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

Pierre Boudier
NVIDIA

Alfredo Buttari
Université de Toulouse

Serge Gratton
Université de Toulouse

Théo Mary
Sorbonne Université

Stéphane Pralet
ATOS

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 on input data and on DNN's parameters. 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 (Grigorescu et al., 2019) and medicine (Rajpurkar et al., 2017). However, many works (Goodfellow et al., 2015; Kurakin et al., 2017; Akhtar and Mian, 2018) 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 crucial to explore their vulnerability against attackers. Several types of attacks are now well known, such as adversarial examples (Szegedy et al., 2014; Madry et al., 2019), poisoning the training data (Biggio et al., 2013), 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 (Garg et al., 2020) despite its potential use to help robust generalization (Wu et al., 2020).

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 targeted 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. We show that when using only perturbations on the data we can outperform state of art algorithms such as PGD (Madry et al., 2019) or DeepFool (Moosavi-Dezfooli et al., 2016), and that by using only perturbations on the network's parameters we can alter a network's accuracy as efficiently.

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 (Higham, 2002; Chaitin-Chatelin and Frayssé, 1996). 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 Wilkinson (1963) in the 1950s and 1960s, with origins in the works of Neumann and Goldstine (1947) and Turing (1948).

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 attacks 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:

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

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 $\overrightarrow{A} = \text{vec}(A)$. We recall some properties of the Kronecker product which will be used in the remainder of this document:

$$(A \bigotimes C)(B \bigotimes D) = (AB) \bigotimes (CD),$$

$$\left\| A \bigotimes B \right\| = \|A\| \, \|B\|,$$

$$|A \bigotimes B| = |A| \bigotimes |B|,$$

$$\text{vec}(AXB) = (B^T \bigotimes A) \, \text{vec}(X).$$

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

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

$$\text{and} \quad \overrightarrow{\Delta A} = \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \Delta x \end{bmatrix}.$$

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

$$\mathcal{A}^+ = \mathcal{A}^T (\mathcal{A}\mathcal{A}^T)^{-1}$$

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 (1967).

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

$$\mathcal{A} = \begin{bmatrix} x^T \bigotimes A_2, & x^T A_1^T \bigotimes I, & A_2 A_1 \end{bmatrix}$$

$$\text{and} \quad \overrightarrow{\Delta A} = \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \overrightarrow{\Delta A_2} \\ \Delta x \end{bmatrix}.$$

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

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

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

$$\begin{aligned}\Delta y &= f_1'(A_1x)\Delta A_1 x + f_1'(A_1x)A_1\Delta x \\ &= \left[x^T \bigotimes f_1'(A_1x), \quad f_1'(A_1x)A_1\right] \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \Delta x \end{bmatrix}.\end{aligned}$$

Let $\mathcal{A} = \left[x^T \bigotimes f_1'(A_1x), \quad f_1'(A_1x)A_1\right]$; 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

$$\begin{aligned}\mathcal{A}\mathcal{A}^T &= x^T x \bigotimes f_1'(A_1x))(f_1'(A_1x))^T \\ &\quad + f_1'(A_1x)A_1(f_1'(A_1x)A_1)^T \\ &= \|x\|^2 (f_1'(A_1x))(f_1'(A_1x))^T \\ &\quad + f_1'(A_1x)A_1(f_1'(A_1x)A_1)^T.\end{aligned}$$

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

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

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

$$\mathcal{A}^T = \begin{bmatrix} x \bigotimes (f_p'(A_py_{p-1})A_p \ldots f_1'(A_1x))^T \\ \vdots \\ y_{i-1} \bigotimes (f_p'(A_py_{p-1})A_p \ldots f_i'(A_iy_{i-1}))^T \\ \vdots \\ y_{p-1} \bigotimes (f_p'(A_py_{p-1}))^T \\ (f_p'(A_py_{p-1})A_p \ldots f_1'(A_1x)A_1)^T \end{bmatrix}.$$

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:

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

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 computes perturbations on both the weights and the input of the network.

3 Adversarial attacks via backward error analysis

3.1 Proposed approach

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

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

subject to

$$\begin{aligned}\widehat{y} &= f_p((A_p + \Delta A_p)f_{p-1}((A_{p-1} + \Delta A_{p-1}) \\ &\quad \ldots f_1((A_1 + \Delta A_1)(x + \Delta x))) \\ &\widehat{y}_i \leq \widehat{y}_j, \ i = 1, \ldots, n.\end{aligned} \tag{5}$$

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

$$\begin{aligned} M_i =& (f_p'(A_p y_{p-1}) A_p f_{p-1}'(A_{p-1} y_{p-2}) \\ & \dots A_{i+1} f_i'(A_i y_{i-1}))^T \mathcal{B} \quad \text{and} \\ M =& (f_p'(A_p y_{p-1}) A_p f_{p-1}'(A_{p-1} y_{p-2}) \\ & \dots A_2 f_1'(A_1 x) A_1)^T \mathcal{B}. \end{aligned}$$

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 depend on a given approximate result $\hat{y}$ and on the network's parameters. The optimization problem can then be 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

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

Once this optimization problem is solved and, thus, $\widehat{y}$ is computed, the adversarial perturbations can be computed using equation 4.

Alternatively, we can reformulate the optimization problem using backward error analysis to only simplify the constraints which reduce the problem to:

$$\begin{aligned} &\textbf{Solve} \\ &\quad \arg\min_{\Delta A_i, \Delta x} \sum_{i=1}^{p} \frac{\|\Delta A_i\|^2}{\|A_i\|^2} + \frac{\|\Delta x\|^2}{\|x\|^2} \\ &\textbf{subject to} \\ &\quad \Delta y = \mathcal{A} \overrightarrow{\Delta A}, \\ &\quad \widehat{y}_i \leq \widehat{y}_j, \ i = 1, \dots, n. \end{aligned} \tag{7}$$

with $\mathcal{A}$ defined as in section 2.4 and

$$\overrightarrow{\Delta A} = \begin{bmatrix} \overrightarrow{\Delta A_1} \\ \vdots \\ \overrightarrow{\Delta A_p} \\ \Delta x \end{bmatrix}.$$

This formulation can be further refined by iterating on the perturbations, for example in the case where we attack the input data we get:

$$\begin{aligned} &\textbf{Solve} \\ &\quad \min_{\delta x} \frac{\|\Delta x + \delta x\|^2}{\|x\|^2} \\ &\textbf{subject to} \\ &\quad \Delta y = \mathcal{A} \delta x, \\ &\quad \widehat{y}_i \leq \widehat{y}_j, \ i = 1, \dots, n, \end{aligned} \tag{8}$$

at the end of each iteration we apply the following modifications:

$$\begin{aligned} \Delta x &\longleftarrow \Delta x + \alpha \delta x \\ x &\longleftarrow x + \alpha \delta x \end{aligned}$$

with α a fixed regularization term. Following the same approach we can also get perturbations on the neural network's parameters.

These two formulations enable us to compute targeted adversarial attacks either on weights or on the input data and on both weights and input data. Note that here we focus on the case where the classifier assigns 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 these constraints.

3.2 Comparison with other approaches

Most of the approaches that generate adversarial examples, including FGSM (Goodfellow et al., 2015) or FGSM-based approaches (Kurakin et al., 2017), SGD (Madry et al., 2019) or SGD-based approaches (Croce and Hein, 2020b), FAB (Croce and Hein, 2020a), etc., use the gradient of the loss function in order to solve a given optimization problem. Usually the optimization problem can be generically formulated as:

$$\begin{aligned} &\textbf{Solve} \\ &\quad \min \|\Delta x\|^2 \\ &\textbf{subject to} \\ &\quad C(x + \Delta x) = j. \end{aligned} \tag{9}$$

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:

$$\begin{aligned} &\textbf{Solve} \\ &\quad \min c \|\Delta x\|^2 + \mathcal{L}(x + \Delta x, j), \end{aligned} \tag{10}$$

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 9 and formulate it as equation 6 or equation 7.

Unlike most existing approaches, our method is generic enough to enable to compute targeted adversarial attacks on both the neural network's parameters and input. Attacks on a network's parameters should not be neglected. In fact in a context where cloud computing is rising in popularity, neural network's inferences which happen on the cloud are subject to multiple threats such as unauthorized access through malicious co-located virtual machines (VM) on a same physical host or root access via host organization. In these situations an intruder can have access to a neural network's parameters and perturb them to launch adversarial attacks on specific inputs using our approach.

Finally, a notable advantage of our approach is that it does not require the knowledge of the loss function and of its gradient in order to find optimal perturbations; unlike other gradient-based adversarial attacks, the proposed attack only depends on the output information and the network's parameters. Indeed, when a neural network is deployed after being trained, no information on the loss function used to train it is available. Although only a few loss functions (e.g., mean square error, cross entropy) are most commonly used there are still numerous cases where non trivial loss functions are used which are difficult, if not impossible, to guess. One commonly occurring example is represented by Generative Adversarial Networks (GAN) (Goodfellow et al., 2014), where a discriminative network acts as a loss function.

4 Experimental results

For all of our experiments we train a fully connected neural network, with *tanh* activation functions, with Keras with Adam's optimizer and a sparse categorical cross entropy's loss on Python on the MNIST (LeCun et al., 2010) or CIFAR10 (Krizhevsky, 2009) database. We describe the neural network's structure in each subsection or on the first row of the corresponding table. If the network is a fully connected neural network with one hidden layer, with 784 nodes in the input layer and 100 nodes in the hidden layer on MNIST then we use the following notation: $(784, 100, 10)$. Once the network is trained, we use its weight matrices to compute adversarial attacks as described before, using MATLAB R2020a. For each experiment we take the 100 first images on the testing data set, we first solve the optimization problem equation 6 or the iterative version of equation 7, as specified in each case, 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. Hence we divide our experimental approach in two parts, one where we focus only on attacks on the neural network's input and one where we only want to attack the neural network's parameters. Neural networks for a given size and database are the same across sections 4.2 and 4.1.

4.1 Attacks on the input data

In this section we perform an attack only on the input and compare it to state of the art attacks such as fast gradient method (FGM) which is the $L2$ norm variation of FGSM (Goodfellow et al., 2015), projected gradient descent (PGD) (Madry et al., 2019) and DeepFool (Moosavi-Dezfooli et al., 2016), using the Foolbox library (Rauber et al., 2020). For each attack we search for the best hyper-parameter needed on Foolbox before comparing the different methods. We give the network accuracy for different ϵ assuming that the attack is successful if:

$$\frac{\|\Delta x\|}{\|x\|} < \epsilon.$$

Hence if we do not find any perturbations that changes the network's result for a given image, such that $\frac{\|\Delta x\|}{\|x\|} < \epsilon$ then we report a network accuracy of 100%.

In figure 1 we show, for multiple input images and classes, the adversarial example resulting from an attack on MNIST, using a $(784, 100, 10)$ network with *tanh* activation functions, computed with the approach proposed in section 3. Above each image is the obtained label and, in parenthesis, the norm of the perturbation.

For perturbations of relative norm greater than approximately 0.1 slight white stains on the perturbed image appear, although a human eye would still classify these perturbed images in their true label.

It is interesting to notice that in the examples of figure 1 we can get multiple adversarial examples for a given input image that are obtained with perturbations of relatively small norm; it clearly shows that our method is efficient in producing adversarial examples for multiple given target classes. In the following comparison we will only use non-targeted attacks to show that this method can compete with, and even outperform, state of the art attacks on a non-targeted scheme.

As seen in table 1, using the iterative version of formulation 7 leads to better results than using the formulation 6, hence we will use this one on the following comparisons.

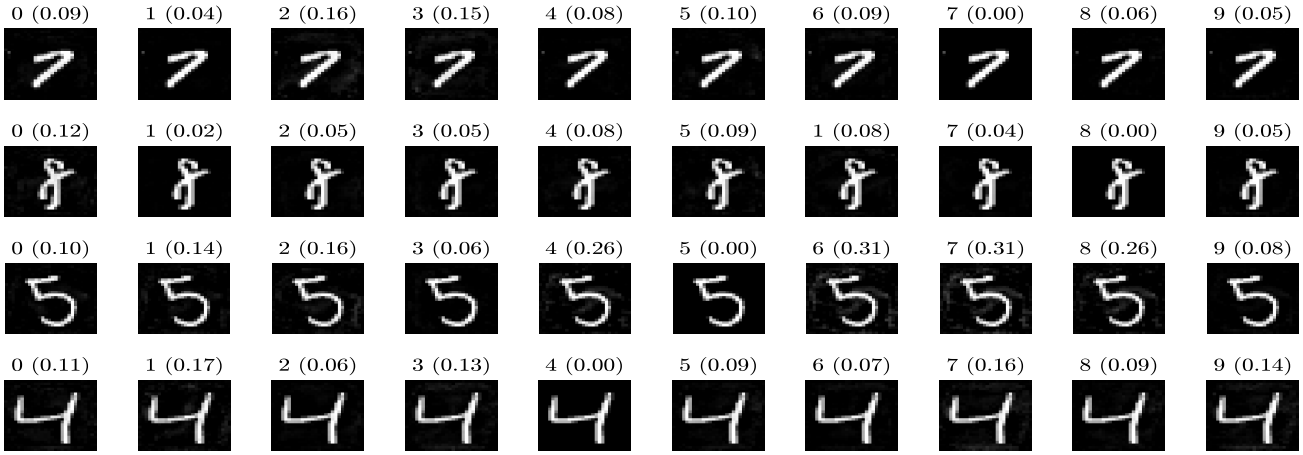

Figure 1: Adversarial Examples Found With BE Attack.

Table 1: Adversarial Attacks On MNIST, $(784, 10)$ Network

ϵ	BE attack (6)	BE attack (8)
0.2	70%	**23%**
0.1	90%	**41%**
0.05	97%	**68%**

Table 2: Adversarial Attacks On MNIST, $(784, 10)$ Network

ϵ	FGM	L2PGD	L2DeepFool	BE attack
0.2	60%	31%	27%	**23%**
0.1	79%	71%	60%	**41%**
0.05	92%	91%	72%	**68%**

In the case of table 2 the backward error attack clearly outperforms state of the art algorithms, finding much more adversarial examples with relative norm smaller than 0.1 compared to DeepFool or PGD.

Table 3: Adversarial Attacks On MNIST, $(784, 100, 10)$ Network

ϵ	FGM	L2PGD	L2DeepFool	BE attack
0.2	49%	**34%**	44%	41%
0.1	80%	81%	78%	**72%**
0.05	94%	96%	93%	**90%**

Here the network is composed of two layers of 768 and 10 neurons each followed by hyperbolic tangent as activation function, and it achieves 56% of accuracy on the CIFAR10 test data. For our computations we use the 100 first test images which are correctly classified.

Table 4: Adversarial Attacks On CIFAR10, $(3072, 768, 10)$ Network

ϵ	FGM	L2PGD	L2DeepFool	BE attack
0.2	1%	2%	**0%**	**0%**
0.1	5%	7%	**0%**	1%
0.05	18%	27%	**3%**	5%
0.01	77%	79%	**55%**	**55%**

The results we get in tables 2, 3, and 4 show that even by perturbing only the input data our method still obtains satisfactory results, in the majority of the cases outperforming PGD and DeepFool, and being at least competitive with these algorithms.

4.2 Attack on weights

In this section we perform an attack only on the neural network's weights. For each attack we give the network accuracy for different ϵ assuming that the attack is successful if:

$$\max_i \frac{\|\Delta A_i\|}{\|A_i\|} < \epsilon.$$

Hence if we do not find any perturbations that changes the network's result for a given image, such that $\max_i \frac{\|\Delta A_i\|}{\|A_i\|} < \epsilon$ then we report a network accuracy of 100%.

Unlike what we have seen in section 4.1, in table 5, formulation 6 leads to better results than using the iterative version of formulation 7, even if this still gives satisfying results. Hence on the following comparisons we will use this version.

Table 5: Adversarial Attacks On Weights On MNIST, $(784, 10)$ Network

ϵ	BE attack (6)	BE attack (7)
0.5	**30%**	39%
0.2	**38%**	51%
0.1	**50%**	61%
0.05	**64%**	83%
0.01	**88%**	100%

Table 6: Adversarial Attacks On Weights On MNIST

ϵ	$(784, 10)$ BE attack	$(784, 100, 10)$ BE attack
0.5	30%	37%
0.2	38%	64%
0.1	50%	72%
0.05	64%	84%
0.01	88%	95%

The results in table 6 show that backward error can be used to efficiently fool a given neural network by perturbing it's weights, moreover it is interesting to note that for a given ϵ the accuracy we can get by perturbing weights is typically of the same order than the one we get before in section 4.1 by perturbing the input data.

Table 7: Adversarial Attacks On Weights On CIFAR10, $(3072, 768, 10)$ Network

ϵ	BE attack
0.05	0%
0.01	4%
5e-3	9%
1e-3	56%
5e-4	73%
1e-4	93%

Here in table 7, even if the data set is more complex and the network has more parameters, these results show that some neural network are less robust than others when it comes to perturbations on their weights. The accuracy we get here for a given ϵ is typically a lot smaller than the one we get for an attack on the input in section 4.1, despite using the same neural network for both attacks. Hence some network are more sensitive to weight perturbations than to input perturbations.

5 Conclusion and discussion

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, without any knowledge of the loss function used to train the neural network, on either the input data or the neural network's parameters.

As seen in section 4, our method can outperform state of the art methods in the case were we attack a given network on the input data. Moreover it also enables to attack a network by perturbing its parameters and hence, for a given input, target a given class.

Our analysis relies on 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 couple of simple data sets (MNIST and CIFAR10). The goal of this paper is to provide a first proof-of-concept that successful adversarial attacks can be built via backward error analysis. We have shown how our approach can compete with state of the art attacks on the input, and how it can also create attacks on the network's parameters, which, to our knowledge, has not been the object of much investigation.

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.

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 even if it is a new approach in development and we do not expect it to have an immediate effect on existing robust models, it shows that models stored on environments that could potentially be the target of an intruder, such as cloud computing environments, could be very sensitive to this sort of attack. 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

Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *CoRR*, abs/1801.00553, 2018. URL http://arxiv.org/abs/1801.00553.

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines, 2013.

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

Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack, 2020a.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, 2020b.

Siddhant Garg, Adarsh Kumar, Vibhor Goel, and Yingyu Liang. Can adversarial weight perturbations inject neural backdoors. *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, Oct 2020. doi: 10.1145/3340531.3412130. URL http://dx.doi.org/10.1145/3340531.3412130.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37, 11 2019. doi: 10.1002/rob.21918.

Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002. ISBN 0-89871-521-0.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world, 2017.

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

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks, 2016.

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

Pranav Rajpurkar, Awni Y. Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y. Ng. Cardiologist-level arrhythmia detection with convolutional neural networks. *CoRR*, abs/1707.01836, 2017. URL http://arxiv.org/abs/1707.01836.

Jonas Rauber, Roland Zimmermann, Matthias Bethge, and Wieland Brendel. Foolbox native: Fast adversarial attacks to benchmark the robustness of machine learning models in pytorch, tensorflow, and jax. *Journal of Open Source Software*, 5(53): 2607, 2020. doi: 10.21105/joss.02607. URL https://doi.org/10.21105/joss.02607.

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. ISSN 0004-5411. doi: 10.1145/321406.321416. URL https://doi.org/10.1145/321406.321416.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

A. M. Turing. Rounding-off errors in matrix processes. *The Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):287–308, 01 1948. ISSN 0033-5614. doi: 10.1093/qjmam/1.1.287. URL https://doi.org/10.1093/qjmam/1.1.287.

J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963. ISBN 0-486-67999-3. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994.

Dongxian Wu, Shu tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization, 2020.