr{D}} \frac{\|\hat{\boldsymbol{r}}(\boldsymbol{x})\|_2}{\|\boldsymbol{x}\|_2}, \tag{15}$$

where $\hat{\boldsymbol{r}}(\boldsymbol{x})$ is the estimated minimal perturbation obtained using DeepFool, and $\mathscr{D}$ denotes the test set[4].

We compare the proposed DeepFool approach to state-of-the-art techniques to compute adversarial perturbations in [18] and [4]. The method in [18] solves a series of penalized optimization problems to find the minimal perturbation, whereas [4] estimates the minimal perturbation by taking the sign of the gradient

$$\hat{\boldsymbol{r}}(\boldsymbol{x}) = \epsilon \operatorname{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right),$$

with J the cost used to train the neural network, $\boldsymbol{\theta}$ is the model parameters, and y is the label of $\boldsymbol{x}$. The method is called *fast gradient sign method*. In practice, in the absence of general rules to choose the parameter ϵ, we chose the smallest ϵ such that 90% of the data are misclassified after perturbation.[5]

4.2. Results

We report in Table 1 the accuracy and average robustness $\hat{\rho}_{\text{adv}}$ of each classifier computed using different methods. We also show the running time required for each method to compute *one* adversarial sample. It can be seen that DeepFool estimates smaller perturbations (hence closer to minimal perturbation defined in (1)) than the ones computed using the competitive approaches. For example, the average perturbation obtained using DeepFool is 5 times lower than the one estimated with [4]. On the ILSVRC2012 challenge dataset, the average perturbation is one order of magnitude smaller compared to the fast gradient method. It should be noted moreover that the proposed approach also yields slightly smaller perturbation vectors than the method in [18]. The proposed approach is hence more accurate in detecting directions that can potentially fool neural networks. As a result, DeepFool can be used as a valuable tool to accurately assess the robustness of classifiers. On

[3]To see this, one can apply Holder's inequality to obtain a lower bound on the ℓ_p norm of the perturbation.

[4]For ILSVRC2012, we used the validation data.

[5]Using this method, we observed empirically that one cannot reach 100% misclassification rate on some datasets. In fact, even by increasing ϵ to be very large, this method can fail in misclassifying all samples.

Classifier	**Test error**	$\hat{\rho}_{\text{adv}}$ [DeepFool]	time	$\hat{\rho}_{\text{adv}}$ [4]	time	$\hat{\rho}_{\text{adv}}$ [18]	time
LeNet (MNIST)	1%	2.0×10^{-1}	110 ms	1.0	20 ms	2.5×10^{-1}	> 4 s
FC500-150-10 (MNIST)	1.7%	1.1×10^{-1}	50 ms	3.9×10^{-1}	10 ms	1.2×10^{-1}	> 2 s
NIN (CIFAR-10)	11.5%	2.3×10^{-2}	1100 ms	1.2×10^{-1}	180 ms	2.4×10^{-2}	>50 s
LeNet (CIFAR-10)	22.6%	3.0×10^{-2}	220 ms	1.3×10^{-1}	50 ms	3.9×10^{-2}	>7 s
CaffeNet (ILSVRC2012)	42.6%	2.7×10^{-3}	510 ms*	3.5×10^{-2}	50 ms*	-	-
GoogLeNet (ILSVRC2012)	31.3%	1.9×10^{-3}	800 ms*	4.7×10^{-2}	80 ms*	-	-

Table 1: The adversarial robustness of different classifiers on different datasets. The time required to compute one sample for each method is given in the time columns. The times are computed on a Mid-2015 MacBook Pro without CUDA support. The asterisk marks determines the values computed using a GTX 750 Ti GPU.

the complexity aspect, the proposed approach is substantially faster than the standard method proposed in [18]. In fact, while the approach [18] involves a costly minimization of a series of objective functions, we observed empirically that DeepFool converges in a few iterations (i.e., less than 3) to a perturbation vector that fools the classifier. Hence, the proposed approach reaches a more accurate perturbation vector compared to state-of-the-art methods, while being computationally efficient. This makes it readily suitable to be used as a baseline method to estimate the robustness of very deep neural networks on large-scale datasets. In that context, we provide the first quantitative evaluation of the robustness of state-of-the-art classifiers on the large-scale ImageNet dataset. It can be seen that despite their very good test accuracy, these methods are extremely unstable to adversarial perturbations: a perturbation that is 1000 smaller in magnitude than the original image is sufficient to fool state-of-the-art deep neural networks.

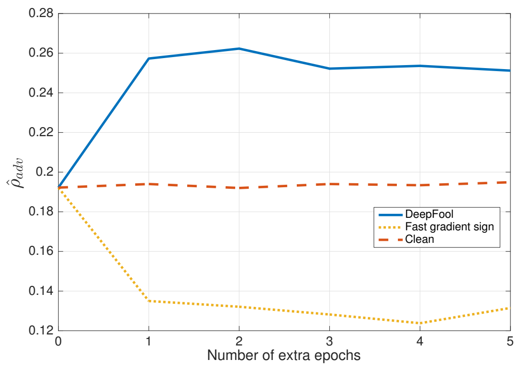
We illustrate in Figure 1 perturbed images generated by the fast gradient sign and DeepFool. It can be observed that the proposed method generates adversarial perturbations which are hardly perceptible, while the fast gradient sign method outputs a perturbation image with higher norm.

It should be noted that, when perturbations are measured using the ℓ_∞ norm, the above conclusions remain unchanged: DeepFool yields adversarial perturbations that are smaller (hence closer to the optimum) compared to other methods for computing adversarial examples. Table 2 reports the ℓ_∞ robustness to adversarial perturbations measured by $\hat{\rho}^{\infty}_{\text{adv}}(f) = \frac{1}{|\mathscr{D}|}\sum_{\boldsymbol{x}\in\mathscr{D}} \frac{\|\hat{\boldsymbol{r}}(\boldsymbol{x})\|_\infty}{\|\boldsymbol{x}\|_\infty}$, where $\hat{\boldsymbol{r}}(\boldsymbol{x})$ is computed respectively using DeepFool (with $p = \infty$, see Section 3.3), and the Fast gradient sign method for MNIST and CIFAR-10 tasks.

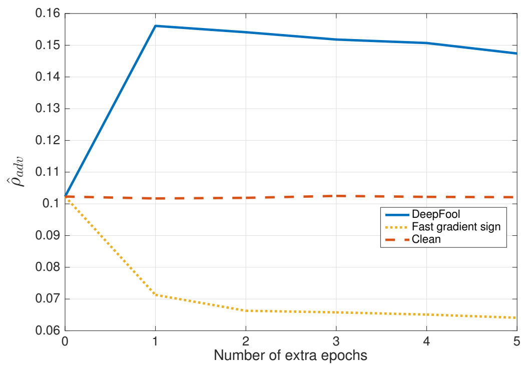
Classifier	DeepFool	Fast gradient sign
LeNet (MNIST)	0.10	0.26
FC500-150-10 (MNIST)	0.04	0.11
NIN (CIFAR-10)	0.008	0.024
LeNet (CIFAR-10)	0.015	0.028

Table 2: Values of $\hat{\rho}^{\infty}_{\text{adv}}$ for four different networks based on DeepFool (smallest l_∞ perturbation) and fast gradient sign method with 90% of misclassification.

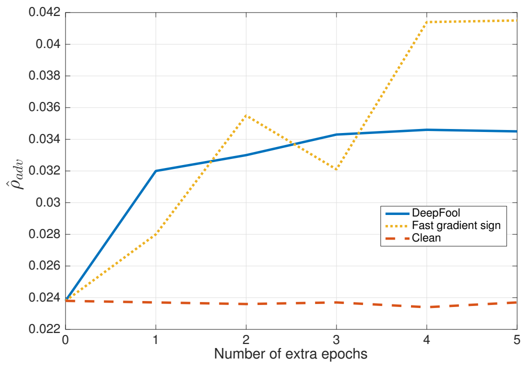
Fine-tuning using adversarial examples In this section, we fine-tune the networks of Table 1 on adversarial examples to build more robust classifiers for the MNIST and CIFAR-10 tasks. Specifically, for each network, we performed two experiments: (i) Fine-tuning the network on DeepFool's adversarial examples, (ii) Fine-tuning the network on the fast gradient sign adversarial examples. We fine-tune the networks by performing 5 additional epochs, with a 50% decreased learning rate only on the perturbed training set. For each experiment, the same training data was used through all 5 extra epochs. For the sake of completeness, we also performed 5 extra epochs on the original data. The evolution of $\hat{\rho}_{\text{adv}}$ for the different fine-tuning strategies is shown in Figures 6a to 6d, *where the robustness $\hat{\rho}_{adv}$ is estimated using DeepFool*, since this is the most accurate method, as shown in Table 1. Observe that fine-tuning with DeepFool adversarial examples significantly increases the robustness of the networks to adversarial perturbations even after one extra epoch. For example, the robustness of the networks on MNIST is improved by 50% and NIN's robustness is increased by about 40%. On the other hand, quite surprisingly, the method in [4] can lead to *a decreased* robustness to adversarial perturbations of the network. We hypothesize that this behavior is due to the fact that perturbations estimated using the fast gradient sign method are much larger than minimal adversarial per-



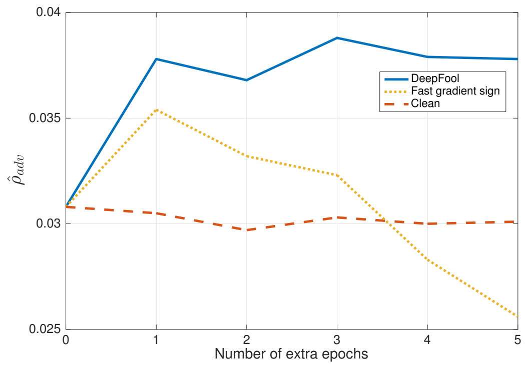
(a) Effect of fine-tuning on adversarial examples computed by two different methods for LeNet on MNIST.



(b) Effect of fine-tuning on adversarial examples computed by two different methods for a fully-connected network on MNIST.



(c) Effect of fine-tuning on adversarial examples computed by two different methods for NIN on CIFAR-10.



(d) Effect of fine-tuning on adversarial examples computed by two different methods for LeNet on CIFAR-10.

Figure 6

turbations. Fine-tuning the network with overly perturbed images decreases the robustness of the networks to adversarial perturbations. To verify this hypothesis, we compare in Figure 7 the adversarial robustness of a network that is fine-tuned with the adversarial examples obtained using DeepFool, where norms of perturbations have been deliberately multiplied by $\alpha = 1, 2, 3$. Interestingly, we see that by magnifying the norms of the adversarial perturbations, the robustness of the fine-tuned network is *decreased*. This might explain why overly perturbed images decrease the robustness of MNIST networks: these perturbations can really change the class of the digits, hence fine-tuning based on these examples can lead to a drop of the robustness (for an illustration, see Figure 8). This lends credence to our hypothesis, and further shows the importance of designing accurate methods to compute minimal perturbations.

Table 3 lists the accuracies of the fine-tuned networks. It can be seen that fine-tuning with DeepFool can improve the accuracy of the networks. Conversely, fine-tuning with the approach in [4] has led to a *decrease* of the test accuracy in all our experiments. This confirms the explanation that the fast gradient sign method outputs *overly perturbed* images

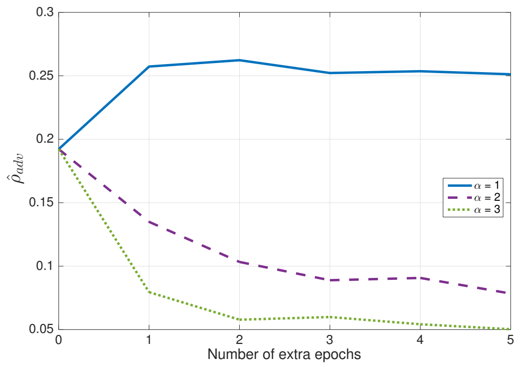


Figure 7: Fine-tuning based on magnified DeepFool's adversarial perturbations.

that lead to images that are unlikely to occur in the test data. Hence, it *decreases* the performance of the method as it acts as a regularizer that does not represent the distribution of the original data. This effect is analogous to geometric data augmentation schemes, where *large* transformations of the original samples have a counter-productive effect on gener-

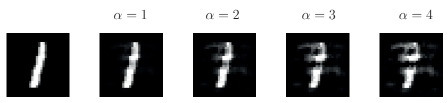


Figure 8: From "1" to "7" : original image classified as "1" and the DeepFool perturbed images classified as "7" using different values of α.

Classifier	DeepFool	Fast gradient sign	Clean
LeNet (MNIST)	0.8%	4.4%	1%
FC500-150-10 (MNIST)	1.5%	4.9%	1.7%
NIN (CIFAR-10)	11.2%	21.2%	11.5%
LeNet (CIFAR-10)	20.0%	28.6%	22.6%

Table 3: The test error of networks after the fine-tuning on adversarial examples (after five epochs). Each columns correspond to a different type of augmented perturbation.

alization.[6]

To emphasize the importance of a correct estimation of the minimal perturbation, we now show that using approximate methods can lead to wrong conclusions regarding the adversarial robustness of networks. We fine-tune the NIN classifier on the fast gradient sign adversarial examples. We follow the procedure described earlier but this time, we decreased the learning rate by 90%. We have evaluated the adversarial robustness of this network at different extra epochs using DeepFool and the *fast gradient sign method*. As one can see in Figure 9, the red plot exaggerates the effect of training on the adversarial examples. Moreover, it is not sensitive enough to demonstrate the loss of robustness at the first extra epoch. These observations confirm that using an *accurate* tool to measure the robustness of classifiers is crucial to derive conclusions about the robustness of networks.

5. Conclusion

In this work, we proposed an algorithm, DeepFool, to compute adversarial examples that fool state-of-the-art classifiers. It is based on an iterative linearization of the classifier to generate minimal perturbations that are sufficient to change classification labels. We provided extensive experimental evidence on three datasets and eight classifiers, showing the superiority of the proposed method over state-of-the-art methods to compute adversarial perturbations, as well as the efficiency of the proposed approach. Due to

[6]While the authors of [4] reported an *increased* generalization performance on the MNIST task (from 0.94% to 0.84%) using adversarial regularization, it should be noted that the their experimental setup is significantly different as [4] trained the network based on a modified cost function, while we performed straightforward fine-tuning.

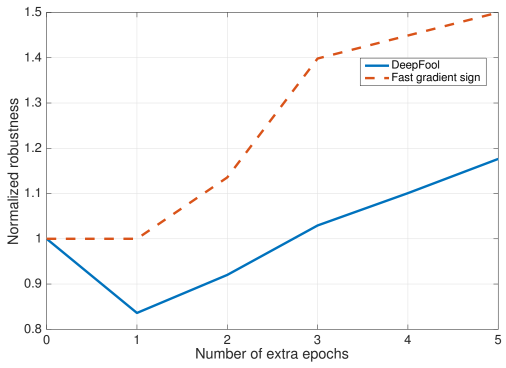


Figure 9: How the adversarial robustness is judged by different methods. The values are normalized by the corresponding $\hat{\rho}_{\text{adv}}$s of the original network.

its accurate estimation of the adversarial perturbations, the proposed DeepFool algorithm provides an efficient and accurate way to evaluate the robustness of classifiers and to enhance their performance by proper fine-tuning. The proposed approach can therefore be used as a reliable tool to accurately estimate the minimal perturbation vectors, and build more robust classifiers.

References

[1] D. Chicco, P. Sadowski, and P. Baldi. Deep autoencoder neural networks for gene ontology annotation predictions. In *ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 533–540, 2014.

[2] A. Fawzi, O. Fawzi, and P. Frossard. Analysis of classifiers' robustness to adversarial perturbations. *CoRR*, abs/1502.02590, 2015.

[3] A. Fawzi and P. Frossard. Manitest: Are classifiers really invariant? In *British Machine Vision Conference (BMVC)*, pages 106.1–106.13, 2015.

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[5] S. Gu and L. Rigazio. Towards deep neural network architectures robust to adversarial examples. *CoRR*, abs/1412.5068, 2014.

[6] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97, 2012.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM International Conference on Multimedia (MM)*, pages 675–678. ACM, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, pages 1097–1105, 2012.

[9] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. 1999.

[10] Y. LeCun, K. Kavukcuoglu, C. Farabet, et al. Convolutional networks and applications in vision. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 253–256, 2010.

[11] M. Lin, Q. Chen, and S. Yan. Network in network. 2014.

[12] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ. Strategies for training large scale neural network language models. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 196–201, 2011.

[13] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.

[14] B. Pepik, R. Benenson, T. Ritschel, and B. Schiele. What is holding back convnets for detection? In *Pattern Recognition*, pages 517–528. Springer, 2015.

[15] A. P. Ruszczyński. *Nonlinear optimization*, volume 13. Princeton university press, 2006.

[16] M. Spencer, J. Eickholt, and J. Cheng. A deep learning network approach to ab initio protein secondary structure prediction. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 12(1):103–112, 2015.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[18] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[19] C.-Y. Tsai and D. Cox. Are deep learning algorithms easily hackable? `http://coxlab.github.io/ostrichinator`.

[20] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *ACM International Conference on Multimedia (MM)*, pages 689–692, 2015.

[21] H. F. Walker and L. T. Watson. Least-change secant update methods for underdetermined systems. *SIAM Journal on numerical analysis*, 27(5):1227–1262, 1990.