IMA Journal of Applied Mathematics (2024) **89**, 175–196
https://doi.org/10.1093/imamat/hxad017
Advance Access publication on 28 June 2023

Adversarial ink: componentwise backward error attacks on deep learning

Lucas Beerens and Desmond J. Higham*
School of Mathematics and The Maxwell Institute for Mathematical Sciences, University of Edinburgh, Edinburgh EH8 9BT, UK
*Corresponding author: d.j.higham@ed.ac.uk

[Received on 28 October 2022; revised on 05 June 2023; accepted on 12 June 2023]

Deep neural networks are capable of state-of-the-art performance in many classification tasks. However, they are known to be vulnerable to adversarial attacks—small perturbations to the input that lead to a change in classification. We address this issue from the perspective of backward error and condition number, concepts that have proved useful in numerical analysis. To do this, we build on the work of Beuzeville, T., Boudier, P., Buttari, A., Gratton, S., Mary, T. and Pralet S. (2021) Adversarial attacks via backward error analysis. hal-03296180, version 3. In particular, we develop a new class of attack algorithms that use componentwise relative perturbations. Such attacks are highly relevant in the case of handwritten documents or printed texts where, for example, the classification of signatures, postcodes, dates or numerical quantities may be altered by changing only the ink consistency and not the background. This makes the perturbed images look natural to the naked eye. Such 'adversarial ink' attacks therefore reveal a weakness that can have a serious impact on safety and security. We illustrate the new attacks on real data and contrast them with existing algorithms. We also study the use of a componentwise condition number to quantify vulnerability.

Keywords: misclassification; stability; conditioning; optimization.

1. Motivation

Over the past decade it has become clear that state of the art deep learning image classification tools are susceptible to adversarial attacks—deliberately constructed perturbations that are intended to go unnoticed by humans but cause a change in the predicted class (Szegedy *et al.*, 2013; Goodfellow *et al.*, 2015). This type of vulnerability is of concern in high stakes application areas, including medical imaging, transport, defence and finance (Marcus, 2018). Consequently there has been a great deal of interest in the design of practical attack and defence strategies (Moosavi-Dezfooli *et al.*, 2016; Papernot *et al.*, 2017; Akhtar & Mian, 2018; Goodfellow *et al.*, 2018; Madry *et al.*, 2018) and, more recently, in theoretical questions concerning the existence and computability of adversarial perturbations (Fawzi *et al.*, 2018; Shafahi *et al.*, 2019; Tyukin *et al.*, 2020; Bastounis *et al.*, 2021; Tyukin *et al.*, 2021).

From the perspective of applied and computational mathematics, the fundamental question to be addressed here concerns well-posedness, or *conditioning*. In particular, *backward error* theory from numerical analysis is highly pertinent. In Beuzeville *et al.* (2021), the authors used the concept of backward error to construct a new form of adversarial attack algorithm. In this work, we build on this idea by focusing on a special class of data perturbation. We develop attack strategies based on *componentwise relative* perturbations; for example, each pixel may be perturbed by a small percentage of its original value. In particular, this approach allows us to preserve the background of a document and perturb only the ink levels in the text. We also test the corresponding condition number as an indicator of vulnerability to attack.

(a) Image from MNIST dataset [16], which is correctly classified as a 7 by a neural network.

(b) Perturbed image using DeepFool [19]. The background is now smeared. This image is classified as an 8.

(c) Perturbed image using Algorithm 4 from section 4. The background is unchanged. This image is classified as an 8.

FIG. 1. Image from MNIST and two adversarial attacks.

To illustrate the main idea, Figure 1 (a) shows a handwritten digit from the MNIST data set (LeCun & Cortes, 2010), and Figure 1 (b) and (c) show adversarial attacks on a trained network described in Section 4. The original image (a) is correctly classified as a 7 by the network. The perturbed images in (b) and (c) are both classified as an 8. For (b), we attacked with the DeepFool algorithm (Moosavi-Dezfooli *et al.*, 2016), which controls the Euclidean norm of the perturbation. Although the image still has the appearance of a 7, we see that the background, where pixels had a value of zero, is noticeably altered. For (c), we computed a componentwise perturbation with the new Algorithm 4 from section 4 and we see that the background is unchanged. The componentwise perturbation in part (c) is compatible with a blotchy pen, imperfect paper or inconsistent handwriting pressure. Indeed, variations in line continuity, line quality and pen control have been widely observed, and are listed among the 21 discriminating elements of handwriting (Huber & Headrick, 1999; Harrison *et al.*, 2009). Hence we argue that this type of 'adversarial ink' attack produces a more natural result than the perturbation in part (b).

The manuscript is organized as follows. In Section 2, we set up some notation and introduce the concept of backward error. Section 3 describes the adversarial attack algorithms in Beuzeville *et al.* (2021), and Section 4 extends these to the componentwise setting. Computational results on the MNIST data set are presented in Section 5; we show illustrative images, summarize the perturbation sizes, consider untargeted attacks, compare against state-of-the-art algorithms, summarize the most likely class changes, look at the use of a condition number to indicate vulnerability to attack, test on different neural networks, and report on black box attacks. Conclusions are given in Section 6.

2. Image classification and backward error

We begin by considering a general image classifier, in the form of a map $F : [0, 1]^n \to \mathbb{R}^c$. Hence, we regard an image as a single vector in $\mathbb{R}^n$. The n components may correspond to individual pixel intensities in the greyscale case, or red, green and blue channel pixel intensities in the colour case. Intensities are assumed to lie in $[0, 1]$. Each image is assigned to one of c classes, according to the largest component of $F(x)$. In practice, the output vector $y = F(x) \in \mathbb{R}^c$ may be passed through a softmax function, so that

$$\frac{e^{y_i}}{\sum_{j=1}^{c} e^{y_j}}$$

is viewed as the probability that x belongs to class i. Since x is assigned to the most likely class, we do not need to include this final layer when considering the classification results.

In numerical analysis, the concept of backward error deals with the following question: given an approximate solution, what is the size of the smallest perturbation to the input which makes this solution exact? In more detail, suppose an approximation algorithm produces a function $\widehat{H}$, instead of the desired function H. For an input x, when the algorithm returns $\widehat{H}(x) = y + \Delta y$ instead of $y = H(x)$, we may ask for the smallest Δx such that $H(x + \Delta x) = y + \Delta y$. In many settings, the size of Δx (the backward error) is more relevant and more amenable to analysis than the size of Δy (the forward error) (Higham, 2002; Corless & Fillion, 2013). For an adversarial attack on a classifier, we may interpret Δy as a *desired* change in the output. Then a question of the same structure arises—what is the smallest perturbation to the input that achieves the desired output? In this setting, we require Δx such that $F(x + \Delta x) = y + \Delta y$.

This approach was exploited in Beuzeville *et al.* (2021), leading to what we describe as Algorithms 1 and 2 in subsections 3.2 and 3.3.

3. Normwise backward error attacks: algorithms 1 and 2

3.1 *Set-up*

In the next two subsections, we describe the data perturbation approach from Beuzeville *et al.* (2021); leading to Algorithms 1 and 2. We cover this existing work in sufficient detail that (a) there are well-defined algorithms that can be implemented in practice, and (b) the new versions, Algorithms 3 and 4 in Section 4, can be introduced naturally and compared computationally.

We formulate all algorithms in terms of linearly constrained linear least-squares problems, for which high quality software is available. Letting $\|\cdot\|_2$ denote the Euclidean norm, these problems have the form

$$\min_z \|C_1 z - k_1\|_2, \quad \text{such that} \quad \begin{cases} C_2 z \leq k_2, \\ C_3 z = k_3, \end{cases} \tag{3.1}$$

where the matrices C_1, C_2, C_3 and vectors z, k_1, k_2, k_3 have appropriate dimensions and where vector equalities and inequalities are to be interpreted in a componentwise sense. (More traditionally, the objective function in (3.1) may be written $\frac{1}{2}\|C_1 z - k_1\|_2^2$, but, of course, the factor $\frac{1}{2}$ and the square may be ignored.) We also assume for now that the Jacobian of the classification map is available; in subsection 5.6, we test the use of a finite-difference approximation to the Jacobian.

3.2 *Linearized algorithm*

We begin by measuring perturbations in the Euclidean norm. Given an image x with $F(x) = y$ and a desired new output $\widehat{y}$, a suitable perturbation may be expressed as

$$\arg\min_{\Delta x} \{\|\Delta x\|_2 \ : \ F(x + \Delta x) = \widehat{y}\}. \tag{3.2}$$

In general, this problem cannot be solved analytically. On the grounds that we are looking for a small perturbation, it is reasonable to linearize, using $F(x + \Delta x) - F(x) \approx \mathscr{A}\Delta x$, where $\mathscr{A} \in \mathbb{R}^{c \times n}$ is the Jacobian of F at x, and F is assumed to be differentiable in a neighbourhood of x. The problem (3.2) then

reduces to

$$\arg\min_{\Delta x} \left\{ \|\Delta x\|_2 \ : \ \mathscr{A}\Delta x = \widehat{y} - y \right\}. \tag{3.3}$$

For any fixed $\widehat{y}$ this is a minimum Euclidean norm linear system. Typically $c \ll n$, so the system is underdetermined. Generically, a solution for this fixed $\widehat{y}$ can be found by introducing the Moore-Penrose inverse (Wang *et al.*, 2018), $\mathscr{A}^\dagger$, to give

$$\arg\min_{\Delta x} \left\{ \|\Delta x\|_2 : \mathscr{A}\Delta x = \widehat{y} - y \right\} = \mathscr{A}^\dagger(\widehat{y} - y). \tag{3.4}$$

Given the solution (3.4), we can use $\widehat{y}$ as an optimization variable. In the targeted case, where we wish the perturbed image to be classified with label c_0, we introduce the misclassification set

$$\mathscr{S} := \{\widehat{y} \in \mathbb{R}^c \ : \ \widehat{y}_{c_0} = \max_{1 \le i \le c} \widehat{y}_i\}. \tag{3.5}$$

To compute an adversarial attack we then solve

$$\arg\min_{\hat{y} \in \mathscr{S}} \|\mathscr{A}^\dagger(\hat{y} - y)\|_2, \tag{3.6}$$

and set $\Delta x = \mathscr{A}^\dagger(\widehat{y} - y)$.

We now show that the problem (3.5)–(3.6) has the form (3.1). The variable to be optimized is $\widehat{y}$, so we will use $z = \hat{y}$ in (3.1). We may take $C_1 = C_1^{[1]} := \mathscr{A}^\dagger$ and $k_1 = k_1^{[1]} := \mathscr{A}^\dagger y$. The misclassification condition is equivalent to

$$\hat{y} - \begin{pmatrix} \widehat{y}_{c_0} \\ \vdots \\ \widehat{y}_{c_0} \end{pmatrix} \le 0.$$

In order to write this in matrix form, we define the matrix $G \in \mathbb{R}^{c \times c}$ with

$$G_{ij} = \begin{cases} 1 & \text{if } j = c_0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

The misclassification condition may then be written

$$(I - G)\hat{y} \le 0.$$

Hence we use $C_2 = C_2^{[1]} := I - G$ and $k_2 = k_2^{[1]} = 0$. There are no equality conditions, so we set $C_3 = C_3^{[1]} := 0$ and $k_3 = k_3^{[1]} := 0$.

This leads to Algorithm 1, as summarized in the displayed pseudocode listing.

Algorithm 1 Basic normwise attack, returning a perturbed image

```
procedure ATTACK(F, x, c_0)
    out ← F(x)
    jac ← Jacobian(F, x)
    pi ← PseudoInverse(jac)
    con ← Constraints(c_0)
    obj ← Objective(pi, out)
    z ← OptimizationVariable
    prob ← Minimize(z, obj, con)
    Solve(prob)
    Δy ← z.value − out
    Δx ← pi · Δy
    return Scale(F, x, Δx, c_0)
end procedure
```

The final step of Algorithm 1 requires further explanation. We note that the constrained linear least-squares problem is not guaranteed to produce a perturbed image with values in $[0, 1]$. For this reason, we prune the entries using

$$\mathrm{Prune}_i(x) = \begin{cases} 0 & \text{if } x_i < 0 \\ x_i & \text{if } x_i \in [0,1] \\ 1 & \text{if } x_i > 1. \end{cases}$$

We also note that the resulting Δx might be unsuccessful; that is, $F(x+\Delta x)$ might not correspond to class c_0. Hence, we regard Δx from (3.6) as a *direction* in which to perturb, and take the smallest increment that results in class c_0. So, the Scale function in Algorithm 1 is defined as

$$\mathrm{Scale}(F, x, \Delta x, c_0) := x + \min\{a \in \mathbb{R} : \arg\max_i F(\mathrm{Prune}(x + a\,\Delta x))_i = c_0\}\,\Delta x. \tag{3.8}$$

This minimization is carried out by computing $\arg\max_i F(\mathrm{Prune}(x + a\,\Delta x))_i$ over a finely spaced range of a values in $(0, \|x\|_2/\|\Delta x\|_2]$. If there is no suitable value of a in this range then we terminate and regard the attack as unsuccessful. We use this range because for larger a the norm of the perturbation before pruning exceeds the norm of the image, at which point it is reasonable to assume that the perturbation is too large to be of interest.

3.3 *Iterative algorithm*

An alternative approach was also proposed in Beuzeville *et al.* (2021). This may be motivated by two ideas.

- Do not exploit the analytical solution of the linearized problem (3.4), and proceed directly with numerical optimization. This allows us to build in constraints that keep all pixel values in the range $[0, 1]$.
- Given that we have linearized the problem, take small steps and proceed iteratively.

The algorithm uses a hyperparameter α indicating the step size for each iteration. Again we will target some class c_0. We start with the perturbation $\Delta x = 0$ and update it with a small multiple of some new δx in every step. At every step, we also have $x_{\text{new}} = x + \Delta x$ and $y_{\text{new}} = F(x_{\text{new}})$. In each of these steps, we solve

$$\arg\min_{\delta x} \|\Delta x + \delta x\|_2, \tag{3.9}$$

under the misclassification condition $\widehat{y} \in \mathscr{S}$, the condition that $\mathscr{A}\delta x = \widehat{y} - y_{\text{new}}$, and the constraint that pixel values lie in the unit interval. Since we now have constraints on both $\widehat{y}$ and δx, we treat them both as optimization variables. Then we update Δx and x_{new} by adding $\alpha\delta x$ to both. Finally $\mathscr{A}$ is recomputed before moving on to the next iteration.

To see that we still have constrained linear least-squares problems of the form (3.1), note that we must repeatedly solve (3.9) under the conditions that $\mathscr{A}\delta x = \widehat{y} - y_{\text{new}}$ and $\widehat{y} \in \mathscr{S}$. Since both δx and $\widehat{y}$ need to be optimized we use $z = \begin{bmatrix} \delta x \\ \widehat{y} \end{bmatrix}$ in (3.1). Thus we use $C_1 = C_1^{[2]} := \begin{bmatrix} I, 0 \end{bmatrix}$ and $k_1 = k_1^{[2]} := -\Delta x$. For the inequality constraints we need to consider the misclassification constraint and the pixel value bound constraint. Keeping in mind that z also includes δx we obtain $\begin{bmatrix} 0, I - G \end{bmatrix} z \leq 0$, for G in (3.7). Pixel values must also lie in the unit interval. This constraint may be written as

$$\begin{bmatrix} I & 0 \\ -I & 0 \end{bmatrix} z \leq \begin{bmatrix} \mathbf{1} - x_{\text{new}} \\ x_{\text{new}} \end{bmatrix},$$

where $\mathbf{1}$ denotes a vector of 1s. Combining the two inequality conditions gives

$$C_2 = C_2^{[2]} := \begin{bmatrix} 0 & I - G \\ I & 0 \\ -I & 0 \end{bmatrix} \quad \text{and} \quad k_2 = k_2^{[2]} := \begin{bmatrix} 0 \\ \mathbf{1} - x_{\text{new}} \\ x_{\text{new}} \end{bmatrix}.$$

The required equality condition is $\mathscr{A}\delta x = \widehat{y} - y_{\text{new}}$; that is, $\begin{bmatrix} -\mathscr{A}, I \end{bmatrix} z = y_{\text{new}}$. Therefore we take $C_3 = C_3^{[2]} := \begin{bmatrix} -\mathscr{A}, I \end{bmatrix}$ and $k_3 = k_3^{[2]} := y_{\text{new}}$.

This leads to Algorithm 2, summarized in displayed the pseudocode. Here we have a prescribed number of iterations, *num*. In Section 4, we examine the performance in terms of the iteration number.

4. Componentwise backward error attacks: algorithms 3 and 4

The minimum Euclidean norm perturbation in (3.2) was motivated by a normwise concept of backward error. Based on the alternative componentwise backward error viewpoint in Higham & Higham (1992); Higham (2002), instead of (3.2) we may consider the problem

$$\arg\min_{\Delta x} \{\epsilon : F(x + \Delta x) = \widehat{y}, \ \ |\Delta x| \leq \epsilon f\},$$

for a given tolerance vector $f \geq 0 \in \mathbb{R}^n$. Here, the absolute value function $|\cdot|$ is applied to each component, so $|\Delta x|_i$ is $|\Delta x_i|$. Unless otherwise indicated, we will use $f = |x|$. In this case, changes are measured in a *relative componentwise* sense, and, in particular, a zero element of x cannot be perturbed.

Algorithm 2 Iterative normwise attack, returning a perturbed image

procedure ATTACK(F, x, c_0, α, num)
  $out \leftarrow F(x)$
  $\Delta x \leftarrow 0$
  $newX \leftarrow x$
  for $i = 1$ **to** num **do**
    $jac \leftarrow$ Jacobian$(F, newX)$
    $pi \leftarrow$ PseudoInverse(jac)
    $con \leftarrow$ Constraints$(jac, newX, out, c_0)$
    $obj \leftarrow$ Objective(Δx)
    $z \leftarrow$ OptimizationVariable
    $prob \leftarrow$ Minimize(z, obj, con)
    Solve$(prob)$
    $\delta x \leftarrow z.value.delta$
    $\Delta x \leftarrow \Delta x + \alpha \cdot \delta x$
    $newX \leftarrow newX + \alpha \cdot \delta x$
    $out \leftarrow F(newX)$
  end for
  return Scale$(F, x, \Delta x, c_0)$
end procedure

Using this type of constraint in an adversarial attack, after linearizing, a componentwise version of (3.3) is given by

$$\arg\min_{\Delta x}\{\epsilon : \mathscr{A}\Delta x = \widehat{y} - y, \quad |\Delta x| \le \epsilon f\}. \tag{4.1}$$

We now write the constraint in a form that fits into the linear optimization framework (3.1), using an idea from (Higham & Higham, 1992, Section 2). We set $\Delta x = Dv$, where $D = \mathrm{diag}(f)$ and v is a vector. The relevant optimization problem is

$$\min\{\epsilon : \mathscr{A}Dv = \widehat{y} - y, \quad |Dv| \le \epsilon f, \quad Dv = \Delta x\}.$$

Since $D = \mathrm{diag}(f)$, we know that the smallest such ϵ will always be equal to $\|v\|_\infty$. Hence the minimization problem can be written

$$\min\{\|v\|_\infty : \mathscr{A}Dv = \widehat{y} - y\}. \tag{4.2}$$

In the absence of an analytical solution to (4.2), we will proceed with an iterative algorithm, along the lines of Algorithm 2, using v and δv in place of Δx and δx, respectively. Again we use x_{new} and y_{new} to keep track of the perturbed image and output during the iterative process. In each iteration of the algorithm, we compute

$$\arg\min_{\delta v} \|v + \delta v\|_\infty,$$

under the conditions that $\widehat{y} - y_{\text{new}} = \mathscr{A}D\delta v$ and $\widehat{y} \in \mathscr{S}$. After each step we update $v \leftarrow v + \alpha\delta v$, where α is a hyperparameter. Then we assign $\Delta x = Dv$ and $x_{\text{new}} = x + \Delta x$. Finally we recompute $\mathscr{A}$.

To fit into the least-squares framework (3.1), we introduce a new variable $u \in \mathbb{R}$. To minimize the infinity norm of $v + \delta v$, we may solve

$$\min\{|u| : |v + \delta v| \leq u\mathbf{1}\}.$$

Here, the constraint may be written as two separate linear inequality constraints. We must also include δv and $\hat{y}$ in the optimization variable. We will write

$$z = \begin{bmatrix} u \\ \hat{y} \\ \delta v \end{bmatrix}.$$

We can now specify the required matrices in (3.1). Since the target function is $|u|$, we use $C_1 = C_1^{[3]} := \begin{bmatrix} 1, 0, \ldots, 0 \end{bmatrix}$ and $k_1 = k_1^{[3]} := 0$. There are five inequality constraints. Two are $\delta v - u\mathbf{1} \leq -v$ and $-\delta v - u\mathbf{1} \leq v$ coming from the infinity norm optimization. A third inequality constraint is $\widehat{y} \in \mathscr{S}$, which we may write as $(I - G)\hat{y} \leq 0$. Finally, to keep the pixel values of the perturbed image within the unit interval we require

$$\begin{bmatrix} 0 & \alpha D \\ 0 & -\alpha D \end{bmatrix} z \leq \begin{bmatrix} \mathbf{1} - x_{\text{new}} \\ x_{\text{new}} \end{bmatrix}.$$

Combining these five constraints we obtain

$$C_2 = C_2^{[3]} := \begin{bmatrix} -\mathbf{1} & 0 & I \\ -\mathbf{1} & 0 & -I \\ 0 & I - G & 0 \\ 0 & 0 & \alpha D \\ 0 & 0 & -\alpha D \end{bmatrix},$$

and

$$k_2 = k_2^{[3]} := \begin{bmatrix} -v \\ v \\ 0 \\ \mathbf{1} - x_{\text{new}} \\ x_{\text{new}} \end{bmatrix}.$$

There is also an equality constraint given by $\widehat{y} - \mathscr{A}D\delta v = y_{\text{new}}$. This results in $C_3 = C_3^{[3]} := \begin{bmatrix} 0 & I & -\mathscr{A}D \end{bmatrix}$ and $k_3 = k_3^{[3]} := y_{\text{new}}$.

This leads to Algorithm 3, which is summarized in the displayed pseudocode.

Algorithm 3 Iterative componentwise attack, returning a perturbed image

procedure ATTACK($F, x, c_0, \alpha, num, f$)
  $out \leftarrow F(x)$
  $v \leftarrow 0$
  $\Delta x \leftarrow 0$
  $newX \leftarrow x$
  $D \leftarrow Diagonal(f)$
  for $i = 1$ **to** num **do**
    $jac \leftarrow$ Jacobian($F, newX$)
    $pi \leftarrow$ PseudoInverse(jac)
    $con \leftarrow$ Constraints(jac, v, out, D, c_0)
    $obj \leftarrow$ Objective(Δx)
    $z \leftarrow$ OptimizationVariable
    $prob \leftarrow$ Minimize(z, obj, con)
    Solve($prob$)
    $\delta v \leftarrow z.value.dv$
    $v \leftarrow v + \alpha \cdot \delta v$
    $\Delta x \leftarrow Dv$
    $newX \leftarrow x + \alpha \cdot \Delta x$
    $out \leftarrow F(newX)$
  end for
  return Scale($F, x, \Delta x, c_0$)
end procedure

One issue with Algorithm 3 is that the problem (4.2) encourages all components of v to achieve the maximum $\|v\|_\infty$. As we will see in section 4, this may lead to perturbations that are very noticeable. We therefore consider an alternative version where (4.2) is changed to

$$\min\{\|Dv\|_2 : \mathscr{A}Dv = \widehat{y} - y\}. \tag{4.3}$$

Because $\Delta x = Dv$, we retain the masking effect where zero values in the tolerance vector f force the corresponding pixels to remain unperturbed. We found that minimizing $\|Dv\|_2$ rather than $\|v\|_\infty$ produced perturbations that appeared less obvious. We will refer to this version as Algorithm 4. It differs from Algorithm 3 only in that $C_1^{[3]}$ is changed to $\left[0, 0, D\right]$ and $k_1^{[3]}$ is changed to $-Dv$.

5. Computational results

We implemented the algorithms in Python using PyTorch Paszke *et al.* (2019) and tested them in a deep learning setting. For the constrained least-squares optimization, we used the Splitting Conic Solver (O'Donoghue *et al.*, 2016) from the CVXPY Python package (Diamond & Boyd, 2016; Agrawal *et al.*, 2018). To evaluate the Jacobian of the classification map, we used the PyTorch function torch.autograd.functional.jacobian, which implements an efficient backpropagation process.

We tested on the MNIST dataset of handwritten digits (LeCun & Cortes, 2010). All images are 28×28 pixels in greyscale. They have a black background, corresponding to a pixel value of zero, with white writing. Hence, choosing a tolerance vector of $f = x$ in Algorithms 3 and 4 causes the background to

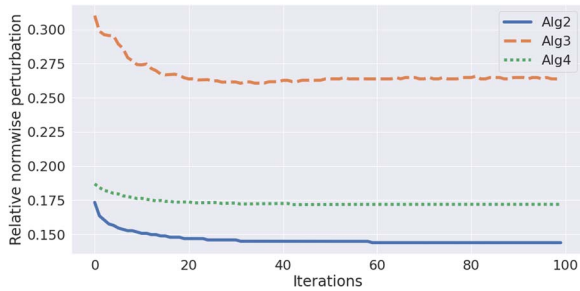


FIG. 2. Performance per iteration for Algorithms 2–4.

remain unperturbed. After applying the algorithms, we display results in the reverse greyscale, so that the background appears white and the 'ink' appears grey–black, which we believe is a more realistic view. This dataset consists of 60,000 training images and 10,000 testing images. Following Beuzeville *et al.* (2021), the network that we use has 784 input nodes and a hidden layer of 100 nodes followed by the output layer of 10 nodes. The layers are fully connected and the first one has a $\tanh$ activation function, chosen because it is differentiable. We consider different activation functions and network architectures in subsection 5.5. All network training is done using the Adam optimizer and a cross-entropy loss function. The accuracy of the trained network on the test data is 97%.

5.1 *Iterations*

First we investigate how Algorithms 2–4 perform with respect to the iteration count. We use $\alpha = 0.1$. After each iteration, using the Scale function in (3.8) we take the smallest successful multiple of the direction produced by the algorithm and record the resulting normwise perturbation size, $\epsilon = \|\Delta x\|_2/\|x\|_2$. In other words, we record the ϵ arising if we terminate at that iteration. Figure 2 shows results for a single image. The horizontal axis gives the iteration count and the vertical axis gives ϵ. The curves, which are similar for other images, indicate that we should use about 30 iterations to get optimal performance; hence we use this value in subsequent experiments. We also note that Algorithm 1 behaves in a similar way to the first step of Algorithm 2, and hence Figure 2 shows that iterating can give a significant benefit. Algorithm 3 produces larger relative two-norm perturbations that do not decrease monotonically with respect to the iteration count. This is to be expected, because the algorithm is optimizing $\|v\|_\infty$.

Next we show examples of the three iterative algorithms successfully attacking images. We chose the first image from each digit class, '0','1','2',...,'9', arising in the training set and systematically targeted each incorrect class. Full results can be seen in the Appendix. In Figure 3, we have picked out one example for attacked images in classes '5' to '9'. In each case, we show the perturbed, incorrectly classified, images from Algorithms 2–4. We also show the size of $\|\Delta x\|_2/\|x\|_2$. We see that Algorithm 2 perturbs the background whereas, by construction, Algorithms 3 and 4 do not. This leads to the background looking dirty or smudgy using Algorithm 2. Whenever Algorithm 3 decides it can perturb the pixels by some relative amount, due to the use of the infinity-norm it does not matter how many pixels are perturbed by that relative amount. This leads to large areas where the black is turned to grey, which is quite noticeable. Algorithm 4 addresses this problem by optimising for the 2-norm, as shown in (4.3). Overall, we see that Algorithm 4 produces images that may have arisen naturally from a slight inconsistency in the ink delivery or the pen pressure.

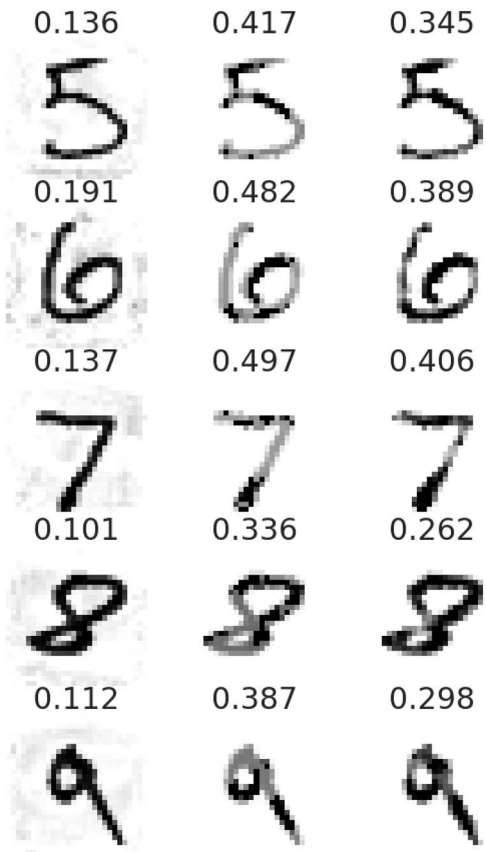


FIG. 3. Comparison of successful attacks created using Algorithms 2–4, from left to right, with the relative 2-norm of the perturbation indicated. Here the original → target examples are 5 → 9, 6 → 1, 7 → 5, 8 → 7 and 9 → 3. Examples for digits 0–9 and all possible choices of target are shown in the Appendix.

In Figure 4, we return to the comparison of perturbation sizes. Here the horizontal axis shows the relative 2-norm of the perturbation. The vertical axis shows the proportion of attacks requiring at least that relative norm of the perturbation to produce the desired classification. So a lower curve indicates better performance. The figure is based on 100 images, resulting in 900 attacks. Algorithm 2 performs best according to this measure. The iterations are seen to significantly improve the performance of these targeted attacks—recall that Algorithm 1 does not iterate. Algorithm 3 performs quite poorly, as is to be expected, since it does not directly control the 2-norm. Algorithm 4, which accounts for the 2-norm while restricting to componentwise attacks shows better performance in this regard.

5.2 *Untargeted attacks*

We now consider the scenario where it is sufficient for an attack to change the classification to *any* new class. We deal with this by targeting all new classes individually and picking the smallest perturbation. We also compare the algorithms with existing approaches designed for this untargeted case. In Figure 5, we compare Algorithms 2 and 4 with DeepFool (Moosavi-Dezfooli *et al.*, 2016) and the ℓ_2 version of

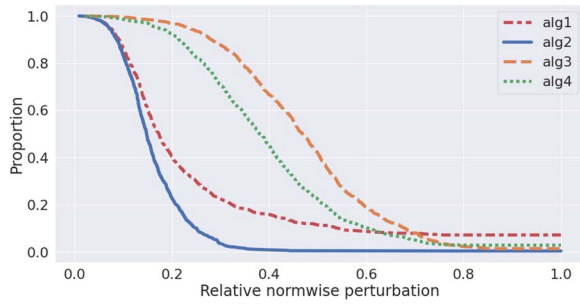


FIG. 4. Comparison between relative 2-norm performances of targeted versions of Algorithms 1–4. The horizontal axis is the relative 2-norm of the perturbation. The vertical axis is the proportion of attacks requiring at least that relative norm of the perturbation to produce the desired classification.

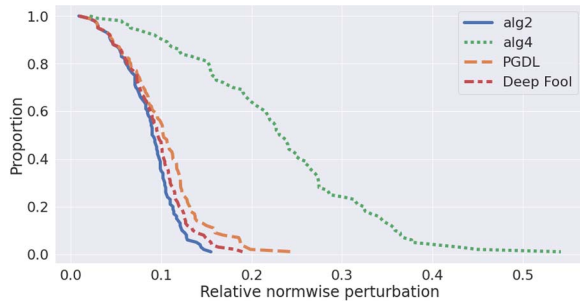


FIG. 5. Comparison between the performances untargeted versions of of Algorithm 2, Algorithm 4, DeepFool and PGD. Axes as for Figure 4.

projected gradient descent (PGD) (Madry *et al.*, 2018), with the performance measure used for Figure 4. The perturbations produced by DeepFool and PGD are scaled in the same manner as that described for Algorithms 1–4. We see that Algorithm 2 gives the best results. We suggest that this slight improvement over Deepfool and PGD arises from (a) the use of $\widehat{y}$ as an 'outer' optimization variable and (b) the use of an iterative procedure to improve the accuracy of the linearizations. These results confirm that Algorithms 3 and 4 are building on a state-of-the-art methodology.

5.3 *Best targets*

Next, we look at the best targets for each class. To do this we attack 1000 images and save both the original image class and the new class of the successful attack with smallest perturbation. We then record, for each original class, the proportion of times that each possible new class arose as the best target. Tables 1 and 2 show the best target proportions for Algorithms 2 and 4, respectively. Corresponding results for Algorithms 1 and 3 are shown in the Appendix: results for Algorithm 1 are similar to those for Algorithm 2, and results for Algorithm 3 are similar to those for Algorithm 4. The rows represent the classes of images that we perturb, and the columns represent the target classes. So, for example, in Table 1, we see that when images of the digit '0' were attacked with Algorithm 2, in 38% of the cases the smallest perturbation arose when the class was changed to '5'. The highest proportion for each original class is highlighted in bold. When comparing the results in Tables 1 and 2, we should

TABLE 1 *Table of best target proportion for attacks made by Algorithm 2. The rows are the original classes and the columns are the target classes*

	Alg. 2: Best target class proportion									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0.00	0.00	0.16	0.05	0.00	**0.38**	0.14	0.13	0.00	0.15
1	0.00	0.00	0.04	**0.42**	0.00	0.10	0.07	0.28	0.08	0.02
2	0.03	0.03	0.00	**0.57**	0.00	0.02	0.07	0.09	0.19	0.01
3	0.01	0.02	0.19	0.00	0.00	**0.57**	0.01	0.04	0.15	0.02
4	0.01	0.00	0.05	0.01	0.00	0.00	0.04	0.15	0.10	**0.64**
5	0.01	0.00	0.02	**0.53**	0.02	0.00	0.10	0.13	0.13	0.05
6	0.02	0.01	0.22	0.01	0.02	**0.52**	0.00	0.09	0.02	0.07
7	0.01	0.02	**0.4**	0.31	0.01	0.00	0.00	0.00	0.02	0.24
8	0.01	0.01	0.31	**0.4**	0.01	0.09	0.02	0.08	0.00	0.06
9	0.00	0.01	0.02	0.09	0.23	0.09	0.00	**0.42**	0.15	0.00

keep in mind that Algorithm 2 may take away ink from the digits and add ink to the background, whereas Algorithm 4 may only take away ink. For the digit class '3' it is notable that Algorithm 2 favours the target class '5', with a proportion of 0.57, and the other significant target classes are '2' and '8'. Algorithm 4 is less likely to perturb from class "3" into class '5' (the proportion is 0.48) and target class '7' has become more frequent (0.10 compared with 0.04 in Algorithm 2). It is intuitively reasonable that convincingly changing a 3 into a 5 or a 2 benefits from both addition and removal of ink and changing a 3 into a 8 benefits from addition alone, both of which are natural for Algorithm 2. Changing 3 into a 7 is more of a subtractive process, which suits Algorithm 4. We also see that for class '1', Algorithm 2 favours target classes '3' and '7', which are likely to require addition of ink, whereas Algorithm 4 has a fairly even spread of proportions—there is no obvious way to remove ink from a '1' in order to approximate a different digit. Perhaps less obvious are the results for class '9'. Here, Algorithm 2 prefers the target class '7', with proportion 0.42, whereas Algorithm 4 prefers class '4', with proportion 0.63 and has class '7' in second place with proportion 0.24. We believe that this effect is explained by the fact that there are two widely used versions of the written digit 4. The version illustrated in the Appendix is close to the digit 9 with the upper portion of the loop removed.

5.4 *Condition numbers*

The backward error concept discussed in Section 2 is traditionally accompanied by a corresponding concept of conditioning (or well-posedness). A condition number measures the worst-case sensitivity of the output to small changes in the input and, by construction, the forward error is approximately bounded by the product of a condition number and a backward error measure (Higham & Higham, 1992; Higham, 2002; Golub & Van Loan, 2013). It follows that when we use a neural network to classify an image, we may also compute an appropriate condition number estimate in order to get a feel for the sensitivity of the output to worst-case perturbations in the input, and hence to adversarial attacks. We note, however, that in the experiments reported so far, realistic attacks were very likely to exist for any input, and hence we view the condition number as a possible means to quantify the *relative* sensitivity.

TABLE 2 *Table of best target proportion for attacks made by Algorithm 4. The rows are the original classes and the columns are the target classes*

	Alg. 4: Best target class proportion									
	0	1	2	3	4	5	6	7	8	9
0	0.00	0.00	0.08	0.01	0.00	**0.49**	0.16	0.11	0.01	0.13
1	0.02	0.00	**0.19**	0.18	0.00	0.06	0.12	0.15	**0.19**	0.08
2	0.06	0.05	0.00	**0.52**	0.03	0.02	0.09	0.10	0.13	0.00
3	0.03	0.02	0.14	0.00	0.05	**0.48**	0.02	0.10	0.12	0.04
4	0.03	0.01	0.08	0.03	0.00	0.03	0.02	0.20	0.25	**0.36**
5	0.06	0.01	0.01	**0.4**	0.06	0.00	0.07	0.06	0.26	0.08
6	0.06	0.02	0.13	0.00	0.29	**0.39**	0.00	0.02	0.07	0.01
7	0.01	0.03	0.20	**0.38**	0.10	0.02	0.00	0.00	0.03	0.24
8	0.03	0.01	0.14	**0.28**	0.09	0.24	0.03	0.06	0.00	0.11
9	0.00	0.01	0.00	0.02	**0.63**	0.02	0.00	0.24	0.08	0.00

In the normwise case, if $\epsilon = \|\Delta x\|_2/\|x\|_2$ is small then

$$\frac{\|F(x) - F(x+\Delta x)\|_2}{\|F(x)\|_2} \approx \frac{\|\mathscr{A}\Delta x\|_2}{\|F(x)\|_2} \lessapprox \frac{\|\mathscr{A}\|_2\|x\|_2}{\|F(x)\|_2}\epsilon =: \mu_2(x)\epsilon.$$

Here, $\mu_2(x)$ may be viewed as a relative normwise condition number.

Similarly, under the constraint $|\Delta x| \le \epsilon f$, where we recall that f is a nonnegative tolerance vector, we have

$$\frac{\|F(x) - F(x+\Delta x)\|_\infty}{\|F(x)\|_\infty} \approx \frac{\|\mathscr{A}\Delta x\|_\infty}{\|F(x)\|_\infty} \lessapprox \frac{\||\mathscr{A}|f\|_\infty}{\|F(x)\|_\infty}\epsilon =: \mu_\infty(x)\epsilon;$$

so $\mu_\infty(x)$ may be viewed as a relative componentwise condition number.

For the normwise case, in Figure 6, we use a collection of 1000 test images that are classified correctly. For each image we compute the best attack from Algorithm 2. The figure scatter plots the attack perturbation size against the normwise condition number, μ_2. We see that a larger value of μ_2 generally corresponds, albeit weakly, to a smaller perturbation. The correlation coefficient is -0.52.

Figure 7 shows corresponding results for the componentwise condition number μ_∞ with attacks from Algorithms 3 and 4. We compare this condition number with the performances corresponding to these two attacks: the relative infinity norm and relative 2-norm respectively. Here, the correlation coefficients are -0.33 and -0.34, respectively, so the condition number is less useful in this case. A possible explanation for this difference is that perturbations are larger, and hence the linearizations are less accurate.

5.5 *Architecture*

So far we used a two layer network with a $\tanh$ activation function. Let us call this Net1. We now consider two further networks. Net2 denotes the network arising when $\tanh$ in Net1 is replaced with a rectified linear unit (ReLu). We note that ReLu is not differentiable at the origin; this did not cause any

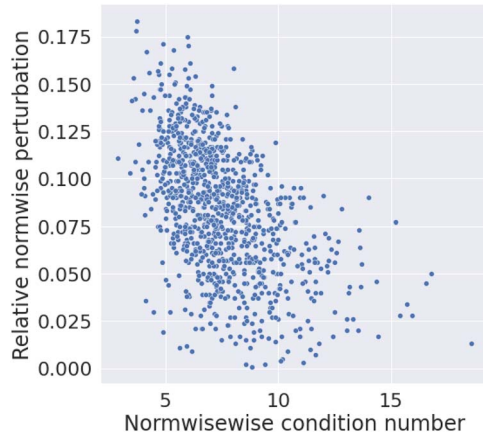
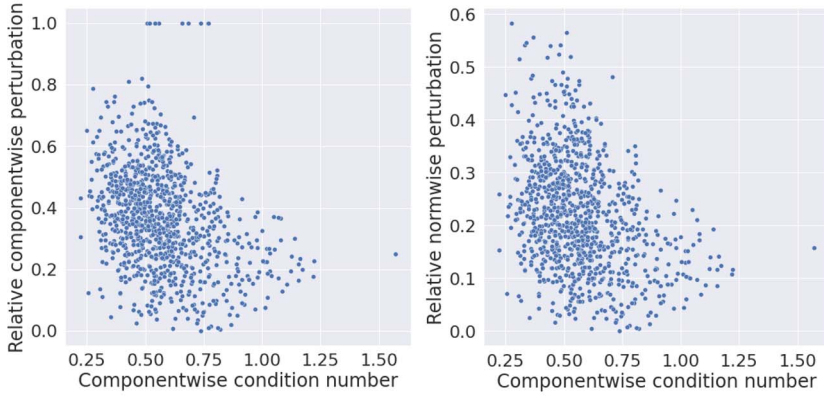


FIG. 6. Scatter plot of relative normwise perturbation for Algorithm 2 against normwise condition number μ_2 for 1000 images.



(a) Algorithm 3 (b) Algorithm 4

FIG. 7. Scatter plot of relative componentwise perturbation and relative 2-norm respectively from Algorithms 3 and 4 against componentwise condition number μ_∞ for 1000 images.

issues in our tests. After training, Net2 has an accuracy of 97%. Net3 is a convolutional neural network (CNN) (Goodfellow *et al.*, 2016) with two convolutional layers that include ReLu and max pool. The first convolutional layer uses Conv2d(1, 16, 5, 1, 2) in PyTorch. The parameters correspond to the number of input channels, number of output channels, the kernel size, the stride and the padding, respectively. This is followed by ReLu and MaxPool with stride 2. The second layer uses Conv2d(16, 32, 5, 1, 2) and is again followed by ReLu and Maxpool with stride 2. The final layer is a fully connected layer leading to an output in $\mathbb{R}^{10}$. Net3 gave an accuracy of 99%.

Figure 8 compares the performance of Algorithms 2 and 4 on these three networks in attacking 100 images without target, using the same measure as Figure 4. We see that for both algorithms changing to a ReLu has little effect. Algorithm 2 finds it more difficult to attack Net3 than Net1 or Net2. For Algorithm 4, this difference appears only in the tail of the graph; so in moving to a more complex architecture, most images remain just as vulnerable to componentwise attack.

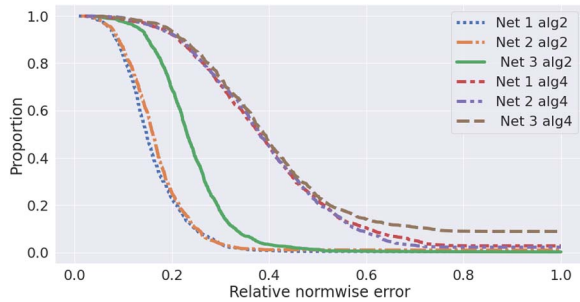


FIG. 8. Comparison between the normwise performances of Algorithms 2 and 4 for different networks. Axes as for Figure 4.

5.6 *Black box attacks*

In a black box setting, the attacker does not have access to the inner workings of the network, and hence cannot directly evaluate the Jacobian. However, with only input and output information it is, of course, possible to approximate the Jacobian using finite differences. We tested the simple Jacobian approximation

$$\left[\frac{F(x+h\cdot e_1)-F(x)}{h} \quad \dots \quad \frac{F(x+h\cdot e_n)-F(x)}{h}\right],$$

where e_i are the standard unit vectors and $h > 0$ is a small parameter; we used $h = 10^{-3}$. Measuring the performance of Algorithms 2 and 4 on Net1 and Net3, we found that results with the exact and approximate Jacobian were essentially identical; see the Appendix. We conclude that these attacks work equally well in a black box setting.

6. Conclusions

Our aim in this work was to show that it is feasible to construct *componentwise* adversarial attacks on image classification systems—here each pixel is perturbed relative to a specified tolerance. In particular this allows us to leave certain pixels unperturbed. We developed algorithms that build on the normwise approach in Beuzeville *et al.* (2021) and make use of the concept of componentwise backward error from Higham & Higham (1992). Compared with state-of-the-art normwise algorithms, when this new approach is applied to greyscale images with a well-defined background it has the advantage that the background can be left unchanged. In the context of physical writing or printing, such 'adversarial ink' is consistent with a blotchy pen, printer or photocopier.

We illustrated the performance of componentwise attacks on three neural networks and in a black box setting. We also showed that the corresponding concept of componentwise condition number has some relevance in signalling vulnerability to attack.

Directions for future work include

- Testing the componentwise algorithms on further data sets, notably those involving monochrome images of handwritten or printed text,
- Testing the componentwise algorithms on other image classification tools (note that the algorithms described here do not rely on any specific form for the classification map),

- The use of object recognition (Srivastava *et al.*, 2021) to identify background pixels, so that the choice of componentwise tolerance vector can be automated in complex images,
- The construction of universal componentwise attacks, where the same perturbation changes the classification of many images that have shared 'non-background' locations,
- The construction of adversarial ink attacks on signatures, postcodes, dates, cheques or entire documents.

Funding

MAC-MIGS Centre for Doctoral Training under EPSRC (grant EP/S023291/1 to L.B.); EPSRC (grants EP/P020720/1 and EP/V046527/1 to D.J.H.). We thank Oliver Sutton for suggesting the phrase adversarial ink, and an anonymous referee for helpful feedback.

Data Availability

Code for the experiments presented here is available at https://github.com/LucasBeerens/adversarial-ink-componentwise-attacks.

References

Agrawal, A., Verschueren, R., Diamond, S. & Boyd, S. (2018) A rewriting system for convex optimization problems. *J. Control Decision*, **5**, 42–60.

Akhtar, N. & Mian, A. (2018) Threat of adversarial attacks on deep learning in computer vision: a survey. *IEEE Access*, **6**, 14410–14430.

Bastounis, A., Hansen, A. C. & Vlaĉić, V. (2021) The mathematics of adversarial attacks in AI–why deep learning is unstable despite the existence of stable neural networks. *arXiv:2109.06098 [cs.LG]*.

Beuzeville, T., Boudier, P., Buttari, A., Gratton, S., Mary, T. & Pralet, S. (2021) Adversarial attacks via backward error analysis. *hal-03296180, version 3*.

Corless, R. M. & Fillion, N. (2013) *A Graduate Introduction to Numerical Methods: From the Viewpoint of Backward Error Analysis*. Berlin: Springer.

Diamond, S. & Boyd, S. (2016) CVXPY: a python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, **17**, 1–5.

Fawzi, A., Fawzi, O. & Frossard, P. (2018) Analysis of classifiers' robustness to adversarial perturbations. *Machine Learning*, **107**, 481–508.

Golub, G. H. & Van Loan, C. F. (2013) *Matrix Computations*, 4th edn. The Johns Hopkins University Press.

Goodfellow I. J., Shlens J., and Szegedy C. (2015) *Explaining and harnessing adversarial examples*, *3rd International Conference on Learning Representations*, San Diego, CA, Bengio Y. and LeCun Y., eds.

Goodfellow, I., Bengio, Y. & Courville, A. (2016) *Deep learning, Adaptive computation and machine learning*. Boston: The MIT Press.

Goodfellow, I. J., McDaniel, P. D. & Papernot, N. (2018) Making machine learning robust against adversarial inputs. *Commun. ACM*, **61**, 56–66.

Harrison, D., Burkes, T. M. & Seiger, D. P. (2009) Handwriting examination: meeting the challenges of science and the law. *Forensic Sci. Commun.*, **11**.

Higham, N. J. (2002) *Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics*, 2nd edn. Philadelphia, PA, USA.

Higham, D. J. & Higham, N. J. (1992) Backward error and condition of structured linear systems. *SIAM J. Matrix Anal. Appl.*, **13**, 162–175.

Huber, R. A. & Headrick, A. M. (1999) *Handwriting Identification: Facts and Fundamentals*. Boca Raton, FA: CRC Press.

LeCun, Y. & Cortes, C. (2010) *MNIST handwritten digit database*.

Madry A., Makelov A., Schmidt L., Tsipras D., and Vladu A. (2018) *Towards deep learning models resistant to adversarial attacks*, *6th International Conference on Learning Representations*, Vancouver, BC, OpenReview.net.

Marcus, G. (2018) Deep learning: a critical appraisal. *arXiv:1801.00631 [cs.AI]*.

Moosavi-Dezfooli S., Fawzi A., and Frossard P. (2016) *Deepfool: A simple and accurate method to fool deep neural networks*, *2016 IEEE Conference on Computer Vision and Pattern Recognition*, NV, USA, IEEE Computer Society, pp. 2574–2582.

O'Donoghue, B., Chu, E., Parikh, N. & Boyd, S. (2016) Conic optimization via operator splitting and homogeneous self-dual embedding. *J. Optim. Theory Appl.*, **169**, 1042–1068.

Papernot N., McDaniel P. D., Goodfellow I. J., Jha S., Celik Z. B., and Swami A. (2017) *Practical black-box attacks against machine learning*, *Proceedings of the ACM Conference on Computer and Communications Security*, Abu Dhabi, UAE, Karri R., Sinanoglu O., Sadeghi A., and Yi X., eds, ACM, pp. 506–519.

Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Kopf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J., and Chintala S. (2019) *PyTorch: an imperative style, high-performance deep learning library*, *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., pp. 8024–8035.

Shafahi A., Huang W., Studer C., Feizi S., and Goldstein T. (2019) *Are adversarial examples inevitable?*, *International Conference on Learning Representations*, New Orleans, USA.

Srivastava, S., Divekar, A. V., Anilkumar, C., Naik, I., Kulkarni, V. & Pattabiraman, V. (2021) Comparative analysis of deep learning image detection algorithms. *J. Big Data*, **8**.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. & Fergus, R. (2013) Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tyukin I. Y., Higham D. J., and Gorban A. N., *On adversarial examples and stealth attacks in artificial intelligence systems*, *2020 International Joint Conference on Neural Networks*, Elasgow: IEEE, 2020, pp. 1–6.

Tyukin, I. Y., Higham, D. J., Bastounis, A., Woldegeorgis, E. & Gorban, A. N. (2021) The feasibility and inevitability of stealth attacks. *arXiv:2106.13997*.

Wang, G., Wei, Y. & Qiao, S. (2018) *Generalized Inverses: Theory and Computations*. *Developments in Mathematics, 53*, 1st edn. 2018. Springer Singapore.

A. Appendix

In Figures A1–A3, we expand on Figure 3 by showing successful attacks produced by Algorithms 2–4, and the relative 2-norm of the perturbation, on an example from each digit class and for all target classes. Here, each image attacked is the first of its class arising in the data set.

Tables A1 and A2 are the analogues of Tables 1 and 2 corresponding to Algorithms 1 and 3, respectively.

Figure A4 shows the performance measures (as described for Figure 4) for exact Jacobian and finite-difference (black box) versions of Algorithms 2 and 4 on Net1 and Net3.

TABLE A1 *Table of best target proportion for attacks made by Algorithm 1. The rows are the original classes and the columns are the target classes*

	Alg. 1: Best target class proportion									
	0	1	2	3	4	5	6	7	8	9
0	0.00	0.00	0.17	0.05	0.00	**0.38**	0.14	0.11	0.00	0.15
1	0.02	0.00	0.04	**0.4**	0.00	0.08	0.06	0.29	0.08	0.02
2	0.03	0.03	0.00	**0.51**	0.00	0.02	0.09	0.09	0.22	0.01
3	0.02	0.02	0.22	0.00	0.00	**0.54**	0.01	0.04	0.13	0.03
4	0.03	0.00	0.05	0.01	0.00	0.00	0.04	0.15	0.09	**0.64**
5	0.02	0.00	0.02	**0.49**	0.02	0.00	0.13	0.13	0.15	0.03
6	0.04	0.01	0.20	0.01	0.02	**0.52**	0.00	0.07	0.01	0.12
7	0.01	0.02	**0.41**	0.33	0.02	0.00	0.00	0.00	0.01	0.21
8	0.02	0.01	0.32	**0.37**	0.01	0.09	0.02	0.08	0.00	0.07
9	0.00	0.01	0.02	0.10	0.23	0.10	0.00	**0.41**	0.14	0.00

TABLE A2 *Table of best target proportion for attacks made by Algorithm 3. The rows are the original classes and the columns are the target classes*

	Alg. 3: Best target class proportion									
	0	1	2	3	4	5	6	7	8	9
0	0.00	0.00	0.06	0.02	0.00	**0.49**	0.18	0.10	0.01	0.13
1	0.02	0.00	0.14	**0.21**	0.00	0.08	0.13	0.10	0.18	0.15
2	0.04	0.07	0.00	**0.47**	0.03	0.02	0.06	0.17	0.14	0.00
3	0.03	0.05	0.12	0.00	0.08	**0.47**	0.02	0.09	0.09	0.05
4	0.03	0.01	0.05	0.05	0.00	0.04	0.01	0.20	**0.37**	0.25
5	0.05	0.02	0.01	**0.38**	0.08	0.00	0.05	0.06	0.20	0.15
6	0.06	0.02	0.13	0.01	0.27	**0.35**	0.00	0.02	0.11	0.02
7	0.01	0.02	0.18	**0.36**	0.07	0.07	0.00	0.00	0.05	0.25
8	0.02	0.01	0.14	0.23	0.11	**0.25**	0.03	0.06	0.00	0.14
9	0.00	0.00	0.00	0.05	**0.59**	0.05	0.00	0.23	0.08	0.00

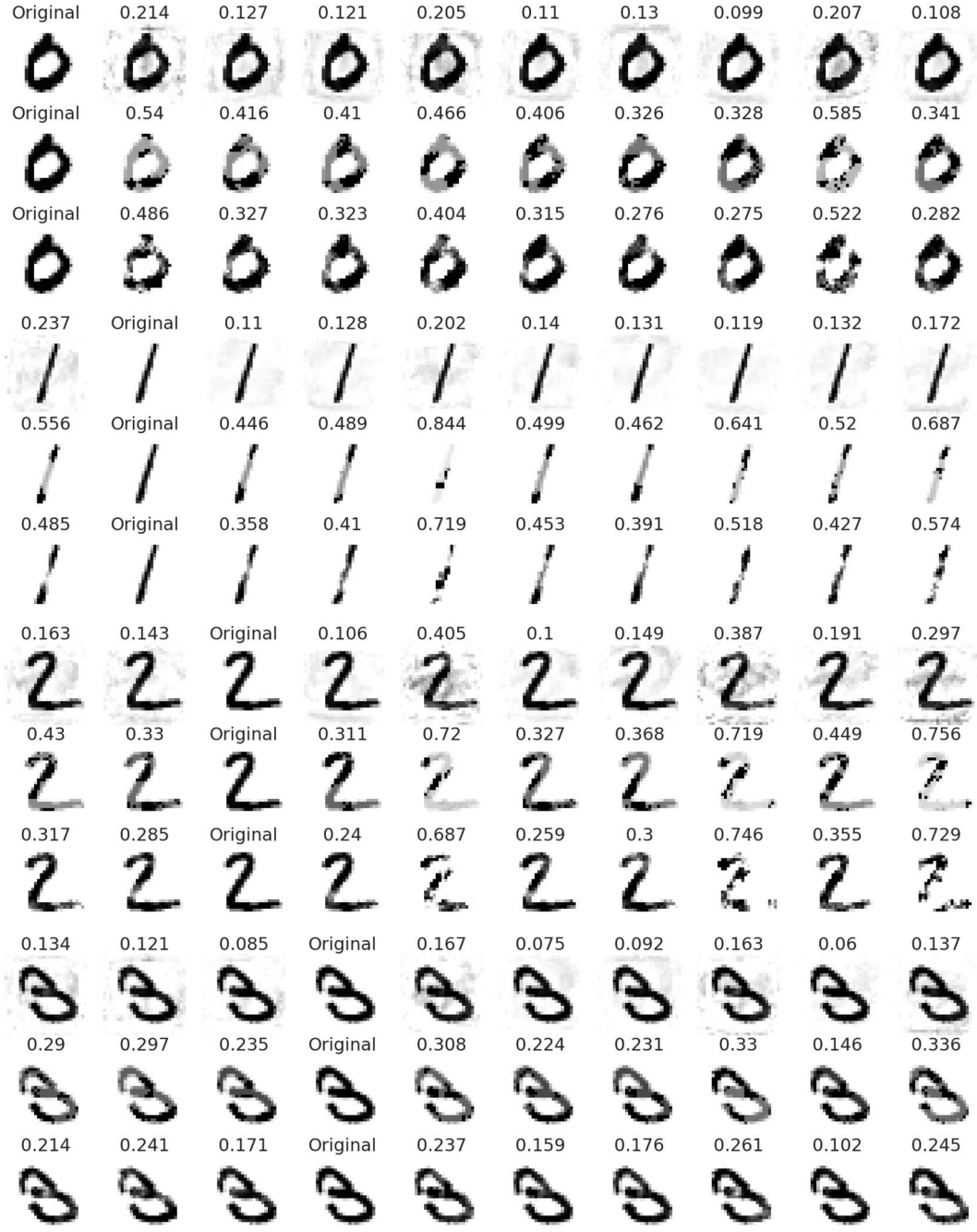


FIG. A1. Row one shows the results of adversarial attacks with Algorithm 2 on an image from class 0. The original image is shown and then, from left to right, we have targets 1,2,3, . . . ,9. The numbers above the images indicate the relative 2-norm of the perturbation. Rows two and three show this information for Algorithms 3 and 4, respectively. This pattern then repeats for images from classes 1–3.

FIG. A2. As in Figure A1, rows corresponds to Algorithms 2–4 in turn and columns indicate target class. In this case, images are from classes 4–7.

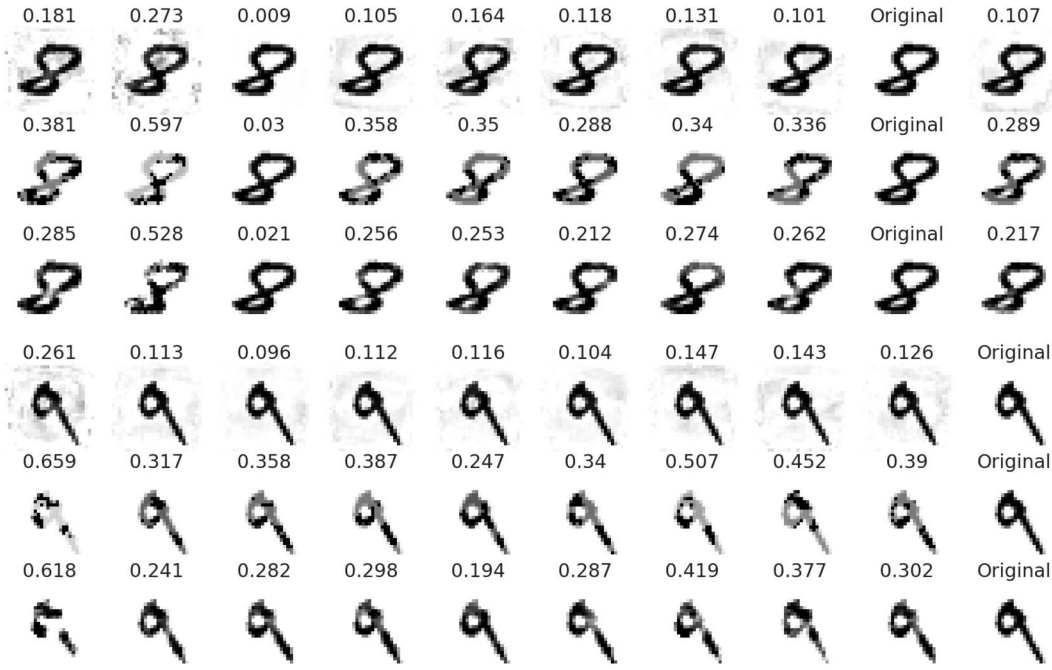


FIG. A3. As in Figure A1, rows corresponds to Algorithms 2–4 in turn and columns indicate target class. In this case images are from classes 8 and 9.

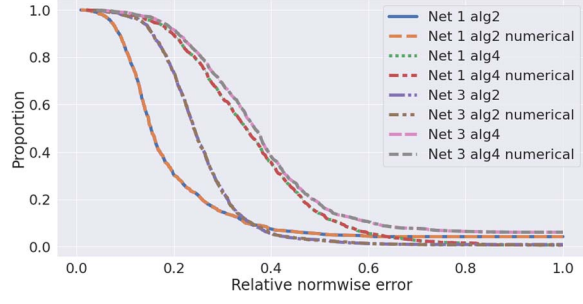


FIG. A4. Comparison between the normwise performances of white box and black box attacks for Algorithms 2 and 4 on three different neural networks. Axes as for Figure 4.