Adversarial attacks via backward error analysis

Théo Beuzeville, Pierre Boudier, Alfredo Buttari, Serge Gratton, Théo Mary, Stéphane Pralet

Adversarial attacks via backward error analysis

Théo Beuzeville
Université de Toulouse
Atos-IRIT
theo.beuzeville@toulouse-inp.fr

Pierre Boudier
NVIDIA
pboudier09@gmail.com

Alfredo Buttari
Université de Toulouse
CNRS-IRIT
alfredo.buttari@irit.fr

Serge Gratton
Université de Toulouse
INPT-IRIT
serge.gratton@enseeiht.fr

Théo Mary
Sorbonne Université
CNRS-LIP6
theo.mary@lip6.fr

Stéphane Pralet
Atos
stephane.pralet@atos.net

Abstract

Backward error (BE) analysis was developed and popularized by James Wilkinson in the 1950s and 1960s, with origins in the works of Neumann and Goldstine (1947) and Turing (1948). It is a fundamental notion used in numerical linear algebra software, both as a theoretical and a practical tool for the rounding error analysis of numerical algorithms. Broadly speaking the backward error quantifies, in terms of perturbation of input data, by how much the output of an algorithm fails to be equal to an expected quantity. For a given computed solution $\widehat{y}$, this amounts to computing the norm of the smallest perturbation Δx of the input data x such that $\widehat{y}$ is an exact solution of a perturbed system: $f(x + \Delta x) = \widehat{y}$. Up to now, BE analysis has been applied to numerous linear algebra problems, always with the objective of quantifying the robustness of algebraic processes with respect to rounding errors stemming from finite precision computations. While deep neural networks (DNN) have achieved an unprecedented success in numerous machine learning tasks in various domains, their robustness to adversarial attacks, rounding errors, or quantization processes has raised considerable concerns from the machine learning community. In this work, we generalize BE analysis to DNN. This enables us to obtain closed formulas and a numerical algorithm for computing adversarial attacks. By construction, these attacks are optimal, and thereby smaller, in norm, than perturbations obtained with existing gradient-based approaches. We produce numerical results that support our theoretical findings and illustrate the relevance of our approach on well-known datasets.

1 Introduction

Adversarial attacks Deep neural networks have become increasingly popular and successful in many machine learning tasks: their efficiency in solving complex problems has led to apply deep learning techniques in safety-critical tasks such as autonomous driving [8] and medicine [14]. However, many works [7, 10, 1] have shown that deep neural network models are vulnerable to adversarial attacks: attacks on a machine learning model that an attacker intentionally designed to cause the model to make mistakes. In order to use DNNs in security-critical scenarios it is thus

Preprint. Under review.

crucial to explore their vulnerability against attackers. Several types of attacks are now well known, such as adversarial examples [16, 12], poisoning the training data [2], etc. Adversarial examples have been one of the most popular approaches. They correspond to slight perturbations on the input data of a neural network, small enough so that they are not noticeable by the human eye but still change the model predictions. Whereas adversarial perturbations are mostly used on the input space, there are few approaches which take interest on a similar notion for the model's parameters [6] despite its potential use to help robust generalization [21].

In this work we propose a novel approach for the construction of adversarial attacks which relies on backward error analysis methods that are more commonly used in scientific computing to assess the effect of finite precision computation or to measure the sensitivity of an algorithm to perturbations on data. Because of its generality, this approach allows us to compute adversarial attacks on the input data as well as on the network parameters; the latter correspond to the case where a malicious user tampers with the values of the weights or biases in order to alter the behavior of the network.

Backward error Numerical error is inherent in machine computation due to its use of floating-point arithmetic. Hence the ability to measure the accuracy of numerical programs is essential [9, 3]. Given a computed solution to a problem, there are two ways to measure its numerical accuracy. *Forward error* analysis directly measures the distance or difference between the computed solution and the exact solution. *Backward error* measures how much the problem's input data must be perturbed to produce the computed solution. Backward error analysis was developed and popularized by James Wilkinson [20] in the 1950s and 1960s, with origins in the works of Neumann and Goldstine (1947) [13] and Turing (1948) [18].

Let us assume $\widehat{y}$ is the computed solution of $y = f(x)$, with $f : \mathbb{R}^n \to \mathbb{R}^n$. The backward error is obtained by asking for what value of x the problem has actually been solved, that is, of what perturbed problem $\widehat{y}$ is the solution. Formally, we have

$$\hat{y} = y + \Delta y = f(x + \Delta x) \tag{1}$$

and the backward error is defined as

$$E_{\text{back}}(\widehat{y}) = \min \{\epsilon : \widehat{y} = f(x + \Delta x), \|\Delta x\| \leq \epsilon \|x\|\}, \tag{2}$$

that is, the minimal norm perturbation Δx of data x such that the perturbed problem $f(x + \Delta x)$ produces the computed solution $\widehat{y}$. As a consequence of this definition, it can be said that:

- If there is an uncertainty in the data (physical measurements, approximations, ...), it is sufficient that the backward error is of the same order as this uncertainty for the computed solution to be satisfactory.
- The algorithm is numerically stable if the backward error is close to the round-off error u of the computer arithmetic used for the computation.

Forward error and backward error are linked by a third quantity, called the problem *conditioning*, which measures how sensitive the solution to a problem is to disturbances in the data. Indeed we have:

$$\text{forward error} \leq \text{condition number} \times \text{backward error}.$$

Our main focus here is to apply backward error analysis to neural networks and to derive mathematical expressions for the backward error defined as in equation (2). We will focus on fully connected deep classification neural networks but the approach is easily generalized to convolutions because a convolution layer, as a fully connected layer, can be expressed as a matrix product. Our objective is to use these expressions to search for perturbations on the neural networks' parameters and input data that produce adversarial examples with no need to have an access to training data but only the model's parameters and a typical example of the data that we aim to misclassify.

Notations We define one layer i of a fully connected or convolutional artificial neural network as

$$y_i = f_i\left(A_i y_{i-1}\right), \; i = 1, ..., p, \tag{3}$$

where y_i is the output of the layer, f_i is the activation function, A_i the weight matrix, y_{i-1} the output of the previous layer, $y_0 = x$ being the network input data. Note that bias can be added as follows:

$$y = f\left([A, \quad b] \begin{bmatrix} x \\ 1 \end{bmatrix}\right),$$

and thus, without loss of generality, we wil assume that one layer of the neural network corresponds to a matrix-vector product and drop b for the sake of readability. We use $\widehat{y}$ as the approximation of the output vector y and note $\Delta y = \widehat{y} - y$ whereas ΔA_i or Δx are perturbations on the matrices or on vectors.

We will always use the 2-norm for vectors and the Frobenius norm for matrices; for the sake of readability, we will drop the subscripts in the norms.

We will use the vec operator which stacks the columns of a matrix one underneath the others and note $\overrightarrow{A} = \mathrm{vec}(A)$. We recall some properties of the Kronecker product which will be used in the remainder of this document:

$$
\begin{aligned}
(A \bigotimes C)(B \bigotimes D) &= (AB) \bigotimes (CD), \\
\left\| A \bigotimes B \right\| &= \|A\| \, \|B\| \,, \\
|A \bigotimes B| &= |A| \bigotimes |B|, \\
\mathrm{vec}(AXB) &= (B^T \bigotimes A) \, \mathrm{vec}(X).
\end{aligned}
$$

2 Backward error analysis for neural networks

The goal of this section is to establish an explicit expression of the backward error for a deep neural network with p fully connected layers defined as in equation (3). We will assume perturbations on both the input data x and the network weights A_i. For the sake of clarity, we will proceed in incremental steps. In section 2.1 we will focus on the simple case of a single layer without activation function and, in sections 2.2 and 2.3, we will extend this result to the cases of two layers without and one layer with activation function, respectively. These steps will provide the ingredients to produce the final and generic result that will be presented in section 2.4.

2.1 One layer without activation function

For a single layer linear neural network the approximated output vector is given by: $\widehat{y} = (A_1 + \Delta A_1)(x + \Delta x)$. Since our goal is to derive adversarial attacks focusing on small perturbations, the second order perturbations $\Delta A_1 \Delta x$ can be safely neglected and dropping it allows us to obtain the following simplified formula

$$
\Delta y = \Delta A_1 x + A_1 \Delta x = \begin{bmatrix} x^T \bigotimes I, & A_1 \end{bmatrix} \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \Delta x \end{bmatrix}.
$$

By this formulation, the ΔA_1 and Δx perturbations can be computed as the solution of the following linear system

$$
\mathcal{A} \overrightarrow{\Delta A} = \Delta y, \quad \text{where } \mathcal{A} = \begin{bmatrix} x^T \bigotimes I, & A_1 \end{bmatrix} \quad \text{and } \overrightarrow{\Delta A} = \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \Delta x \end{bmatrix}.
$$

Note that this is an under determined system whose minimum norm solution is given by $\overrightarrow{\Delta A} = \mathcal{A}^+ \Delta y$ with

$$
\mathcal{A}^+ = \mathcal{A}^T (\mathcal{A} \mathcal{A}^T)^{-1} \quad \text{and } \mathcal{A} \mathcal{A}^T = \begin{bmatrix} x^T \bigotimes I, & A_1 \end{bmatrix} \begin{bmatrix} x \bigotimes I \\ A_1^T \end{bmatrix} = A_1 A_1^T + \|x\|^2 I.
$$

Writing $\mathcal{B} = (A_1 A_1^T + \|x\|^2 I)^{-1}$, we obtain

$$
\begin{bmatrix} \overrightarrow{\Delta A_1} \\ \Delta x \end{bmatrix} = \begin{bmatrix} x \bigotimes I \\ A_1^T \end{bmatrix} \mathcal{B} \Delta y
$$

and so

$$
\begin{cases} \Delta A_1 &= \mathcal{B} \Delta y x^T, \\ \Delta x &= A_1^T \mathcal{B} \Delta y. \end{cases}
$$

Note that if we ignore perturbations Δx on the input, we recover the well-known result of Rigal and Gaches [15].

2.2 Two layers without activation function

In this section the goal is to establish an explicit expression of the backward error for a two layers neural network without activation function in order to show how adding layers affects the backward error. For a two layers linear neural network the computed output vector is given by: $\widehat{y}_2 = (A_2 + \Delta A_2)(A_1 + \Delta A_1)(x + \Delta x)$. Once more we drop lower order perturbations and obtain

$$\begin{aligned}\Delta y &= A_2 \Delta A_1 x + \Delta A_2 A_1 x + A_2 A_1 \Delta x \\ &= \begin{bmatrix} x^T \bigotimes A_2, & x^T A_1^T \bigotimes I, & A_2 A_1 \end{bmatrix} \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \overrightarrow{\Delta A_2} \\ \Delta x \end{bmatrix}.\end{aligned}$$

Again, this formulation allows us to compute the perturbations as the minimum norm solution of an underdetermined linear system $\mathcal{A}\overrightarrow{\Delta A} = \Delta y$, where

$$\mathcal{A} = \begin{bmatrix} x^T \bigotimes A_2, & x^T A_1^T \bigotimes I, & A_2 A_1 \end{bmatrix} \quad \text{and} \quad \overrightarrow{\Delta A} = \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \overrightarrow{\Delta A_2} \\ \Delta x \end{bmatrix}.$$

As in the previous section, this is achieved computing $\mathcal{A}^+ \Delta y$ with $\mathcal{A}^+ = \mathcal{A}^T (\mathcal{A}\mathcal{A}^T)^{-1}$ and

$$\begin{aligned}\mathcal{A}\mathcal{A}^T &= \begin{bmatrix} x^T \bigotimes A_2, & x^T A_1^T \bigotimes I, & A_2 A_1 \end{bmatrix} \begin{bmatrix} x \bigotimes A_2^T \\ A_1 x \bigotimes I \\ (A_2 A_1)^T \end{bmatrix} \\ &= \|x\|^2 A_2 A_2^T + \|A_1 x\|^2 I + A_2 A_1 (A_2 A_1)^T.\end{aligned}$$

Writing $\mathcal{B} = (\|x\|^2 A_2 A_2^T + \|A_1 x\|^2 I + A_2 A_1 (A_2 A_1)^T)^{-1}$, we obtain

$$\begin{bmatrix} \overrightarrow{\Delta A_1} \\ \overrightarrow{\Delta A_2} \\ \Delta x \end{bmatrix} = \begin{bmatrix} x \bigotimes A_2^T \\ A_1 x \bigotimes I \\ (A_2 A_1)^T \end{bmatrix} \mathcal{B} \Delta y,$$

which leads to

$$\begin{cases} \overrightarrow{\Delta A_1} & = x \bigotimes A_2^T \mathcal{B} \Delta y = \mathrm{vec}(A_2^T \mathcal{B} \Delta y x^T), \\ \overrightarrow{\Delta A_2} & = A_1 x \bigotimes I \mathcal{B} \Delta y = \mathrm{vec}(I \mathcal{B} \Delta y x^T A_1^T), \\ \Delta x & = (A_2 A_1)^T \mathcal{B} \Delta y, \end{cases}$$

or, equivalently,

$$\begin{cases} \Delta A_1 & = A_2^T \mathcal{B} \Delta y x^T, \\ \Delta A_2 & = \mathcal{B} \Delta y x^T A_1^T, \\ \Delta x & = A_1^T A_2^T \mathcal{B} \Delta y. \end{cases}$$

2.3 One layer with activation function

We now turn our attention to the case of a single layer with activation function to show how nonlinear functions affect the backward error. Let $\widehat{y} = f_1((A_1 + \Delta A_1)(x + \Delta x)))$ and suppose that f_1 is differentiable; a first-order Taylor expansion gives us

$$\Delta y = f_1'(A_1 x) \Delta A_1 x + f_1'(A_1 x) A_1 \Delta x = \begin{bmatrix} x^T \bigotimes f_1'(A_1 x), & f_1'(A_1 x) A_1 \end{bmatrix} \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \Delta x \end{bmatrix}.$$

Let $\mathcal{A} = \begin{bmatrix} x^T \bigotimes f_1'(A_1 x), & f_1'(A_1 x) A_1 \end{bmatrix}$; following the same approach as in section 2.1, we have

$$\mathcal{A}^T (\mathcal{A}\mathcal{A}^T)^{-1} \Delta y = \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \Delta x \end{bmatrix}$$

with

$$\begin{aligned}\mathcal{A}\mathcal{A}^T &= x^T x \bigotimes f_1'(A_1 x)) (f_1'(A_1 x))^T + f_1'(A_1 x) A_1 (f_1'(A_1 x) A_1)^T \\ &= \|x\|^2 (f_1'(A_1 x)) (f_1'(A_1 x))^T + f_1'(A_1 x) A_1 (f_1'(A_1 x) A_1)^T.\end{aligned}$$

Writing $\mathcal{B} = (\mathcal{A}\mathcal{A}^T)^{-1}$, we obtain

$$\begin{cases} \Delta A_1 &= (f_1'(A_1x))^T\mathcal{B}\Delta y x^T, \\ \Delta x &= (f_1'(A_1x)A_1)^T\mathcal{B}\Delta y. \end{cases}$$

2.4 Backward error analysis for a generic neural network

In this section we will use the results of the previous three sections and generalize them to the case of a deep neural network with p layers and activation functions.

For a neural network with p layers defined as in equation (3), the computed result is

$$\widehat{y}_p = f_p((A_p + \Delta A_p)f_{p-1}((A_{p-1} + \Delta A_{p-1})\dots(A_2 + \Delta A_2)f_1((A_1 + \Delta A_1)(x + \Delta x))\dots)).$$

Assuming the activation functions $(f_i)_{i=1,\dots,p}$ are differentiable, with first order approximations we have

$$\mathcal{A}^T = \begin{bmatrix} x \bigotimes (f_p'(A_p y_{p-1})A_p f_{p-1}'(A_{p-1}y_{p-2})\dots A_2 f_1'(A_1x))^T \\ \vdots \\ y_{i-1} \bigotimes (f_p'(A_p y_{p-1})A_p f_{p-1}'(A_{p-1}y_{p-2})\dots A_{i+1} f_i'(A_i y_{i-1}))^T \\ \vdots \\ y_{p-1} \bigotimes (f_p'(A_p y_{p-1}))^T \\ (f_p'(A_p y_{p-1})A_p f_{p-1}'(A_{p-1}y_{p-2})\dots A_2 f_1'(A_1x)A_1)^T \end{bmatrix}.$$

Let $\mathcal{B} = (\mathcal{A}\mathcal{A}^T)^{-1}$, we then can show that the perturbations associated with the given approximation $\widehat{y}_p$ are:

$$\begin{cases} \Delta A_1 &= (f_p'(A_p y_{p-1})A_p f_{p-1}'(A_{p-1}y_{p-2}))^T\mathcal{B}\Delta y x^T, \\ &\vdots \\ \Delta A_i &= (f_p'(A_p y_{p-1})A_p f_{p-1}'(A_{p-1}y_{p-2})\dots A_{i+1} f_i'(A_i y_{i-1}))^T\mathcal{B}\Delta y y_{i-1}^T, \\ &\vdots \\ \Delta A_p &= (f_p'(A_p y_{p-1}))^T\mathcal{B}\Delta y y_{p-1}^T, \\ \Delta x &= (f_p'(A_p y_{p-1})A_p f_{p-1}'(A_{p-1}y_{p-2})\dots A_2 f_1'(A_1x)A_1)^T\mathcal{B}\Delta y. \end{cases} \tag{4}$$

Thanks to this backward error analysis of neural networks, we have thus obtained a general expression for perturbations to yield a given approximate result. Our analysis is for a general arbitrary network with any number of layers and with activation functions, and allows perturbations made on both the weights and the input of the network.

3 Adversarial attacks via backward error analysis

In this section we present a novel approach for producing adversarial attacks to classification neural networks that relies on the backward error analysis presented in section 2. The approach consists in computing the smallest norm perturbation on input data or network weights such that, for a given input x, the computed $\widehat{y}$ results in a misclassification, that is, it erroneously affects the input to class j instead of the expected one. Mathematically, the adversarial perturbation is defined as the solution of the following minimization problem

Solve

$$\underset{\Delta A_i, \Delta x}{\arg\min} \sum_{i=1}^{p} \frac{\|\Delta A_i\|^2}{\|A_i\|^2} + \frac{\|\Delta x\|^2}{\|x\|^2} \tag{5}$$

subject to

$$\widehat{y} = f_p((A_p + \Delta A_p)f_{p-1}((A_{p-1} + \Delta A_{p-1})\dots f_1((A_1 + \Delta A_1)(x + \Delta x))))$$
$$\widehat{y}_i \leq \widehat{y}_j,\ i = 1, \dots, n.$$

From section 2 we know that we can express the perturbations as:

$$\begin{cases} \Delta A_i &= M_i \Delta y y_{i-1}^T \\ \Delta x &= M\Delta y \end{cases}$$

where

$$M_i = (f'_p(A_p y_{p-1}) A_p f'_{p-1}(A_{p-1} y_{p-2}) \dots A_{i+1} f'_i(A_i y_{i-1}))^T \mathcal{B} \quad \text{and}$$
$$M = (f'_p(A_p y_{p-1}) A_p f'_{p-1}(A_{p-1} y_{p-2}) \dots A_2 f'_1(A_1 x) A_1)^T \mathcal{B}.$$

Therefore

$$\frac{\|\Delta A_i\|^2}{\|A_i\|^2} = \frac{\left\|M_i \Delta y y_{i-1}^T\right\|^2}{\|A_i\|^2} = \frac{\|M_i \Delta y\|^2 \|y_{i-1}\|^2}{\|A_i\|^2},$$

where we have used the fact that $\|xy^T\|_F^2 = \|x\|_2^2 \|y\|_2^2$.

Hence, using backward error analysis, we express the perturbations as variables which only depends on a given approximate result $\hat{y}$ and on the network's parameters. The optimization problem is then reduced to:

$$\begin{aligned} &\textbf{Solve} \\ &\quad \arg\min_{\widehat{y}} \|\mathcal{M}(\widehat{y} - y)\|^2 \\ &\textbf{subject to} \\ &\quad \widehat{y}_i \leq \widehat{y}_j, \ i = 1, \dots, n, \end{aligned} \tag{6}$$

with

$$\mathcal{M} = \begin{bmatrix} \frac{\|x\|}{\|A_1\|} M_1 \\ \vdots \\ \frac{\|y_{i-1}\|}{\|A_i\|} M_i \\ \vdots \\ \frac{\|y_{p-1}\|}{\|A_p\|} M_p \\ \frac{1}{\|x\|} M \end{bmatrix}.$$

Note that here we focus on the case where the classifier assign the input data to the j-th class when $\widehat{y}_i \leq \widehat{y}_j, \ i = 1, \dots, n$. But this can easily be generalized to other types of classification by modifying those constraints. Once this optimization problem is solved and, thus, $\widehat{y}$ is computed, the adversarial perturbations can be computed using equation (4).

Most of the approaches that generate adversarial examples (FGSM [7] or based on FGSM [10], SGD [12] or based on SGD [5], FAB [4]...) use the gradient of the loss function in order to solve a given optimization problem. Usually the optimization problem can be generically formulated as:

$$\begin{aligned} &\textbf{Solve} \\ &\quad \min \|\Delta x\|^2 \\ &\textbf{subject to} \\ &\quad C(x + \Delta x) = j. \end{aligned} \tag{7}$$

Where j is the target class and $C(x + \Delta x)$ the class of the perturbed image. This problem being difficult to solve, the above-mentioned approaches commonly resort to solving the following problem:

$$\begin{aligned} &\textbf{Solve} \\ &\quad \min c \|\Delta x\|^2 + \mathcal{L}(x + \Delta x, j), \end{aligned} \tag{8}$$

where $\mathcal{L}$ is the loss of a given image with respect to a given target class.

Unlike these methods, our approach relies on a first order approximation of Δx or ΔA_i resulting from the BE analysis. This enables us to simplify equation (7) and formulate it as equation (6). A notable advantage of our approach is that it does not require the loss gradient in order to find optimal perturbations.

4 Experimental results

For all of our experiments we train a fully connected neural network with Keras with Adam's optimizer and a sparse categorical cross entropy's loss on Python on the MNIST database [11]. We

describe the neural network's structure in each subsection. Once the network is trained, we use its weight matrices to compute adversarial attacks as described before, using MATLAB R2020a. For each data in our set, we first solve the optimization problem (6) using MATLAB's `lsqlin` function from the optimization toolbox, and then we find the corresponding perturbations on the input or on the model's weight. We report for each image its label and the targeted label, as label $\mapsto$ targeted label, and the norm of the perturbations needed to create an adversarial example. We show in green each successful attempt to create an adversarial attack to the targeted class.

4.1 Attack on weights

4.1.1 One layer without activation function

Here the network is just one layer of 10 neurons and it achieves 92% of accuracy on the test data. The norm values reported in the table below is $\frac{\|\Delta A_1\|}{\|A_1\|}$.

Table 1: Norm of the adversarial attacks on weights for a single layer neural network.

Class	$7 \mapsto 0$	$7 \mapsto 1$	$7 \mapsto 2$	$7 \mapsto 3$	$7 \mapsto 4$	$7 \mapsto 5$	$7 \mapsto 6$	$7 \mapsto 8$	$7 \mapsto 9$
Norm	0.0492	$< 10^{-4}$	0.0283	0.0284	0.0120	0.0438	0.0288	0.0087	0.0102
Class	$8 \mapsto 0$	$8 \mapsto 1$	$8 \mapsto 2$	$8 \mapsto 3$	$8 \mapsto 4$	$8 \mapsto 5$	$8 \mapsto 6$	$8 \mapsto 7$	$8 \mapsto 9$
Norm	0.0487	0.0008	0.0183	0.0053	0.0394	0.0299	0.0365	0.0214	0.0099
Class	$5 \mapsto 0$	$5 \mapsto 1$	$5 \mapsto 2$	$5 \mapsto 3$	$5 \mapsto 4$	$5 \mapsto 6$	$5 \mapsto 7$	$5 \mapsto 8$	$5 \mapsto 9$
Norm	0.0328	0.0563	0.0344	0.0036	0.0144	0.0400	0.0285	0.0047	0.0187
Class	$4 \mapsto 0$	$4 \mapsto 1$	$4 \mapsto 2$	$4 \mapsto 3$	$4 \mapsto 5$	$4 \mapsto 6$	$4 \mapsto 7$	$4 \mapsto 8$	$4 \mapsto 9$
Norm	0.0391	0.0451	0.0146	0.0067	0.0447	0.0357	0.0181	0.0247	0.0184

In this case, as we said in section 2.3, we find the same result as Rigal and Gaches, so we have an explicit formula enabling us to compute optimal perturbations, hence we successfully misclassify each image to each possible class, with very small perturbations (their norm is typically around 1% that of the weight's norm).

4.1.2 Two layers with activation function

Here the network is composed of two layers of 100 and 10 neurons each followed by hyperbolic tangent as activation function, and it achieves 97% of accuracy on the test data. The norm values reported in the table below correspond to $\max(\frac{\|\Delta A_1\|}{\|A_1\|}, \frac{\|\Delta A_2\|}{\|A_2\|})$.

Table 2: Norm of the adversarial attacks on weights for a two layers neural network.

Class	$7 \mapsto 0$	$7 \mapsto 1$	$7 \mapsto 2$	$7 \mapsto 3$	$7 \mapsto 4$	$7 \mapsto 5$	$7 \mapsto 6$	$7 \mapsto 8$	$7 \mapsto 9$
Norm	0.0802	0.0663	0.0774	0.0787	0.0672	0.0760	0.0717	0.0667	0.0642
Class	$8 \mapsto 0$	$8 \mapsto 1$	$8 \mapsto 2$	$8 \mapsto 3$	$8 \mapsto 4$	$8 \mapsto 5$	$8 \mapsto 6$	$8 \mapsto 7$	$8 \mapsto 9$
Norm	0.0591	0.0170	0.1092	0.1180	0.0648	0.0860	0.0543	0.0218	0.0839
Class	$5 \mapsto 0$	$5 \mapsto 1$	$5 \mapsto 2$	$5 \mapsto 3$	$5 \mapsto 4$	$5 \mapsto 6$	$5 \mapsto 7$	$5 \mapsto 8$	$5 \mapsto 9$
Norm	0.0857	0.0955	0.0865	0.0704	0.1004	0.0916	0.0982	0.0927	0.0984
Class	$4 \mapsto 0$	$4 \mapsto 1$	$4 \mapsto 2$	$4 \mapsto 3$	$4 \mapsto 5$	$4 \mapsto 6$	$4 \mapsto 7$	$4 \mapsto 8$	$4 \mapsto 9$
Norm	0.0772	0.0855	0.1027	0.1118	0.1002	0.0789	0.0959	0.0791	0.1028

For this setting, some tests did not achieve the expected result, as reported by the red cells; in these cases, the perturbation resulted in a misclassification in a class other than the target one. This is likely due to a malfunctioning of the optimizer used for solving the problem in equation (6). We reserve to future work the use of more efficient optimizers. We can remark that the reported values are, on average, larger than those reported in the previous section due to the higher complexity of the network or, again, because of the inefficiency of the optimizer.

4.2 Attacks on the input data

4.2.1 One layer without activation function

In figure 1 we show, for multiple input images and classes, the perturbed image resulting from an adversarial attack computed with the approach proposed in section 3. Above each image is the obtained label and, in parenthesis, the norm of the perturbation.

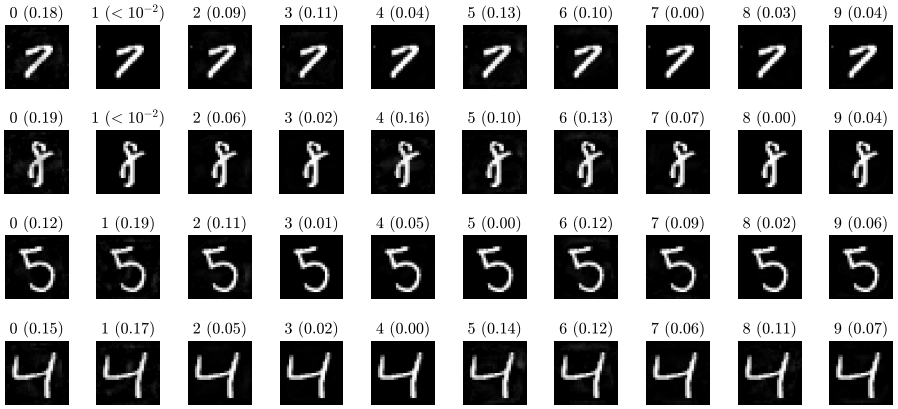

Figure 1: Adversarial examples found with BE attack

We remark that for perturbations of relative norm greater than approximately 0.1 we start noticing slight white stains on the perturbed image, but a human eye would still classify these perturbed images in their original class.

4.2.2 Two layers with activation function

Here we compare, for four different images, all adversarial examples found by our method and by non-targeted and targeted FGSM in figures 2 and 3 and 4, respectively. Above each image is the obtained label and, in parenthesis, the norm of the perturbation. Adversarial examples which did not result in the intended class are not showed here, but as in 4.1.2, in these cases, the perturbation resulted in a misclassification in a class other than the target one.

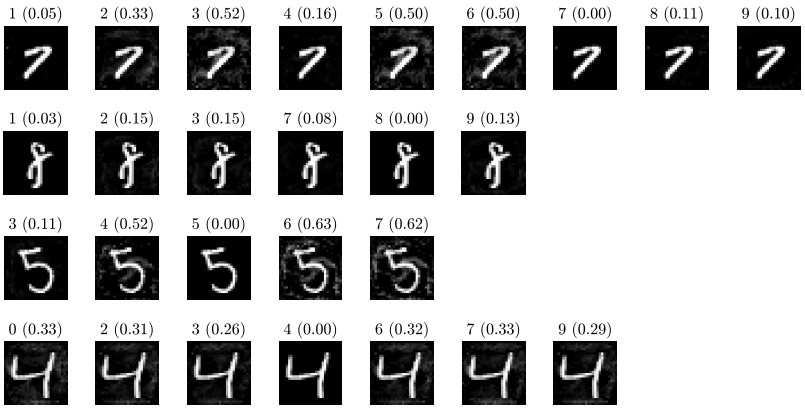

Figure 2: Adversarial examples found with BE attack

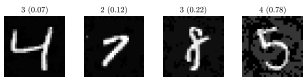

Figure 3: Adversarial examples found with FGSM attack

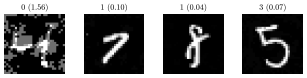

Figure 4: Adversarial examples found with targeted FGSM attack

There are multiple differences between the images resulting from these two methods:

- For a given image and a given target label, the two methods often give different results in terms of perturbations norm. In some cases FGSM or targeted FGSM will achieve better performance (4 labeled as 3) and in some others our method will (4 labeled as 0). Whereas FGSM can attain smaller perturbation's norm for a given image and a given label our method can still find better results for the same image with another label, as it can successfully lead to misclassify a given image to a variety of target labels.
- Perturbation patterns are quite different; if we compare the two images representing a 5 which are labeled as a 4, for FGSM perturbations are uniform on a given set of pixels, and as reported in [19] it's a result which is common to all currently used gradient-based methods. On the contrary, for our method, perturbations are more diffuse. This would be helpful when one wants to train a neural network with adversarial images. Indeed, adversarial training is the most investigated [7, 17] way to make DNNs robust. Adversarial training consist on injecting adversarial examples on the training data set, hence network which are trained with adversarial examples obtained by gradient methods learn to resist to those kinds of perturbations. Hence those network may not be robust to adversarial examples obtained by backward error approach.

5 Conclusion and limitations

We have performed a backward error analysis of generic deep neural networks. Our analysis provides formulas and a numerical algorithm that can be used to construct adversarial attacks in a novel way, on either the input data or the neural network's parameters.

Our analysis makes uses of first order approximations, which means that, in the case where the perturbations needed to attain a given output vector are large, the not-so-small second order terms could make the results inexact. However, this should not be a problem in the context of adversarial attacks, which focus on small perturbations.

Our experiments focus on neural networks with few layers, trained on a simple dataset (MNIST). The goal of this paper is to provide a first proof-of-concept that successful adversarial attacks can be built via backward error analysis, and to show how they are different from most of the state-of-the-art attacks. These preliminary results illustrate the potential of backward error analysis, and we expect that our method can be further improved and refined to target deeper networks using more robust optimization solvers. Once this is achieved, further statistical studies to compare performances of this approach to state-of-the-art attacks will be considered.

6 Broader impact

The existence of new types of adversarial attacks poses potential security threats to machine learning models. This work shows how to construct adversarial attacks on a neural network's weights and input data. However as it is a new approach in development we do not expect it to have an immediate effect on existing robust models. Moreover such attacks often enable to develop more robust deep learning systems by using them to train neural networks. On the other hand we think that by using classical numerical analysis tools, such as backward error, we could develop new ways of evaluating the robustness of neural networks.

References

[1] N. Akhtar and A. Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *CoRR*, abs/1801.00553, 2018.

[2] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines, 2013.

[3] F. Chaitin-Chatelin and V. Frayssé. Lectures on finite precision computations. In *Software, environments, tools*, 1996.

[4] F. Croce and M. Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack, 2020.

[5] F. Croce and M. Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020.

[6] S. Garg, A. Kumar, V. Goel, and Y. Liang. Can adversarial weight perturbations inject neural backdoors. *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, Oct 2020.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015.

[8] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37, 11 2019.

[9] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[10] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world, 2017.

[11] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[12] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[13] J. Neumann and H. Goldstine. Numerical inverting of matrices of high order. *Bulletin of the American Mathematical Society*, 53:1021–1099, 1947.

[14] P. Rajpurkar, A. Y. Hannun, M. Haghpanahi, C. Bourn, and A. Y. Ng. Cardiologist-level arrhythmia detection with convolutional neural networks. *CoRR*, abs/1707.01836, 2017.

[15] J. L. Rigal and J. Gaches. On the compatibility of a given solution with the data of a linear system. *J. ACM*, 14(3):543–548, July 1967.

[16] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2014.

[17] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses, 2020.

[18] A. M. Turing. Rounding-off errors in matrix processes. *The Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):287–308, 01 1948.

[19] X. Wang, K. He, C. Song, L. Wang, and J. E. Hopcroft. At-gan: An adversarial generator model for non-constrained adversarial examples, 2020.

[20] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994.

[21] D. Wu, S. tao Xia, and Y. Wang. Adversarial weight perturbation helps robust generalization, 2020.