Backward error analysis of artificial neural networks with applications to floating-point computations and adversarial attacks

Théo Beuzeville

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Université de Toulouse

Doctorat de l'Université de Toulouse

préparé à Toulouse INP

Analyse inverse des erreurs des réseaux de neurones artificiels avec applications aux calculs en virgule flottante et aux attaques adverses

Thèse présentée et soutenue, le 7 juin 2024 par

Théo BEUZEVILLE

École doctorale
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse

Spécialité
Informatique et Télécommunications

Unité de recherche
IRIT - Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Alfredo BUTTARI

Composition du jury
M. Philippe LANGLOIS, Président et rapporteur, Université de Perpignan Via Domitia
M. Nicolas BRISEBARRE, Rapporteur, École Normale Supérieure de Lyon
Mme Fabienne JÉZÉQUEL, Examinatrice, Université Paris-Panthéon-Assas
M. Ehouarn SIMON, Examinateur, Toulouse INP
M. Alfredo BUTTARI, Directeur de thèse, Toulouse INP
M. Nicolas WINCKLER, Co-directeur de thèse du monde socio-économique, Eviden

Membres invités
M. Théo MARY, Sorbonne Université
M. Serge GRATTON, Toulouse INP

Abstract

The use of artificial intelligence, whose implementations are often based on artificial neural networks, is now becoming widespread across a wide variety of tasks. These deep learning models indeed yield much better results than many specialized algorithms previously used and are therefore being deployed on a large scale. It is in this context of very rapid development that issues related to the storage of these models emerge, since they are sometimes very deep and therefore comprise up to billions of parameters, as well as issues related to their computational performance, both in terms of accuracy and time- and energy-related costs. For all these reasons, the use of reduced precision is increasingly being considered. On the other hand, it has been noted that neural networks suffer from a lack of interpretability, given that they are often very deep models trained on vast amounts of data. Consequently, they are highly sensitive to small perturbations in the data they process. Adversarial attacks are an example of this; since these are perturbations often imperceptible to the human eye, constructed to deceive a neural network, causing it to fail in processing the so-called adversarial example.

The aim of this thesis is therefore to provide tools to better understand, explain, and predict the sensitivity of artificial neural networks to various types of perturbations. To this end, we first extended to artificial neural networks some well-known concepts from numerical linear algebra, such as condition number and backward error. These quantities allow to better understand the impact of perturbations on a mathematical function or system, depending on which variables are perturbed or not. We then use this backward error analysis to demonstrate how to extend the principle of adversarial attacks to the case where not only the data processed by the networks is perturbed but also their own parameters. This provides a new perspective on neural networks' robustness and allows, for example, to better control quantization to reduce the precision of their storage. We then improved this approach, obtained through backward error analysis, to develop attacks on network input comparable to state-of-the-art methods. Finally, we extended approaches of roundoff error analysis, which until now had been approached from a practical standpoint or verified by software, in neural networks by providing a theoretical analysis based on existing work in numerical linear algebra. This analysis allows for obtaining bounds on forward and backward errors when using floating-point arithmetic. These bounds both ensure the proper functioning of neural networks once trained, and provide recommendations on architectures and training methods to enhance the robustness of neural networks.

Key words: artificial neural networks, floating-point, error analysis, adversarial attacks, rounding errors, backward error

Résumé

L'utilisation d'intelligences artificielles, dont les implémentations reposent souvent sur des réseaux de neurones artificiels, se démocratise maintenant dans une grande variété de tâches. En effet, ces modèles d'apprentissage profond produisent des résultats bien meilleurs que de nombreux algorithmes spécialisés précédemment utilisés et sont donc amenés à être déployés à grande échelle. C'est dans ce contexte de développement très rapide que des problématiques liées au stockage de ces modèles émergent, car ils sont parfois très profonds et comprennent donc jusqu'à des milliards de paramètres, ainsi que des problématiques liées à leurs performances en termes de calcul tant d'un point de vue de précision que de coût en temps et en énergie. Pour toutes ces raisons, l'utilisation de précision réduite est de plus en plus indispensable. D'autre part, il a été noté que les réseaux de neurones souffrent d'un manque d'interprétabilité, étant donné qu'ils sont souvent des modèles très profonds, entraînés sur de vastes quantités de données. Par conséquent, ils sont très sensibles aux perturbations qui peuvent toucher les données qu'ils traitent. Les attaques adverses en sont un exemple ; ces perturbations, souvent imperceptibles à l'œil humain, sont conçues pour tromper un réseau de neurones, le faisant échouer dans le traitement de ce qu'on appelle un exemple adverse.

Le but de cette thèse est donc de fournir des outils pour mieux comprendre, expliquer et prédire la sensibilité des réseaux de neurones artificiels à divers types de perturbations. À cette fin, nous avons d'abord étendu à des réseaux de neurones artificiels certains concepts bien connus de l'algèbre linéaire numérique, tels que le conditionnement et l'erreur inverse. Nous avons donc établi des formules explicites permettant de calculer ces quantités et trouvé des moyens de les calculer lorsque nous ne pouvions pas obtenir de formule. Ces quantités permettent de mieux comprendre l'impact des perturbations sur une fonction mathématique ou un système, selon les variables qui sont perturbées ou non. Nous avons ensuite utilisé cette analyse d'erreur inverse pour démontrer comment étendre le principe des attaques adverses au cas où, non seulement les données traitées par les réseaux sont perturbées, mais également leurs propres paramètres. Cela offre une nouvelle perspective sur la robustesse des réseaux neuronaux et permet, par exemple, de mieux contrôler la quantification des paramètres pour ensuite réduire la précision arithmétique utilisée et donc faciliter leur stockage. Nous avons ensuite amélioré cette approche, obtenue par l'analyse d'erreur inverse, pour développer des attaques sur les données des réseaux comparables à l'état de l'art. Enfin, nous avons étendu les approches d'analyse des erreurs d'arrondi, qui jusqu'à présent avaient été abordées d'un point de vue pratique ou vérifiées par des logiciels, dans les réseaux de neurones en fournissant une analyse théorique basée sur des travaux existants en algèbre linéaire numérique. Cette analyse permet d'obtenir des bornes sur les erreurs directes et inverses lors de l'utilisation d'arithmétiques flottantes. Ces bornes permettent à la fois d'assurer le bon fonctionnement des réseaux de neurones une fois entraînés, mais également de formuler des recommandations concernant les architectures et les méthodes d'entraînement afin d'améliorer la robustesse des réseaux de neurones.

Mots clés : réseaux de neurones artificiels, virgule flottante, analyse d'erreur, attaques adverses, erreurs d'arrondi, erreur inverse

Remerciements

Je tiens tout d'abord à remercier ceux qui ont rendu cette thèse possible. En particulier mes encadrants, à commencer par Serge Gratton qui m'a introduit au monde de la recherche dès ma deuxième année d'école d'ingénieurs et vers qui je me suis tourné par la suite pour ce sujet de thèse. En plus de son implication et de ses efforts lors de la création de ce projet de thèse, Serge a joué un rôle essentiel pendant son développement grâce à ses remarques et réflexions toujours constructives. Je tiens aussi à exprimer mes remerciements sincères envers Stéphane Pralet, dont la contribution à l'élaboration de cette thèse et son financement par Atos ont été essentiels. Grâce à lui, j'ai toujours bénéficié d'une grande liberté pour mener mes recherches. Je tiens également à remercier Nicolas Winckler, qui a pris la suite de Stéphane. Son encadrement en fin de thèse et son point de vue extérieur ont grandement aidé à présenter mes travaux de manière plus claire. Je remercie également Théo Mary, qui a suivi cette thèse depuis le début. Une grande partie des travaux présentés ici se base sur ses recherches récentes, et les interactions avec lui ont toujours été source de discussions et de travaux intellectuellement stimulants. Son aide et son intérêt pour cette thèse m'ont permis d'avancer avec confiance dans mon travail. Je tiens à remercier tout particulièrement Alfredo Buttari, mon directeur de thèse, pour sa bienveillance et sa présence toujours rassurante tout au long de ces trois années de thèse. Alfredo a toujours su se rendre disponible, être à l'écoute, et son intelligence scientifique, mais également humaine, m'ont énormément apporté et aidé pendant cette période.

J'aimerais remercier Philippe Langlois et Nicolas Brisebarre d'avoir accepté d'évaluer mes travaux. Leur intérêt pour ma thèse, le temps qu'ils ont consacré à fournir un rapport, leurs remarques et leurs questions pertinentes ont été extrêmement précieux. Je remercie également mes examinateurs, Fabienne Jézéquel et Ehouarn Simon, pour leurs retours et discussions sur mon travail.

Je remercie tous les chercheurs de l'IRIT, et j'ai une pensée particulière pour tous les doctorants et docteurs de l'équipe APO qui ont participé à l'ambiance au laboratoire qui a rendu les journées de travail plus agréables.

Cette thèse marque l'aboutissement de longues années d'études. Je tiens à remercier sincèrement les enseignants qui m'ont aidé durant ces années, en particulier M. Desanlis, Mme Devicque, Mme Direz et M. Gozard. Je remercie également mes amis qui m'ont accompagné pendant toutes ces années, depuis le collège pour certains, et mes pensées vont évidemment vers toute la classe euro du lycée, la team corrézienne de club pro, les sup 2, la team 5/2, etc. Je pense évidemment aussi à toutes les personnes qui m'ont entouré pendant les études, en particulier mes clubs d'athlétisme, l'ECA et le SATUC, ainsi que leurs entraineurs et athlètes.

Pour finir, je remercie toute ma famille, merci à ma sœur qui m'a permis de mieux grandir, à mon père qui m'a très certainement poussé à rentrer dans le monde du travail le plus tard possible et donc à poursuivre mes études le plus longtemps possible, et bien sûr, à ma mère qui m'a toujours accompagné et aidé pendant toutes ces années.

Contents

List of Tables

List of Figures

Introduction

The term Artificial Intelligence (AI) refers to the ability of a computer or a system to perform tasks that are commonly associated with human intelligence or intelligent beings, such as learning, reasoning, problem-solving, understanding natural language and interacting with the environment. Artificial Intelligence has now a wide range of applications, from virtual assistants and recommendation systems to autonomous vehicles and medical diagnosis. This recent development of AI is largely due to Deep Learning, a subset of Machine Learning (ML), which explains why those terms are often confused. Machine learning is one of the fields of AI and consists in building algorithms that are able to learn from available data and then generalize to unseen data. Indeed, hard-coding knowledge in order to get AI systems seems impractical and it is therefore natural that systems that are able to extract knowledge from data are more desirable. We are therefore moving from a model-based design approach, rooted on algorithmic logic, to a data-centric approach where it is much more difficult to get explicit proof of accuracy or confidence bounds.

Deep Neural Networks (DNNs) are a specific type of ML model that achieve state-of-the-art performance in many machine learning tasks and in various types of applications. They are known to be universal approximators [59, 101]. In fact, many results assert that a small approximation error can be achieved if the network size is sufficiently large and provide upper bounds which depend on the network size. Those theoretical results are often quoted to justify the empirical efficiency of deep neural networks: indeed, they exceed human accuracy in many tasks and even in playing games [104, 20, 81]. Their efficiency in solving complex problems has led to apply deep learning techniques in safety-critical tasks, from medicine with the detection of cancer [22] to driving cars [124]. However, DNNs are sensitive to various perturbations. For example, the response of the neural networks can be sensitive to environmental perturbations such as variations in image brightness, macroscopic transformations that preserve the semantics of the data [52, 33, 34] (e.g. zoom level, translations, rotations) and infinitesimal transformations of the adversarial type [108, 72, 40, 84] but also to rounding errors, or quantization processes [24, 119]. This raises concerns and has lead to high interest in finding new approaches to make them more robust.

However, the work on neural networks error analysis available in the literature is mainly based on experimental approaches [77, 64, 90] and very few studies address a more theoretical framework.

The objective of this thesis is therefore to propose a theoretical framework for the error analysis in neural networks based on traditional work in numerical linear algebra. We will hence develop numerical analysis methods and tools applicable in a context in which errors may come from environmental perturbations, adversarial transformations, and arithmetic constraints in order to estimate the prediction quality and stability of neural networks. This allows us to understand what are the sources of errors and instabilities, and thus to contribute to a more explainable AI.

We will start this manuscript by investigating mathematical tools that enable to quantify a system's sensitivity to perturbations. Developed and popularized by James Wilkinson in the 1950s and 1960s [120, 122], with origins in the works of Neumann and Goldstine [86] (1947) and Turing [114] (1948), backward error analysis 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. In numerical analysis, *backward error* is a particularly well-established tool [56, 113], as it enables one to know if an inexact solution to a problem is in fact the exact solution to a nearby problem with slightly perturbed input data. Then depending on prior knowledge on the problem, such as uncertainty on the input data, one can say that said problem is backward stable if the backward error is close to these uncertainties which essentially means that the algorithm has computed a solution which is as good as it can be. We will hence begin by introducing different mathematical quantities typically used for error analysis and robustness analysis in numerical linear algebra, such as backward error and *condition number*.

As a first contribution, this thesis develops generic definitions of these quantities for deep neural networks, as well as generic formulas for computing them numerically. This theoretical framework enables us to have tools to quantify and better explain how generic perturbations affect artificial neural networks. This leads us to have a closer look at a more specific type of perturbations, *adversarial attacks* on neural networks, since we will demonstrate that computing the backward error for classification neural networks amounts to design adversarial attacks. First introduced in 2014 by Szegedy et al. [108], adversarial attacks most frequently refer to carefully crafted perturbations applied to input data with the intention of deceiving the neural network into making incorrect predictions or classifications. These perturbations are often imperceptible to the human eye but can still cause significant disruptions in the network's computations. Adversarial attacks may come in various forms, including adversarial examples, where the adversary seeks to perturb input data to produce incorrect predictions, and poisoning attacks [128], where the training data is manipulated to compromise the integrity of the model. The existence of adversarial attacks challenges the robustness and reliability of neural network models. Therefore, understanding the mechanisms behind adversarial attacks is essential to develop more resilient and trustworthy AI systems. The motivations for constructing adversarial attacks are diverse, ranging from exploring vulnerabilities in neural network architectures to finding ways to make

neural networks more robust via *adversarial training* [46, 60, 76] but also highlighting the potential lack of integrity of AI-driven decision-making processes. We will therefore introduce a new form of attacks, based on a backward error analysis, that consists in perturbing at the same time the inputs of a neural network as well as its parameters. We will then adapt this attack to be competitive with state-of-the-art attacks when perturbing the input data.

Finally, the last type of perturbations we will be interested in are those which are specific to the use of finite precision in computations. Indeed, machines perform the computations with finite precision, so they make arithmetic errors, rounding errors. Machine learning methods and their software implementations, even the most advanced ones, are therefore subject to rounding errors resulting from the use of finite precision arithmetic in calculations. Numerical accuracy dictates the precision and reliability of computational results. In the context of AI, where algorithms process vast amounts of data and execute complex computations, maintaining numerical accuracy is crucial. The success of deep learning mostly relies on the increasing size of both the number of parameters of the neural networks used but also on the size of training datasets. This leads to higher energy and computational costs for the training of deep learning models, as well as limiting their ability to be integrated into an embedded environment for an inference phase with more constraining hardware than during training. Therefore, in modern machine learning, low-precision arithmetics are becoming increasingly attractive due to both their higher speed and their lower energy consumption [5, 50, 126]. Proposed solutions demonstrate significant enhancements in power consumption, processing speed, and memory usage by recommending the replacement of widely used 32-bit floating-point arithmetic by a low-precision approach.

However, rounding errors can have a considerable impact on the robustness of AI methods and tools, where robustness is defined as the ability to maintain correct behaviour in the presence of disturbances of different nature and origin. To be able to deploy neural networks in critical systems, we have to ensure that these errors do not modify their functioning, their operation and properties. Hence, the ability to measure the accuracy of numerical programs, to analyse the errors due to rounding in the computations, and therefore to estimate which computations are more sensitive to changes of arithmetic is essential [56, 19].

On a more practical level, software such as CADNA [39, 65] and FLUCTUAT [48] can be used to assess, with some precision, how reliable the result of an algorithm is. FLUCTUAT is used by many companies to check the robustness of their software. Depending on the software used, the robustness and validity of the result obtained by the algorithm will be assessed differently. CADNA is based on a stochastic approach and can therefore produce unsatisfying results depending on the used algorithm, whereas FLUCTUAT is more often chosen in industry, for example in fields such as aerospace or nuclear engineering, when certifications require deterministic results, since it provides both a worst-case analysis (interval semantics) and knowledge of the

sources of errors (error series semantics) [78]. The extension of these types of verifier for artificial neural networks is still in progress [105] and raises many questions in terms of computation time as they are based on computationally expensive methods such as SMT (Satisfiability Modulo Theories) solving [68] or mixed integer linear programming [112], but also because of their own rounding error [66].

Although universities and industry are increasingly interested in the safety implications of artificial intelligence, the possibility of integrating these algorithms into critical equipment has not yet received sufficient attention. In particular, what we will be concentrating on now, is to produce theoretical and experimental results enabling us to understand the propagation of rounding errors in neural network architectures. We will therefore provide a rounding error analysis of artificial neural networks, which leads us to obtaining bounds on backward and forward errors. Those bounds enable us to better understand how sensitive to perturbations and rounding errors neural networks are, depending on the choice of architecture, training and scaling.

This manuscript is organized as follows.

We will present in Chapter 1 a background on the use of floating-point arithmetic, rounding error analysis and artificial intelligence.

In Chapter 2 we will first establish formulas and ways to compute the backward error of artificial neural networks; existing work focuses on numerical linear algebra, therefore our goal will be to extend it by integrating activation functions, which induce nonlinearities. We also show how to compute the condition number, which, in turn, helps in establishing bounds on the forward error once we have determined bounds on the backward error.

In Chapter 3 we will show how the question of computing the backward error of a classifier naturally leads to adversarial attacks. We will then show how we can extend the use of adversarial attacks, which in the literature focus primarily on perturbing the input data, to the classifier's parameters. We will then propose an improvement of this method to compete with state-of-the-art adversarial attacks on the input data.

In Chapter 4 we will focus on producing a rounding error analysis of neural networks, therefore finding bounds on the backward and forward error. First using the deterministic rounding error analysis approach and then showing how it can be extended to probabilistic bounds that are sharper.

Finally, the manuscript concludes with a summary of the main findings and some perspectives of future works.

Chapter 1

Background

This manuscript will focus on the effects of perturbations on multilayered artificial neural networks. In this chapter, we recall key notions and results that will serve as the basis for our approach to quantify and predict the effects of perturbations on artificial neural networks.

1.1 Measures of error

When solving a problem using numerical computations, uncertainties on the result can have multiple sources.

The first one being uncertainties in the input data, which may be due to measurement uncertainty at acquisition time, as well as errors coming from prior computations and even errors due to the storage of this data, since they have to be stored using finite precision.

This input data is then processed by an algorithm using finite precision arithmetic, therefore rounding errors will occur within each performed operation.

In order to be able to quantify how these errors affect computations, we will introduce different measures of error. For these quantities to be useful and not overestimated, the choice of a relevant metric is crucial. In this work we will mainly focus on a componentwise analysis as this metric enables to take into account the structure of matrices, such as sparsity or scaling. Unlike normwise metrics, when using componentwise metrics, each element of a perturbation on the data is measured relatively to a given tolerance, which can for example be its absolute value.

1.1.1 Forward error

Let $\widehat{y}$ be a computed result that is an approximation of $y = f(x)$, with $f : \mathbb{R}^n \to \mathbb{R}^m$. Multiple ways to estimate the quality of $\widehat{y}$ exists. Given a computed solution to a problem, the most well-known measure of error is the forward error, which measures directly the difference, or distance, between the computed solution $\widehat{y}$ and the exact solution y; this quantity is therefore directly linked with a norm.

Formally, for some given norm $\|\cdot\|$, we have the absolute forward error defined as

$$\varepsilon_{\text{fwd}}^{\text{abs}}(\widehat{y}) = \|\widehat{y} - y\|$$

as well as the relative forward error, defined for $y \neq 0$,

$$\varepsilon_{\text{fwd}}^{\text{rel}}(\widehat{y}) = \frac{\|\widehat{y} - y\|}{\|y\|}. \tag{1.1}$$

The componentwise absolute forward error is widely used in error analysis and perturbation analysis, and is defined as

$$\varepsilon_{\text{fwd}}^{\text{abs}}(\widehat{y}) = \max_i |\widehat{y}_i - y_i|$$

and its associated componentwise relative forward error as

$$\varepsilon_{\text{fwd}}^{\text{rel}}(\widehat{y}) = \max_i \frac{|\widehat{y}_i - y_i|}{|y_i|}. \tag{1.2}$$

The relative forward error is undefined for $y = 0$, but has the property of being scale invariant. It is therefore the most commonly used type of error measurement.

1.1.2 Backward error

When the forward error is large, we cannot distinguish if either the mathematical problem is sensitive to perturbations or the algorithm used to solve the problem behaves badly when perturbations exist on data or computations. The backward error is a quantity that makes it possible to discriminate between these two cases. The backward error is obtained by asking for what perturbed value of x the problem has actually been solved, i.e. what is the perturbation Δx such that $\widehat{y}$ is the exact solution of $f(x + \Delta x)$. The backward error is then defined as the smallest perturbation Δx.

Formally we have $\widehat{y} = f(x + \Delta x)$, and we can define the normwise backward error as:

$$\varepsilon_{\text{bwd}}^{\text{abs}}(\widehat{y}) = \min\{\varepsilon : \widehat{y} = f(x + \Delta x),\ \|\Delta x\| \leq \varepsilon\}$$

as well as the relative backward error

$$\varepsilon_{\text{bwd}}^{\text{rel}}(\widehat{y}) = \min\{\varepsilon : \widehat{y} = f(x + \Delta x),\ \|\Delta x\| \leq \varepsilon\|x\|\}. \tag{1.3}$$

The absolute componentwise backward error is

$$\varepsilon_{\text{bwd}}^{\text{abs}}(\widehat{y}) = \min\{\varepsilon : \widehat{y} = f(x + \Delta x),\ |\Delta x| \leq \varepsilon\}$$

and the relative componentwise backward error

$$\varepsilon_{\text{bwd}}^{\text{rel}}(\widehat{y}) = \min\{\varepsilon : \widehat{y} = f(x + \Delta x),\ |\Delta x| \leq \varepsilon |x|\}\,. \tag{1.4}$$

It is then said that if there is an uncertainty in the data or computations (physical measurements, approximations, rounding error...), it is sufficient that the backward error is of the same order as this uncertainty for the computed solution $\widehat{y}$ to be as good as one could expect. An algorithm which consistently generates a small backward error is said to be *backward stable*. In a backward stable algorithm, any errors introduced during its execution have a comparable impact to that of a minor perturbation in the input data. We show in Figure 1.1 the difference between forward and backward error when computing tanh for a given input x and perturbed input $x + \Delta x$.

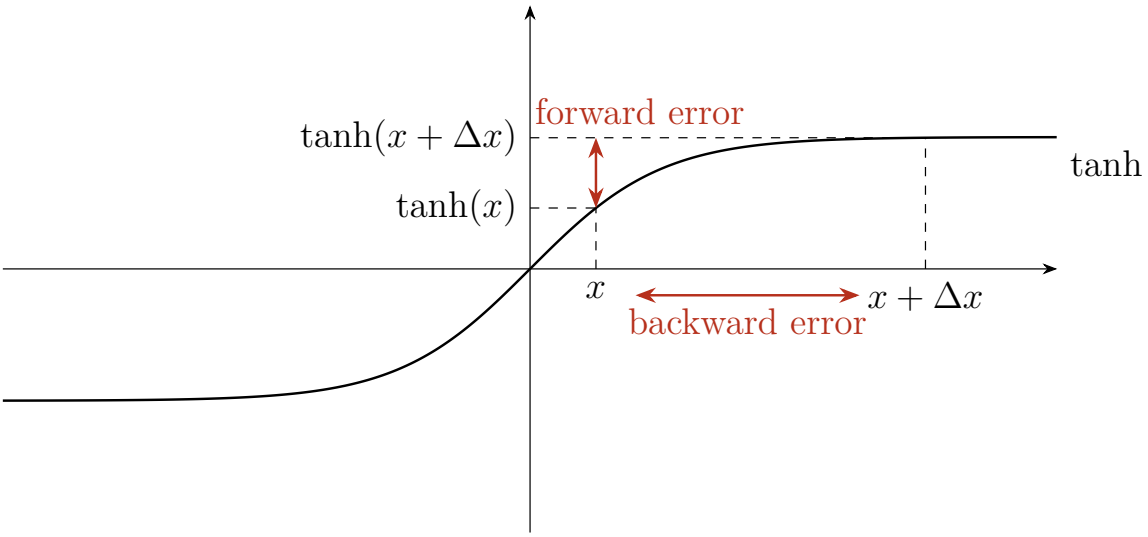


FIGURE 1.1: Backward error and forward error illustration.

Note that one can more generally define the componentwise backward error as:

$$\varepsilon_{\text{bwd}}(\widehat{y}) = \min\{\varepsilon : \widehat{y} = f(x + \Delta x),\ |\Delta x| \leq \varepsilon t\}$$

where t, the tolerance against which the perturbations on x are measured, is assumed to have nonnegative entries. In the rest of this work, the inequalities and absolute values of matrices and vectors are to be taken componentwise. The same holds for divisions when we want to switch to a relative error. For the particular choice of tolerance $t = |x|$, ε_{bwd} is the componentwise relative backward error; in the following sections we will focus on this choice of tolerance.

1.1.3 Forward and backward error for a linear system

As a very well-known example, let us consider in this section a linear system $Ax = b$, with $A \in \mathbb{R}^{n\times n}$ and $(x, b) \in \mathbb{R}^n \times \mathbb{R}^n$. We will illustrate in this case how to compute the componentwise forward and backward error as we will later build on this approach to extend it to the case of artificial neural networks.

Given an approximate solution $\widehat{x}$ to the linear system, obtaining a formula to compute the forward error is quite straightforward since we have

$$\varepsilon_{\text{fwd}} = \max_i \frac{|\widehat{x}_i - x_i|}{|x_i|}. \tag{1.5}$$

Whereas in this case the backward error is defined as:

$$\varepsilon_{\text{bwd}} = \min\{\varepsilon \geq 0 : (A + \Delta A)\widehat{x} = b + \Delta b,\ |\Delta A| \leq \varepsilon|A|,\ |\Delta b| \leq \varepsilon|b|\} \tag{1.6}$$

and obtaining an explicit formula is not as straightforward, as we have to find perturbed data $A + \Delta A$ and $b + \Delta b$ such that $\widehat{x}$ is solution of the perturbed linear system.

To find a closed formula for the expression of the backward error we hence need to solve the minimization problem of equation (1.6). The closed formula of the componentwise backward error for a linear system is a well-known result of Oettli and Prager [89]; the proof resorts to a now common approach which is to find a lower bound of ε_{bwd} and then show that there exist perturbations such that this lower bound is attained [54]. Let us define the residual $r = b - A\widehat{x}$; we then have:

$$(A + \Delta A)\widehat{x} = b + \Delta b$$

which leads to

$$\Delta A\widehat{x} - \Delta b = r$$

and by taking the absolute value we have

$$|\Delta A||\widehat{x}| + |\Delta b| \geq |r|.$$

We know that ε_{bwd} has to verify the inequalities in equation (1.6), hence

$$\varepsilon_{\text{bwd}}(|A||\widehat{x}| + |b|) \geq |r|.$$

Therefore

$$\varepsilon_{\text{min}} = \max_i \frac{|r_i|}{(|A||\widehat{x}| + |b|)_i}$$

is a lower bound for ε_{bwd}.

Oettli and Prager [89] then define the perturbations

$$\Delta A = \text{diag}\left(\frac{r_i}{(|A||\widehat{x}| + |b|)_i}\right)|A|\text{diag}(\text{sign}(\widehat{x}_i))$$

and

$$\Delta b = -\text{diag}\left(\frac{r_i}{(|A||\widehat{x}| + |b|)_i}\right)|b|.$$

We have $|\Delta A| \leq \varepsilon_{\min}|A|$ and $|\Delta b| \leq \varepsilon_{\min}|b|$ with equality for at least one component. Moreover $(A + \Delta A)\widehat{x} = b + \Delta b$, hence the lower bound $\varepsilon_{\min}$ is attained for these perturbations. Therefore, the componentwise relative backward error for the linear system $Ax = b$ is:

$$\varepsilon_{\text{bwd}} = \max_i \frac{|r_i|}{(|A||\widehat{x}| + |b|)_i}. \tag{1.7}$$

Note that the equivalent result for the normwise relative backward error defined as

$$\varepsilon_{\text{bwd}} = \min\{\varepsilon \geq 0 : (A + \Delta A)\widehat{x} = b + \Delta b,\ \|\Delta A\| \leq \varepsilon\|A\|,\ \|\Delta b\| \leq \varepsilon\|b\|\} \tag{1.8}$$

is given by Rigal and Gaches [94] by

$$\varepsilon_{\text{bwd}} = \frac{\|r\|}{(\|A\|\|\widehat{x}\| + \|b\|)}. \tag{1.9}$$

Note that an interesting property of the backward error is that, unlike the forward error, it does not depend on the exact solution x.

These results can be extended to derive formulas in the case of a matrix–vector plus bias $y = Ax + b$, where $y \in \mathbb{R}^n$. This operation, essentially, corresponds to a layer of a fully connected neural network with no activation function and will therefore be of special interest to us for the coming chapters.

Indeed, assuming perturbations on the matrix A and bias b, the computed solution in that case is $\widehat{y}$, instead of $\widehat{x}$, and verifies

$$\widehat{y} = (A + \Delta A)x + b + \Delta b$$

and the componentwise relative backward error is

$$\varepsilon_{\text{bwd}} = \min\{\varepsilon \geq 0 : \widehat{y} = (A + \Delta A)x + b + \Delta b,\ |\Delta A| \leq \varepsilon|A|,\ |\Delta b| \leq \varepsilon|b|\}. \tag{1.10}$$

Using the same approach as Oettli and Prager [89] we get that the componentwise relative backward error for a fully connected layer is given by

$$\varepsilon_{\text{bwd}} = \max_i \frac{|\widehat{y} - y|_i}{(|A||x| + |b|)_i}.$$

Indeed, it is easy to show that

$$\max_i \frac{|\widehat{y} - y|_i}{(|A||x| + |b|)_i}$$

is a lower bound for ε_{bwd} and this bound is attained for the perturbations

$$\Delta A = \text{diag}\left(\frac{\widehat{y}_i - y_i}{(|A||x| + |b|)_i}\right) |A|\text{diag}(\text{sign}(x_i))$$

and

$$\Delta b = \text{diag}\left(\frac{\widehat{y}_i - y_i}{(|A||x| + |b|)_i}\right) |b|.$$

1.1.4 Condition number

Forward error and backward error are linked by the condition number of the problem, which measures how sensitive the solution to a problem is to perturbations in the data. The use of condition number in numerical analysis is therefore particularly prominent [106, 4, 55, 30, 29] to understand and diminish the impact of perturbations on a system.

To define the problem conditioning let us say that we have $y + \Delta y = f(x + \Delta x)$, with $f : \mathbb{R}^n \to \mathbb{R}^m$ twice differentiable, then there is $\theta \in]0, 1[$ such that:

$$\Delta y = f(x + \Delta x) - f(x) = f'(x)\Delta x + \frac{f''(x + \theta\Delta x)}{2!}(\Delta x)^2.$$

So

$$\frac{\|\Delta y\|}{\|y\|} \leq \frac{\|f'(x)\|\|x\|}{\|f(x)\|}\frac{\|\Delta x\|}{\|x\|} + O(\|\Delta x\|^2) \tag{1.11}$$

with

$$\kappa_f(x) = \frac{\|f'(x)\|\|x\|}{\|f(x)\|} \tag{1.12}$$

the relative condition number of f which measures the relative change in the output for a given relative change in the input. The componentwise relative condition number,

defined by Gohberg and Koltracht [44], is

$$\kappa_f(x) = \|\mathrm{diag}(f(x))^{-1} f'(x) \mathrm{diag}(x)\|_\infty. \tag{1.13}$$

We then have from equation (1.11) the following useful inequality:

$$\text{forward error} \leq \text{condition number} \times \text{backward error}. \tag{1.14}$$

Note that, in our case, all quantities are taken relatively but this inequality holds whether or not quantities are relative.

More generally, for a given function $f : \mathbb{R}^n \to \mathbb{R}^m$, the condition number of the problem f at x is defined by Rice [93] and Lyubich [75], in the case of absolute normwise quantities we have

$$\kappa_f(x) = \lim_{\varepsilon \to 0} \sup_{\|\Delta x\| \leq \varepsilon} \left(\frac{\|f(x + \Delta x) - f(x)\|}{\|\Delta x\|} \right) \tag{1.15}$$

and the relative condition number is

$$\kappa_f(x) = \lim_{\varepsilon \to 0} \sup_{\|\Delta x\| \leq \varepsilon \|x\|} \left(\frac{\|f(x + \Delta x) - f(x)\|}{\|f(x)\|} \Big/ \frac{\|\Delta x\|}{\|x\|} \right) \tag{1.16}$$

while in the case of absolute componentwise quantities we have

$$\kappa_f(x) = \lim_{\varepsilon \to 0} \sup_{\max_i |\Delta x_i| \leq \varepsilon} \left(\max_i \frac{|f_i(x + \Delta x) - f_i(x)|}{|\Delta x_i|} \right) \tag{1.17}$$

and the relative condition number is

$$\kappa_f(x) = \lim_{\varepsilon \to 0} \sup_{\max_i |\Delta x_i| \leq \varepsilon |x_i|} \left(\max_i \frac{|f_i(x + \Delta x) - f_i(x)|}{|f_i(x)|} \Big/ \max_i \frac{|\Delta x_i|}{|x_i|} \right). \tag{1.18}$$

If f is differentiable then we have the formulas given in equations (1.12) and (1.13) that stand.

It is said that a given problem is *well conditioned* when small perturbations on the input variable lead to small changes in the output, which therefore means that the condition number is small. Alternatively, when the condition number is large the problem is said to be *ill conditioned*, in which case small perturbations on input data will lead to large changes in output. The interpretation of "small" and "large" may vary based on the context of the application. Specifically, there are instances where it is more suitable to evaluate the scale of perturbations with an absolute approach and others where it is more appropriate to evaluate it relatively to a given tolerance.

1.2 Floating-point arithmetic

1.2.1 Floating-point number representation

In order to represent real numbers, a computer has a finite precision at its disposal; numbers therefore have to be represented under this constraint. The typical choice for storing numbers is to fix a constant number of bits, which are binary digits, and use this number of bits to approximately represent a desired set of numbers. Computers typically allow users to choose from a set of such representations or data types, which can vary depending on the number of bits utilized but also on whether the representation is done in a fixed-point format or floating-point format. Floating-point formats are very popular since they enable to represent numbers of very different orders of magnitude. In this work we will focus on floating-point representations as defined in the IEEE 754 standard for binary floating-point arithmetic [63], since this is the most widely used format.

A floating-point system representation [85] is hence a way of representing numbers in finite precision and the set of numbers that can be represented is therefore a subset of the real numbers. It is based on the scientific notation where a nonzero real number y is expressed in decimal as:

$$y = s \times m \times 10^E$$

where $s = \pm 1$ is the *sign*, m is the *significand* or *mantissa*, which is an integer such that $1 \leq m \leq 9$ and E an integer *exponent*. Note that requiring $1 \leq m \leq 9$ allows for having a unique representation for each number. In this case, the format is then said to be *normalized*. It is always possible to find such m in this case by multiplying or dividing by 10 as long as the requirement is not satisfied. For example, if we take the real number 0.00012345, its scientific representation will be 1.2345×10^{-4}. It is obtained by multiplying 4 times the number by 10, and hence the decimal point floats 4 times, which is why it is called a floating-point representation.

In a more generic case, let y be a floating-point number, for a given radix β, it can then be expressed as:

$$y = s \times m \times \beta^E \tag{1.19}$$

or alternatively by expanding m in base β

$$y = s \times (d_0.d_1 d_2 \dots d_{p-1})_\beta \times \beta^E, \tag{1.20}$$

where p is the precision, the exponent E is such that $E_{\min} \leq E \leq E_{\max}$ and $1 \leq d_0 \leq \beta - 1$ in order to have a normalized representation. Note that in the case of binary representation for computers $\beta = 2$ and therefore this digit d_0 is not stored since it is implicitly set to 1.

Because the mantissa has a fixed amount of digits, the representation's precision diminishes as the exponent of the number rises. This results in larger gaps between

neighbouring representable numbers as we move away from zero. We illustrate that in Figure 1.2 using a toy binary floating-point number system in which the precision p is fixed to 3 and the exponent can take values in $\{-1,\ 0,\ 1\}$.

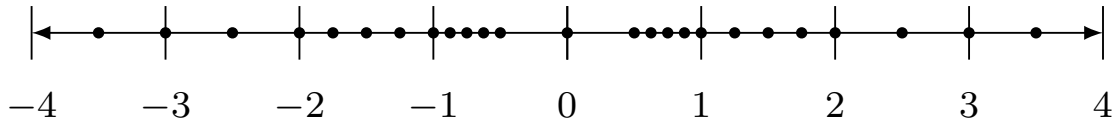


FIGURE 1.2: Representable numbers with a toy floating-point number system.

In Figure 1.2 we can clearly see that there is an underflow gap around zero in this floating-point arithmetic representation. The IEEE 754 standard make use of *subnormal* numbers, also known as *denormalized* numbers, to extend the range of floating-point number representation. This allows for preserving precision when very small values are encountered and therefore spanning the gap observed around zero, as seen in Figure 1.3. However, subnormal numbers are represented differently. Their leading bit is zero and they are interpreted with the value of the smallest allowed exponent. Moreover, they are usually not supported in hardware and, thus, their occurrence may significantly degrade performance.

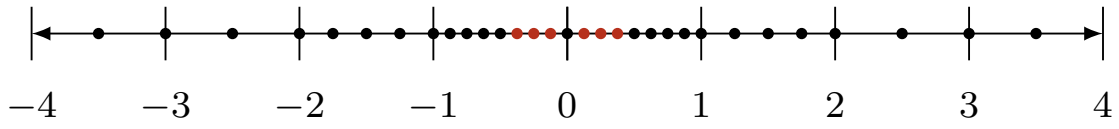


FIGURE 1.3: Representable numbers with a toy floating-point number system including subnormal numbers.

1.2.2 Rounding

In the context of floating-point arithmetic, since a finite number of bits is used to represent numbers, rounding principles are crucial for handling real numbers. Let us define $x \in \mathbb{R}$ and $\mathrm{fl}(x)$ denotes a close representable number in floating-point precision; the process of finding such a number is then called *rounding*. $\mathrm{fl}(x)$ therefore designates the rounding of x in the rest of the manuscript. The IEEE 754 standard defines rules for rounding to ensure consistency and predictability across different hardware platforms. Four different rounding modes are defined as follows:

- Round down (or round towards negative infinity), where the number is rounded to the nearest representable floating-point number that is less than or equal to the original number.

- Round up (or round towards positive infinity), where the number is rounded to the nearest representable floating-point number that is greater than or equal to the original number.
- Round towards zero, where the number is rounded down if it is greater than zero and rounded up if it is smaller than zero.
- Round to nearest, where the number is rounded either up or down, depending on which is nearer. In the case where the number falls exactly halfway between two representable floating-point numbers two strategies exists. The ties to even strategy consists in choosing the one with an even least significant digit. While the ties to away strategy consists in rounding down if the number is smaller than zero and rounding up if it is greater than zero.

1.2.3 Rounding errors

For any representation using finite precision, two quantities that dictate their usage are the range of representable numbers as well as the *machine epsilon* or *unit roundoff* u, which is an upper bound on the relative approximation error due to rounding.

The range of representation is the maximum and minimum positive numbers that it can express. Back to equation (1.20) we clearly can state that the minimum positive normalized number has to be

$$(1.00\ldots0)_\beta \times \beta^{E_{\min}} = \beta^{E_{\min}}$$

while the maximum number is

$$((\beta-1).(\beta-1)(\beta-1)\ldots(\beta-1))_\beta \times \beta^{E_{\max}} = (\beta - \beta^{-(p-1)})\beta^{E_{\max}}.$$

Let us say that we have $x \in \mathbb{R}$ that is in the range of representation of the floating-point numbers. Then the relative error introduced by the representation is defined as:

$$|\delta| = \frac{|x - \mathrm{fl}(x)|}{|x|}.$$

Theorem 1.1 (Theorem 2.2 from Higham [56])**.** *If $x \in \mathbb{R}$ lies in the range of representation of a floating-point representation then $\mathrm{fl}(x) = x(1+\delta)$, $|\delta| \leq u$, where u is called the machine epsilon.*

Proof. As defined in section 1.2.1, if x is in the normalized range then it can be expressed as

$$x = (1.d_0\ldots d_{p-1}d_p\ldots)_\beta \times \beta^E$$

while, depending on the rounding, we have

$$\mathrm{fl}(x) = (1.d_0 \dots d_{p-2} d'_{p-1})_\beta \times \beta^E$$

therefore, regardless of the rounding mode, we have:

$$|\delta| = \frac{|x - \mathrm{fl}(x)|}{|x|} \leq \frac{\beta^{-(p-1)} \times \beta^E}{\beta^E} = \beta^{-(p-1)},$$

and assuming round to nearest we have:

$$|\delta| = \frac{|x - \mathrm{fl}(x)|}{|x|} \leq \frac{\frac{\beta^{-(p-1)}}{2} \times \beta^E}{\beta^E} = \frac{\beta^{-(p-1)}}{2}. \tag{1.21}$$

□

The machine epsilon u therefore depends on the rounding mode. It is sometimes also defined as the difference between one and the next larger floating-point number, in which case

$$\varepsilon_M = \beta^{-(p-1)}$$

and is an upper bound for the relative error due to rounding, independently of which rounding mode is used.

Note that, as said in section 1.2.1, the fact that the relative error made when representing a number in floating-point depends on u, shows that the absolute gap between x and its representation increases as x becomes larger.

In fact, as seen in equation (1.21), for a given exponent E, assuming round to nearest, the biggest absolute error that can be made for a given real number $(1.d_0 \dots d_{p-1} d_p \dots)_\beta \times \beta^E$ is

$$\frac{\beta^{-(p-1)}}{2} \times \beta^E.$$

Since the value of such a real number can fluctuate between β^E and $\beta \times \beta^E$, this means that the relative error ranges between

$$\frac{\beta^{-(p-1)}}{2}$$

and

$$\frac{\beta^{-p}}{2}.$$

Therefore, the relative error can vary from up to a factor β for a couple of neighboured real numbers, this phenomenon is called *wobbling precision* and is one of the reasons why small bases such as $\beta = 2$ are often recommended.

1.2.4 Floating-point formats

The IEEE 754 standard was established by the Institute of Electrical and Electronics Engineers (IEEE) in 1985 and has been revised several times since then. IEEE 754 defines the format of floating-point numbers, the rules for arithmetic operations on these numbers as well as representations for special values such as positive, negative infinity, "Not a Number" (NaN) to handle exceptional conditions, subnormal numbers, that are non normalized numbers, and also signed zero. This standard is followed by almost all machines. The IEEE standard defines multiple floating-point number representations including FP16, FP32, and FP64 (half, single and double precision formats).

TABLE 1.1: Floating-point formats.

Format	$E_{\min}$	$E_{\max}$	Mantissa	Range	u
BFloat16	-126	127	7 bits	1.18×10^{-38} to 3.39×10^{38}	3.91×10^{-3}
FP16	-14	15	10 bits	6.10×10^{-5} to 6.55×10^{4}	9.77×10^{-4}
FP32	-126	127	23 bits	1.18×10^{-38} to 3.40×10^{38}	5.96×10^{-8}
FP64	-1022	1023	52 bits	2.23×10^{-308} to 1.80×10^{308}	1.11×10^{-16}

Single precision is widely used due to its balance between precision and storage efficiency while double precision provides higher precision, making it more suitable for tasks where numerical accuracy is critical. Half precision is used in scenarios where reduced precision is acceptable, and storage or computational efficiency is crucial. The unique requirements and challenges of AI applications have accelerated the development and adoption of 16-bit formats, offering improved memory and energy efficiency as well as compatibility with emerging industry's hardware. The BFloat16 [67] (Brain Floating-Point 16-bit) format is one of the most popular, it is a floating-point number representation designed for neural network training and inference in machine learning applications. It was introduced by Google as part of the TensorFlow Processing Unit (TPU) architecture and is a shortened 16-bit version of the 32-bit IEEE 754 single-precision floating-point format. In the specific context of deep learning this format is attractive since the range of values it can represent is the same as that of the FP32, but it also retains the advantages of a 16-bit format for storage and computation speed.

1.3 Rounding error analysis

We have seen that representing real numbers with finite precision in a computer system leads to discrepancies between the true value of a number and its computer

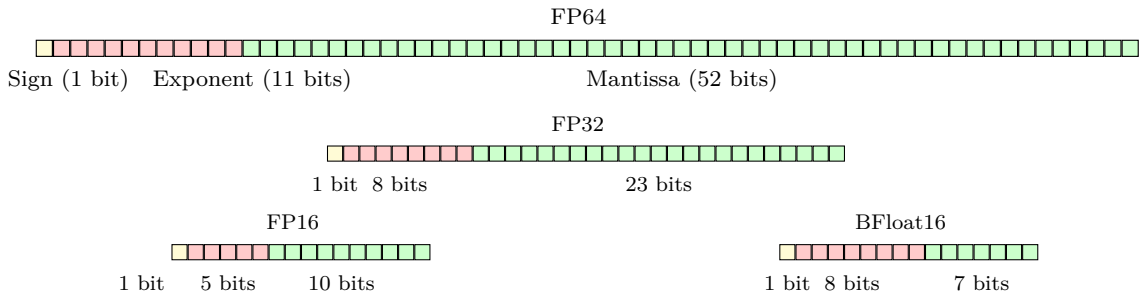


FIGURE 1.4: Bits distribution on different arithmetic precisions.

representation. Those gaps are called rounding errors and can accumulate in computations, especially in iterative processes or complex algorithms that involve many arithmetic operations. Over time, these errors can affect the accuracy of the final result. Rounding error analysis aims to quantify and understand the impact of these errors on the stability and accuracy of algorithms but also provide guidelines to better manage them.

1.3.1 Model of arithmetic

In order to evaluate the effect of rounding errors during computations the concept of *correctly rounded* arithmetic is essential, it is one of the key aspects of IEEE 754; this property aims to provide consistent and predictable results across different hardware platforms. Indeed, most of the time, the result of an arithmetic operation between two floating-point numbers is not a floating-point number. It is therefore necessary to define a strategy which ensures that arithmetic operations behave similarly on a wide range of computers. Hence the IEEE standard requires that, if the result of a floating-point operation is not a floating-point number, the computed result has to be the rounded value of the exact result. For example, if we note $\oplus$ the addition between two floating-point numbers, we have

$$a \oplus b = \mathrm{fl}(a + b),$$

where $\mathrm{fl}(a + b)$ is the floating-point representation of the exact result $a + b$.

As defined in section 1.2.3, u, the unit roundoff is an upper bound for the relative error due to rounding when using floating-point arithmetic. This typically means that for a set of basic operations ($\mathrm{op} \in \{+, -, \times, /, \sqrt{}\}$) we have the following classical model for floating-point arithmetic [121]:

Model 1.1 (Standard model for floating-point computations)**.**

$$\mathrm{fl}(a \ \mathrm{op} \ b) = (a \ \mathrm{op} \ b)(1 + \delta), \ |\delta| \leq u.$$

Note that in view of the IEEE 754 previously stated requirements, this model stands for IEEE standard arithmetic.

1.3.2 Classical backward error analysis

Traditional error analysis in numerical linear algebra is typically based on bounds obtained on the distance between one and the product

$$\prod_{k=1}^{n}(1+\delta_k).$$

Indeed, for each operation between floats, Model 1.1 stands. Hence, a sequence of basic operations will bring up such a product. These products are typically simplified using the following lemma, which corresponds to Lemma 3.1 from Higham [56].

Lemma 1.2 (Deterministic error bound)**.** *If* $|\delta_k| \leq u$ *and* $\rho_k = \pm 1$ *for* $k = 1, \ldots, n$*, and* $nu < 1$*, then*

$$\prod_{k=1}^{n}(1+\delta_k)^{\rho_k} = 1+\theta_n, \;\; |\theta_n| \leq \gamma_n.$$

The constant γ_n is defined as

$$\gamma_n = \frac{nu}{1-nu},$$

with $nu < 1$, n being the number of elementary operations considered, and u the machine epsilon [56].

1.3.2.1 Inner product error analysis

To illustrate how rounding error analysis works, we will first analyse the inner product operation, $y = a^T b$, with $(a, b) \in \mathbb{R}^n \times \mathbb{R}^n$. Assuming a left to right evaluation, we will note $y_i = a_1 b_1 + \ldots + a_i b_i$ the i-th partial sum. From the standard Model 1.1 we have

$$\widehat{y}_1 = \mathrm{fl}(y_1) = \mathrm{fl}(a_1 b_1) = a_1 b_1 (1+\delta_1)$$

with $|\delta_1| \leq u$, and then

$$\widehat{y}_2 = \mathrm{fl}(a_1 b_1 (1+\delta_1) + a_2 b_2 (1+\delta_2)) = (a_1 b_1 (1+\delta_1) + a_2 b_2 (1+\delta_2))(1+\varepsilon_2)$$

with $|\delta_i| \leq u$ for $i = 1, 2$ and $|\varepsilon_2| \leq u$. Using a simple recurrence, we can then show that

$$\widehat{y} = \sum_{i=1}^{n} a_i b_i (1+\delta_i) \prod_{j=\max(i,2)}^{n} (1+\varepsilon_j), \tag{1.22}$$

where each δ_i and ε_i are the rounding errors introduced, respectively, by multiplications and additions between floating-point numbers. Applying Lemma 1.2 to equation (1.22) we get

$$\widehat{y} = \sum_{i=1}^{n} a_i b_i (1 + \psi_i)$$

where $|\psi_i| \leq \gamma_{n-\max(i,2)+2} \leq \gamma_n$. It follows that γ_n is a bound for the backward error of the inner product because we have

$$\widehat{y} = \mathrm{fl}(a^T b) = (a + \Delta a)^T b = a^T (b + \Delta b), \ |\Delta a| \leq \gamma_n |a|, \ |\Delta b| \leq \gamma_n |b|. \tag{1.23}$$

Accordingly, the inner product performed in finite precision will produce the same result as the exact solution to a slightly perturbed version of the input data. Indeed, γ_n is equivalent to nu when $nu \ll 1$, therefore, perturbations are, in this case, very small. As explained in section 1.1.2, such a method with a small backward error will typically be called backward stable. Note that the definition of small will be context dependent. In our case, when using finite precision computations, having a backward error close to the unit roundoff, shows that the result is as good as one could hope for.

Moreover, we can also bound the forward error since

$$\widehat{y} - y = \sum_{i=1}^{n} a_i b_i \psi_i.$$

Therefore

$$|\widehat{y} - y| \leq \gamma_n \sum_{i=1}^{n} |a_i b_i| = \gamma_n |a|^T |b|$$

and

$$\frac{|\widehat{y} - y|}{|y|} \leq \gamma_n \frac{|a|^T |b|}{|a^T b|}. \tag{1.24}$$

This shows that, if $|a^T b| \ll |a|^T |b|$, one cannot guarantee that the relative forward error will be small.

1.3.2.2 Matrix–vector product error analysis

We now take interest in the matrix–vector product operation since it represents the basic operation in linear algebra and of many layers of neural networks. Here, finding bounds on the backward error is a well-known result [56]. However, we will still show how to obtain them and to then extend these results to less straightforward cases in Chapter 4.

Let $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$. We are interested in the floating-point computation of $y = Ax$. Without any loss of generality let $y_i = a_i^T x$, with a_i the i-th row of A. We

have

$$y_i = \sum_{k=1}^{n} (a_{ik} x_k),$$

then based on Model 1.1, assuming that the sum is evaluated from left to right, we have

$$\widehat{y}_i = (((a_{i1}x_1)(1+\delta_1) + (a_{i2}x_2)(1+\delta_2))(1+\varepsilon_2) + \ldots + (a_{in}x_n)(1+\delta_n))(1+\varepsilon_n).$$

Hence

$$\widehat{y}_i = \sum_{k=1}^{n} ((a_{ik}x_k)(1+\delta_k) \prod_{j=\max(k,2)}^{n} (1+\varepsilon_j)).$$

If we assume that $|\delta_k| \leq u$ for $k = 1, \ldots, n$, $|\varepsilon_j| \leq u$ for $j = 2, \ldots, n$ and $nu < 1$, then Lemma 1.2 from Higham [56] can be applied and therefore

$$(1+\delta_k) \prod_{j=\max(k,2)}^{n} (1+\varepsilon_j) = 1 + \psi_k$$

with $|\psi_k| \leq \gamma_{n-\max(k,2)+2}$, which then implies that $|\psi_k| \leq \gamma_n$. Therefore

$$\widehat{y}_i = (a_i + \Delta a_i)^T x$$

with $|\Delta a_i| \leq \gamma_n |a_i|$. Combining the m rows gives

$$\widehat{y} = (A + \Delta A)x$$

with

$$|\Delta A| \leq \gamma_n |A|.$$

This result is a direct extension of the inner product case, which is logical since each component of the output of a matrix–vector product is the result of an inner product. This enables to show that each relative perturbation on the input matrix that are needed to get the computed output $\widehat{y}$ are bounded by γ_n, hence the perturbations are small when $nu < 1$ which shows that the algorithm is numerically stable.

1.3.3 Probabilistic backward error analysis

For large problems or computations in reduced precision, bounds found using classical rounding error analysis, which are worst-case bounds, may be less useful because too pessimistic. Indeed, these traditional bounds involve the number n of elementary operations performed, which is empirically often replaced by its square root. Recent approaches [57, 58, 23] enable to relax this constant by making probabilistic assumptions about the rounding errors. They state that deterministic results can easily be

interchanged with their probabilistic counterparts, which essentially means that the previous constant γ_n, with high probability, can be replaced with

$$\widetilde{\gamma}_n(\lambda) = \exp\left(\frac{\lambda\sqrt{n}u + nu^2}{1-u}\right) - 1 = \lambda\sqrt{n}u + O(u^2), \tag{1.25}$$

when we can assume that the rounding errors done in computations can be modelled by independent random variables of mean zero. We will later derive bounds for artificial neural networks, which are based on the results from Connolly, Higham, and Mary [23]. We will therefore introduce their main theorems and in some cases proofs, since we will later base our approach on these same theorems and/or proofs.

From Higham and Mary [57] we have the main result in the following theorem:

Theorem 1.3 (Probabilistic error bound)**.** *Let $\delta_1, \ldots, \delta_n$ be independent random variables of mean zero with $|\delta_k| \leq u$ for all k. Then for $\rho_i = \pm 1,\ i = 1, \ldots, n$ and any constant $\lambda > 0$,*

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1 + \theta_n,\ |\theta_n| \leq \widetilde{\gamma}_n(\lambda)$$

holds with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2(1-u)^2}{2}\right).$$

Connolly, Higham, and Mary [23] show that the probabilistic assumptions made in the previous theorem can be replaced by weaker ones given in Model 1.2.

Model 1.2 (Probabilistic model of rounding errors)**.** *Let $\delta_1, \ldots, \delta_n$ be random variables of mean zero such that $\mathbb{E}(\delta_{k+1} \mid \delta_1, \ldots, \delta_k) = \mathbb{E}(\delta_{k+1})$ for any $k = 1, \ldots, n-1$.*

In order to introduce this modified theorem, we will need to first provide the concept of martingale.

Definition 1.1 (Martingale)**.** *A sequence of random variables $E_0, \ldots, E_n$ is said to be a martingale if it satisfies for all k*

$$\begin{aligned}&\mathbb{E}(|E_k|) < +\infty,\\ &\mathbb{E}(E_k \mid E_0, \ldots, E_{k-1}) = E_{k-1}.\end{aligned}$$

We also introduce the Azuma–Hoeffding inequality.

Lemma 1.4 (Azuma–Hoeffding inequality)**.** *If $E_0, \ldots, E_n$ is a martingale that satisfies $|E_k - E_{k-1}| \leq c_k$ for $k = 1, \ldots, n$, then for all $\lambda > 0$,*

$$\Pr\left(|E_n - E_0| \geq \lambda\left(\sum_{k=1}^{n} c_k^2\right)^{\frac{1}{2}}\right) \leq 2\exp\left(\frac{-\lambda^2}{2}\right).$$

We can now state the following main result from [23]:

Theorem 1.5 (Probabilistic error bound)**.** *Let $\delta_1, \ldots, \delta_n$ be random variables of mean zero with $|\delta_k| \leq u$ for all k such that $\mathbb{E}(\delta_{k+1} \mid \delta_1, \ldots, \delta_k) = \mathbb{E}(\delta_{k+1}) = 0$ for $k = 1, \ldots, n-1$, then for $\rho_i = \pm 1,\ i = 1, \ldots, n$ and any constant $\lambda > 0$,*

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1 + \theta_n,\ |\theta_n| \leq \widetilde{\gamma}_n(\lambda)$$

holds with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

Proof. Let

$$E_k = \sum_{i=1}^{k} \rho_i \delta_i$$

for $k = 1, \ldots, n$ and $E_0 = 0$. Since $|\delta_k| \leq u$ for $k = 1, \ldots, n$, we have $|E_k| \leq ku$, and therefore $\mathbb{E}(|E_k|) < +\infty$. Moreover, for $k = 1, \ldots, n-1$,

$$\mathbb{E}(E_{k+1} \mid E_1, \ldots, E_k) = E_k + \rho_{k+1}\mathbb{E}(\delta_{k+1} \mid \delta_1, \ldots, \delta_k) = E_k,$$

hence $E_0, \ldots, E_{n+1}$ is a martingale. Then, in order to apply the Azuma–Hoeffding inequality, we need to bound $|E_{k+1} - E_k|$. Since we know that $|E_{k+1} - E_k| = |\rho_{k+1}\delta_{k+1}|$, and that $|\rho_{k+1}\delta_{k+1}| \leq u$, for $k = 1, \ldots, n-1$, we therefore can use Lemma 1.4, and get

$$\Pr\left(|E_n - E_0| \geq \lambda\left(\sum_{k=1}^{n} u^2\right)^{\frac{1}{2}}\right) \leq 2\exp\left(\frac{-\lambda^2}{2}\right).$$

Hence

$$|E_n| = |\sum_{i=1}^{n} \rho_i \delta_i| \leq \lambda\sqrt{n}u \tag{1.26}$$

with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

Let

$$\phi = \prod_{i=1}^{n}(1+\delta_i)^{\rho_i},$$

then

$$\log(\phi) = \sum_{i=1}^{n} \rho_i \log(1+\delta_i),$$

yet for all $i \in [1; n]$ we have $|\delta_i| \leq u < 1$, we can therefore use a Taylor expansion which gives us

$$\log(1+\delta_i) = \sum_{k=1}^{+\infty} \frac{(-1)^{k+1}\delta_i^k}{k}$$

then we can get lower and upper bounds as

$$\delta_i - \sum_{k=2}^{+\infty} |\delta_i|^k \leq \log(1+\delta_i) \leq \delta_i + \sum_{k=2}^{+\infty} |\delta_i|^k$$

and since $|\delta_i| \leq u < 1$ we therefore have

$$\delta_i - \frac{\delta_i^2}{1-|\delta_i|} \leq \log(1+\delta_i) \leq \delta_i + \frac{\delta_i^2}{1-|\delta_i|}$$

which can then be further weakened as

$$\delta_i - \frac{u^2}{1-u} \leq \log(1+\delta_i) \leq \delta_i + \frac{u^2}{1-u}.$$

For $\rho_i = \pm 1$ it is clear that

$$\rho_i\delta_i - \frac{u^2}{1-u} \leq \rho_i \log(1+\delta_i) \leq \rho_i\delta_i + \frac{u^2}{1-u}$$

and then summing the inequality for $i = 1, \ldots, n$ we get

$$E_n - \frac{nu^2}{1-u} \leq \log(\phi) \leq E_n + \frac{nu^2}{1-u}.$$

We can then use the bound on E_n found on equation (1.26) to get

$$-\lambda\sqrt{n}u - \frac{nu^2}{1-u} \leq \log(\phi) \leq \lambda\sqrt{n}u + \frac{nu^2}{1-u},$$

with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

The bound is then further weakened by dividing $\lambda\sqrt{n}u$ by $1-u$, then, by taking the exponential of those values, and using the definition of $\widetilde{\gamma}_n(\lambda)$ in equation (1.25) we obtain

$$\frac{1}{1+\widetilde{\gamma}_n(\lambda)} = 1 - \frac{\widetilde{\gamma}_n(\lambda)}{1+\widetilde{\gamma}_n(\lambda)} \leq \phi \leq 1 + \widetilde{\gamma}_n(\lambda).$$

Yet

$$\phi = \prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1 + \theta_n$$

and therefore $|\theta_n| \leq \widetilde{\gamma}_n(\lambda)$ holds with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

□

Theorem 1.5 can thus be interchanged with Lemma 1.2 provided that the probabilistic assumptions hold. In the following sections, we will demonstrate the impact this has on the bounds obtained on the inner product and matrix–vector product in section 1.3.2.

Note that another significant result from Connolly, Higham, and Mary [23] is that Model 1.2 holds when stochastic rounding [28] is used.

1.3.3.1 Application to the inner product

Theorem 1.5 allows saying that equation (1.22) can be written as

$$\widehat{y} = \sum_{i=1}^{n} a_i b_i (1+\psi_i) \tag{1.27}$$

where $|\psi_i| \leq \widetilde{\gamma}_{n-\max(i,2)+2}(\lambda) \leq \widetilde{\gamma}_n(\lambda)$ holds for any given i with probability at least

$$P(\lambda) = 1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

Let us define

$$Q(\lambda, n) = 1 - n(1 - P(\lambda)).$$

The bound fails to hold for a given i with probability at most $1 - P(\lambda)$, therefore it fails to hold for at least one i with probability at most

$$n(1 - P(\lambda)).$$

Hence, the bounds hold for all i with probability at least $Q(\lambda, n)$. This means that the deterministic bounds obtained in equation (1.23) can be replaced by their probabilistic counterpart $\widetilde{\gamma}_n(\lambda)$, provided that the rounding errors satisfy the probabilistic model, with probability at least $Q(\lambda, n)$.

1.3.3.2 Application to the matrix–vector product

Since for any given output component $\widehat{y}_i$ is obtained via an inner product, we know from section 1.3.3.1 that

$$\widehat{y}_i = (a_i + \Delta a_i)^T x \tag{1.28}$$

with $|\Delta a_i| \leq \widetilde{\gamma}_n(\lambda)|a_i|$, holds with probability at least $Q(\lambda, n)$. This means that the bound fails to hold for a given i with probability at most $1 - Q(\lambda, n)$, therefore it fails to hold for at least one row i with probability at most $m(1 - Q(\lambda, n))$. We can then combine the m rows, as done on section 1.3.2.2, which means that

$$\widehat{y} = (A + \Delta A)x$$

with

$$|\Delta A| \leq \widetilde{\gamma}_n(\lambda)|A|,$$

holds with probability at least $1 - m(1 - Q(\lambda, n)) = Q(\lambda, mn)$. The deterministic bounds obtained in section 1.3.2.2 can therefore be replaced by their probabilistic counterpart, provided that the rounding errors satisfy the probabilistic model, with probability at least $Q(\lambda, mn)$.

1.4 Artificial intelligence

Ada Lovelace, who was perhaps the world's first programmer, may also be the first person to have foreseen the eventuality of AI back in 1842 in her notes from the first ever published computer program [74, 99, 13]. But it is only after the World War II that Alan Turing was the first person to conduct substantial research in this field, driven by the first work, now generally recognized as AI, of the neurologist Warren McCulloch and the logician Walter Pitts [79]. They proposed a model of artificial neurons in which true/false values of logic were mapped onto the on/off activity of neurons and the 0/1 of individual states in Turing machines. They also showed that all the logical gates (and, or, not, etc.) could be implemented by simple structures only composed of artificial neurons, that are neural networks. Last, but not least, they demonstrated that any computable function could be computed by some neural network. The link between Turing computation, human intelligence and mathematical logic was therefore made in this unified theoretical approach.

Since the end of the 1990s AI has gained significant interest through the field of machine learning; indeed breakthroughs in image and speech recognition, natural language processing, and gaming were made using deep learning, helped by the availability and usability of big data.

1.4.1 Machine learning

Mitchell [80] provides the following definition of learning: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." Machine Learning is now used in a wide variety of tasks from regression, anomaly detection, denoising, to classification. While, at first, supervised learning was the most widely used form of experience, there now exists a wide variety of learning approaches, such as e.g. unsupervised, self-supervised, semi-supervised, or reinforcement learning [45, 82]. Machine learning training encompasses a broad set of algorithms and techniques used to enable computers to learn from data without being explicitly programmed. It includes not only artificial neural networks but also other algorithms like decision trees, support vector machines, clustering algorithms, etc.

In supervised learning, the algorithm is trained on a labelled dataset, where the input data is paired with corresponding output labels. The goal is then to learn a function that maps from input features to the correct output labels. In unsupervised learning instead, the goal is to train a model on an unlabelled dataset; the algorithm then tries to find patterns, structure within the data. Semi-supervised learning is a combination of supervised and unsupervised learning, as the model is trained on a dataset that contains both labelled and unlabelled examples. Reinforcement learning involves a learning system interacting with its environment and making decisions. Depending on the success of these actions, the system is receiving feedback in the form of rewards or penalties.

In the context of classification, the machine is asked to provide a function, or model, that will map an input vector to a category or class. In order to learn this function the learning system is often provided by the user with a dataset which contains a set of input vectors $(x_i)_{i=1,\dots,N}$, each belonging to $\mathbb{R}^n$, and their associated ground-truth label $(y_i)_{i=1,\dots,N}$, which is the desired output of the model, each belonging to the set of labels $[1; C]$, where C is the number of classes of the classification problem.

Solving a supervised learning problem means finding a function whose predictions are as close as possible to the true labels. To formalise this, we introduce the notion of cost function or *loss function*.

Definition 1.2 (Loss function)**.** *Let $(y_i)_{i=1,\dots,N} \in \mathcal{Y}$ be some given labels, $x \in \mathbb{R}^n$ a given input vector, and $f : \mathbb{R}^n \to \mathcal{Y}$ a prediction model. A cost function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a function used to quantify the quality of f. The larger $\mathcal{L}(y, f(x))$ is, the further the predicted label $f(x)$ is from the true value y.*

In the context of supervised learning, the goal of the training will hence be, given a cost function $\mathcal{L}$, to look for a prediction function f which minimises this cost function over all possible values of x. The machine often has a finite number of x values in its dataset. These input data used to train the model are usually divided into three

datasets: training, validation, and test sets. This division allows for the evaluation of a model's performance on different subsets of data and helps ensure that the model generalizes well to new, unseen examples. The training set is the portion of the dataset used to train the machine learning model. The model typically adjusts its set of parameters θ, with $f(x;\theta)$ being the model, during the training phase, which consists in minimizing the loss function which measures the error between the prediction of the model and the desired output [53, 71].

Indeed, given a training set $(x_i, y_i)_{i=1,\dots,N}$ and a loss function, for example, the mean squared error is defined as

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i;\theta) - y_i)^2. \tag{1.29}$$

The training of a neural network consists in finding parameters θ^* such that:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta). \tag{1.30}$$

A standard optimization method used to solve this problem is the *gradient descent* which consists in performing steps in the opposite direction to the gradient. At each iteration of this method, the model's parameters are therefore updated with:

$$\theta \leftarrow \theta - \alpha\nabla\mathcal{L}(\theta),$$

where the step size α is often referred to as *learning rate* in machine learning. In modern machine learning and especially in deep learning, the number of parameters and training points can be prohibitively large. It is therefore impossible to compute the full gradient at each training iteration. To tackle this problem the simplest alternative is to randomly choose, at each iteration, a single point k for which the gradient will be computed, leading to the following modified iteration

$$\theta \leftarrow \theta - \alpha\nabla\mathcal{L}_k(\theta),$$

where $\mathcal{L}_k$ is the loss function for $N = 1$ at the point (x_k, y_k). This method is called *stochastic* gradient descent [95]. Many improvements on the basic stochastic gradient descent algorithm have been proposed, among them Adagrad [37], RMSprop [111] and Adam [69], which have shown excellent performance in deep learning applications [102].

During the model's training, its parameters will therefore be iteratively updated using the training dataset. The validation set is then used to fine-tune the model's hyperparameters, which can be, for example, the learning rate or the size of labelled data it has access to during a training iteration, and to assess its performance during training. Indeed, after each epoch (a certain number of training iterations), the model is evaluated on the validation set, and the performance metrics are monitored. A

fundamental issue in supervised machine learning is overfitting [110], which is the fact that a model does not generalize well from training data to unseen data. This is often the case when one trains the machine learning algorithm for too long and hence gets predictions that fit perfectly the training dataset but do not capture information and won't be able to generalize to new data. The validation set hence helps prevent overfitting. If the model performs well on the training set but poorly on the validation set, adjustments to the model or its hyperparameters may be necessary.

Finally, the test set serves as a final evaluation to estimate how well the model is expected to perform on new, unseen data. Since the validation set can be used to modify hyperparameters and better adjust the learning model, it can hence gradually be learnt by the model. The test set is therefore crucial for providing an unbiased assessment of the model's generalization ability and its performance in real-world scenarios.

The performance of the model for a classification task is often evaluated by measuring the accuracy of the provided function, which is the proportion of examples in a given dataset that are correctly classified.

1.4.2 Artificial neural networks and deep learning

Artificial neural networks are a specific type of mathematical model that is commonly used in machine learning. Artificial neural networks attempt to mimic the neurons of the human brain in order to perform a wide range of tasks. The formalisation of neurons as a mathematical object was done in 1943 by McCulloch and Pitts [79] but it is in 1958 that Rosenblatt introduced the concept of perceptron [98], illustrated in Figure 1.5. In a biological neuron, dendrites receive input signals from other neurons; in the artificial neurons those inputs are represented as a vector $x = (x_1, x_2, \ldots, x_n)$, this incoming signal is then weighted so that some connection between neurons are stronger than others; the neuron's weights represented by a vector $a = (a_1, a_2, \ldots, a_n)$ in a perceptron are used to simulate this behaviour. The core of the neuron, the soma, then performs a weighted sum to which it adds a bias so that the signal that gets out of the soma is $y = a^T x + b$. It is then processed by the axon, which gets its signal from the summation behaviour which occurs inside the soma. Once the signal coming from the soma reaches a certain potential, the axon will transmit it. In an artificial neuron, this is done by adding an activation function ϕ such that the output of a neuron is therefore $y = \phi(a^T x + b)$. Note that this expression can be simplified to $y = \phi(a^T x)$ by adding the bias to the weight vector $a = (b, a_1, a_2, \ldots, a_n)$ and one to the input vector $x = (1, x_1, x_2, \ldots, x_n)$.

Deep Learning is a field of machine learning which make use of deep neural networks, also called deep feedforward neural networks or also Multilayer Perceptrons (MLPs), which are artificial neural networks composed of multiple layers of artificial neurons between the network input and output layers, to solve machine learning problems.

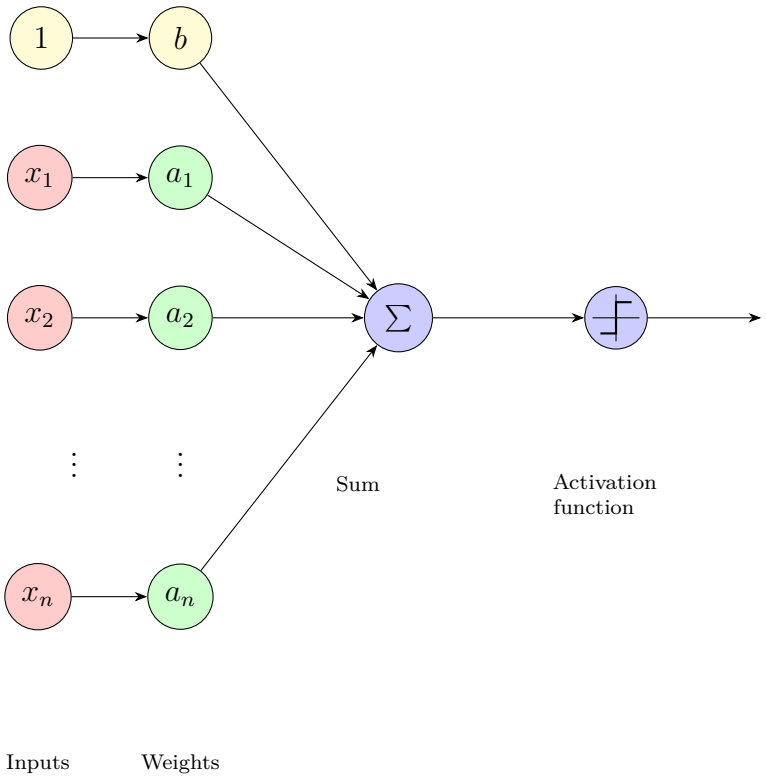


FIGURE 1.5: Perceptron's architecture.

As illustrated in Figure 1.6, each layer consists of a set of neurons, these neurons are connected to neurons in the adjacent layers so that the connections between neurons in one layer to neurons in the next layer are represented by weights which can then be organized into a matrix, often referred to as the weight matrix for example in case of a fully connected neural network or kernel for a convolutional neural network.

In this manuscript, we aim to apply a backward error analysis to artificial neural networks. In order to do so, we will focus here on feed-forward neural networks whose layers can be expressed as a matrix–vector product, which is immediate for fully connected or convolutional layers. Let us say that we have a feed-forward network of depth $p \in \mathbb{N}$ layers, each layer with its associated weight matrix $A_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and activation function ϕ_i applied entrywise. For a given input $x \in \mathbb{R}^{n_0}$ we then have the following expression for the output of this neural network:

$$y = m(x) = \phi_p(A_p \phi_{p-1}(A_{p-1} \dots A_2 \phi_1(A_1 x) \dots)). \tag{1.31}$$

Note that, here, and in the following chapters, we simplify the expression of the layers by integrating the potential layer's bias b_i in the weight matrix A_i. Moreover, this expression also encompasses convolutional layers, since they can be expressed as

matrix–vector products.

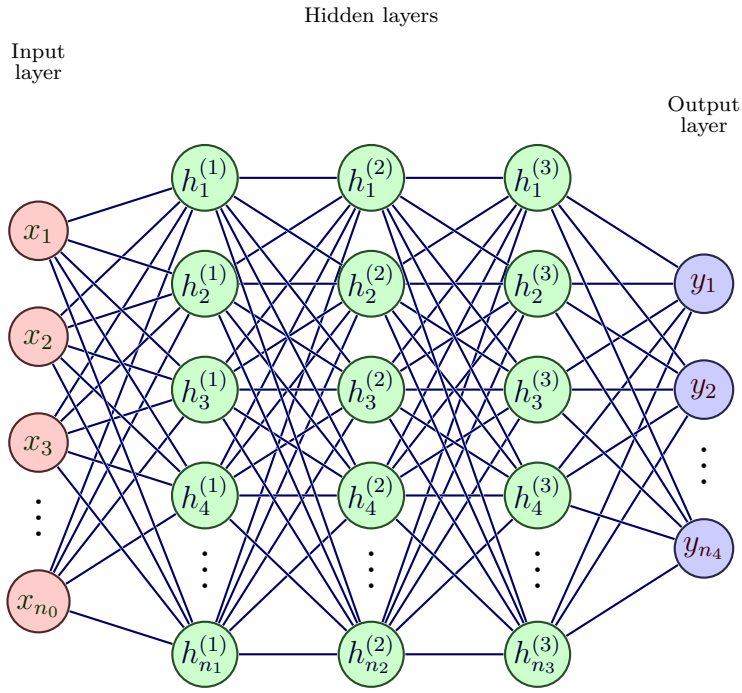


FIGURE 1.6: Fully connected artificial neural network's architecture.

1.4.3 Adversarial attacks

Despite their empirical efficiency, many works underline the sensitivity of DNNs to adversarial attacks [108, 72, 40, 84]. Among adversarial attacks, artificial neural networks are vulnerable to *adversarial examples*, which are perturbations applied to an input that would not fool a human but are sufficient to fool the model into making a wrong prediction. Figure 1.7 shows an example of how a slight perturbation on an image can trigger an erroneous classification by a neural network which works correctly on the unperturbed image. Adversarial examples are considered to be a significant obstacle to the deployment of neural networks models in safety-critical tasks, due to the clear security threat that these attacks represent; this also raises questions regarding the robustness and ability of a neural network to generalize in the context of new distributions. Exact computation of a neural network robustness, when possible, does not scale well for large neural networks; for instance, the problem of verifying the robustness of a ReLU (Rectified Linear Unit) neural network can be formalized as a Mixed Integer Programming problem, which is NP hard [68, 112]. For this reason, many different approaches have been developed to find adversarial examples in order to more efficiently evaluate neural network robustness. Finding adversarial

FIGURE 1.7: The camouflage fools the Mask R-CNN object detector (on the bottom), whereas plain colours (on top) are being correctly detected [130].

examples with small norm perturbations is crucial to assess the vulnerability of a neural network against attackers. Indeed, such examples do not only provide for a better understanding of a model's robustness but also allow for improving it by integrating them into the training process — a defence known as adversarial training [46, 76]. The literature on adversarial attacks is very abundant and many methods have been proposed. A complete review is out of the scope of this manuscript but we make a brief survey of some of the most popular and successful methods. 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 [41, 116, 115] despite its potential use to help robust generalization [125]. In this case, the adversarial attack on a neural network's parameters has been introduced and called *stealth attack* by Tyukin, Higham, and Gorban [115].

In this work we will at first focus on adversarial examples which are inputs of a neural network perturbed in such a way that they are classified in a different class than expected whereas a human would still correctly recognize them and assign to them the correct label. Finding *targeted* adversarial examples amounts to computing the smallest norm perturbation on the input data x such that the perturbed input $x + \Delta x$ is misclassified by the neural network in a prescribed target class j instead of

the expected true label i. Mathematically, the targeted adversarial perturbation is defined as the solution of the following optimization problem:

$$\begin{aligned} \underset{\Delta x}{\text{minimize}} \quad & \|\Delta x\|^2 \\ \text{subject to} \quad & \text{Class}(x + \Delta x) = j, \end{aligned} \tag{1.32}$$

where j is the target class and $\text{Class}(x+\Delta x)$ the class of the perturbed input. Different types of norms can be used to measure the size of the perturbation and methods have been proposed that can handle one or the other or even multiple types. In most cases a classifier will associate with an input x the label j out of C classes when the j-th component of the classifier's output $m(x)$ is the maximum of its C components, that is:

$$\underset{i=1,\dots,C}{\arg\max}\, m_i(x) = j.$$

Hence, the constraints $\text{Class}(x + \Delta x) = j$ can be formulated as

$$m_i(x + \Delta x) \leq m_j(x + \Delta x), \;\; i = 1, \dots, C. \tag{1.33}$$

In an *untargeted* attack, instead, one will search for a perturbation that leads to a misclassification regardless of the output class. In this case, the constraints of the minimization problem become

$$\text{Class}(x + \Delta x) \neq i, \tag{1.34}$$

where i is the true label associated with the input x. Hence for most classifiers the constraints can then be formulated as

$$m_i(x + \Delta x) \leq \max_{j \neq i} m_j(x + \Delta x). \tag{1.35}$$

In general the original problem (1.32) is too complex to be solved directly, hence state-of-the-art methods to compute adversarial examples attempt to approximately solve it. Numerous approaches that have been proposed and are commonly used to generate adversarial examples resort to solving the following penalty problem:

$$\underset{\Delta x}{\text{minimize}} \quad \|\Delta x\|^2 + c\, \mathcal{L}(x + \Delta x, j), \tag{1.36}$$

where $\mathcal{L}$ is a given loss function, potentially different from the one used to train the neural network, of the input with respect to a given target class. This was first introduced by Szegedy et al. [108] who formalized the minimization problem and introduced the term of adversarial sample. In their work, they proposed an attack using a Large Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm to solve the

problem (1.36). The L-BFGS attack uses a line-search algorithm to find the optimal value of the weight c, which makes it expensive; to overcome this limitation, Goodfellow, Shlens, and Szegedy [46] proposed a so-called Fast Gradient Sign Method (FGSM). This attack uses only one step in the direction of the sign of the gradient to generate an adversarial example; hence, the obtained perturbation can be expressed as:

$$\Delta x = \varepsilon \operatorname{sign}(\nabla_x \mathcal{L}_f(x, j)) \tag{1.37}$$

where $\mathcal{L}_f$ is the cost function used during training.

A more powerful direct alternative to FGSM is Projected Gradient Descent [76] (PGD) which is an iterative version of FGSM, producing smaller perturbations at the cost of being computationally more expensive. The most well-known state-of-the-art adversarial attack using this so-called penalty method is the Carlini–Wagner attack [18]. Penalty methods transform the constrained optimization problem into an unconstrained one by coupling the need to minimize the perturbation norm and the need to misclassify the input; the resulting optimization problem is easier to solve than the problem in equation (1.32) but this comes at the expense of needing to find an optimal loss function as well as an optimal weight c, which is, for the weight, often achieved using expensive line-search algorithms.

All the above methods aim to optimize both the misclassification and minimal norm criteria at the same time. This is often achieved using a line-search algorithm to find the best balance between the two criteria. Other methods aim to accelerate the generation of adversarial samples, tailoring algorithms specifically for a given norm. Amongst them, DeepFool [84] iteratively perturbs the input by linearizing the model around the current point and then find the closest decision boundary for the l_2-norm or l_∞-norm. DDN-attack [97] is another method which decouples the two objectives by using projections on a l_2-ball centred on the original input at each iteration, whereas FAB-attack [25] uses both projections and linear approximations of the neural network to produce competitive adversarial examples.

A recent approach called ALMA [96] takes advantage of the Augmented Lagrangian method to attenuate the drawbacks of penalty-based attacks. Indeed, with this method the penalty is adaptively modified during the optimization iterations to estimate and converge to the optimal penalty term, which corresponds to the Lagrangian multiplier of the optimization problem.

1.5 Notations, definitions and properties

In this section notations, definitions and properties of quantities used in this thesis are introduced.

1.5.1 Neural networks notations

In this manuscript, when considering a generic feed-forward neural network, we define it with a depth of $p \in \mathbb{N}$ layers, each layer has its associated weight matrix $A_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and activation function ϕ_i applied entrywise. For a given input $x \in \mathbb{R}^{n_0}$ we will denote the output:

$$y = m(x) = \phi_p(A_p \phi_{p-1}(A_{p-1} \dots A_2 \phi_1(A_1 x) \dots)).$$

For the rest of this document, let us define $y_i \in \mathbb{R}^{n_i}$ the output of the i-th layer of a neural network, y_0 being the input x. In this case we have, for example,

$$y = y_p = \phi_p(A_p y_{p-1})$$

where $\phi_p(A_p y_{p-1})$ is a vector since the activation function ϕ_p is applied entrywise, we also define $\phi_i'(A_i y_{i-1})$, for $i = 1, \dots, p$, as the following diagonal matrix

$$[\phi_i'(A_i y_{i-1})]_{j,j} = \phi_i'([A_i y_{i-1}]_j).$$

1.5.2 Operators and functions

We will use the "vec" operator which stacks the columns of a matrix one underneath the others and note $\overrightarrow{A} = \mathrm{vec}(A)$. Let $A = (a_{ij})_{1 \le i \le m,\ 1 \le j \le n} \in \mathbb{R}^{m \times n}$, we then have

$$\mathrm{vec}(A) = (a_{11}, \dots, a_{m1}, a_{12}, \dots, a_{m2}, \dots, a_{1n}, \dots, a_{mn}) \in \mathbb{R}^{mn \times 1}.$$

We will also use the "diag" operator which, when given a vector argument, creates a matrix with the vector values along the diagonal. For example, given a vector $a = (a_1, a_2, \dots, a_n)$

$$D = \mathrm{diag}(a) = \mathrm{diag}(a_1, \dots, a_n) \in \mathbb{R}^{n \times n}$$

is a matrix whose entries outside the main diagonal are all zero and entries in the diagonal are given by the vector a. The "sign" function of a real number is a function which is defined as follows:

$$\mathrm{sign} = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0 \end{cases}$$

and is meant to be applied componentwise if applied to vectors or matrices. More generally, divisions and inequalities between vectors are also defined componentwise.

1.5.3 Kronecker product definition and properties

Consider three matrices $A \in \mathbb{R}^{m\times n}$, $X \in \mathbb{R}^{n\times p}$ and $B \in \mathbb{R}^{p\times q}$, then the Kronecker product $A \bigotimes B$ is a specialization of the tensor product which produces the following $mp \times nq$ matrix:

$$A \bigotimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

We recall some properties of the Kronecker product which will be used in the remainder of this document

$$(A \bigotimes C)(B \bigotimes D) = (AB) \bigotimes (CD), \tag{1.38}$$

$$\|A \bigotimes B\| = \|A\|\|B\|, \tag{1.39}$$

$$|A \bigotimes B| = |A| \bigotimes |B|, \tag{1.40}$$

$$\mathrm{vec}(AXB) = (B^T \bigotimes A)\,\mathrm{vec}(X), \tag{1.41}$$

$$(A \bigotimes B)^T = A^T \bigotimes B^T. \tag{1.42}$$

1.5.4 Notations

In our work, we will focus on the Frobenius norm for the matrices, and the l_2-norm for the vectors, and, for the sake of readability, we will drop the subscript on the $\|\cdot\|_F$ and $\|\cdot\|_2$ operators in the remainder of this document.

We note the distinction between the continuous and discrete intervals from a to b, the former denoted by $[a, b]$ and the latter represented by $[a; b]$.

Chapter 2

Backward error and condition number for artificial neural networks

Forward and backward errors are two quantities that enable to estimate the accuracy of a given computed solution compared to the exact result. A third quantity, the condition number, links the other two and enables to measure the sensitivity of a system to perturbations. Unlike the forward error, which can be quite straightforward to compute, the backward error and condition number often require a more detailed analysis.

Indeed, the backward error represents the solution to a minimization problem, which often means that closed formulas are not readily derived, as seen in section 1.1.3, whereas for the condition number, even if formulas exist when the function is derivable, there is still work to do in order to understand how to apply them depending on which perturbations are considered to measure the sensitivity of a problem. The goal of this chapter is therefore to explain how to establish explicit expressions of the backward error as well as the condition number for artificial neural networks, depending on which perturbations are considered.

For the sake of clarity, we will proceed in incremental steps. We will first show in section 2.1.1 how the nonlinearity of the activation function affects the computation of the backward error and we will provide formulas to compute the componentwise and normwise backward errors, assuming perturbations on the neural network's weights. We will detail in section 2.1.2 how the condition number is obtained in a componentwise and normwise case for a single layer.

These steps will provide the ingredients to produce the final, more generic results, which shows how chaining layers in a general artificial neural network impacts computations, in section 2.2. These final results will provide a better understanding of how assumptions that are made on which variables are perturbed impact formulas for the backward error and condition number. In section 2.5 we will attempt to validate these formulas; to do so we will, at first, compute the backward error and condition number

for random neural networks of small sizes, which have quasilinear behaviour. Moreover from equation (1.14), one expects a direct link between forward, backward errors and condition number, and because we know exactly how to compute the forward error, comparing the forward error to the product of backward error by the condition number will provide another way of validating our formulas.

2.1 Single layer neural network

An entire layer of a neural network is typically composed of a matrix–vector product followed by an activation function, which can be written as $y = \phi(Ax)$, where the application of ϕ is to be taken componentwise, $A \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$.

In order to compute the backward error and condition number, one has to assume that perturbations are made on some given variables; in these first sections, we will assume that perturbations are carried by the neural network's parameters and not by its input data. This choice allows us to have a better understanding of how the activation function changes the approach which is typically used to obtain formulas for the backward error by Oettli and Prager [89]. Indeed, in numerical linear algebra, backward error and condition number are typically used to quantify the impact of rounding errors on computations. Therefore, when encountering a matrix–vector product, one would assume perturbations on the matrix. We will however later show in section 2.2 how the computations of the backward error and condition number can be extended to accommodate perturbations in any given parameters and/or input.

2.1.1 Backward error expression

This section focuses on the computation of the backward error for a single layer of neural network with nonlinear activation function. When there are no activation functions, and therefore no nonlinearities, backward error can be computed using standard numerical linear algebra approaches, as shown in section 1.1.3.

However, in the machine learning context, activation functions are typically chosen so that they are differentiable, since their gradient is necessary for the training phase. Given that we typically focus on small norm perturbations (e.g. rounding errors, adversarial attacks), the application of an activation function can therefore be reduced to a linear case by using a first order approximation. This will enable us, in case of a single layer neural network, to get back to standard backward error formulas.

Note that many methods already exist for dealing with functions that are not differentiable at certain points, since gradients are needed during the training phase. The most widely known example is ReLU, in which case one typically sets the value of the derivative to zero at zero [7].

2.1.1.1 Componentwise backward error

For a single layer neural network as previously defined, the componentwise relative backward error is:

$$\varepsilon_{\text{bwd}} = \min\{\varepsilon \geq 0 : \widehat{y} = \phi((A + \Delta A)x),\ |\Delta A| \leq \varepsilon |A|\}. \tag{2.1}$$

For a given output component $\widehat{y}_i$, let a_i be the i-th row of A; assuming ϕ is differentiable at $a_i^T x$ we have, with a first order approximation:

$$\widehat{y}_i = \phi((a_i + \Delta a_i)^T x) = \phi(a_i^T x) + \phi'(a_i^T x)\Delta a_i^T x,$$

which means that

$$\widehat{y}_i - y_i = \phi'(a_i^T x)\Delta a_i^T x.$$

We are now ready to use a similar approach to that used in section 1.1.3, which means finding a lower bound for the backward error and show that it is attained. Using equation (2.1) we have

$$|\widehat{y}_i - y_i| \leq \varepsilon_{\text{bwd}} |\phi'(a_i^T x)||a_i^T||x|,$$

which leads to

$$\frac{|\widehat{y}_i - y_i|}{|\phi'(a_i^T x)||a_i^T||x|} \leq \varepsilon_{\text{bwd}}. \tag{2.2}$$

Then

$$\varepsilon_{\text{min}} = \max_i \frac{|\widehat{y} - y|_i}{|\phi'((Ax)_i)|(|A||x|)_i}$$

is a lower bound for ε_{bwd}.

Let us define the perturbation

$$\Delta A = \text{diag}\left(\text{sign}(\phi'(a_i^T x))\frac{\widehat{y}_i - y_i}{|\phi'(a_i^T x)||a_i^T||x|}\right)|A|\text{diag}(\text{sign}(x_i)),$$

then $|\Delta A| \leq \varepsilon_{\text{min}}|A|$, with equality for at least one component and at first order we obtain the desired equality $\widehat{y} = \phi((A + \Delta A)x)$. This shows that the lower bound ε_{min} is attained for this perturbation, therefore the componentwise relative backward error for such a single layer neural network is:

$$\varepsilon_{\text{bwd}} = \max_i \frac{|\widehat{y} - y|_i}{|\phi'((Ax)_i)|(|A||x|)_i}. \tag{2.3}$$

2.1.1.2 Normwise backward error

The normwise relative backward error is defined as:

$$\varepsilon_{\text{bwd}} = \min\{\varepsilon \geq 0 : \widehat{y} = \phi((A + \Delta A)x),\ \|\Delta A\| \leq \varepsilon \|A\|\}. \tag{2.4}$$

The approach to get a formula here is the same as in the previous section, we make use of a first order expression which is given by

$$\widehat{y} = \phi((A + \Delta A)x) = \phi(Ax) + \phi'(Ax)\Delta Ax, \tag{2.5}$$

to obtain a lower bound on the backward error with

$$\frac{\|\phi'(Ax)^{-1}(\widehat{y} - y)\|}{\|A\|\|x\|} \leq \varepsilon_{\text{bwd}}. \tag{2.6}$$

Which therefore means that

$$\varepsilon_{\min} = \frac{\|\phi'(Ax)^{-1}(\widehat{y} - y)\|}{\|A\|\|x\|}$$

is the lower bound for ε_{bwd}. Note that, since the application of ϕ is componentwise the matrix $\phi'(Ax)$ is diagonal, therefore, assuming its entries are non-zero, $\phi'(Ax)^{-1}$ exists. Let us then define the perturbation

$$\Delta A = \|A\| \frac{\phi'(Ax)^{-1}(\widehat{y} - y)x^T}{\|A\|\|x\|^2}.$$

With this given perturbation, we have

$$\|\Delta A\| = \|A\| \frac{\|\phi'(Ax)^{-1}(\widehat{y} - y)x^T\|}{\|A\|\|x\|^2}.$$

Since for any vectors a and b we have $\|ab^T\|_F = \|a\|_2\|b\|_2$, we can then say that

$$\|\Delta A\| = \|A\| \frac{\|\phi'(Ax)^{-1}(\widehat{y} - y)\|}{\|A\|\|x\|}.$$

Therefore $\|\Delta A\| = \varepsilon_{\min}\|A\|$, and at first order, we obtain the desired equality $\widehat{y} = \phi((A+\Delta A)x)$. This means that the lower bound $\varepsilon_{\min}$ of the backward error is attained and therefore the normwise relative backward error for a single layer neural network is:

$$\varepsilon_{\text{bwd}} = \frac{\|\phi'(Ax)^{-1}(\widehat{y} - y)\|}{\|A\|\|x\|}. \tag{2.7}$$

2.1.2 Condition number expression

As seen in section 1.1.4, the condition number is quite straightforward to obtain when one has access to first order information. Indeed, once we have equation (2.5) then we get

$$\widehat{y} - y = (x^T \bigotimes \phi'(Ax))\overrightarrow{\Delta A} + O((\overrightarrow{\Delta A})^2) \tag{2.8}$$

which then, depending on the metric, gives access to a direct relationship between forward and backward error. We therefore detail in this section how to obtain the condition number for each metric, knowing that in the more generic case of a deeper neural network it will be derived similarly.

2.1.2.1 Componentwise condition number

In case of a componentwise metric, as defined in equation (1.18), the relative condition number is

$$\kappa_\phi(A, x) = \lim_{\varepsilon \to 0} \sup_{\max_i |\overrightarrow{\Delta A}_i| \leq \varepsilon |\overrightarrow{A}_i|} \left(\max_i \frac{|\phi((A + \Delta A)x) - \phi(Ax)|_i}{|\phi(Ax)|_i} \Big/ \max_i \frac{|\overrightarrow{\Delta A}_i|}{|\overrightarrow{A}_i|} \right).$$

Then, assuming all entries are non-zero, we can rewrite equation (2.8) as

$$\frac{\widehat{y} - y}{y} = \mathrm{diag}(\phi(Ax))^{-1}(x^T \bigotimes \phi'(Ax))\mathrm{diag}(\overrightarrow{A})\frac{\overrightarrow{\Delta A}}{\overrightarrow{A}} + O(\overrightarrow{\Delta A}^2),$$

where the divisions are meant componentwise. Moreover, we have

$$\lim_{\varepsilon \to 0} \sup_{\max_i |\overrightarrow{\Delta A}_i| \leq \varepsilon |\overrightarrow{A}_i|} \max_i \left| \mathrm{diag}(\phi(Ax))^{-1}(x^T \bigotimes \phi'(Ax))\mathrm{diag}(\overrightarrow{A})\frac{\overrightarrow{\Delta A}}{\overrightarrow{A}} \right|_i$$

$$= \lim_{\varepsilon \to 0} \sup_{\max_i |\overrightarrow{\Delta A}_i| \leq \varepsilon |\overrightarrow{A}_i|} \|\mathrm{diag}(\phi(Ax))^{-1}(x^T \bigotimes \phi'(Ax))\mathrm{diag}(\overrightarrow{A})\|_\infty \max_i \frac{|\overrightarrow{\Delta A}_i|}{|\overrightarrow{A}_i|}$$

which then means that we get the relative componentwise condition number of the neural network in the form of:

$$\kappa_\phi(A, x) = \|\mathrm{diag}(\phi(Ax))^{-1}(x^T \bigotimes \phi'(Ax))\mathrm{diag}(\overrightarrow{A})\|_\infty. \tag{2.9}$$

2.1.2.2 Normwise condition number

In case of a normwise metric, as defined in equation (1.16), the relative condition number is

$$\kappa_\phi(A,x) = \lim_{\varepsilon\to 0} \sup_{\|\Delta A\|\leq\varepsilon\|A\|} \left(\frac{\|\phi((A+\Delta A)x) - \phi(Ax)\|}{\|\phi(Ax)\|} \Big/ \frac{\|\Delta A\|}{\|A\|} \right).$$

Yet from equation (2.8) it is clear that

$$\kappa_\phi(A,x) = \lim_{\varepsilon\to 0} \sup_{\|\Delta A\|\leq\varepsilon\|A\|} \left(\frac{\|(x^T \bigotimes \phi'(Ax))\overrightarrow{\Delta A}\|}{\|\phi(Ax)\|} \Big/ \frac{\|\Delta A\|}{\|A\|} \right),$$

which then leads to

$$\kappa_\phi(A,x) = \lim_{\varepsilon\to 0} \sup_{\|\Delta A\|\leq\varepsilon\|A\|} \left(\frac{\|x^T \bigotimes \phi'(Ax)\|\|\overrightarrow{\Delta A}\|}{\|\phi(Ax)\|} \Big/ \frac{\|\Delta A\|}{\|A\|} \right).$$

Therefore, since $\|\overrightarrow{A}\|_2 = \|A\|_F$, the relative normwise condition number of the neural network is given by:

$$\kappa_\phi(A,x) = \frac{\|x^T \bigotimes \phi'(Ax)\|\|A\|}{\|\phi(Ax)\|}.$$

2.2 General neural network

Our work focuses on producing a general theoretical framework to evaluate a neural network's sensibility to various perturbations. We will therefore consider the case in which both the input and the parameters are perturbed. In the previous section, we showed how to compute the backward error and condition number for a single layer of neural network. This will serve as the initial building block for constructing formulas that enable to compute these quantities in case a given neural network is perturbed on its input and parameters.

We will therefore show, in this section, how the chaining of layers impacts computations. These generic formulas will also allow us to explicitly define, for cases where perturbations are applied only on the input or on a given set of parameters, how the backward error and condition number are impacted.

Similarly to section 2.1, we will use a first order approximation of the model with respect to the perturbed parameters. In order to get generic formulas, we will consider perturbations on the parameters $(A_i)_{i=1,\dots,p}$ and input x. Consider a given neural network model, assuming perturbations on the model's parameters and on its input

we have

$$\widehat{y} = \phi_p((A_p + \Delta A_p)\phi_{p-1}((A_{p-1} + \Delta A_{p-1}) \dots \phi_1((A_1 + \Delta A_1)(x + \Delta x)) \dots)).$$

A first order approximation leads to the following equality:

$$\begin{aligned} \widehat{y} - y = & \phi_p'(A_p y_{p-1})\Delta A_p y_{p-1} + \dots \\ & + \phi_p'(A_p y_{p-1})A_p\phi_{p-1}'(A_{p-1}y_{p-2}) \dots A_{i+1}\phi_i'(A_i y_{i-1})\Delta A_i y_{i-1} \\ & + \dots + \phi_p'(A_p y_{p-1})A_p\phi_{p-1}'(A_{p-1}y_{p-2}) \dots A_2\phi_1'(A_1 x)\Delta A_1 x \\ & + \phi_p'(A_p y_{p-1})A_p\phi_{p-1}'(A_{p-1}y_{p-2}) \dots A_2\phi_1'(A_1 x)A_1\Delta x. \end{aligned} \tag{2.10}$$

For the sake of readability let us define, for $i = 1, \dots, p$, the Jacobian of our neural network model m computed with respect to the parameters A_i

$$J_m^i(A, x) = \phi_p'(A_p y_{p-1})A_p\phi_{p-1}'(A_{p-1}y_{p-2}) \dots A_{i+1}\phi_i'(A_i y_{i-1})$$

and the Jacobian of the model with respect to the input

$$J_m^0(A, x) = \phi_p'(A_p y_{p-1})A_p\phi_{p-1}'(A_{p-1}y_{p-2}) \dots A_2\phi_1'(A_1 x)A_1.$$

Note that for $p = 1$ and no perturbation on the input we fall back to the case of section 2.1.1 and $J_m^i(A, x)$ is a diagonal matrix.

Following equation (2.10) we therefore have

$$\widehat{y} - y = \sum_{i=1}^{p} J_m^i(A, x)\Delta A_i y_{i-1} + J_m^0(A, x)\Delta x. \tag{2.11}$$

This expression will be the starting point to derive backward error formulas.

We will however also need a different expression in which the terms are rearranged, using the Kronecker product, to have the form of a linear system. Indeed, using the property of equation (1.41) of the Kronecker product we have

$$\widehat{y} = y + \sum_{i=1}^{p} (y_{i-1}^T \bigotimes J_m^i(A, x)) \operatorname{vec}(\Delta A_i) + J_m^0(A, x)\Delta x. \tag{2.12}$$

Let us then define the vector $\overrightarrow{\Delta A}$ as the concatenation of all the vectorized perturbations

$$\overrightarrow{\Delta A} = \begin{bmatrix} \Delta x \\ \overrightarrow{\Delta A_1} \\ \vdots \\ \overrightarrow{\Delta A_p} \end{bmatrix} \tag{2.13}$$

and the Jacobian matrix of our model with respect to the input and parameters

$$\mathcal{J}_m(A,x) = \left[J_m^0(A,x), \quad y_0^T \otimes J_m^1(A,x), \quad \ldots, \quad y_{p-1}^T \otimes J_m^p(A,x)\right]. \tag{2.14}$$

We then can rewrite equation (2.12) as the following linear system

$$\widehat{y} = y + \mathcal{J}_m(A,x)\overrightarrow{\Delta A}. \tag{2.15}$$

2.2.1 Backward error expression

To obtain formulas for the backward error, we will therefore proceed in the same manner as in section 2.1.1.

2.2.1.1 Componentwise backward error

We can define, using a first order approximation, the componentwise relative backward error as:

$$\begin{aligned}\varepsilon_{\text{bwd}} = \min\{\varepsilon \geq 0 : \widehat{y} - y = \sum_{i=1}^{p} J_m^i(A,x)\Delta A_i y_{i-1} + J_m^0(A,x)\Delta x, \\ |\Delta A_i| \leq \varepsilon |A_i|, \ i = 1,\ldots,p, \ |\Delta x| \leq \varepsilon |x|\}.\end{aligned} \tag{2.16}$$

Yet equation (2.11) implies

$$|\widehat{y} - y| \leq \sum_{i=1}^{p} |J_m^i(A,x)||\Delta A_i||y_{i-1}| + |J_m^0(A,x)||\Delta x|.$$

Then from the definition of the backward error from equation (2.16) we get

$$|\widehat{y} - y| \leq \varepsilon_{\text{bwd}} \left(\sum_{i=1}^{p} |J_m^i(A,x)||A_i||y_{i-1}| + |J_m^0(A,x)||x| \right)$$

which means that

$$\frac{|\widehat{y} - y|}{\sum_{i=1}^{p} |J_m^i(A,x)||A_i||y_{i-1}| + |J_m^0(A,x)||x|} \leq \varepsilon_{\text{bwd}}.$$

Therefore

$$\varepsilon_{\min} = \max_i \frac{|\widehat{y} - y|_i}{(\sum_{i=1}^{p} |J_m^i(A,x)||A_i||y_{i-1}| + |J_m^0(A,x)||x|)_i}$$

is a lower bound for ε_{bwd}.

So far, we have followed the same path as in the previous section; however, here, finding perturbations that satisfy the equality case is much harder. Indeed, in the case of a single layer followed by an activation function as in section 2.1.1, the Jacobian matrix is diagonal and therefore finding a perturbation that satisfies the equality $\widehat{y} = \phi((A + \Delta A)x) = \phi(Ax) + J_m(A, x)\Delta Ax$ was much easier. Whereas in the general case, the Jacobian matrices are not diagonal anymore.

In order to compute the backward error, we will therefore need the expression of equation (2.15), where $\mathcal{J}_m(A, x)$ is not diagonal.

In that case, finding the backward error defined as in equation (2.16) is equivalent, at first order, to solving the following optimization problem

$$\begin{aligned} \underset{\overrightarrow{\Delta A}}{\text{minimize}} \quad & \|\overrightarrow{\Delta A}\|_\infty \\ \text{subject to} \quad & \widehat{y} - y = \mathcal{J}_m(A, x)\overrightarrow{\Delta A}, \end{aligned} \tag{2.17}$$

where $\mathcal{J}_m(A, x) \in \mathbb{R}^{M \times N}$. Since we defined the sizes of the weight matrices and of the output as in equation (1.31), we have

$$\begin{aligned} M &= n_p, \\ N &= \sum_{i=1}^{p} n_i \times n_{i-1} + n_0 \end{aligned} \tag{2.18}$$

and therefore the system

$$\widehat{y} - y = \mathcal{J}_m(A, x)\overrightarrow{\Delta A}$$

is underdetermined. We hence have no closed formula to compute the componentwise backward error for a general neural network, however multiple methods [38, 1, 109, 17] focus on solving problems that are similar in form to the problem (2.17).

2.2.1.2 Normwise backward error

In the case where a first order approximation is given by equation (2.11), we cannot easily get an equivalent of the equation (2.6) that we obtained in a single layer case. However, knowing that equation (2.15) stands, since, as seen in equation (2.18), $\mathcal{J}_m(A, x)$ has a number of rows much smaller than its number of columns, we can make use of the Moore–Penrose generalized inverse, or, equivalently, Moore–Penrose pseudoinverse [83, 91, 35, 118], which for our given expression, defines the matrix

$$\mathcal{J}_m(A, x)^\dagger = \mathcal{J}_m(A, x)^T(\mathcal{J}_m(A, x)\mathcal{J}_m(A, x)^T)^{-1}$$

that gives the minimum norm solution of the system in equation (2.15). Let us then define the vector $\overrightarrow{A}$ as the concatenation of all the vectorized input and parameters

$$\overrightarrow{A} = \begin{bmatrix} x \\ \overrightarrow{A}_1 \\ \vdots \\ \overrightarrow{A}_p \end{bmatrix}. \tag{2.19}$$

Since the expression of the normwise backward error, knowing that $\|A\|_F = \|\overrightarrow{A}\|_2$, here is

$$\begin{aligned} \varepsilon_{\text{bwd}} = \min\{\varepsilon \geq 0 : \widehat{y} - y = \mathcal{J}_m(A, x)\overrightarrow{\Delta A}, \\ \|\overrightarrow{\Delta A_i}\| \leq \varepsilon \|\overrightarrow{A_i}\|,\ i = 1, \ldots, p,\ \|\Delta x\| \leq \varepsilon \|x\|\} \end{aligned} \tag{2.20}$$

then using the Moore–Penrose generalized inverse we have an explicit formula for the normwise backward error in

$$\varepsilon_{\text{bwd}} = \frac{\|\mathcal{J}_m(A, x)^\dagger (\widehat{y} - y)\|}{\|\overrightarrow{A}\|}. \tag{2.21}$$

Note that for $p = 1$, if we assume perturbations only on the parameters, we have

$$\widehat{y} - y = (x^T \bigotimes \phi'(Ax))\overrightarrow{\Delta A}$$

and

$$\mathcal{J}_m(A, x)^\dagger = (x^T \bigotimes \phi'(Ax))^\dagger = (x \bigotimes \phi'(Ax)^T)(\|x\|^2 I \bigotimes \phi'(Ax)^2)^{-1}.$$

Which means that

$$\begin{aligned} (x^T \bigotimes \phi'(Ax))^\dagger (\widehat{y} - y) &= (\frac{x}{\|x\|^2} \bigotimes \phi'(Ax)^{-1})(\widehat{y} - y) \\ &= \frac{\phi'(Ax)^{-1}(\widehat{y} - y)x^T}{\|x\|^2} \end{aligned}$$

and therefore the formula of equation (2.21), since for any given vectors a and b the equality $\|ab^T\|_F = \|a\|_2\|b\|_2$ stands, leads to

$$\varepsilon_{\text{bwd}} = \frac{\|\phi'(Ax)^{-1}(\widehat{y} - y)x^T\|}{\|A\|\|x\|^2} = \frac{\|\phi'(Ax)^{-1}(\widehat{y} - y)\|}{\|A\|\|x\|},$$

which is the same backward error as the one defined in equation (2.7).

2.2.2 Condition number expression

Consider a given neural network model, assuming perturbations on the model's parameters and on its input, we have the equation (2.2) that stands. We can define the vector $\overrightarrow{\Delta A}$ as in equation (2.13) and the Jacobian matrix $\mathcal{J}_m(A, x)$ as in equation (2.14). This leads to the first order expression of equation (2.15)

$$\widehat{y} - y = \mathcal{J}_m(A, x)\overrightarrow{\Delta A}$$

which serves as a starting point to express the condition number for each metric. Since from here the approach to obtain formulas are the same as in section 2.1.2, we directly get the following results.

2.2.2.1 Componentwise condition number

We can then define the absolute componentwise condition number as:

$$\kappa_m(A, x) = \|\mathcal{J}_m(A, x)\|_\infty$$

and the relative componentwise condition number is

$$\kappa_m(A, x) = \|\text{diag}(m(x))^{-1}\mathcal{J}_m(A, x)\text{diag}(\overrightarrow{A})\|_\infty.$$

2.2.2.2 Normwise condition number

In the case of a normwise metric, we get the absolute normwise condition number of the neural network in:

$$\kappa_\phi(A, x) = \|\mathcal{J}_m(A, x)\|$$

and the relative normwise condition number is

$$\kappa_\phi(A, x) = \frac{\|\mathcal{J}_m(A, x)\|\|A\|}{\|m(x)\|}.$$

Note that equation (2.15) stands when we consider perturbations on the input and all parameters. More generally, if we want to consider the backward error when perturbations are carried by a given set of variables, the vector $\overrightarrow{\Delta A}$ has to contain all those variables and the matrix $\mathcal{J}_m(A, x)$ has to be the Jacobian of the model with respect to each of those variables. Depending on which variables are considered to be perturbed, the expression of the Jacobian matrix and the vector containing the perturbed variables varies. Therefore, in order to have the following relation between forward, backward error and condition number

$$\varepsilon_{\text{fwd}} \leq \kappa_m(A, x)\varepsilon_{\text{bwd}},$$

one must ensure that quantities are defined with the same appropriate metrics and perturbations.

2.3 Link with artificial neural networks' robustness

Recent research from Beerens and Higham [6], building on our work in Chapter 2 and Chapter 3, uses the concept of componentwise backward error to generate adversarial attacks that aim to alter the ink consistency of images and showed that an increase in condition number is typically linked with a decrease in norm of adversarial attacks, resulting in less robust neural networks. Indeed, the intuition is that a given problem with high condition number will be more sensitive to small perturbations. Therefore, it is not surprising that adversarial attacks, that are perturbations on input data which lead the neural network to misclassify them, are more efficient on neural networks with high condition number. We can hence expect well conditioned neural networks to behave better in terms of robustness.

Other researches focus on using Lipschitz regularity of neural networks to better understand their sensitiveness to adversarial attacks. This was first introduced by Szegedy et al. [108]. Since then, k-Lipschitz neural networks [117, 8, 103] have been popularized and make use of the Lipschitz constant, which is an upper bound on the relationship between input perturbations and output variation, to obtain neural networks that are more robust and explainable.

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called Lipschitz continuous if there exists a constant k such that for all $(x, y) \in \mathbb{R}^m \times \mathbb{R}^m$

$$\|f(x) - f(y)\| \leq k\|x - y\|.$$

If there are such k's, then the infimum of them exists and is called the Lipschitz constant of f. Then $\mathrm{Lip}(f)$ is this constant and from Hutchinson [62] we have

$$\mathrm{Lip}(f) = \sup_{x \neq y} \left(\frac{\|f(x) - f(y)\|}{\|x - y\|} \right).$$

However, the absolute condition number of f is defined as in equation (1.15) by

$$\kappa_f(x) = \lim_{\varepsilon \to 0} \sup_{\|\Delta x\| \leq \varepsilon} \left(\frac{\|f(x + \Delta x) - f(x)\|}{\|\Delta x\|} \right).$$

We therefore can say that for a given point x we have

$$\kappa_f(x) \leq \mathrm{Lip}(f).$$

This link between condition number and Lipschitz regularity confirms previous expectations that the condition number can be an important quantity to evaluate neural networks' robustness.

2.4 Custom relative error

As defined in sections 1.1.1 and 1.1.2, for a computed result $\widehat{y}$ that is an approximation of $y = f(x)$, with $f : \mathbb{R}^n \to \mathbb{R}^m$, we formally have the relative forward error defined as

$$\varepsilon_{\text{fwd}}(\widehat{y}) = \max_i \frac{|\widehat{y}_i - y_i|}{|y_i|}$$

and the relative backward error as

$$\varepsilon_{\text{bwd}}(\widehat{y}) = \min \{\varepsilon : \widehat{y} = f(x + \Delta x),\ |\Delta x| \leq \varepsilon |x|\}\,.$$

For our error analysis and computations, in order to use the relative error we suppose that the exact solution y or the input x is not zero in the case of a normwise metric, and that all components of y and x are non-zero in the case of a componentwise metric. In order to take these cases into account and tackle problems occurring for small values, we introduce a custom relative error defined as

$$\varepsilon_{\text{fwd}}(\widehat{y}) = \max_i \frac{|\widehat{y}_i - y_i|}{|y_i| + \varepsilon_y} \tag{2.22}$$

and similarly the custom relative backward error defined as

$$\varepsilon_{\text{bwd}}(\widehat{y}) = \min \{\varepsilon : \widehat{y} = f(x + \Delta x),\ |\Delta x| \leq \varepsilon(|x| + \varepsilon_x)\}\,, \tag{2.23}$$

with $\varepsilon_y,\ \varepsilon_x \geq 0$ being arbitrarily fixed constants. In this case if $|y| \gg \varepsilon_y$ or $|x| \gg \varepsilon_x$ then this custom relative error is equivalent to the one defined in sections 1.1.1 and 1.1.2, or else, if $1 \gg |y|$ or $1 \gg |x|$, we can choose $\varepsilon_y,\ \varepsilon_x = 1$ and recover the absolute error.

With this new metric of error comes a new relationship between backward and forward error. Indeed at first order we have

$$\widehat{y} - y = J_f(x)\Delta x$$

which means that

$$\frac{\widehat{y} - y}{y + \varepsilon_y} = \text{diag}(f(x) + \varepsilon_y)^{-1} J_f(x) \text{diag}(x + \varepsilon_x) \frac{\Delta x}{x + \varepsilon_x}$$

and hence the custom condition number is

$$\kappa_m(x) = \|\text{diag}(f(x) + \varepsilon_y)^{-1} J_f(x) \text{diag}(x + \varepsilon_x)\|_\infty.$$

2.5 Numerical experiments

In this section, we aim to validate the formulas which have been derived in the previous sections. This typically involves computing the backward error, forward error and condition number for a given neural network whose computations are done in some given precision.

We choose to verify our formulas by applying them in the case where perturbations are due to rounding errors since this will enable us to have preliminary results on the effects of rounding errors for some given neural networks and to compare them with results already known in linear algebra. For example, we expect that for a single layer neural network with tanh activation function, when inputs and parameters are close to zero, we should get backward and forward errors that align with those of the matrix–vector product operation.

In these experiments we will focus on the componentwise metrics since they are the one we will use for the rounding error analysis in Chapter 4.

2.5.1 Experimental setup

These experiments are carried out with Python 3.8. Computations are performed in single precision while "exact" quantities are computed in double precision. In this case, errors are coming from the use of reduced precision, we will therefore assume that perturbations are carried by the neural network's parameters when computing backward errors and condition numbers.

We showed that in order to compute the backward error of deep neural networks, an optimization problem needs to be solved; this will be done using the CVXPY library [32, 2].

The experiments will at first be done on untrained neural networks randomly initialized with different distributions. One being a Gaussian $\mathcal{N}(\mu, \sigma)$ with

$$\begin{aligned} \mu &= 0, \\ \sigma &= \frac{1}{\sqrt{n}}, \end{aligned}$$

n being the number of neurons of a given layer. The other two distributions we use are uniform distributions $\mathcal{U}(a, b)$ where

$$a = 0,$$
$$b = \frac{1}{\sqrt{n}},$$

which is hence not centred in zero and allows for observing different behaviours and another one where

$$a = -\frac{1}{\sqrt{n}},$$
$$b = \frac{1}{\sqrt{n}}.$$

These choices come from both the observation that trained layers' weight values typically converge to these types of distributions and the Xavier's initialization from Glorot and Bengio [43] which is the most widely used type of parameters' initialization before training the neural networks. This type of distribution considers the number of parameters of each layer to determine the scale of the random initialization. This allows the activation functions and gradients to work effectively during both the forward phase and the backpropagation used during training.

For each neural network we randomly generate weights and inputs from these distributions $N_{\text{test}} = 10$ times and then compute the average $\varepsilon^{\text{mean}}$, maximum ε^{max}, backward ε_{bwd} and forward errors ε_{fwd}, as well as the condition number κ_m.

The trained neural networks architectures are provided in Table 2.1, they are initialized with Xavier's initialization and trained on FashionMNIST [127], this dataset allows for a more challenging classification task than MNIST while maintaining the same input sizes. Networks are trained using a cross entropy loss and Adam optimizer with a default learning rate of 10^{-3} and batch size 128. Both networks attain approximately 90% accuracy on the testing dataset when trained without overfitting.

2.5.2 Backward error computation

In order to assess the numerical solution obtained by solving the optimization problem of equation (2.17), we compare it with the theoretical formula of the backward error found in equation (2.3) for a single layer random neural network of size n and with a tanh activation function. Figure 2.1 shows the evolution of the backward error, computed either via the formula of equation (2.3) or via the optimization problem of equation (2.17), as a function of the number of neurons in the layer. In Figure 2.1a we compare the performance of different solvers that are available in the CVXPY library, using the theoretical formula as a reference. Note that the ECOS [36] solver is the one

TABLE 2.1: Neural networks architectures details.

Convolutional model		Fully connected model	
Layer	Shape	Layer	Shape
conv1-ReLU	$1 \times 6 \times 25$	linear1-tanh	784×500
max-pooling	$2 \times 2 \times s2$	linear2-tanh	500×500
conv2-ReLU	$6 \times 12 \times 25$	linear3-tanh	500×500
max-pooling	$2 \times 2 \times s2$	linear4-ReLU	500×10
linear3-tanh	192×500		
linear4-tanh	500×120		
linear5-tanh	120×60		
linear6-ReLU	60×10		

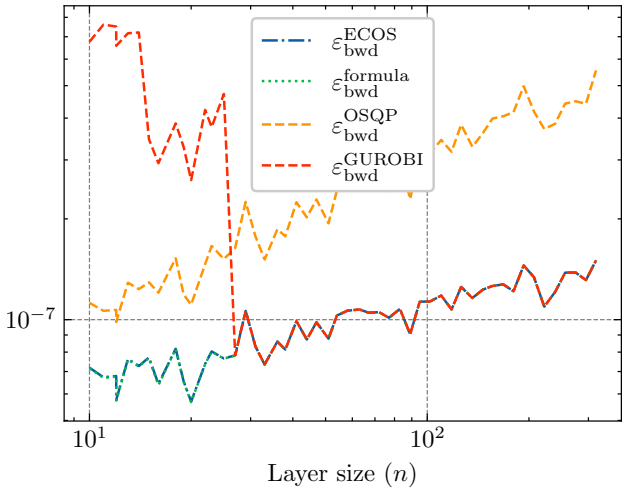
which is selected by default by CVXPY. Indeed, this library tries to automatically select the most efficient solver depending on the optimization problem.

First we can remark that the backward error we get in Figure 2.1, as one could expect, is close to the machine epsilon of the FP32 format, given in Table 1.1. These initial findings also allow us to rule out certain solvers which do not perform well compared to ECOS. The only solver that appears to give comparable results to ECOS, namely GUROBI [51], in fact, at first, does not perform well, like OSQP [107], and then CVXPY falls back on using ECOS since GUROBI raised an error. Using ECOS the relative error between the backward error computed using the optimization problem and using the formula is approximately of order 10^{-5} and, as seen in Figures 2.1b and 2.1c, is negligible in our experiments that aim to validate the bounds found later in Chapter 4. This is therefore the solver we will use in the following experiments of this manuscript.

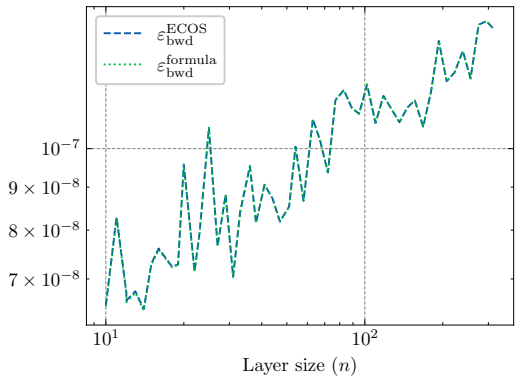
Note that we also tried different formulations of the optimization problem, as described by Earle [38], and other solvers, such as MOSEK [3], but ultimately, using built-in functions from CVXPY to express the optimization problem described by equation (2.17) leads to better results.

2.5.3 Condition number and bound on the forward error

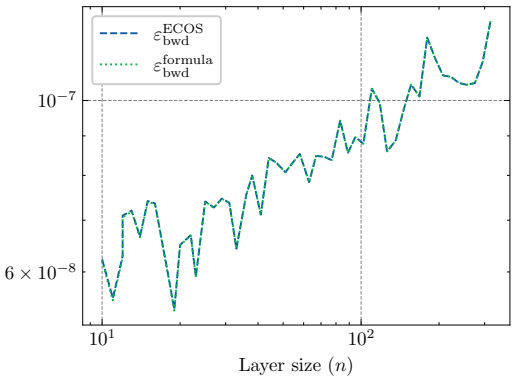
We have seen in section 1.1.4 that the inequality (1.14), which links condition number, backward error and forward error, stands as long as quantities are consistently defined. In Figure 2.2 we show the evolution of the backward error, forward error and the product of the condition number by the backward error. Inequality (1.14) is clearly satisfied, since the product of the condition number by the backward error follows the same trend as, and bounds, the forward error. In Figure 2.2b, backward error, forward error and its bound are very similar. Indeed, in case of a single layer neural network



(a) Parameters and entries taken from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution.

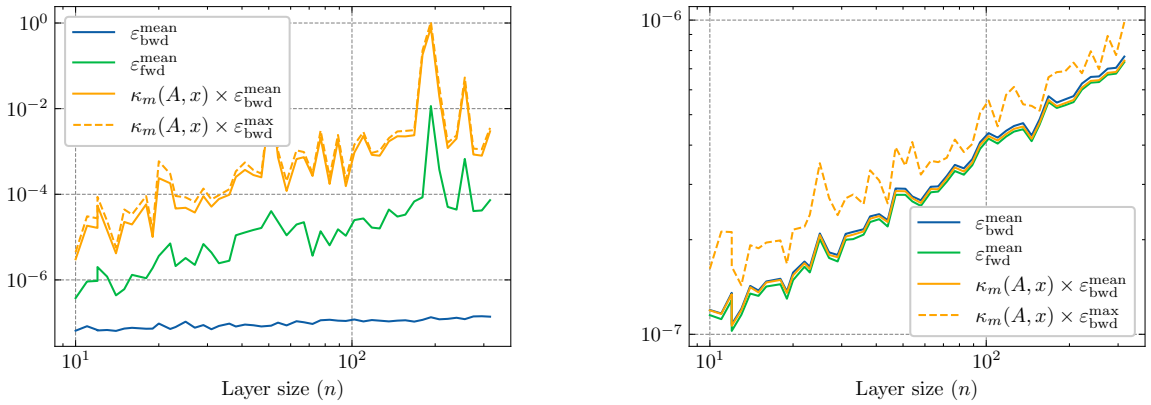


(b) Parameters and entries taken from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution.



(c) Parameters and entries taken from a $\mathcal{U}(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}})$ distribution.

FIGURE 2.1: Backward error, computed via equation (2.3) vs. problem (2.17), for a single layer neural network of size n.



(a) Parameters and entries taken from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution.

(b) Parameters and entries taken from a $\mathcal{U}(0, \frac{1}{\sqrt{n}})$ distribution.

FIGURE 2.2: Backward error, forward error and condition number for a single layer neural network of size n with random parameters and entries.

with tanh activation, the behaviour of the neural network is quasilinear around zero and the condition number, which is the one defined in equation (2.9), is close to one.

To further validate our formulas, we will now apply them to a deeper fully connected network whose architecture is given on the right side of Table 2.1. As with random networks, we compare the forward error to its bound of inequality (1.14). For random networks, we could easily generate networks of different sizes. Here, to have comparable networks, we take different states of the same network architecture at various stages of its training. Figure 2.3 therefore presents the evolution of the forward error and its bound using the computed backward error and condition number during training. Figure 2.3 clearly demonstrates that the forward error, which is computed exactly, is sharply predicted by the product of the backward error and the conditioning.

Note that the forward error increases drastically from Figure 2.2a, when random values are taken from a mean zero distribution, to Figure 2.2b. This is due to the use of a relative forward error when the processed values are close to zero. Since the matrix–vector product may result in arbitrary small values, then the relative forward error may consequently be much larger in case of mean zero distributions. This observation is in fact also true for the inner product, as shown in equation (1.24), since for this particular choice of values' distribution the absolute value of the inner product can be arbitrarily smaller than the inner product of the absolute values.

In order to tackle this issue and have a more coherent metric for values that are close to zero, we have introduced a custom relative error in section 2.4.

Since this new metric has been defined, we apply it to the case of a two-layer neural network with tanh activation and same size n in Figure 2.4. When using parameters

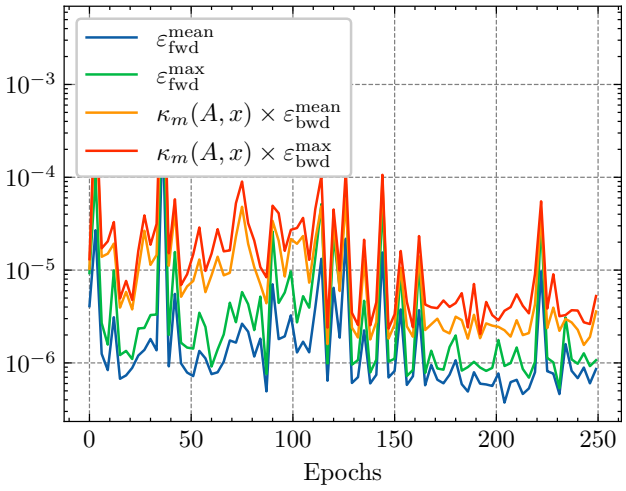


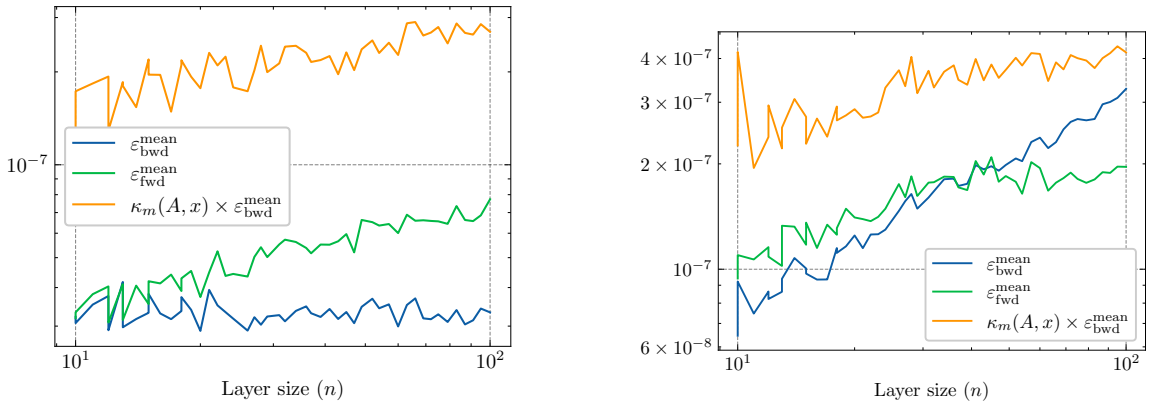
FIGURE 2.3: Backward error, forward error and condition number for a small fully connected network during training.

and entries taken from a zero-mean distribution, we set the arbitrary constant of equation (2.22) to one. This shows that backward and forward error are of similar order of magnitude, which is to be expected in a quasilinear context. The condition number multiplied by the backward error still bounds and follow the same increase as the forward error. Moreover, in Figure 2.4b, the backward error increases as the number of neurons in the layer increases. This is to be expected since the number of neurons per layers defines the number of operations that will be performed by this layer. Since the number of floating-point operations increases with n then the backward error is also expected to increase. However, the backward error in Figure 2.4a does not grow as the number of neurons n increases, this is in line with the results of Higham and Mary [57], since the input and parameters both have mean zero. These behaviours will be explored in more detail in Chapter 4, as we will specifically analyse the impact of rounding errors in the computations of artificial neural networks.

Overall the results we get using the custom relative error are more coherent, we will therefore use this metric in the following experiments of this manuscript when we deal with values that are close to zero.

2.5.4 Condition number during neural network training

During our experiments on classification neural networks, we noticed that an increase in the condition number of a neural network with respect to its parameters is directly linked with overfitting during training. In order to confirm these findings, we trained a convolutional network, whose architecture is given on the left side of Table 2.1, for a sufficiently high number of epochs and with a learning rate 10 times larger than the



(a) Parameters and entries taken from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution.

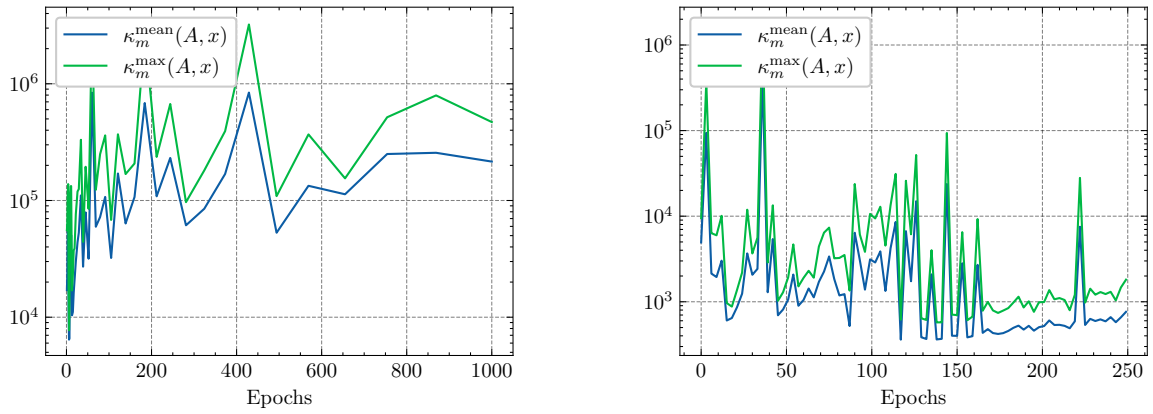
(b) Parameters and entries taken from a $\mathcal{U}(0, \frac{1}{\sqrt{n}})$ distribution.

FIGURE 2.4: Backward error, forward error and condition number for a two-layer neural network, each layer is of size n with random parameters and entries.

regular training setup. We then computed at each epoch the average and maximum condition number for a set of images.

Figure 2.5a shows the evolution of the condition number during such a training, while Figure 2.5b shows the condition number of the same neural network for a regular training setup without overfitting. Figure 2.5 should be compared with Figure 2.6 which shows the evolution of the training and testing losses during the training of both neural networks.

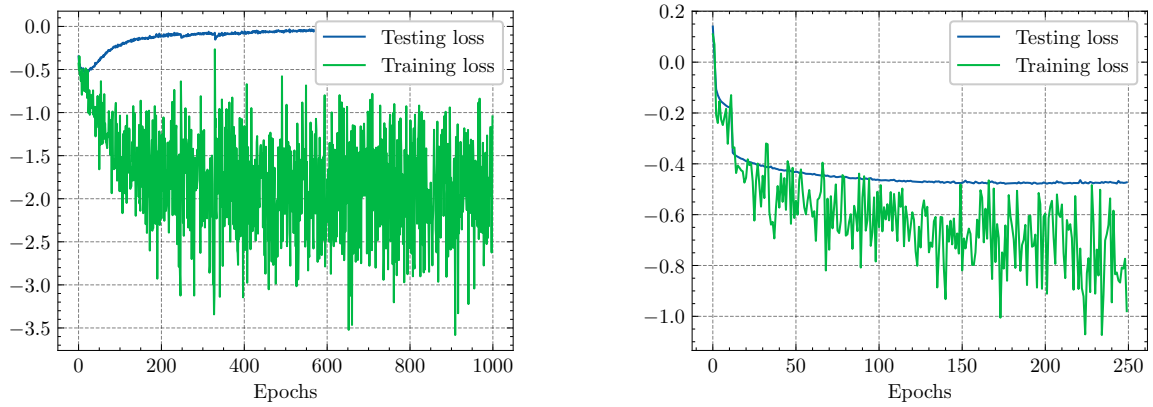
Figures 2.5 and 2.6 allow us to clearly distinguish the change in behaviour between the training phase without overfitting and the phase with overfitting. Indeed, in Figure 2.6a the testing loss starts to increase after around 30 epochs due to the higher learning rate, while in Figure 2.6b the learning rate is reduced and therefore there is no overfitting. By comparing these curves with those of the conditioning in Figure 2.5, a clear trend can be observed. When the network does not overfit, its conditioning is typically of the order of 10^4. In Figure 2.6a the condition number quickly increases, while in Figure 2.6b it is much more stable. As noticed in section 2.3, the condition number of neural networks appears to be a clear indicator of their robustness, implying that networks with higher condition numbers are less robust. These insights, combined with the initial results on trained neural networks from Figure 2.5, led us to propose an approach that involves using the condition number during the training of neural networks. Indeed, since an increase in the condition number typically leads to a set of undesirable properties, it is therefore a useful quantity to monitor, and even penalize, during the training of the network. Since, as stated in section 1.4.1, the training of a



(a) Overfitting neural network.

(b) Regularly trained neural network.

FIGURE 2.5: Comparison of the condition number evolution for an overfitting neural network and a regularly trained neural network.



(a) Overfitting neural network.

(b) Regularly trained neural network.

FIGURE 2.6: Comparison of the losses on training and testing dataset for an overfitting neural network and a regularly trained neural network.

neural network consists in finding parameters θ^* such that:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta).$$

Our proposed approach consists in adding the condition number of the network with respect to its parameters, which means that the problem becomes

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta) + \alpha\kappa_m(\theta, x),$$

with $\alpha \in \mathbb{R}^+$. Therefore, by introducing penalty when the condition number increases, we aim to reduce overfitting, additionally this will produce neural networks with smaller condition numbers, which also guarantees more robustness to perturbations, such as adversarial attacks.

Note we have not yet fully explored this idea which is the subject of an ongoing patent application.

2.6 Conclusion and discussion

We have shown in this chapter how significant quantities, that are commonly used in numerical linear algebra to better understand and evaluate numerical stability of algorithms, such as backward error and condition number, can be extended and computed for artificial neural networks. This analysis takes into account any neural network whose layers' operations are equivalent to a matrix–vector product followed by an activation function, such as fully connected or convolutional layers. We have shown that, with a first order approximation, we can obtain closed formulas to compute the componentwise and normwise backward errors for an entire layer of such a neural network. However, for larger neural networks, it is necessary to solve an optimization problem to obtain the componentwise backward error, whereas we can obtain an explicit formula for the normwise backward error by using the Moore–Penrose generalized inverse. On the other hand, the condition number can always be computed once we have access to the Jacobian matrix of the neural network.

We have implemented a Python code that is generic enough so that, given a neural network, it can compute all these quantities, regardless of the structure of the network. Using this code, we were able to perform an initial validation of our formulas on both randomly generated neural networks using known results from numerical linear algebra and deeper trained neural networks using the inequality (1.14).

We then focused more specifically on the conditioning of neural networks. Indeed, this quantity is commonly used to quantify the sensitivity of a system to perturbations in its data. We highlighted the connection between this quantity and the Lipschitz constant. In the particular case of neural networks, the Lipschitz constant is the

subject of active research to improve the robustness and explainability of networks. This link, combined with the experimental observations we made and those of Beerens and Higham [6], led us to propose a method for training neural networks by taking into account their condition number. The aim is to obtain more robust and explainable networks.

All these quantities are generic enough to assess the robustness of neural networks with respect to any parameter or input, as well as for a wide variety of perturbations. Section 2.5 focuses particularly on perturbations coming from rounding errors because it allows, for simple networks, a direct comparison with results obtained in linear algebra, notably from Higham and Mary [57].

In the coming chapters, we will use the backward error and condition number to better understand specific perturbations, such as adversarial attacks and rounding errors.

Chapter 3

Adversarial attacks on artificial neural networks

Section 1.4.3 introduced the concept of adversarial attacks on artificial neural networks. These are small perturbations applied to the inputs of classification neural networks designed to deceive their decision-making process. On the other hand, in Chapter 2, the concept of backward error was developed for generic neural networks. This concept also raises the question of finding small perturbations, but in that case it is to produce a given computed output.

If we apply the concept of backward error to the particular case of a classification neural network, the exact output value y leads to the ground-truth label. Then there exist multiple computed output $\widehat{y}$ that lead to the same class as y. However, if the class of $\widehat{y}$ is the same as y then the backward error is zero, since the minimal norm perturbations to obtain this class are zero.

Therefore, computing the backward error for a classification neural network, whose final outputs are discrete classes, amounts to finding the smallest perturbations on a network's parameters and/or input that changes the true output's class. Hence, there appears to be a close connection between adversarial attacks and backward error; we will attempt to explore this further in this chapter.

To do so, we will present two novel approaches for creating adversarial attacks. The first proposed approach relies on backward error analysis methods, presented in Chapter 2, that are more commonly used in scientific computing to assess the effects of finite precision computations or to measure the sensitivity of an algorithm to perturbations on data. Section 3.1.1 demonstrates how, because of its generality, the concept of backward error allows for computing targeted adversarial attacks on the input data as well as on the neural network's parameters; the latter correspond, for example, to the case where a malicious user tampers with the values of the weights or biases in order to alter the behaviour of the network. Section 3.1.2 shows how this practically enables the creation of adversarial attacks. After establishing how the backward error approach enables the creation of attacks, we will demonstrate how this new type of attack compares, both theoretically in section 3.1.3 and practically in

section 3.1.4 with pioneering adversarial attack algorithms.

The second approach aims to produce perturbations with smaller norms with respect to existing, state-of-the-art methods. To tackle this objective, we propose an approach that relies on second order information and, more precisely, on Sequential Quadratic Programming (SQP), which is a well known and widely studied method for solving constrained optimization problems. The base algorithm is presented in the section 3.2.1. We discuss the practical limitations of a baseline SQP-based method and propose some improvements to overcome them in section 3.2.2, which lead to a hybrid approach, mixing first order and second order iterations in order to achieve better convergence and lower execution time. Finally, in section 3.2.3, we present experimental results showing how the proposed approach compares to the state of the art, both in terms of performance in computation time and in creating efficient attacks.

Section 1.4.3 showed that both targeted and untargeted schemes exist. It has however been argued, notably by Carlini and Wagner [18], that computing an untargeted attack is often a less accurate approach than running a targeted attack for each target class and then take the smallest perturbation. For this reason, in our work we focus on computing a targeted attack, knowing that we can then use it in an untargeted context using this method. Our method can easily be extended to the case of untargeted attacks but we consider this is out of the scope of the present work. The following sections will thus be based on the following formulation of the optimization problem, equivalent to finding adversarial attacks for a given target label j,

$$\begin{aligned} \underset{\Delta x}{\text{minimize}} \quad & \|\Delta x\|^2 \\ \text{subject to} \quad & m_i(x+\Delta x) \leq m_j(x+\Delta x), \\ & i = 1, \ldots, C, \end{aligned} \tag{3.1}$$

where

$$m(x) = \phi_p(A_p \phi_{p-1}(A_{p-1} \ldots A_2 \phi_1(A_1 x) \ldots))$$

is the output of the classifier with C classes.

Note that this chapter focuses on adversarial attacks for the l_2-norm for the vectors, which, along with the l_∞-norm, is one of the most studied settings in the literature [27]. Moreover, the case of the l_∞-norm has been addressed by Beerens and Higham [6], whose work takes as a starting point the following study on the l_2-norm which we presented in [9].

3.1 Adversarial attacks via backward error analysis

Chapter 2 provides explicit definitions, formulas and methods to calculate the backward error, for different metric, for a generic deep neural network. This work will serve as the starting point for this section, in which we aim to extend these definitions and formulas to the case of classification networks. This extension of the results from the previous chapter will lead us to delve into the concept of adversarial attacks and to propose new approaches based on the backward error.

Note that we will focus on the normwise backward error expression, knowing that the componentwise approach was later developed by Beerens and Higham [6].

3.1.1 Backward error and its application to adversarial attacks

In this section, we will use the results of the previous chapter and generalize them to the case of a deep neural network used for classification tasks. Section 1.1.2 states that, for a given function f, input x and computed output $\widehat{y}$, the backward error as a quantity which is obtained by asking for what perturbed value of x the function has actually been applied, i.e. what is the perturbation Δx such that $\widehat{y}$ is the exact solution of $f(x + \Delta x)$, and in this case the backward error is the smallest value of $\|\Delta x\|$.

Usually, this measure of error is used to quantify rounding errors, so the computed result $\widehat{y}$ is obtained after computations. Now, if we consider this concept from the perspective of adversarial attacks, we could set a desired target output $\widehat{y}$ that we want to achieve from a given input x. We then arrive at a question of the same order as the backward error: what are the minimal perturbations necessary to achieve this result? Therefore, in that case, using the backward error formulas that we obtained before, we could compute the perturbations that are necessary to reach such a $\widehat{y}$.

For a generic neural network with p layers, the computed result is

$$\widehat{y} = m(x + \Delta x, A_1 + \Delta A_1, \dots, A_p + \Delta A_p)$$

and the normwise relative backward error was given, with a first order approximation, by equation (2.20)

$$\begin{aligned}\varepsilon_{\text{bwd}} = \min\{\varepsilon \geq 0 : \widehat{y} = y + \mathcal{J}_m(A, x)\overrightarrow{\Delta A}, \\ \|\overrightarrow{\Delta A_i}\| \leq \varepsilon \|\overrightarrow{A_i}\|,\ i = 1, \dots, p,\ \|\Delta x\| \leq \varepsilon \|x\|\},\end{aligned}$$

where $\mathcal{J}_m(A, x)$ is the Jacobian of the model with respect to its parameters and input.

Then the Moore–Penrose generalized inverse gives an explicit formula to find the backward error in:

$$\overrightarrow{\Delta A} = \mathcal{J}_m(A,x)^T(\mathcal{J}_m(A,x)\mathcal{J}_m(A,x)^T)^{-1}(\widehat{y}-y). \tag{3.2}$$

Using the expression of the Jacobian in equation (2.14), this can be written

$$\begin{bmatrix} \Delta x \\ \overrightarrow{\Delta A_1} \\ \vdots \\ \overrightarrow{\Delta A_p} \end{bmatrix} = \begin{bmatrix} J_m^0(A,x)^T \\ y_0 \otimes J_m^1(A,x)^T \\ \vdots \\ y_{p-1} \otimes J_m^p(A,x)^T \end{bmatrix} (\mathcal{J}_m(A,x)\mathcal{J}_m(A,x)^T)^{-1}\Delta y. \tag{3.3}$$

For the sake of readability, for $i = 0, \dots, p$, let us note

$$J_i^\dagger = J_m^i(A,x)^T(\mathcal{J}_m(A,x)\mathcal{J}_m(A,x)^T)^{-1}.$$

It follows from equation (3.3), and the property of equation (1.41) of the Kronecker product, that the perturbations associated with the given output $\widehat{y}$ are:

$$\begin{cases} \Delta x = J_0^\dagger \Delta y, \\ \Delta A_1 = J_1^\dagger \Delta y x^T, \\ \quad \vdots \\ \Delta A_i = J_i^\dagger \Delta y y_{i-1}^T, \\ \quad \vdots \\ \Delta A_p = J_p^\dagger \Delta y y_{p-1}^T. \end{cases} \tag{3.4}$$

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. This method could therefore be applied to create adversarial attacks on all kinds of networks by perturbing, depending on the context, either the input, a given set of parameters, or both. Since adversarial examples were initially studied and popularized in the context of classification neural networks, we will focus on applying our method in this context in section 3.1.2.

3.1.2 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 3.1.1. 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 assigns the input to class j instead of the expected one. As stated in section 1.4.3 the most common way for the classification decision to be made in neural networks is based on the maximum component of the output. Therefore, mathematically, the adversarial perturbation is defined as the solution of the following minimization problem

$$\begin{aligned} \underset{\Delta A_i,\, \Delta x}{\arg\min} \quad & \sum_{i=1}^{p} \frac{\|\Delta A_i\|^2}{\|A_i\|^2} + \frac{\|\Delta x\|^2}{\|x\|^2} \\ \text{subject to} \quad & \widehat{y} = \phi_p((A_p + \Delta A_p)\phi_{p-1}((A_{p-1} + \Delta A_{p-1}) \\ & \ldots \phi_1((A_1 + \Delta A_1)(x + \Delta x)))), \\ & \widehat{y}_i \leq \widehat{y}_j, \ i = 1, \ldots, C. \end{aligned} \tag{3.5}$$

The optimization problem, in this form, is very challenging to solve because of the equality constraint and the potentially very large size of the variables. However, from section 3.1.1 we know that, we can express the minimum norm perturbations that satisfy the equality constraint of problem (3.5) at first order as functions of $\widehat{y}$ with equation (3.4). This allows to have the following equality

$$\sum_{i=1}^{p} \frac{\|\Delta A_i\|^2}{\|A_i\|^2} + \frac{\|\Delta x\|^2}{\|x\|^2} = \sum_{i=1}^{p} \frac{\|J_i^{\dagger} \Delta y y_{i-1}^T\|_F^2}{\|A_i\|_F^2} + \frac{\|J_0^{\dagger} \Delta y\|_2^2}{\|x\|^2} \tag{3.6}$$

where all quantities are known except $\widehat{y}$ in $\Delta y = \widehat{y} - y$. This means that we can alternatively use $\widehat{y}$ as a way to find adversarial attacks, therefore using it as an optimization variable. Indeed, using backward error analysis, we express the perturbations as variables which only depend on a given approximate result $\widehat{y}$ and on the network's parameters.

Since we can express the norm of the perturbations as in equation (3.6), we can use the fact that for any given vectors a and b we have $\|ab^T\|_F = \|a\|_2 \|b\|_2$ to obtain

$$\begin{aligned} \sum_{i=1}^{p} \frac{\|\Delta A_i\|^2}{\|A_i\|^2} + \frac{\|\Delta x\|^2}{\|x\|^2} &= \sum_{i=1}^{p} \frac{\|y_{i-1}\|_2^2}{\|A_i\|_F^2} \|J_i^{\dagger} \Delta y\|_2^2 + \frac{1}{\|x\|^2} \|J_0^{\dagger} \Delta y\|_2^2 \\ &= \|J^{\dagger}(\widehat{y} - y)\|_2^2 \end{aligned} \tag{3.7}$$

with

$$J^{\dagger} = \begin{bmatrix} \frac{\|x\|}{\|A_1\|} J_1^{\dagger} \\ \vdots \\ \frac{\|y_{i-1}\|}{\|A_i\|} J_i^{\dagger} \\ \vdots \\ \frac{\|y_{p-1}\|}{\|A_p\|} J_p^{\dagger} \\ \frac{1}{\|x\|} J_0^{\dagger} \end{bmatrix}.$$

Since equation (3.7) is obtained for minimal norm perturbations that satisfy the equality constraint of the optimization problem (3.5), this problem can therefore be reduced to

$$\begin{aligned} \underset{\widehat{y}}{\arg\min} \quad & \|J^{\dagger}(\widehat{y} - y)\|^2 \\ \text{subject to} \quad & \widehat{y}_i \leq \widehat{y}_j, \ i = 1, \ldots, C. \end{aligned} \tag{3.8}$$

By using this formulation we are no longer directly seeking minimal-norm perturbations, but rather an output vector $\widehat{y}$ of size C, which, through the backward error approach, allows us to obtain minimal-norm perturbations. Indeed, once the optimization problem is solved and, thus, $\widehat{y}$ is computed, the adversarial perturbations can be computed using equation (3.4).

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

$$\begin{aligned} \underset{\Delta A_i, \Delta x}{\arg\min} \quad & \sum_{i=1}^{p} \frac{\|\Delta A_i\|^2}{\|A_i\|^2} + \frac{\|\Delta x\|^2}{\|x\|^2} \\ \text{subject to} \quad & \Delta y = \mathcal{J}_m(A, x)\overrightarrow{\Delta A}, \\ & \widehat{y}_i \leq \widehat{y}_j, \ i = 1, \ldots, C. \end{aligned} \tag{3.9}$$

This formulation can be further refined by iterating on the perturbations, for example in the case where we attack the input data we solve at each iteration the following optimization problem:

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \|\Delta x + d\|^2 \\ \text{subject to} \quad & \Delta y = J_m^0(A, x)d, \\ & \widehat{y}_i \leq \widehat{y}_j, \ i = 1, \ldots, C \end{aligned} \tag{3.10}$$

and at the end of each iteration, the following modifications are applied:

$$\Delta x \leftarrow \Delta x + \alpha d,$$
$$x \leftarrow x + \alpha d,$$

with α a fixed regularization term. This approach leads to the Algorithm 1.

Algorithm 1 Iterative backward error attack

Input: a starting point for the algorithm x; a number of iterations N_{iter}.
$x_1 \leftarrow x$
$\Delta x \leftarrow 0$
for $k = 1, \dots, N_{\text{iter}}$ **do**
  $\underset{d}{\text{minimize}} \, \|\Delta x + d\|^2$
  subject to $\Delta y = J_m^0(A, x_k)d, \ \widehat{y}_i \leq \widehat{y}_j, \ i = 1, \dots, C$
  $\Delta x \leftarrow \Delta x + \alpha d$
  $x_{k+1} \leftarrow x_k + \alpha d$
end for
Output: a perturbation Δx.

Alternatively, we can also use this method to generate attacks on the network parameters. In this case, we need to adapt the Jacobian matrix, which must then be taken with respect to the perturbed 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. The formulation of the problem as given in equation (3.8) has the clear advantage of greatly reducing the size of the variable to be optimized, which could be a considerable advantage when focusing on attacks on a neural network's parameters. On the other hand, the formulation in Algorithm 1 might be more suitable when the variable to optimize is of smaller size, for example when considering attacks only on the input, or only on a given small set of parameters. This version allows for taking small steps iteratively, thus neglecting the effects of model linearisation.

Note that here we focus on the case where the classifier assigns the input data to the j-th class when $\widehat{y}_i \leq \widehat{y}_j, \ i = 1, \dots, C$, but this can easily be generalized to other types of classification by modifying these constraints.

3.1.3 Comparison with other approaches

Most of the approaches that generate adversarial examples, including FGSM [46] or FGSM-based approaches [72], SGD (Stochastic Gradient Descent) [76] or SGD-based approaches [26], FAB [25], 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 in equation (1.32):

$$\begin{aligned} \underset{\Delta x}{\text{minimize}} \quad & \|\Delta x\|^2 \\ \text{subject to} \quad & \text{Class}(x + \Delta x) = j, \end{aligned}$$

where j is the target class and $\text{Class}(x + \Delta x)$ the class of the perturbed image. This problem being difficult to solve, the above-mentioned approaches commonly resort to solving the problem of equation (1.36)

$$\underset{\Delta x}{\text{minimize}} \quad \|\Delta x\|^2 + c\, \mathcal{L}(x + \Delta x, j)$$

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 backward error analysis. This enables us to simplify equation (1.32) and formulate it as in equation (3.8) or equation (3.9).

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) [47], where a discriminative network acts as a loss function.

3.1.4 Numerical experiments

In this section, the objective will be to evaluate the performance of the previously defined attacks. Given that attacks on inputs are the most well-known and studied in

the literature, we will first compare our method in the context where only the inputs are perturbed. This allows us to assess the robustness of our method compared to what is done in the literature and justify its extension to attacks on the parameters.

Finally, we will evaluate the performance of our attack when applying perturbations to the parameters of neural networks by comparing it with the results previously obtained on the inputs.

Note that these experiments will be done on neural networks whose task is to classify images, since it is the most studied case.

3.1.4.1 Experimental setup

For all of our experiments we train a fully connected neural network, using tanh activation functions. Neural network models are trained using the Keras library [21] and Adam's optimizer [69] with a sparse categorical cross entropy's loss on Python.

We will apply our different versions of the backward error attack to image classification cases, using the MNIST [31] and CIFAR10 [70] datasets since they are the most well-known and commonly used datasets for creating adversarial attacks.

The neural network's structure is described 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 export its parameters and use them to compute adversarial attacks as described before, using MATLAB R2020a. For each experiment we take the 100 first images on the testing dataset, we first solve the optimization problem of equation (3.8) or the iterative version of equation (3.9), 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. The Jacobian matrix is computed using the formula of equation (2.14).

For all these experiments, and for each adversarial-example-generating algorithm, we set a same maximum number of iterations after which we consider the attack to have failed for a fixed threshold ε.

The experimental approach is divided 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 3.1.4.3 and 3.1.4.2.

3.1.4.2 Adversarial attacks on a neural network's input

In Figure 3.1 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 Algorithm 1. For perturbations

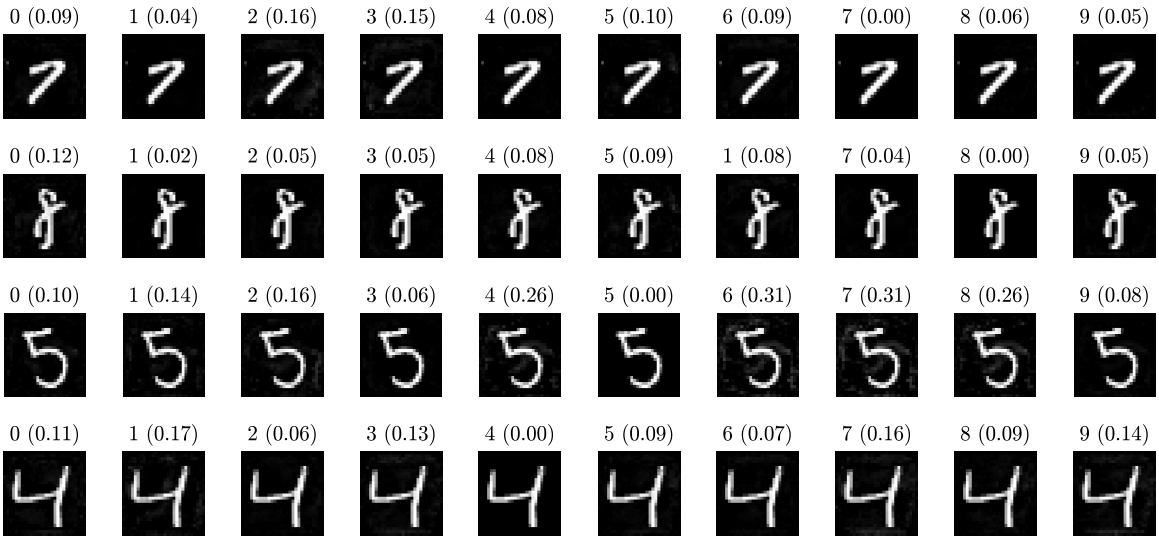


FIGURE 3.1: Adversarial examples found with BE Attack; above each image is the obtained label and, in parentheses, the norm of the perturbation.

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 3.1 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, other attacks in an untargeted setting.

In this section we perform an attack only on the input and compare it to well-know attacks such as Fast Gradient Method (FGM) which is the l_2-norm variation of FGSM [46], Projected Gradient Descent (PGD) [76] and DeepFool [84], using the Foolbox library [92]. For each attack, we search for the best hyperparameter needed on Foolbox before comparing the different methods. We give the network accuracy, defined as the percentage of correctly classified inputs, for different ε assuming that the attack is successful if:

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

Hence, if we do not find any perturbations that change the network's result for a given image, such that

$$\frac{\|\Delta x\|}{\|x\|} < \varepsilon$$

then we report a network accuracy of 100%.

TABLE 3.1: Adversarial attacks on MNIST, $(784, 10)$ neural network.

ε	BE attack (3.8)	BE attack (3.10)
0.2	70%	**23%**
0.1	90%	**41%**
0.05	97%	**68%**

As seen in Table 3.1, using the iterative version given in Algorithm 1 gives better results than using the formulation of equation (3.8), hence we will use this one on the following comparisons. In the case of Table 3.2 we compare the backward error attack

TABLE 3.2: Adversarial attacks on MNIST, $(784, 10)$ neural 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%**

to other attacks, this results show that our method outperforms FGM and L2PGD, while finding many more adversarial examples with relative norm smaller than 0.1 compared to DeepFool. In Table 3.3 we increase the size of the neural network, while

TABLE 3.3: Adversarial attacks on MNIST, $(784, 100, 10)$ neural 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%**

the backward error attack still gives satisfying results for smaller norm perturbations, it shows that for perturbations of relative norm closer to 0.2 results of the different attacks seem to even out. However, our attack still performs well in cases of smaller norms.

In the next experiments, we will attempt to confirm these initial findings with a network trained on a more complex database, namely CIFAR10. In Table 3.4 the

TABLE 3.4: Adversarial attacks on CIFAR10, $(3072, 768, 10)$ neural 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%**

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.

The results we get in Tables 3.2, 3.3, and 3.4 show that even by perturbing only the input data our method still obtains satisfactory results, in the majority of the cases outperforming PGD and FGSM, it is also performs better than Deepfool in many instances, and otherwise, it at least compares well.

With this method, we are therefore able to generate effective attacks compared to the attacks used as examples. However, one of the major contributions of our attack is that it also allows for generating perturbations on the parameters. Unfortunately, to our knowledge, there are no other attacks that allow for this but these results suggest that this works compares well to existing attacks when perturbing the inputs.

3.1.4.3 Adversarial attacks on a neural network's parameters

In this section, we perform an attack only on the neural network's parameters by applying perturbations on all of its weights. Note that, however, our attack is generic enough to produce perturbations on any given set of parameters. For each attack we give the network accuracy for different ε assuming that the attack is successful if:

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

Hence, if we do not find any perturbations that change the network's result for a given image, such that

$$\max_i \frac{\|\Delta A_i\|}{\|A_i\|} < \varepsilon$$

then we report a network accuracy of 100%.

TABLE 3.5: Adversarial attacks on weights on MNIST, $(784, 10)$ neural network.

ε	BE attack (3.8)	BE attack (3.9)
0.5	**30%**	39%
0.2	**38%**	51%
0.1	**50%**	61%
0.05	**64%**	83%
0.01	**88%**	100%

Unlike what we have seen in section 3.1.4.2, in Table 3.5, formulation of equation (3.8) leads to better results than using the iterative version of equation (3.10), even if this still gives satisfying results. This corresponds to what we expected, given that in this case the optimization variable is large and thus the problem is more difficult to solve. Hence, on the following comparisons we will use the version of equation (3.8).

TABLE 3.6: 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 3.6 show that backward error can be used to efficiently fool a given neural network by perturbing its 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 as the one we get before in section 3.1.4.2 by perturbing the input data.

In Table 3.7, the dataset is more complex and the neural network has more parameters, these results show that perturbations on the parameters are, in that case, more efficient than those on the input. Indeed, 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 3.1.4.2, despite using the same neural network for both attacks. This was not the case for the neural networks trained on MNIST, which shows that some neural

TABLE 3.7: Adversarial attacks on weights on CIFAR10, $(3072, 768, 10)$ neural network.

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

network are less robust than others when it comes to perturbations on their parameters. Moreover, some networks are more sensitive to parameters perturbations than to input perturbations.

3.2 Adversarial attacks via Sequential Quadratic Programming

In the previous section, we have used the backward error formulas obtained in Chapter 2 to create adversarial attacks on the parameters and inputs of a neural network. This has led us to produce comparisons of the backward error attacks with some standard methods of adversarial attacks on inputs. In order to do so, we proposed an iterative version of the backward error attack, with Algorithm 1, which performed well compared to DeepFool, FGSM and PGD. In this section, we will explain how this iterative algorithm can be further refined using a well-known optimization method, namely Sequential Quadratic Programming.

3.2.1 Local Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is one of the most well-known and successful methods for constrained nonlinear optimization; the first reference to SQP-type algorithms appears in the PhD thesis of Wilson [123]. An exhaustive review of this method is out of the scope of this manuscript but, in this section, we will explain the general idea behind it; we refer the reader to the above-mentioned thesis or to any optimization textbook such as the one by Boggs and Tolle [14] or Nocedal and Wright [87]. Although SQP is generic enough to solve optimization problems with both equality and inequality constraints, in this work we will focus on the case where only

inequality constraints are present because this is enough for computing adversarial attacks, as explained in the next section. Therefore, our review of the SQP method below is limited to this case for the sake of conciseness.

Let us assume that one has to solve the following minimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0. \end{aligned} \tag{3.11}$$

The idea behind SQP is to solve this problem iteratively: at each iteration x_k the problem is modelled by a constrained quadratic programming subproblem whose minimizer is used to build iterate x_{k+1}. One common choice is to solve the Karush–Kuhn–Tucker (KKT) equations using Newton's method. The Lagrangian of this problem is

$$L(x, \lambda) = f(x) + \lambda^T g(x), \tag{3.12}$$

where the vector λ corresponds to the Lagrangian multipliers.

To satisfy the KKT conditions one must find (x, λ) such that

$$\begin{aligned} & \nabla L(x, \lambda) = 0 \\ & g(x) \leq 0. \end{aligned} \tag{3.13}$$

Applying Newton's method to the above equation generates iterates that are identical to those one gets by solving the following quadratic problem:

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} L(x_k, \lambda_k) d \\ \text{subject to} \quad & \nabla g(x_k)^T d + g(x_k) \leq 0. \end{aligned} \tag{3.14}$$

This leads to the local SQP method in Algorithm 2 where (d_x, d_λ) are the primal dual solution of the quadratic optimization subproblem.

Algorithm 2 local SQP

1: **Input:** a starting point for the algorithm x_1; a number of iterations N_{iter}.
2: **for** $k = 1, \ldots, N_{\text{iter}}$ **do**
3: $\quad \underset{d}{\text{minimize}} \, \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} L(x_k, \lambda_k) d$
4: $\quad$ subject to $\nabla g(x_k)^T d + g(x_k) \leq 0$
5: $\quad x_{k+1} \leftarrow x_k + d_x$
6: $\quad \lambda_{k+1} \leftarrow d_\lambda$
7: **end for**
8: **Output:** an approximate solution $(x_{N_{\text{iter}}+1}, \lambda_{N_{\text{iter}}+1})$.

3.2.2 Proposed approach

In this section, we will adapt the local SQP algorithm described in the previous section to solve adversarial attack problems. Essentially, this amounts to defining an objective function and inequality constraints that correspond to the problem in equation (1.32). We will focus on a targeted formulation of this problem knowing that, as explained before, an untargeted version can be obtained by applying the produced algorithm to all classes and then taking the smallest successful perturbation.

Let us assume that $m : \mathbb{R}^n \to \mathbb{R}^C$ is a classifier such that m assigns to an input $x \in \mathbb{R}^n$ the label i, i being one of the C classes, when

$$m_j(x) \leq m_i(x), \ j = 1, \ldots, C. \tag{3.15}$$

Then, as stated in equation (3.1), one must solve the following optimization problem in order to find an adversarial perturbation:

$$\begin{aligned} \underset{\Delta x}{\text{minimize}} \quad & \frac{1}{2}\|\Delta x\|^2 \\ \text{subject to} \quad & m_i(x + \Delta x) \leq m_j(x + \Delta x), \\ & i = 1, \ldots, C, \end{aligned}$$

where j is a target class that is different from the true label of the input x.

Assuming $x_k = x + \Delta x$ is the perturbed input, for the specific case of adversarial attacks on a neural network classifier, the objective function of the optimization problem can be defined as

$$f : x_k \mapsto \frac{1}{2}\|x - x_k\|^2. \tag{3.16}$$

Let us define I_C the identity matrix of size C, e_j the j-th canonical vector of size C and $\mathbf{1_C}$ the all-ones vector of size C. Then the inequality constraints of the optimization problem can be expressed as

$$g(x_k) = (I_C - \mathbf{1_C}^T e_j) m(x_k) \leq 0. \tag{3.17}$$

3.2.2.1 Improvements to the basic SQP algorithm

Algorithm 2 can readily be used to solve the optimization problem with f and g defined as in the previous section. Nevertheless, this naive approach suffers from some limitations. First, SQP being a local algorithm, its convergence is not guaranteed for any given starting point. This means that Algorithm 2 may fail to compute a successful adversarial attack or to compute one with a small norm; this behaviour was indeed observed in a preliminary experimental evaluation.

The second major drawback relates to the use of the Hessian of the Lagrangian $\nabla^2_{xx} L(x_k, \lambda_k)$ which, essentially, amounts to computing second order derivatives of the neural network function that appears in the constraints defined in equation (3.17). Note that whether this term has to be explicitly computed depends on the optimizer which is chosen to solve the quadratic subproblem on lines 3–4 of Algorithm 2. If the optimizer relies on a direct method, this term has to be computed explicitly; for other choices, it may be sufficient to provide the local optimizer with a function to apply this operator to a vector. In all cases, computing or applying this term may be excessively expensive (both in terms of operations and memory).

In order to overcome or mitigate these two limitations, we propose a few modifications.

The first improvement consists in using a hybrid approach where a first order method is used prior to the SQP iterations. The objective is to provide SQP with a better starting point so that it is more likely to converge. Let us say that one seeks to solve the optimization problem (3.11). Using a first order development of the constraints function we obtain

$$g(x_{k+1}) = g(x_k + d) = \nabla g(x_k)^T d + g(x_k).$$

Then a first order iterative problem close in spirit to SQP would be the following:

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & f(x_k + d) \\ \text{subject to} \quad & \nabla g(x_k)^T d + g(x_k) \leq 0. \end{aligned} \tag{3.18}$$

This quadratic problem allows iterating on the perturbations while linearizing the constraints and has therefore the same purpose as our approach in Algorithm 1. In our case, where f is defined as in equation (3.16), we have:

$$\begin{aligned} f(x_k + d) &= \frac{1}{2}\|x - x_k - d\|^2 \\ &= \frac{1}{2}\langle x - x_k | x - x_k \rangle - \langle d | x - x_k \rangle + \frac{1}{2}\langle d | d \rangle . \end{aligned} \tag{3.19}$$

By noting that

$$\nabla f(x_k)^T d = -\langle d | x - x_k \rangle \tag{3.20}$$

we obtain

$$f(x_k + d) = \frac{1}{2}\langle x - x_k | x - x_k \rangle + \nabla f(x_k)^T d + \frac{1}{2} d^T I d. \tag{3.21}$$

Hence, solving problem (3.18) is equivalent to solving

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \nabla f(x_k)^T d + \frac{1}{2} d^T I d \\ \text{subject to} \quad & \nabla g(x_k)^T d + g(x_k) \leq 0. \end{aligned} \tag{3.22}$$

This is achieved by performing iterations of Algorithm 2 where $\nabla^2_{xx} L(x_k, \lambda_k)$ in line 3 is replaced with the identity matrix.

A second minor improvement consists in adding regularization terms to the update of the current solution at iteration k; these are called α for the first order iterations updates and β for the second order iterations update. The use of these regularization terms was found to drastically improve the convergence in practice and more will be said in the experimental evaluation.

Finally, the choice of the starting point for the first order iterations is still critical to speed up the convergence of the method. In a basic implementation a natural choice would be to set $x_1 = x$, that is, the starting point for the optimization solver is the unperturbed input. Assuming that a targeted adversarial attack towards class j must be computed, we have found that a better starting point is a randomly selected input belonging to class j.

Combining all these improvements leads us to the final algorithm, which we called *S*equential *QU*adratic Programming *AT*tack (SQUAT) and which is described in Algorithm 3. Here, a total number of N_{iter} iterations is performed; alternatively, the method can be stopped when a perturbation of small enough norm is computed. Out of these iterations, the first N_1 only use first order information; these are relatively cheap and serve to get closer to an optimal solution. The remaining iterations, that are based on the SQP method and, thus, rely on second order information, refine the solution computed by the previous ones. The value of N_1 as well as of α and β has to be carefully chosen to achieve fast convergence. Some heuristics on the choice of these parameters are presented in the experiment section.

3.2.3 Numerical experiments

3.2.3.1 Experimental setup

In order to compare our method with the current state of the art, we chose to use the code provided by Rony et al. [96]. In the following comparisons we will only use non-targeted attacks to show that our method can compete with, and even outperform, state-of-the-art attacks on a non-targeted scheme. More precisely, we will compare our method to state-of-the-art methods for the l_2-norm: C&W from Carlini and Wagner [18], DDN from Rony et al. [97], FAB from Croce and Hein [25], and finally ALMA from Rony et al. [96]. The experiments are done on a SmallCNN neural network from Zhang et al. [129], in order to evaluate our attack against defences, we use this

Algorithm 3 SQUAT

Input: a target label j; a total number of iterations N_{iter}; a number of iterations before using second order information N_1; a couple of fixed regularization terms α and β for first and second order iterations respectively; an image x^j belonging to class j.
$x_1 \leftarrow x^j$
for $k = 1, \dots, N_{\text{iter}}$ **do**
  if $k = 1, \dots, N_1$ **then**
    $\underset{d}{\text{minimize}}\, \nabla f(x_k)^T d + \frac{1}{2} d^T I d$
  else if $k > N_1$ **then**
    $\underset{d}{\text{minimize}}\, \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} L(x_k, \lambda_k) d$
  end if
  subject to $\nabla g(x_k)^T d + g(x_k) \leq 0$
  if $k \leq N_1$ **then**
    $x_{k+1} \leftarrow x_k + \alpha d_x$
  else if $k > N_1$ **then**
    $x_{k+1} \leftarrow x_k + \beta d_x$
    $\lambda_{k+1} \leftarrow \beta d_\lambda$
  end if
end for
Output: an approximate solution $(x_{N_{\text{iter}}+1}, \lambda_{N_{\text{iter}}+1})$.

network in different training setups. First it is trained regularly and then trained to be robust to adversarial attacks: first using l_∞-TRADES defence [129] and then using l_2-DDN defence [97] on the MNIST dataset [31]. We will denote these neural networks, respectively, SmallCNN for the regularly trained model, SmallCNN-TRADES and SmallCNN-DDN for the adversarially trained models.

3.2.3.2 Metrics

We assess the performance of an attack by how much the accuracy of a given neural network decreases depending on how small perturbations are in terms of norm. This is measured using a metric called *robust accuracy*, which measures the accuracy of the neural network when all its inputs are subject to attacks of a given norm. Therefore, for a given value ε, assuming all the inputs of a dataset are attacked with perturbations of norms smaller than ε, it corresponds to the percentage of input data that have been successfully classified to the ground-truth class. Hence, the lower the robust accuracy for a given threshold, the better the attack. In Figures 3.3, 3.4 and 3.5 we plot curves showing the robust accuracy as a function of ε.

For all the experiments, we chose to evaluate the attacks on the first 500 images of the testing dataset. To summarize these results, we also present the median distance between the found adversarial example and the original input for multiple attacks and neural networks in Table 3.9. The median distance here essentially represents the threshold value ε such that the network achieves a robust accuracy of 50% for a given attack.

3.2.3.3 Algorithm implementation and complexity

The complexity of an attack on a deep learning system can be measured in terms of forward and backward evaluations of the model as these operations usually require more computational power than other operations involved in these algorithms. In our case, most of the operations are done computing first and second order derivatives of the neural network model to define the quadratic subproblems given in lines 5 and 7 of Algorithm 3. These subproblems are then solved using the Python CVXPY library [32, 2]. For the experiments reported in this section, the algorithm used by the solver is the default one, namely, Operator Splitting Quadratic Program [107] (OSQP) which implements an Alternating Direction Method of Multipliers [15] (ADMM) variant. In this case, the solver requires the entire matrix involved in the problem's formulation, which, in our case, implies computing the Hessian of the Lagrangian. Other types of optimizer could be used to solve the quadratic subproblem and other implementation choices could be made in order to speed up the execution but exploring all these parameters is out of the scope of this document as we aim to show how a simple SQP-type approach could efficiently be used to create competitive adversarial attacks.

The choice of the number of iterations N_{iter} has, clearly, a considerable impact on the cost of our method. Essentially, we can afford to do many first order cheap iterations (N_1) in order to get as close as possible to an optimal solution, such that only few expensive second order iterations are needed to reach convergence. In our experiments, N_1 is fixed to a relatively large number. As for the second order iterations, the number is not fixed but we keep on iterating as long as the constraints of the quadratic subproblem are satisfied because this means that the algorithm has successfully improved the solution obtained at the previous iteration. The number of second order iteration is, typically, of the order of a few tens.

For the following experiments we chose to use a budget of 4000 iterations per image for ALMA, DDN, FAB and C&W while for SQUAT we use a budget of approximately 2000 iterations as those iterations can be more expensive. As an example, for the SmallCNN-TRADES neural network, this typically results in a total of 4000 forward and 4000 backward propagations for DDN and ALMA, 8000 forward and 40000 backward for FAB, 30000 forward and 30000 backward for C&W. In this case, the SQUAT attack performed in average 20 iterations with second order information per image while having set $N_1 = 2000$ first order iterations per image. For the first order

iterations one forward and $C = 10$ backward propagations are needed, while in order to compute a Hessian matrix–vector multiplication one forward and one backward propagations would be needed, but to compute the full Hessian here the cost scale as the input size n so we make the assumption that computing the Hessian costs as much as n forward and n backward propagations, which gives an average of 18000 forward and 36000 backward propagations in total.

The learning rate α used in the first order iterations, has to be carefully chosen. In our experiments, in order to keep the computational cost comparable to other methods, we chose not to use a line-search algorithm to find an optimal learning rate value; instead, we only run the first order part of our attack with few iterations in order to compare the algorithm performance depending on α. We compare, in Table 3.8, the median distance of the attacks perturbations depending on the value of α, and, in Figure 3.2, an example of how changes on α impact the final results of the SQUAT algorithm by showing robust accuracy curves on the SmallCNN-TRADES. The results presented in Table 3.8 are consistent in terms of median distances compared to which defence is used. Indeed, our attack is based on the l_2-norm, so it is logical that the defence via DDN, which is a l_2-norm defence, performs better in countering it than the l_∞-norm defence of TRADES and the regular training setup. This is confirmed by the fact that the median distances are typically larger when using the DDN defence compared to the other setups.

Training setup					
Regular		DDN		TRADES	
α	Median distance	α	Median distance	α	Median distance
0.8	1.80	0.7	3.68	0.2	1.89
0.9	**1.76**	0.8	3.55	0.3	1.70
1	1.79	0.9	**3.52**	0.4	**1.68**
1.1	1.83	1	3.58	0.5	2.01

TABLE 3.8: Adversarial attack performance on SmallCNN depending on the choice of α and the training setup.

As seen in Table 3.8, and particularly for the more robust networks, the behaviour of the algorithm can be very sensitive to the choice of α. For the SmallCNN and SmallCNN-DDN neural network we chose $\alpha = 0.9$ but, as seen in Figure 3.2, the median distance does not always give enough information and using robust accuracy curves we chose in this case to use $\alpha = 0.3$ as it gave a better performance trade-off between smaller and larger norm. The second hyperparameter β, which is the learning rate when using second order information, is fixed at $\beta = 0.15$ for all our experiments. Indeed, tuning this parameter as it has been done for α can be costly; moreover the goal here is to improve a good enough starting point, it is not an exploration phase

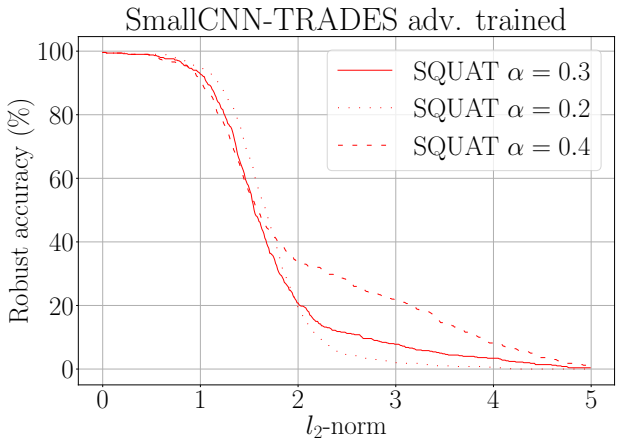


FIGURE 3.2: Robust accuracy curves for the SmallCNN-TRADES adversarially trained model on MNIST.

anymore, hence one does not want to make large steps and we found that usually any $\beta \leq 0.2$ can be a good choice, knowing that the smaller β is the more steps will be needed to converge.

3.2.3.4 Results

As shown on the robust accuracy curves of all different attacks on Figure 3.3 and on the median distances of Table 3.9 we observe that our proposed attack SQUAT outperforms other state-of-the-art attacks on the regularly trained SmallCNN. Indeed, for a given threshold, the robust accuracy is globally lower with SQUAT than with other attacks. This is even clearer for smaller norms, as seen on the right panel of Figure 3.3: for norms in the $[0, 1]$ range, all other attacks have comparable results and only our method stands out. Finally, it must be noted that SQUAT and ALMA attain 100% attack success rate approximately at the same perturbation norm which is smaller than the other methods.

On Figure 3.4 and Figure 3.5 which give robust accuracy curves on the SmallCNN-DDN and SmallCNN-TRADES we can see that all the methods behave approximately the same for norms belonging in the $[0, 1]$ interval; this is not surprising as the neural networks are much more robust and hence less affected by small norm perturbations. On the l_2-DDN adversarially trained network all the attacks are significantly less efficient than for the regularly trained network, this is expected as these attacks are designed for the l_2-norm and the SmallCNN-DDN is designed to be robust to these types of attacks. As seen in Figure 3.4, in this setup ALMA, SQUAT perform similarly, with a slight advantage for ALMA when perturbations norm are in the $[2, 5]$ range. On the other hand, C&W, FAB and DDN all perform similarly for this neural

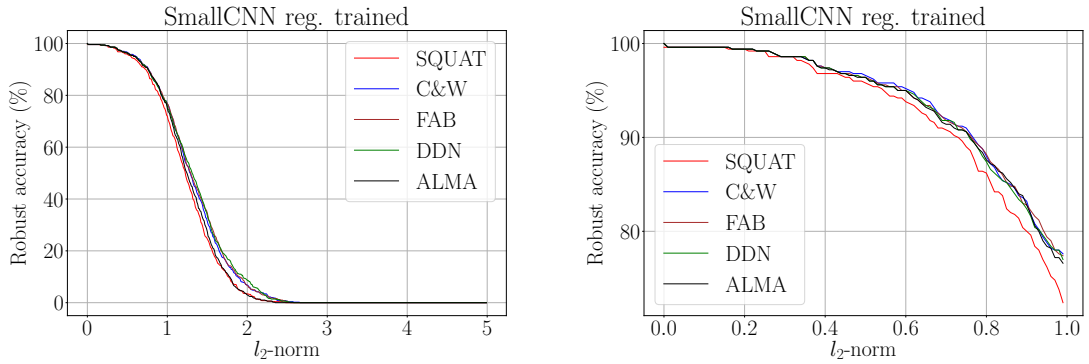


FIGURE 3.3: Robust accuracy curves for the SmallCNN regularly trained model on MNIST.

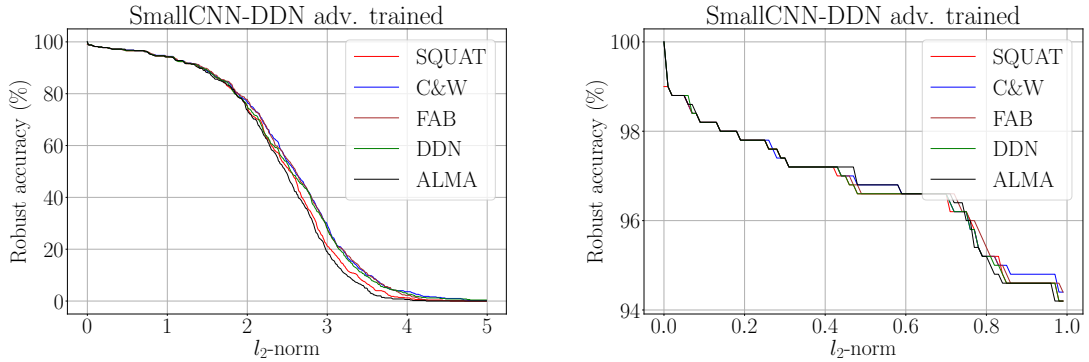


FIGURE 3.4: Robust accuracy curves for the SmallCNN-DDN adversarially trained model on MNIST.

network. Median distances obtained on the l_∞-TRADES adversarially trained neural network are closer to those we get on the regularly trained one except for the C&W attack which performs poorly compared to other state-of-the-art attacks. In this case, SQUAT perform similarly to FAB, outperforming DDN and C&W in terms of median distance of adversarial perturbations, whereas globally ALMA get better results on this training setup.

3.3 Conclusion and discussion

In section 3.1 we have performed a backward error analysis of generic deep neural networks which 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

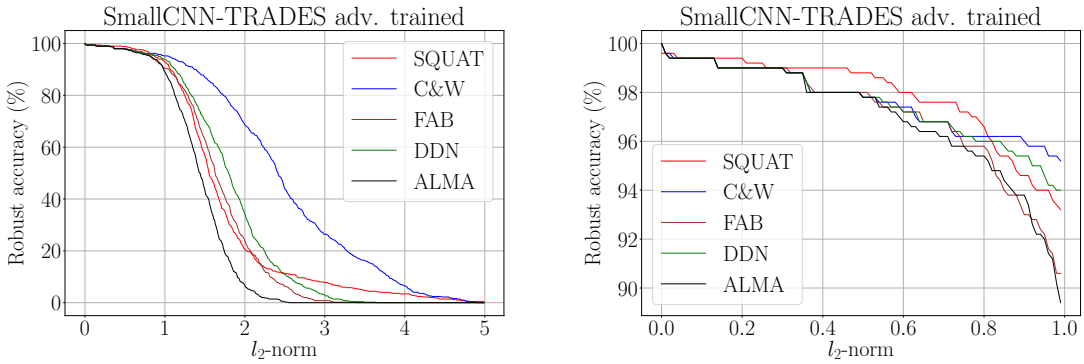


FIGURE 3.5: Robust accuracy curves for the SmallCNN-TRADES adversarially trained model on MNIST.

	Training setup		
	Regular	DDN	TRADES
Attack	Median distance	Median distance	Median distance
SQUAT	**1.22**	2.53	1.55
FAB	1.29	2.61	1.63
DDN	1.31	2.55	1.80
ALMA	1.24	**2.48**	**1.44**
C&W	1.28	2.59	2.41

TABLE 3.9: Adversarial attacks performances on SmallCNN depending on the training setup.

used to train the neural network, on either the input data or the neural network's parameters.

As seen in section 3.1.4, our method can outperform well-known methods in the case where the attack is performed on the input data of a given network. Moreover, most significantly, this method is generic enough to consider attacks on any set of inputs and parameters which 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 datasets (MNIST and CIFAR10). The goal of this approach 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 well-known 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.

This discovery of adversarial attacks on a neural network's parameters has led us to take interest in the quantization of deep neural networks and propose a method for quantizing a deep neural network. Quantization of deep neural networks refers to the process of reducing the number of bits used to represent, for example, the weights of the network. Indeed, the adversarial attack method we proposed in section 3.1.2 allows for determining the smallest perturbation of the neural network's parameters, beyond which the output of the deep neural network is erroneous. Therefore, for a target inference accuracy of the neural network, we can adjust the values of the parameters as long as this does not result in larger perturbations than the adversarial limit. Thus, one can vary the arithmetic precision according to the perturbation threshold while maintaining the target accuracy. Note that this approach can be modulated since the proposed attack can be applied to a subset or all of the parameters. This method has been patented [11] and could be the subject of more in-depth future research.

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.

This leads us to the second part of this chapter, section 3.2, where we have proposed an improvement of the iterative Algorithm 1 and shown how to compute adversarial attacks on deep neural networks using a Sequential Quadratic Programming based approach, adapting the basic algorithm to this specific case, in particular by first using a first order variation of SQP in order to then use second order information to improve the resulting perturbations.

The goal of this section is to provide a first look of how successful adversarial attacks can be built using second order information and using existing optimization algorithms. As seen in the experimental results section, we have shown that our approach can compete and even outperform others state-of-the-art attacks on models that are regularly and adversarially trained.

As seen in the work by Rony et al. [96], using existing optimization methods to attack neural networks often requires substantial adjustments. Our approach, proposed in [10], relies on few, simple, modifications of the original SQP algorithm, which enable the use of this method in the specific case of adversarial attacks on a neural network and still obtains competitive results.

Many SQP-type algorithms and solvers [16] have been developed and could be used in our approach, notably Gill, Murray, and Saunders [42] designed a software using algorithm with limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. Those types of solvers, using Hessian approximation or taking advantage of Hessian–vector product could be of special interest for designing a more robust and less expensive SQP-approach to create adversarial examples in the future.

The existence of new types of adversarial attacks poses potential security threats to machine learning models. Hence, designing adversarial attacks and defences is a subject of great interest. Indeed, they enable to better understand neural networks sensitivity and improve their robustness, notably by the means of adversarial training. This work shows how to construct adversarial attacks on a neural network's parameters 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.

Chapter 4

Rounding error analysis for artificial neural networks

This chapter presents a rounding error analysis for artificial neural networks. The goal of such an analysis is to provide bounds on the backward error in order to explain and quantify how the use of a given arithmetic precision impacts the stability and accuracy of an algorithm. Using the backward error over the forward error provides several advantages. Indeed, by focusing on backward error, the analysis can offer insights into the root causes of instabilities in the neural network's predictions and behaviours. Moreover, once bounds on the backward error are found, bounds on the forward error can be directly derived using the condition number of the problem, as seen in Chapter 2.

The primary contribution of this chapter will be to explain in section 4.1 how to integrate the nonlinear activation functions into the deterministic analysis of rounding errors. This requires understanding how rounding errors produced by the computation of the function can be interpreted as errors on the input of the function. Subsequently, we will integrate this analysis to obtain bounds for the computations of a single layer neural network and this will finally lead us to generalize these bounds to the case of deeper neural networks.

In the second part of this chapter, namely section 4.2, we will focus on integrating these functions in the probabilistic setting. Two versions will be proposed, one where the interactions between the errors made during the layer application and those due to the activation function is deterministic and one where these interactions are treated in a probabilistic manner.

4.1 Deterministic bounds

4.1.1 Activation function

Here and for the remainder of this document we will assume that for all activation functions, similarly to Model 1.1, the following model stands:

Model 4.1 (Floating-point arithmetic model for activation functions)**.**

$$\mathrm{fl}(f(x)) = f(x)(1+\delta_f),\ |\delta_f| \leq \ell u.$$

Knowing that for each activation function, the constant ℓ has to be evaluated for each framework. As an example, for NVIDIA GPUs (Graphics Processing Units) the constant for activation functions such as tanh or ReLU can be found on the Appendix E of the documentation [88].

To integrate activation functions into the backward error analysis, we need to know how the rounding error obtained by applying the function, according to Model 4.1, can be interpreted back as a perturbation on the data.

Let us define $f : \mathbb{R}^n \to \mathbb{R}^n$. Assuming that f is differentiable at the point x and that $f(x) \neq 0$, at first order, we have

$$f(x+\delta) = f(x) + \delta f'(x),$$

which means that

$$f(x+\delta) = f(x)(1+\delta\frac{f'(x)}{f(x)}).$$

Let us define

$$K_f(x) = \frac{xf'(x)}{f(x)}$$

and note $\kappa_f(x) = |K_f(x)|$, which essentially represents the componentwise condition number of f at x, then

$$f(x+\delta) = f(x)(1+\frac{\delta}{x}K_f(x)). \tag{4.1}$$

Here we want to use equation (4.1) to understand the impact of rounding errors, in terms of perturbations on the input data x, when Model 4.1 stands. Let f be a given activation function, differentiable at x, that satisfies this model. The computed solution $\widehat{y}$ then satisfies

$$\widehat{y} = f(x)(1+\delta_f) = f(x)(1+\frac{\delta_f}{x}K_f(x)\frac{x}{K_f(x)}).$$

Replacing δ in equation (4.1) by

$$\Delta x = \delta_f \frac{x}{K_f(x)},$$

means that we have

$$\begin{aligned}\widehat{y} = f(x)(1+\delta_f) &= f(x + \delta_f \frac{x}{K_f(x)}) \\ &= f(x + \Delta x),\end{aligned}$$

with

$$|\Delta x| \leq \frac{\ell u}{\kappa_f(x)}|x|.$$

This result shows that the relative perturbation on the input that is needed to get the computed output $\widehat{y}$ is bounded by

$$\frac{\ell u}{\kappa_f(x)}.$$

Therefore, the perturbations are small when $\kappa_f(x) \geq \ell u$. This states that the bigger the condition number of f is, the smaller the perturbation needed on the input will be to attain a same output. This is consistent with the fact that functions with small condition number are less sensitive to perturbations.

4.1.2 Entire layer

We now search to combine the results from section 1.3.2.2 and section 4.1.1 to obtain rounding error bounds for a complete layer of artificial neural network.

Consider an entire layer composed of a matrix–vector product followed by an activation function, let $y = \phi(Ax)$ with x a given input vector. Each output component $\widehat{y}_i$ then satisfies:

$$\widehat{y}_i = \phi\left(\sum_{k=1}^{n}(a_{ik}x_k)(1+\varepsilon_k)\prod_{j=\max(k,2)}^{n}(1+\delta_j)\right)(1+\delta_\phi).$$

Note that each rounding errors ε_k, δ_j and δ_ϕ also depend on i but are not indexed to simplify the notation.

From section 4.1.1 we know how to take into account rounding errors introduced by the activation function in terms of perturbations on the input data, therefore

$$\widehat{y}_i = \phi\left(\sum_{k=1}^{n}\left((a_{ik}x_k)(1+\varepsilon_k)\prod_{j=\max(k,2)}^{n}(1+\delta_j)\right)\left(1+\frac{\delta_\phi}{K_\phi(a_i^T x)}\right)\right),$$

which means that

$$\widehat{y}_i = \phi\left(\sum_{k=1}^{n}(a_{ik}x_k)(1+\psi_k)\left(1+\frac{\delta_\phi}{K_\phi(a_i^T x)}\right)\right). \tag{4.2}$$

Equation (4.2) clearly shows two distinct perturbation terms, namely

$$(1+\psi_k) = (1+\varepsilon_k)\prod_{j=\max(k,2)}^{n}(1+\delta_j)$$

and

$$\left(1+\frac{\delta_\phi}{K_\phi(a_i^T x)}\right)$$

arising from the rounding errors of the matrix–vector product and of the activation function, respectively. The goal here is to express the product of these two terms

$$(1+\psi_k)\left(1+\frac{\delta_\phi}{K_\phi(a_i^T x)}\right) = 1+\psi_k+\frac{\delta_\phi}{K_\phi(a_i^T x)}+\frac{\psi_k\delta_\phi}{K_\phi(a_i^T x)},$$

as $1+\Phi_k$, where

$$\Phi_k = \psi_k+\frac{\delta_\phi}{K_\phi(a_i^T x)}+\frac{\psi_k\delta_\phi}{K_\phi(a_i^T x)}$$

and to bound the absolute value of Φ_k. Yet from Lemma 1.2 we know that $|\psi_k| \leq \gamma_n$, therefore since $|\delta_\phi| \leq \ell u$, we have

$$|\Phi_k| = \left|\psi_k+\frac{\delta_\phi}{K_\phi(a_i^T x)}+\frac{\psi_k\delta_\phi}{K_\phi(a_i^T x)}\right| \leq \frac{nu}{1-nu}+\frac{\ell u}{\kappa_\phi(a_i^T x)}+\frac{\ell n u^2}{\kappa_\phi(a_i^T x)(1-nu)}.$$

Gathering all terms under the same denominator leads to

$$|\Phi_k| \leq \frac{\kappa_\phi(a_i^T x)nu+\ell u(1-nu)+\ell n u^2}{\kappa_\phi(a_i^T x)(1-nu)},$$

which means that

$$|\Phi_k| \leq \kappa_\phi(a_i^T x)\frac{nu+\frac{\ell u}{\kappa_\phi(a_i^T x)}}{\kappa_\phi(a_i^T x)(1-nu)}$$

and therefore

$$|\Phi_k| \leq \frac{(n+\frac{\ell}{\kappa_\phi(a_i^T x)})u}{1-nu},$$

which can finally be further weakened to

$$|\Phi_k| \leq \frac{(n + \frac{\ell}{\kappa_\phi(a_i^T x)})u}{1 - (n + \frac{\ell}{\kappa_\phi(a_i^T x)})u} = \gamma_{n+\ell/\kappa_\phi(a_i^T x)}.$$

This means that equation (4.2) can be written as

$$\widehat{y}_i = \phi\left(\sum_{k=1}^{n} a_{ik} x_k (1 + \Phi_k)\right),$$

where

$$|\Phi_k| \leq \gamma_{n+\ell/\kappa_\phi(a_i^T x)}.$$

Combining the m rows gives us

$$\widehat{y} = \phi((A + \Delta A)x), \ |\Delta A| \leq \gamma_{n+\ell/\kappa_\phi(Ax)}|A|.$$

Note that the inequality is to be taken componentwise, with $\kappa_\phi(Ax)$ being equal to $\kappa_\phi(a_i^T x)$ if the component is on the i-th row. This result shows that each relative perturbation on the input is bounded by $\gamma_{n+\ell/\kappa_\phi(Ax)}$. Hence, the perturbations are small when

$$\left(n + \frac{\ell}{\kappa_\phi(Ax)}\right) u \leq 1.$$

This can be interpreted as a combination of the results obtained in section 1.3.2.2 and section 4.1.1. The bounds reflect the fact that n basic operations are done by the matrix–vector product and then the ℓ errors due to the activation are amplified or reduced depending on its condition number.

4.1.3 General neural network

Bounds on the backward error were previously obtained for a single layer of artificial neural network, we now take interest in the case of a general neural network of p layers.

In order to better explain our main result, let us first develop how rounding error propagates on a couple of layers. Let $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{p \times m}$ and $x \in \mathbb{R}^n$.

Assuming that we compute $y = \phi_2(B\phi_1(Ax))$ by first performing $z = \phi_1(Ax)$ and then $y = \phi_2(Bz)$, we thus have two layer applications in a row. From section 4.1.2, for a given output component $\widehat{z}_i$, the following equation follows

$$\widehat{z}_i = \phi_1(\sum_{k=1}^{n}(a_{ik}x_k)(1 + \psi_k)(1 + \frac{\delta_{\phi_1}}{K_{\phi_1}(a_i^T x)})).$$

Then for the second layer application, the output component $\widehat{y}_i$ satisfies

$$\widehat{y}_i = \phi_2(\sum_{k=1}^{m}(b_{ik}\widehat{z}_k)(1+\psi_k)(1+\frac{\delta_{\phi_2}}{K_{\phi_2}(b_i^T\widehat{z})})).$$

Hence, applying the results of the previous section leads to

$$\widehat{y} = \phi_2((B+\Delta B)\phi_1((A+\Delta A)x)),$$

with

$$|\Delta A| \leq \gamma_{n+\ell_{\phi_1}/\kappa_{\phi_1}(Ax)}|A|,\ |\Delta B| \leq \gamma_{m+\ell_{\phi_2}/\kappa_{\phi_2}(B\widehat{z})}|B|.$$

This result enables us to distinguish a pattern for bounds on the backward error when chaining neural networks' layers. Indeed, for a given layer, the bound combines, first the aggregation of errors due to the matrix–vector product, in the form of the number of columns of the weight matrix, n_{i-1}, and then the impact of the activation function.

Findings derived from a couple of layers imply, through induction, that the subsequent key result can be formulated:

Theorem 4.1 (Deterministic error bounds for artificial neural networks)**.** *Consider a neural network composed of p layers whose output can be expressed as follows*

$$y = \phi_p(A_p\phi_{p-1}(A_{p-1}\dots A_2\phi_1(A_1x)\dots)),$$

with $A_i \in \mathbb{R}^{n_i\times n_{i-1}}$, for $i=1,\dots,p$ and $x\in\mathbb{R}^{n_0}$. Then, if y is evaluated in floating-point arithmetic, the computed result $\widehat{y}$ satisfies

$$\widehat{y} = \phi_p((A_p+\Delta A_p)\phi_{p-1}((A_{p-1}+\Delta A_{p-1})\dots(A_2+\Delta A_2)\phi_1((A_1+\Delta A_1)x)\dots)),$$

with

$$|\Delta A_i| \leq \gamma_{n_{i-1}+\ell_{\phi_i}/\kappa_{\phi_i}(A_i\widehat{y}_{i-1})}|A_i|,\ i=1,\dots,p.$$

These bounds are obtained by using both the standard model of arithmetic, given by Model 1.1, for the matrix–vector computations, and the Model 4.1 for the computation of the activation function. These results enable us to quantify the impact of activation functions in the context of computations performed by neural networks in finite precision. This impact appears in the form of the constant ℓ_{ϕ_i}, which, for each activation function ϕ_i, corresponds to the rounding errors introduced by the application of the function, and this constant is then amplified or not depending on the condition number of the activation.

4.2 Probabilistic bounds

The goal of this section is to show how the bounds obtained on artificial neural networks in section 4.1.3 can be adapted to the probabilistic case. As shown in [57, 58, 23], Theorem 1.5 enables to directly replace the deterministic γ_n with its probabilistic equivalent $\widetilde{\gamma}_n(\lambda)$ in most numerical linear algebra operations when the performed computations induce rounding errors that follow Model 1.2.

However, since activation functions are not standard numerical linear algebra operations, as shown in section 4.1.2, we will demonstrate how this changes the approach and results we had in Theorem 4.1.

Two different ways of integrating activation functions in the probabilistic error analysis can be distinguished. In section 4.2.1 we will first take interest in the case where we make probabilistic assumptions on the errors due to linear algebra operations, such as matrix–vector product, and use only deterministic assumptions for their interaction with errors made for the computation of the activation function. In section 4.2.2 we will handle the interaction between the matrix–vector rounding errors and those of the activation functions using a probabilistic model.

4.2.1 Deterministic activation function's rounding error

As shown in section 4.1.2, the result of a neural network layer followed by an activation function results in a product of rounding error terms of the following form, seen in equation (4.2)

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}\left(1+\frac{\delta_{n+1}}{K_\phi(x)}\right). \tag{4.3}$$

From Theorem 1.5 we know that, if the first n rounding error terms satisfy Model 1.2, then the following result stands:

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1+\theta_n,$$

with $|\theta_n| \leq \widetilde{\gamma}_n(\lambda)$, with probability at least

$$1-2\exp\left(\frac{-\lambda^2}{2}\right).$$

The goal of this section is to find a similar result, expressing the rounding error product terms of equation (4.3) as $1+\Psi_n$ where the absolute value of Ψ_n is bounded by a given constant with some given probability.

Theorem 4.2 (Deterministic error bounds for activation function)**.** *Let $\delta_1, \ldots, \delta_n$ be random variables of mean zero with $|\delta_k| \leq u$ for all k such that $\mathbb{E}(\delta_{k+1} \mid \delta_1, \ldots, \delta_k) =$*

$\mathbb{E}(\delta_{k+1}) = 0$ *for* $k = 1, \ldots, n-1$. *Let* $|\delta_{n+1}| \leq \ell u$. *Then for* $\rho_i = \pm 1,\ i = 1, \ldots, n$ *and any constant* $\lambda > 0$,

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}\left(1+\frac{\delta_{n+1}}{K_\phi(x)}\right) = 1+\Psi_n,$$
$$|\Psi_n| \leq \widetilde{\gamma}_n(\lambda) + \frac{\ell}{\kappa_\phi(x)}u + O(u^2)$$

holds with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

Proof. From equation (4.3) we get

$$\begin{aligned}\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}(1+\frac{\delta_{n+1}}{K_\phi(x)}) &= (1+\theta_n)(1+\frac{\delta_{n+1}}{K_\phi(x)}) \\ &= 1+\theta_n+\frac{\delta_{n+1}}{K_\phi(x)}+\theta_n\frac{\delta_{n+1}}{K_\phi(x)}.\end{aligned}$$

Then let

$$\Psi_n = \theta_n + \frac{\delta_{n+1}}{K_\phi(x)} + \theta_n\frac{\delta_{n+1}}{K_\phi(x)},$$

we have

$$|\Psi_n| = \left|\theta_n + \frac{\delta_{n+1}}{K_\phi(x)} + \theta_n\frac{\delta_{n+1}}{K_\phi(x)}\right|,$$

which implies

$$|\Psi_n| \leq |\theta_n| + \left|\frac{\delta_{n+1}}{K_\phi(x)}\right| + \left|\theta_n\frac{\delta_{n+1}}{K_\phi(x)}\right|.$$

Using Theorem 1.3, we get that

$$\begin{aligned}|\Psi_n| &\leq \widetilde{\gamma}_n(\lambda) + \frac{\ell u}{\kappa_\phi(x)} + \widetilde{\gamma}_n(\lambda)\frac{\ell u}{\kappa_\phi(x)} \\ &\leq \widetilde{\gamma}_n(\lambda) + \frac{\ell u}{\kappa_\phi(x)} + O(u^2),\end{aligned}$$

holds with probability at least

$$P(\lambda) = 1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

□

This theorem provides bounds on the expression in equation (4.2), which result from rounding errors arising from both matrix–vector product and the application of the activation function. It combines a probabilistic approach for the first n rounding errors introduced by the matrix–vector product to a deterministic approach for their interaction with errors arising from the activation function. Once this result is established we are ready to give bounds for a general artificial neural network in the following theorem:

Theorem 4.3 (Mixed error bounds for artificial neural networks)**.** *Consider a general neural network composed of p layers whose output can be expressed as follows*

$$y = \phi_p(A_p\phi_{p-1}(A_{p-1}\dots A_2\phi_1(A_1x)\dots)),$$

with $A_i \in \mathbb{R}^{n_i\times n_{i-1}}$, for $i = 1,\dots,p$ and $x \in \mathbb{R}^{n_0}$, when y is evaluated in floating-point arithmetic and the computations occurring during the matrix–vector product generate rounding errors that satisfy Model 1.2, the computed result $\widehat{y}$ satisfies

$$\widehat{y} = \phi_p((A_p + \Delta A_p)\phi_{p-1}((A_{p-1} + \Delta A_{p-1})\dots(A_2 + \Delta A_2)\phi_1((A_1 + \Delta A_1)x)\dots))$$

with

$$|\Delta A_i| \leq (\widetilde{\gamma}_{n_{i-1}}(\lambda) + \frac{\ell_{\phi_i}u}{\kappa_{\phi_i}(A_i\widehat{y}_{i-1})})|A_i|,\ \ i = 1,\dots,p,$$

with probability at least

$$Q(\lambda, \sum_{i=1}^{p} n_in_{i-1}),$$

where

$$Q(\lambda, n) = 1 - n(1 - P(\lambda)).$$

Proof. The proof to obtain the bounds is almost identical to the work of section 4.1.3, replacing Lemma 1.2 by Theorem 4.2. For a given layer i the bound holds with probability at least $Q(\lambda, n_in_{i-1})$, by the same logic as in section 1.3.3.2. Therefore, the bound fails to hold for a given i with probability at most $1 - Q(\lambda, n_in_{i-1})$, where Q is defined as in section 1.3.3.1, hence it fails to hold for at least one layer i with probability at most $\sum_{i=1}^{p}(1 - Q(\lambda, n_in_{i-1})))$. This means that the bound holds for any layer with probability at least

$$1 - \left(\sum_{i=1}^{p} n_in_{i-1}(1 - P(\lambda))\right) = 1 - (1 - P(\lambda))\sum_{i=1}^{p} n_in_{i-1} = Q(\lambda, \sum_{i=1}^{p} n_in_{i-1}).$$

□

This result, which combines a probabilistic approach for errors arising from the matrix–vector product and a deterministic approach for the accumulation of these errors with those of the activation function, logically yields a bound reflecting the approach. Indeed, we obtain a bound that adds to the usual bound on the matrix–vector product, $\widetilde{\gamma}_n(\lambda)$, a second term which reflects the addition of the errors coming from the activation function, ℓu, which are amplified or not depending on the value of the condition number.

Compared to the deterministic approach of Theorem 4.1, this approach only adds the assumptions of the standard probabilistic Model 1.2 for the computations of the matrix–vector product.

4.2.2 Probabilistic activation function's rounding error

In order to fully use the potential improvement of probabilistic error analysis, we want to adapt the proof of Theorem 4.6 from Connolly, Higham, and Mary [23] to the particular case of a product of rounding errors of the form of equation (4.3). The following proofs will thus be adaptations based on the proof from Theorem 4.6 of [23], which has been introduced in section 1.3.3 as Theorem 1.5.

The result of a neural network layer followed by an activation function results in a product of rounding error terms of the form

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}\left(1+\frac{\delta_{n+1}}{K_\phi(\widehat{c})}\right). \tag{4.4}$$

Where $\widehat{c}$ is the computed output of a matrix–vector product and $\delta_1, \ldots, \delta_n$ its associated rounding errors. To integrate rounding errors arising from the activation function into our analysis, we will need to make assumptions about these errors.

In the following lemma we will make the assumption that the rounding errors $\delta_1, \ldots, \delta_n$ are random variables, then $\widehat{c}$, which is a function of these rounding errors, will be a random variable. Then the value of $\widehat{c}$ will fluctuate with respect to the values of the rounding errors. We will assume that in this set of values there exists $\zeta > 0$ such that $\kappa_\phi(\widehat{c}) \geq \zeta$.

For the sake of readability, let us note for the remainder of this section $K_\phi = K_\phi(\widehat{c})$ and $\kappa_\phi = |K_\phi(\widehat{c})| = \kappa_\phi(\widehat{c})$.

Lemma 4.4. *Let $\delta_1, \ldots, \delta_{n+1}$ be random variables of mean zero with $|\delta_k| \leq u$ for all $k = 1, \ldots, n$ and $|\delta_{n+1}| \leq \ell u$, such that $\mathbb{E}(\delta_{k+1} \mid \delta_1, \ldots, \delta_k) = \mathbb{E}(\delta_{k+1}) = 0$ for $k = 1, \ldots, n$. Let K_ϕ be a function of $\delta_1, \ldots, \delta_n$ and assume that there exists $\zeta > 0$*

such that $\kappa_\phi \geq \zeta$. *Let* $E_k = \sum_{i=1}^{k} \rho_i \delta_i$ *for* $k = 1, \ldots, n$, $E_0 = 0$ *and*

$$E_{n+1} = \sum_{i=1}^{n} \rho_i \delta_i + \rho_{n+1} \frac{\delta_{n+1}}{K_\phi},$$

where $\rho_i = \pm 1$, $i = 1, \ldots, n+1$. *Then for any constant* $\lambda > 0$,

$$|E_{n+1}| \leq \lambda \sqrt{n + \frac{\ell^2}{\zeta^2}} u$$

holds with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

Proof. Since $|\delta_k| \leq u$ for $k = 1, \ldots, n$, we have $|E_k| \leq ku$, and since $|\delta_{n+1}| \leq \ell u$ and $\zeta \leq |K_\phi| = \kappa_\phi$, we have

$$|E_{n+1}| \leq (n + \frac{\ell}{\zeta})u.$$

Hence $\mathbb{E}(|E_k|) < \infty$ for all $k = 1, \ldots, n+1$. Moreover, for $k = 1, \ldots, n-1$

$$\mathbb{E}(E_{k+1} \mid E_1, \ldots, E_k) = E_k + \rho_{k+1}\mathbb{E}(\delta_{k+1} \mid \delta_1, \ldots, \delta_k) = E_k.$$

We also have

$$\mathbb{E}(E_{n+1} \mid E_1, \ldots, E_n) = E_n + \rho_{n+1}\mathbb{E}(\frac{\delta_{n+1}}{K_\phi} \mid \delta_1, \ldots, \delta_n).$$

Since K_ϕ depends only on $\delta_1, \ldots, \delta_n$, K_ϕ is then fixed when $\delta_1, \ldots, \delta_n$ are fixed, therefore

$$\mathbb{E}(\frac{\delta_{n+1}}{K_\phi} \mid \delta_1, \ldots, \delta_n) = \frac{\mathbb{E}(\delta_{n+1} \mid \delta_1, \ldots, \delta_n)}{K_\phi} = 0,$$

which implies that $\mathbb{E}(E_{n+1} \mid E_1, \ldots, E_n) = E_n$ and hence $E_0, \ldots, E_{n+1}$ is a martingale. Moreover, $|E_{k+1} - E_k| \leq u$ for $k = 1, \ldots, n-1$ and

$$|E_{n+1} - E_n| \leq \frac{\ell u}{\zeta}.$$

By the Azuma–Hoeffding inequality, given by Lemma 1.4, we therefore have for any $\lambda > 0$,

$$\Pr\left(|E_{n+1} - E_0| \geq \lambda \left(nu^2 + \frac{\ell^2 u^2}{\zeta^2}\right)^{\frac{1}{2}}\right) \leq 2\exp\left(\frac{-\lambda^2}{2}\right)$$

which means that

$$\Pr\left(|E_{n+1}| \geq \lambda\sqrt{n+\frac{\ell^2}{\zeta^2}}u\right) \leq 2\exp\left(\frac{-\lambda^2}{2}\right)$$

and concludes the proof. □

We are then ready to state our main result:

Theorem 4.5 (Probabilistic error bound for activation function)**.** *Let $\delta_1, \ldots, \delta_{n+1}$ be random variables of mean zero with $|\delta_k| \leq u$ for all $k = 1, \ldots, n$ and $|\delta_{n+1}| \leq \ell u$ such that $\mathbb{E}(\delta_{k+1} \mid \delta_1, \ldots, \delta_k) = \mathbb{E}(\delta_{k+1}) = 0$ for $k = 1, \ldots, n$. Let K_ϕ be a function of $\delta_1, \ldots, \delta_n$ and assume that there exists $\zeta > 0$ such that $\kappa_\phi \geq \zeta$. Then for $\rho_i = \pm 1,\ i = 1, \ldots, n$ and any constant $\lambda > 0$,*

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}(1+\frac{\delta_{n+1}}{K_\phi})^{\rho_{n+1}} = 1+\theta_n,\ |\theta_n| \leq \lambda\sqrt{n+\frac{\ell^2}{\zeta^2}}u + O(u^2),$$

holds with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

Proof. Let $E_k = \sum_{i=1}^{k} \rho_i\delta_i$ for $k = 1, \ldots, n$, $E_0 = 0$ and

$$E_{n+1} = \sum_{i=1}^{k} \rho_i\delta_i + \rho_{n+1}\frac{\delta_{n+1}}{K_\phi}.$$

From Lemma 4.4 we know that

$$|E_{n+1}| \leq \lambda\sqrt{n+\frac{\ell^2}{\zeta^2}}u,$$

holds with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right).$$

We now will use the bound we found for E_{n+1} to bound the product of rounding error terms of equation (4.4). By taking the logarithm of this product we have

$$\log(\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}(1+\frac{\delta_{n+1}}{K_\phi})^{\rho_{n+1}}) = \sum_{i=1}^{n}\rho_i\log(1+\delta_i) + \rho_{n+1}\log(1+\frac{\delta_{n+1}}{K_\phi}).$$

Using the Taylor expansion of $\log(1+\delta_i)$, since $|\delta_i| \leq u < 1$, we have

$$\log(1+\delta_i) = \sum_{k=1}^{+\infty} (-1)^{k+1} \frac{\delta_i^k}{k}.$$

We can therefore bound $\log(1+\delta_i)$ since

$$-\delta_i - \sum_{k=2}^{+\infty} \frac{\delta_i^k}{k} \leq \log(1+\delta_i) \leq \delta_i + \sum_{k=2}^{+\infty} \frac{\delta_i^k}{k}.$$

These bounds can be further weakened to

$$-\delta_i - \sum_{k=2}^{+\infty} \delta_i^k \leq \log(1+\delta_i) \leq \delta_i + \sum_{k=2}^{+\infty} \delta_i^k$$

and then, taking the closed form of these geometric series, we have

$$-\delta_i - \frac{|\delta_i|^2}{1-|\delta_i|} \leq \log(1+\delta_i) \leq \delta_i + \frac{|\delta_i|^2}{1-|\delta_i|}, \tag{4.5}$$

which then implies, since $|\delta_i| \leq u$, that

$$-\delta_i - \frac{u^2}{1-u} \leq \log(1+\delta_i) \leq \delta_i + \frac{u^2}{1-u}.$$

For $\rho_i = \pm 1$ we hence have

$$-\rho_i\delta_i - \frac{u^2}{1-u} \leq \rho_i \log(1+\delta_i) \leq \rho_i\delta_i + \frac{u^2}{1-u}$$

which, by adding the inequalities for $i = 1, \ldots, n$, then means

$$-E_n - \frac{nu^2}{1-u} \leq \sum_{i=1}^{n} \rho_i \log(1+\delta_i) \leq E_n + \frac{nu^2}{1-u}. \tag{4.6}$$

At this point equation (4.6) enables us to obtain bounds on the error terms coming from the n first rounding errors, we now need to incorporate the error term that comes from the activation function. Assuming that

$$\frac{\ell u}{\zeta} < 1, \tag{4.7}$$

we have

$$\log(1+\frac{\delta_{n+1}}{K_\phi}) = \sum_{k=1}^{+\infty}(-1)^{k+1}\frac{\delta_{n+1}^k}{kK_\phi^k}$$

and hence, as in equation (4.5), we get

$$-\frac{\delta_{n+1}}{K_\phi} - \frac{|\frac{\delta_{n+1}}{K_\phi}|^2}{1-|\frac{\delta_{n+1}}{K_\phi}|} \leq \log(1+\frac{\delta_{n+1}}{K_\phi}) \leq \frac{\delta_{n+1}}{K_\phi} + \frac{|\frac{\delta_{n+1}}{K_\phi}|^2}{1-|\frac{\delta_{n+1}}{K_\phi}|}$$

which implies

$$-\rho_{n+1}\frac{\delta_{n+1}}{K_\phi} - \frac{\left(\frac{\ell u}{\zeta}\right)^2}{1-\frac{\ell u}{\zeta}} \leq \rho_{n+1}\log(1+\frac{\delta_{n+1}}{K_\phi}) \leq \rho_{n+1}\frac{\delta_{n+1}}{K_\phi} + \frac{\left(\frac{\ell u}{\zeta}\right)^2}{1-\frac{\ell u}{\zeta}}. \tag{4.8}$$

Adding inequalities from equation (4.6) and equation (4.8) we get

$$\begin{aligned}
&-E_{n+1} - \frac{nu^2}{1-u} - \frac{\left(\frac{\ell u}{\zeta}\right)^2}{1-\frac{\ell u}{\zeta}} \\
\leq &\log(\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}(1+\frac{\delta_{n+1}}{K_\phi})^{\rho_{n+1}}) \\
\leq &E_{n+1} + \frac{nu^2}{1-u} + \frac{\left(\frac{\ell u}{\zeta}\right)^2}{1-\frac{\ell u}{\zeta}}.
\end{aligned}$$

Then using the bound we computed for E_{n+1} from Lemma 4.4 we can weaken the inequality to obtain

$$\begin{aligned}
&-\lambda\sqrt{n+\frac{\ell^2}{\zeta^2}}u - \frac{nu^2}{1-u} - \frac{\left(\frac{\ell u}{\zeta}\right)^2}{1-\frac{\ell u}{\zeta}} \\
\leq &\log(\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}(1+\frac{\delta_{n+1}}{K_\phi})^{\rho_{n+1}}) \\
\leq &\lambda\sqrt{n+\frac{\ell^2}{\zeta^2}}u + \frac{nu^2}{1-u} + \frac{\left(\frac{\ell u}{\zeta}\right)^2}{1-\frac{\ell u}{\zeta}}
\end{aligned}$$

which holds with probability at least

$$1-2\exp\left(\frac{-\lambda^2}{2}\right).$$

We now will exponentiate the previous inequality, knowing that

$$e^t \leq 1 + \frac{t}{1-t}, \ t \in [0, 1[$$

and using the Taylor expansion of

$$\frac{t}{1-t}$$

we obtain the final bound on the rounding error term

$$1 - (\lambda\sqrt{n + \frac{\ell^2}{\zeta^2}}u + O(u^2)) \leq \prod_{i=1}^{n}(1+\delta_i)^{\rho_i}(1 + \frac{\delta_{n+1}}{K_\phi})^{\rho_{n+1}} \leq 1 + \lambda\sqrt{n + \frac{\ell^2}{\zeta^2}}u + O(u^2).$$

Hence, we can state that

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}(1 + \frac{\delta_{n+1}}{K_\phi})^{\rho_{n+1}} = 1 + \theta_n, \ |\theta_n| \leq \lambda\sqrt{n + \frac{\ell^2}{\zeta^2}}u + O(u^2)$$

holds with probability at least

$$1 - 2\exp\left(\frac{-\lambda^2}{2}\right),$$

which concludes the proof. □

We recall that by a deterministic approach in section 4.1.2 we were previously able to bound the backward error for a layer of neural networks as follows

$$\widehat{y} = \phi((A + \Delta A)x), \ |\Delta A| \leq \gamma_{n+\ell/\kappa_\phi(Ax)}|A|.$$

Theorem 4.5 shows that the deterministic bound

$$\gamma_{n+\ell/\kappa_\phi(Ax)} = (n + \frac{\ell}{\kappa_\phi(Ax)})u + O(u^2)$$

can easily be replaced by its probabilistic counterpart assuming mean independence of rounding errors, an assumption weaker than independence,

$$\widetilde{\gamma}_{n+\ell^2/\kappa_\phi(Ax)^2}(\lambda) = \lambda\sqrt{n + \frac{\ell^2}{\kappa_\phi(Ax)^2}}u + O(u^2),$$

with probability at least $Q(\lambda, mn)$. This means that the result obtained for the bounds on the backward error for a general neural network in Theorem 4.1 can be replaced by the following theorem:

Theorem 4.6 (Probabilistic error bounds for artificial neural networks)**.** *Consider a general neural network composed of p layers whose output can be expressed as follows*

$$y = \phi_p(A_p\phi_{p-1}(A_{p-1}\dots A_2\phi_1(A_1x)\dots)),$$

with $A_i \in \mathbb{R}^{n_i\times n_{i-1}}$, for $i = 1,\dots,p$ and $x \in \mathbb{R}^{n_0}$, when y is evaluated in floating-point arithmetic and the computations occurring during the layer application generate rounding errors that satisfy Model 1.2, the computed result $\widehat{y}$ satisfies

$$\widehat{y} = \phi_p((A_p+\Delta A_p)\phi_{p-1}((A_{p-1}+\Delta A_{p-1})\dots(A_2+\Delta A_2)\phi_1((A_1+\Delta A_1)x)\dots))$$

with

$$|\Delta A_i| \leq \widetilde{\gamma}_{n_{i-1}+\ell_{\phi_i}^2/\kappa_{\phi_i}(A_i\widehat{y}_{i-1})^2}(\lambda)|A_i|,\ \ i = 1,\dots,p,$$

with probability at least

$$Q(\lambda, \sum_{i=1}^{p} n_i n_{i-1}).$$

This final theorem concludes this section in which we established two different bounds, via a mixed and full probabilistic approach. We provide in Table 4.1 a brief summary of the bounds we obtained in this chapter. The probabilistic approach

	Theorem 4.1	Theorem 4.3	Theorem 4.6
Bound	$\gamma_{n+\ell_\phi/\kappa_\phi(Ax)}$	$\widetilde{\gamma}_n(\lambda) + \ell_\phi u/\kappa_\phi(Ax)$	$\widetilde{\gamma}_{n+\ell_\phi^2/\kappa_\phi(Ax)^2}(\lambda)$
Equivalent	$(n + \frac{\ell_\phi}{\kappa_\phi(Ax)})u$	$(\lambda\sqrt{n} + \frac{\ell_\phi}{\kappa_\phi(Ax)})u$	$\lambda\sqrt{n + \frac{\ell_\phi^2}{\kappa_\phi(Ax)^2}}u$

TABLE 4.1: Summary of the obtained bounds and their equivalents when $nu \ll 1$.

typically relaxes the deterministic bounds by reducing the impact of n consecutive rounding errors from nu to $\sqrt{n}u$. This is the observed difference between bounds of Theorem 4.1 and of Theorem 4.3 using probabilistic approach only on the matrix–vector product. The bound is then further refined by taking into account the ℓ_ϕ rounding errors introduced by the activation function. These ℓ_ϕ errors cannot be assumed to be mean independent between each other. Therefore, we only assume that the global rounding error δ_{n+1}, which encompasses all ℓ_ϕ intermediate errors, is of mean zero and is mean independent of the errors $\delta_1,\dots,\delta_n$ coming from the matrix–vector product, which means that $\mathbb{E}(\delta_{n+1} \mid \delta_1,\dots,\delta_n) = \mathbb{E}(\delta_{n+1})$. This leads to the bound of Theorem 4.6.

Compared to the mixed bounds of Theorem 4.3, this approach thus adds the assumption that there is mean independence between the errors made during matrix–vector computations and errors made during the computation of the activation function. Moreover, it also assumes that the condition number of the activation is strictly positive at the computation point. This assumption should not be a problem, since if the conditioning is zero, then the function is constant, in which case there are no rounding errors.

Note that in the case of linear activation functions where each $\kappa_{\phi_i} = 1$, the backward error is bounded by $\widetilde{\gamma}_{n_{i-1}+\ell_{\phi_i}^2}(\lambda)$, which corresponds to the addition of ℓ_{ϕ_i} rounding errors due to the application of the activation function to the n_{i-1} errors due to the first n_{i-1} basic operations. Probabilistic assumptions are made over the mean independence of the rounding errors for each operation. Moreover, since the ℓ_{ϕ_i} rounding errors made during the activation are not necessarily independent, we did not make any assumptions for these errors and the corresponding term is therefore squared in the bound.

These bounds show that the activation function's condition number is a quantity that can dictate whether the backward error is large or not. Indeed, the ℓ_{ϕ_i} rounding errors can be amplified or not by the condition number of ϕ_i.

We can interpret this in terms of adversarial attacks on the parameters. Since we have shown, in Chapter 3, a direct link between backward error and adversarial attacks on a neural network's parameters, we can expect these two quantities to behave similarly when the condition number varies. In section 2.2.2 we showed that a small condition number is typically linked with more robust neural networks. This means that when the condition number increases we expect adversarial attacks to be more efficient, as demonstrated by Beerens and Higham [6] and Savostianova et al. [100], and therefore have smaller norms. The bounds in Theorem 4.6 show that for a fixed $\widehat{y}$ and a given layer, if the condition number of the activation function increases then the backward error will decrease and therefore the perturbations needed on the layer's parameters will have smaller norm. This is consistent with the fact that in this case, it will be easier to find adversarial attacks with smaller norm.

In terms of rounding error coming from the use of reduced arithmetic precision, our bounds suggest that one should use higher precision for the activation functions. Indeed, it seems that the error terms coming from the activation function can be arbitrarily large depending on the condition number. For example, given a layer with tanh activation, if the result of the matrix–vector falls within the threshold region of the hyperbolic tangent function, the condition number tends to zero, and using low precision on the activation function would result in quickly pushing all outputs to one or minus one. In this context, it would be appropriate to use higher precision to avoid this phenomenon. This finding seems to align with results of Hubara et al. [61] which show that we expect neural networks with low precision to behave better when adding more precision to its activation functions than to its parameters.

4.3 Numerical experiments

In this section, we will seek to validate the bounds obtained in Chapter 4. To do so, we will use the formulas obtained in Chapter 2 to compute the backward error and the condition number for different neural networks. Once these quantities obtained, we can compare the computed value of the backward error with its different theoretical bounds. Additionally, we will compare the value of the forward error with its corresponding bound, which is obtained by multiplying the condition number by the bound on the backward error.

4.3.1 Experimental setup

The experimental setup is essentially the same as in section 2.5. The matrix–vector product computations have been implemented in C using loops so that the code corresponds to the floating-point Model 1.1 and to our analysis. If the matrix–vector product is implemented differently, using blocking [12] for example, we typically expect our bounds to be more pessimistic.

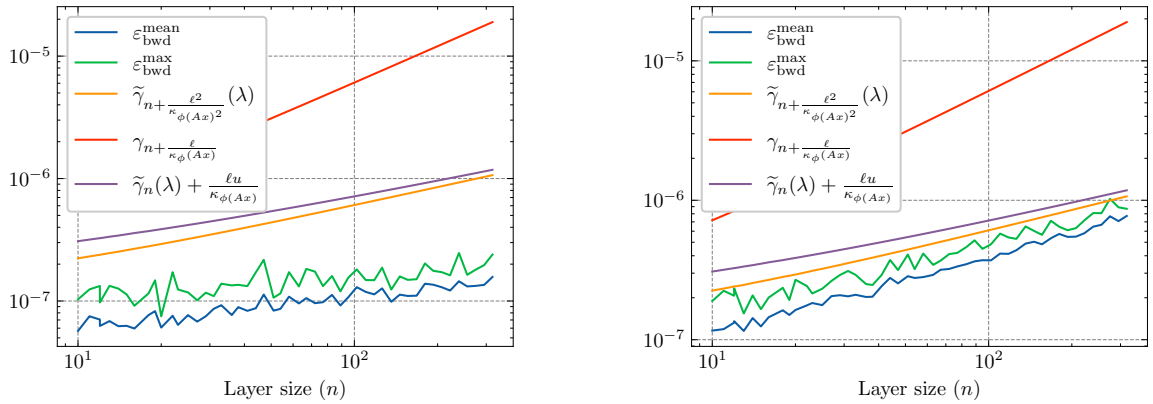
The experiments are first conducted with untrained neural networks, whose parameters and inputs are randomly generated in the same manner as in section 2.5. For each randomly generated neural network we run the experiment $N_{\text{test}} = 10$ times and then compute the average, $\varepsilon^{\text{mean}}$, and maximum, ε^{max}, errors, to compare them with their associated deterministic and probabilistic bounds. The probabilistic bounds are computed using $\lambda = 1$, as in the experiments of Higham and Mary [57]. Note that $\kappa_\phi(Ax)$ is a vector since it is a componentwise condition number, we therefore choose to take its smallest component in order to get the worst-case componentwise bound. Experiments on untrained neural networks enable us to better evaluate the sharpness of the bounds since it makes it easier to vary some parameters, such as the size of the layers. Moreover, it also allows for a more straightforward comparison of the impact of either the number of layers with a fixed number of neurons, or the number of neurons per layer with a fixed number of layers, on the computations performed by the networks.

In this chapter, three different bounds have been obtained for the backward error: a deterministic bound in Theorem 4.1, a mixed bound in Theorem 4.3, and a probabilistic bound in Theorem 4.6 as shown in Table 4.1.

The aim of the initial experiments is to compare these bounds to better understand the relevance of probabilistic approaches. To achieve this objective, we will revisit the experiments conducted in section 2.5 and incorporate the bounds associated with backward and forward errors. Next, these bounds are further analysed on both deeper random and trained networks.

4.3.2 Backward, forward errors and their bounds on random neural networks

To obtain results showing the evolution of backward error for different network sizes, we start the experiments with random neural networks generated using different distributions. This setup allows a better understanding of the effect of the neuron number and parameters distribution. Figure 4.1 presents the case of a single layer neural network with tanh activation and size n. It shows the evolution of the backward error and its corresponding theoretical bounds as a function of the number of neurons n. Similarly, the results for the forward error with the theoretical bounds are shown in Figure 4.2. We observe in Figure 4.1 that the probabilistic bound captures almost

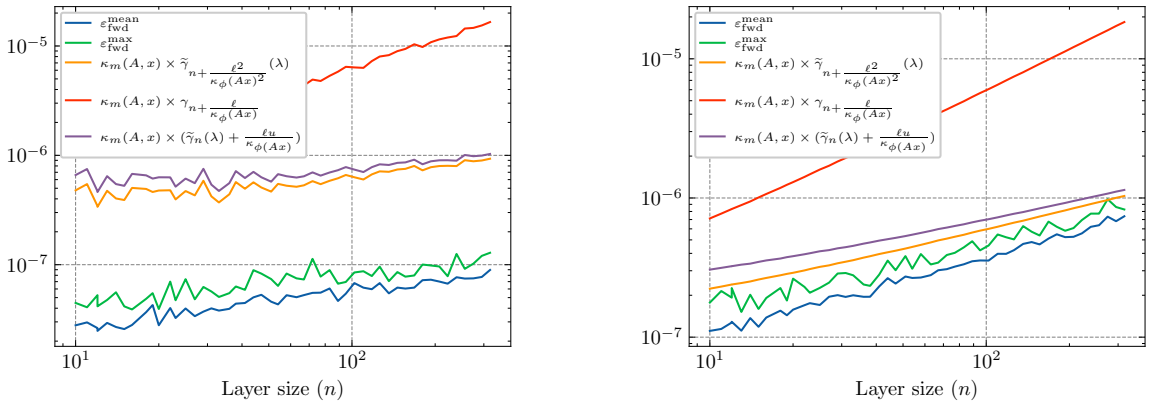


(a) Parameters and entries taken from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution.

(b) Parameters and entries taken from a $\mathcal{U}(0, \frac{1}{\sqrt{n}})$ distribution.

FIGURE 4.1: Backward error and its bounds for a single layer neural network of size n with random parameters and entries.

exactly the behaviour of the backward error for input data and parameters taken from a uniform positive distribution, in Figure 4.1b, while being more pessimistic for the Gaussian distribution in Figure 4.1a. Higham and Mary [58] indeed showed that when floating-point computations are performed between values that can be represented by random variables of zero mean, then the backward error does not increase with n but instead remains close to the unit roundoff. These results were therefore expected since, in this case, the neural network has a quasi linear behaviour and is therefore comparable to the case of a matrix–vector product [57]. Thus, in the absence of any assumptions regarding the distributions of the parameters and entries, the probabilistic bounds cannot be further enhanced in this case. Figures 4.1 and 4.2 clearly illustrate the differences between the various bounds obtained in this chapter. The gain between the deterministic approach and the two bounds using probabilistic assumptions is significant and is mostly due to the assumptions made



(a) Parameters and entries taken from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution.

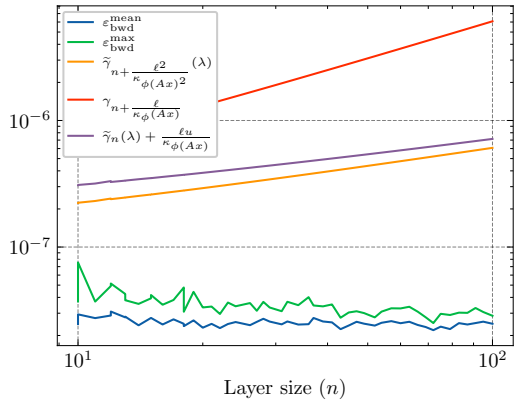
(b) Parameters and entries taken from a $\mathcal{U}(0, \frac{1}{\sqrt{n}})$ distribution.

FIGURE 4.2: Forward error and its bounds for a single layer neural network of size n with random parameters and entries.

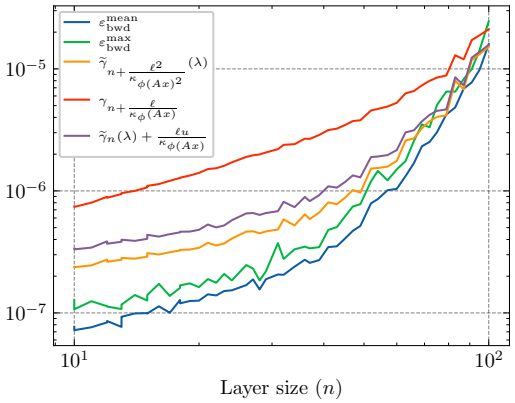
about the matrix–vector product operations. The full probabilistic approach allows for having slightly sharper bounds, especially when n decreases since, in that case, errors introduced by the matrix–vector product get smaller while errors introduced by the activation function do not depend on n and therefore remain the same.

Next, the experiments focus on the case of deeper fully connected neural networks with random initialization, and tanh as activation function. In this case, for a uniform positive distribution of input data, we start to get nonlinear behaviour. Indeed after few layers, or if the layer size is big enough, the accumulation of positive values will, after a tanh activation function, results in an output close to one. We therefore expect in this case the forward error to get smaller while the backward error will increase. Indeed, when the input values are large enough it requires higher perturbations to get an output smaller than one. Bounds on the backward error do not hold anymore in Figure 4.3b. This expectation is reasonable given that the model utilized to derive bounds on the backward error depends on the condition number not approaching zero. Therefore, as the condition number approaches zero, the model becomes invalid. However, it still indicates that the backward error is beginning to increase significantly, suggesting potentially unexpected behaviour in the neural network. Figure 4.4 provides confirmation on the behaviour of the forward error. Indeed, since we have an exact formula to compute this quantity, the fact that the bound on the forward error captures its behaviour confirms the validity of the computed bounds and condition number. In Figure 4.4b, we can observe a significant decrease in the forward error, which aligns with the observation that the output values are all pushed towards one by the tanh activation function.

Figure 4.5 provides further insights since in that case the last layer's activation is

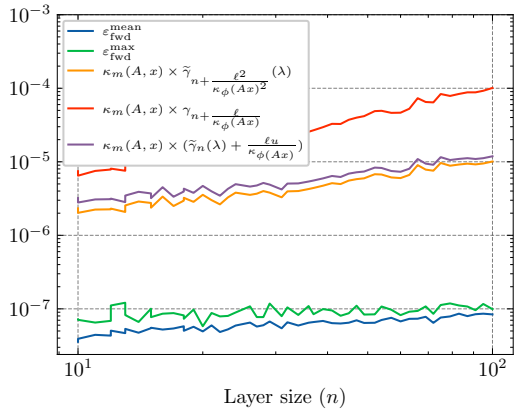


(a) Parameters and entries taken from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution.

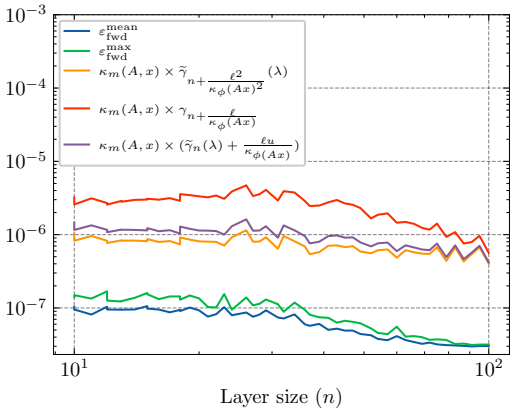


(b) Parameters and entries taken from a $\mathcal{U}(0, \frac{1}{\sqrt{n}})$ distribution.

FIGURE 4.3: Backward error and its bounds for a three-layer neural network, each layer is of size n with random parameters and entries.



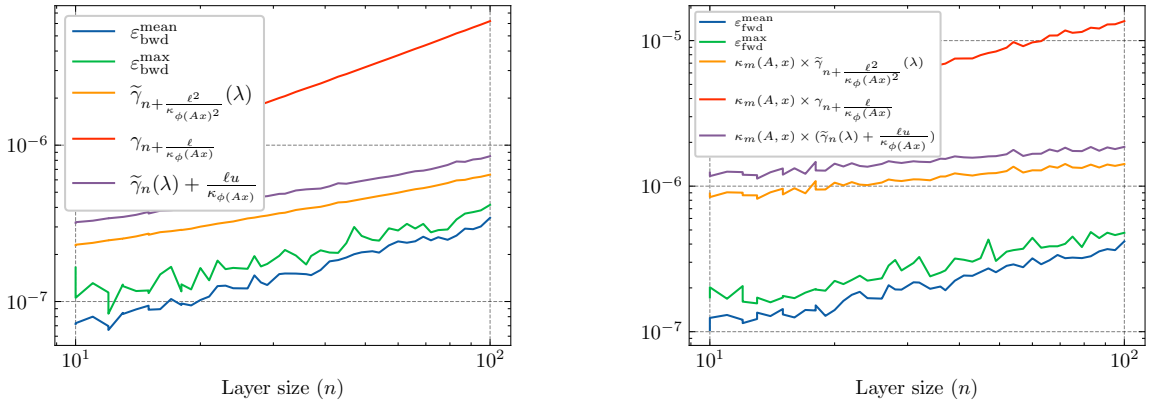
(a) Parameters and entries taken from a $\mathcal{N}(0, \frac{1}{\sqrt{n}})$ distribution.



(b) Parameters and entries taken from a $\mathcal{U}(0, \frac{1}{\sqrt{n}})$ distribution.

FIGURE 4.4: Forward error and its bounds for a three-layer neural network, each layer is of size n with random parameters and entries.

switched from a tanh to a ReLU, in which case we expect that the last layer will not cause the backward error to increase as much as in Figure 4.3b by pushing values to one.



(a) Backward error and its bounds.

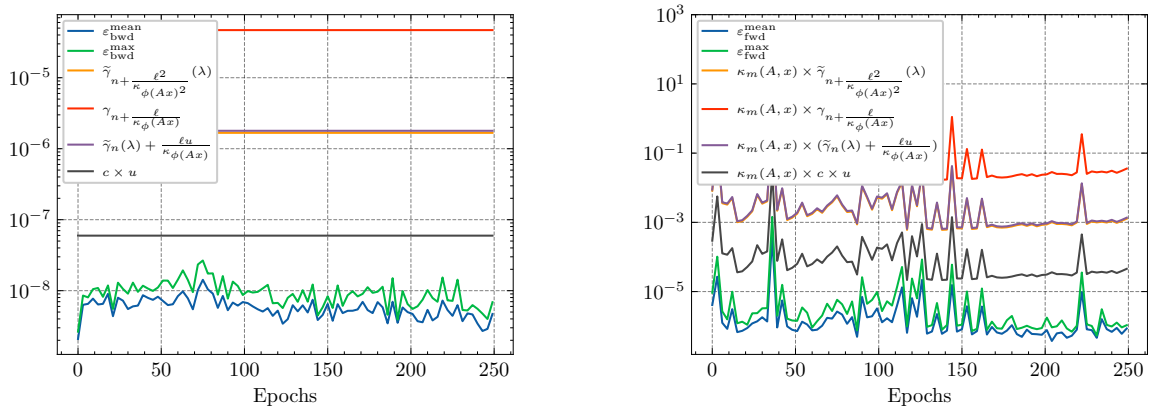
(b) Forward error and its bounds.

FIGURE 4.5: Three-layer ReLU-ended neural network, each layer is of size n with random parameters and entries taken from a $\mathcal{U}(0, \frac{1}{\sqrt{n}})$ distribution.

4.3.3 Backward, forward error and their bounds on trained neural networks

To better assess the robustness of the obtained bounds, we will apply our results to trained networks, as previously shown in Chapter 2. Our experiments consist in training a network and, for each training step, evaluating the backward error, forward error, conditioning, and associated bounds on $N_{\text{test}} = 10$ images from the testing dataset. Figure 4.6b shows the results for the fully connected network given on the right side of Table 2.1. We present in Figure 4.6a the backward error and its bounds and in Figure 4.6b the forward error bounded by the product of the backward error and the conditioning. Bounds on the backward error provide information that is in good agreement with the errors' behaviour. However, they seem to be rather pessimistic. Since these neural networks are trained using Glorot and Bengio [43] initialization, and that their parameters typically converge to a zero-mean Gaussian distribution, we are therefore in a case where results from Higham and Mary [58] are applicable. In Figure 4.6 we implement these results by integrating a bound that is directly proportional to u.

In order to estimate the constant c in these results, we use the Theorem 3.3 of Higham and Mary [58]. Since in our case parameters typically follow a normal distribution, let say $\mathcal{N}(0, \sigma)$, then with very high probability we know that these



(a) Backward error and its bounds.

(b) Forward error and its bounds.

FIGURE 4.6: Errors evolution and their bounds during the training of a small connected network on FashionMNIST.

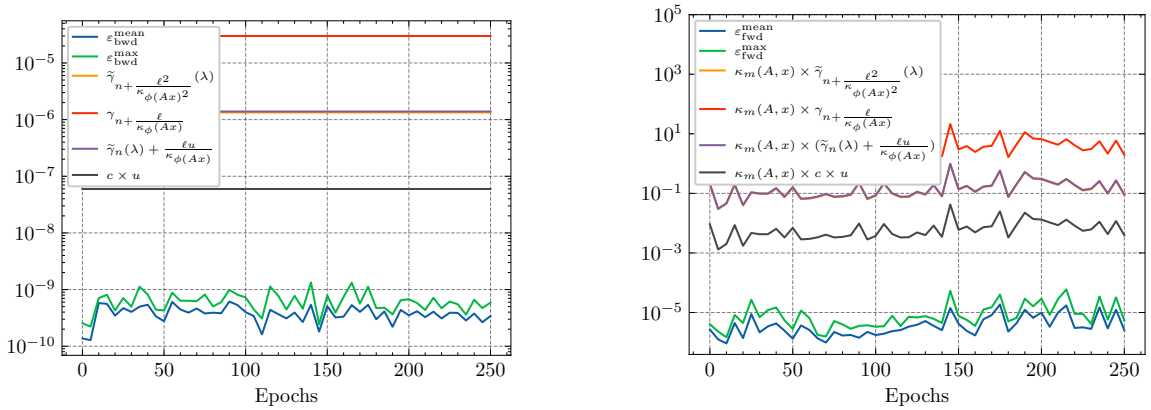
parameters are bounded by 2σ. Moreover, the absolute value of these parameters then follow the folded normal distribution of mean $\sqrt{2/\pi}\sigma$ [73]. This implies that c will typically be of the same order as

$$\frac{2\sigma}{\sqrt{2/\pi}\sigma} = \sqrt{2\pi}.$$

The resulting bounds are in that case much sharper with respect to both the computed forward and backward error. Figure 4.7 provides similar experiments for the convolutional neural network given on the left side of Table 2.1. These results show that the convolutional layer has an approximately 10 times lower backward error, compared to the fully connected network of Figure 4.6, while maintaining the same order of magnitude for the forward error. This means that the condition number of this neural network is approximately 10 times larger than for the fully connected one, suggesting that convolutional layers typically lead to networks that are more sensitive to perturbations on their input and/or parameters.

Note that the backward error bound involves the number n of columns of a given layer's weight matrix. In case of a convolutional layer, since the weight matrix is sparse, we can ignore a significant amount of columns, in fact a row comprises at most k non-zero components, with k the kernel size. Therefore, for convolutional layers, the number n is set to $k = 5$ in our case. This means that we expect convolutional layers to have much smaller backward errors than fully connected ones.

However, in Figure 4.7, since the neural network also comprises fully connected layers, the bounds do not benefit from these observations.



(a) Backward error and its bounds.

(b) Forward error and its bounds.

FIGURE 4.7: Errors evolution and their bounds during the training of a convolutional network on FashionMNIST.

4.4 Conclusion and discussion

Chapter 2 provided definitions and formulas to compute the backward error and condition number of artificial neural networks. In Chapter 4 we build on these results by establishing theoretical bounds for the backward error of artificial neural networks for the first time. This chapter therefore lays the foundations for a theoretical analysis of rounding errors in neural networks. This then allows us to construct bounds using more assumptions, to obtain results that are more in line with practice and valid with a certain probability. With this theoretical approach, we have therefore demonstrated how to obtain deterministic bounds in Theorem 4.1, which were then further refined using Model 1.2 into mixed bounds in Theorem 4.3, and probabilistic bounds in Theorem 4.6.

Probabilistic bounds are in overall good agreement with the computed backward error for both random and trained artificial neural networks. Furthermore, upon obtaining these bounds on the backward error, we were able to derive bounds on the forward error, as it is constrained by the product of the backward error and the condition number. These bounds on the forward error are also consistent with the computed results, confirming both our computation methods from Chapter 2 and our bounds from Chapter 4. We however showed that, with further assumptions on the neural network parameters, results from Higham and Mary [58] could be used to improve bounds. These results shown in Figure 4.6 and Figure 4.7 are preliminary and therefore need further investigation.

The derived bounds on the backward error allowed to provide insight on the identification of layers or building blocks that have the most impact on a neural network stability and sensitiveness to small perturbations such as rounding errors.

This allows for a better understanding and control on the design or choice of neural network architectures and training setup. Indeed, our results suggest that the use of zero-mean and rightly scaled initialization, such as proposed by Glorot and Bengio [43], in addition to maintaining the magnitudes of the activation functions from one layer to another during the propagation phase, can also lead to significant rounding errors reduction. Moreover, our results also suggest that while fully connected layers are fairly resilient to changes in precision, activation functions should be given more priority in terms of precision, particularly when applied to values that induce a condition number close to zero.

Conclusion

Summary of contributions

This manuscript has established three major axes of contributions. Firstly, in Chapter 1, quantities enabling the evaluation of a neural network's sensitivity to perturbations have been identified. The first contribution, provided in Chapter 2, has therefore been to establish direct formulas for computing these quantities, for different metrics, in the case of artificial neural networks. This allows extending the concept of backward error to neural networks, a well-known key concept in numerical linear algebra, and brings with it a whole set of analysis and tools that can now be applied to these networks. The first analysis that has been directly applied is related to conditioning, which links backward and forward errors. This quantity is typically linked to the sensitivity of a function. Preliminary experimental results and a theoretical connection established with the Lipschitz constant directly led us to consider training methods to generate networks with low conditioning, which are expected to be more robust and explainable. A code designed to compute the backward error and the condition number for generic neural networks was developed and allowed us to validate the obtained formulas.

After considering the backward error for generic networks, we naturally wondered how this concept could extend to classification networks whose outputs are classes and therefore integers. This led us, in Chapter 3, to take a deeper interest in adversarial attacks. These are small perturbations of neural networks inputs, causing a change in output class. Taking this problem from the perspective of backward error, we thus focused on adversarial attacks that would perturb both the inputs and parameters of the network. Therefore, in section 3.1, such an approach was proposed and was then extended in section 3.2 to produce competitive adversarial attacks on input data using a sequential quadratic programming-type algorithm. By creating an algorithm capable of generating attacks on the parameters of a neural network, we have also paved the way towards new quantization techniques, since these attacks make it possible to define the smallest perturbations on the parameters allowing a change of class.

Finally, in Chapter 4 we returned to the most well-known application of the backward error concept, the analysis of rounding errors in floating-point computations. We have therefore demonstrated how existing analyses in numerical linear algebra, notably from Higham [56], can be extended to artificial neural networks. This meant taking into account the nonlinearities coming from the activation functions as well as

the layering of neural networks to finally obtain deterministic bounds. These bounds served as a basis for then using the latest results from Connolly, Higham, and Mary [23] to derive mixed and probabilistic bounds. These bounds were then validated by building on the code obtained in Chapter 2 to generate bounds for randomly generated and trained convolutional and fully connected neural networks.

Perspectives

Many points remain open and deserve to be mentioned. Firstly, concerning the computation of backward errors for neural networks, this manuscript started from existing approaches in numerical linear algebra and thus employed a formulation of layers as matrix–vector products. This establishes a foundation upon which to work, both from the perspective of the computation of the error itself and from the rounding error analysis. Indeed, most existing layers base their computations upon such formulations, in other cases, the formulas and bounds obtained will need to be adapted, building upon existing work.

By bridging the gap between error analysis in numerical linear algebra and error analysis in neural networks, we believe that we can provide a set of tools that can help both better explain and understand choices that have been made empirically. Furthermore, in a context where the use of reduced and mixed precision becomes increasingly indispensable, the theoretical understanding of the impact of rounding errors on different elementary blocks of neural networks is becoming crucial. To this end, Chapter 4 provides tools to better mitigate the impact of reduced precision by identifying blocks that are more sensitive than others.

In the context of the increasingly widespread use of machine learning algorithms, domains requiring high reliability in their algorithmic computations have so far been hesitant to use machine learning algorithms, often because they are not easily explainable, partly due to adversarial attacks, but also because of the very high number of parameters and hence computations performed. Indeed, since these models can be highly sophisticated, the cost of using verification tools such as CADNA or FLUCTUAT, to ensure correct behaviour of algorithms when using finite precision, can be prohibitively large. Recent research of Graillat et al. [49], presenting the PROMISE algorithm, aims at using CADNA to produce mixed-precision quantization of neural networks that ensure a given accuracy. To do so, a brute-force algorithm would consist in exploring all possible precision configurations of the network's parameters. Since this approach is computationally infeasible, PROMISE makes use of a modified version of the delta debugging algorithm based on a divide-and-conquer method. By providing tools that can identify the most sensitive blocks of a given network, the rounding error analysis performed in Chapter 4 could help in guiding such software so that they only target robust layers and therefore gain significant computation time.

The final point that certainly deserves more attention is related to adversarial attacks. While this topic is highly active in the literature, it has so far mainly been focused in terms of attacks on the network inputs. However, as shown in Chapter 3, attacks on a neural network's parameters should not be neglected. Indeed, these attacks can be considered on one or multiple layers, as well as under different metrics so that they provide important insights into the sensitivity of a network layers.

Bibliography

[1] Nabih N. Abdelmalek. "Minimum L_∞ solution of underdetermined systems of linear equations". In: *Journal of Approximation Theory* 20.1 (1977), pp. 57–69.

[2] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen P. Boyd. "A Rewriting System for Convex Optimization Problems". In: *CoRR* abs/1709.04494 (2017).

[3] MOSEK ApS. *The MOSEK optimization toolbox for CVXPY manual. Version 10.0.* 2022. URL: https://docs.mosek.com/latest/faq/faq.html#cvxpy.

[4] Mario Arioli, James Weldon Demmel, and Iain S. Duff. "Solving sparse linear systems with sparse backward error". In: *SIAM Journal on Matrix Analysis and Applications* 10.2 (1989), pp. 165–190.

[5] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. "Scalable methods for 8-bit training of neural networks". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 5151–5159.

[6] Lucas Beerens and Desmond J. Higham. "Adversarial Ink: Componentwise Backward Error Attacks on Deep Learning". In: *CoRR* abs/2306.02918 (2023).

[7] David Bertoin, Jérôme Bolte, Sébastien Gerchinovitz, and Edouard Pauwels. "Numerical influence of ReLU'(0) on backpropagation". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan. 2021, pp. 468–479.

[8] Louis Béthune, Thibaut Boissin, Mathieu Serrurier, Franck Mamalet, Corentin Friedrich, and Alberto González-Sanz. "Pay attention to your loss : understanding misconceptions about Lipschitz neural networks". In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh. 2022.

[9] Théo Beuzeville, Pierre Boudier, Alfredo Buttari, Serge Gratton, Théo Mary, and Stéphane Pralet. "Adversarial attacks via backward error analysis". In: *hal-03296180* (Dec. 2021).

[10] Théo Beuzeville, Alfredo Buttari, Serge Gratton, Théo Mary, and Erkan Ulker. "Adversarial attacks via Sequential Quadratic Programming". In: *hal-03752184* (Aug. 2022).

[11] Théo Beuzeville, Alfredo Buttari, Serge Gratton, and Stéphane Pralet. "Method, computer program and device for quantizing a deep neural network". US 2023/0334301 (United States). Oct. 2023.

[12] Pierre Blanchard, Nicholas J. Higham, and Théo Mary. "A Class of Fast and Accurate Summation Algorithms". In: *SIAM J. Sci. Comput.* 42.3 (2020), A1541–A1557.

[13] Margaret A. Boden. *Artificial Intelligence: A Very Short Introduction*. Oxford University Press, Aug. 2018. ISBN: 9780199602919.

[14] Paul T. Boggs and Jon W. Tolle. "Sequential quadratic programming". In: *Acta numerica* 4 (1995), pp. 1–51.

[15] Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Found. Trends Mach. Learn.* 3.1 (2011), pp. 1–122.

[16] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. "Knitro: An integrated package for nonlinear optimization". In: *Large-scale nonlinear optimization* (2006), pp. 35–59.

[17] James A. Cadzow. "An Efficient Algorithmic Procedure for Obtaining a Minimum l_∞-Norm Solution to a System of Consistent Linear Equations". In: *SIAM Journal on Numerical Analysis* 11.6 (1974), pp. 1151–1165.

[18] Nicholas Carlini and David A. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 39–57.

[19] Françoise Chaitin-Chatelin and Valérie Frayssé. *Lectures on finite precision computations*. Software, environments, tools. SIAM, 1996. ISBN: 978-0-89871-358-9.

[20] Kumar Chellapilla and David B. Fogel. "Evolving neural networks to play checkers without relying on expert knowledge". In: *IEEE Trans. Neural Networks* 10.6 (1999), pp. 1382–1391.

[21] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[22] Dan C. Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. "Mitosis Detection in Breast Cancer Histology Images with Deep Neural Networks". In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2013 - 16th International Conference, Nagoya, Japan, September 22-26, 2013, Proceedings, Part II*. Ed. by Kensaku Mori, Ichiro Sakuma, Yoshinobu Sato, Christian Barillot, and Nassir Navab. Vol. 8150. Lecture Notes in Computer Science. Springer, 2013, pp. 411–418.

[23] Michael P. Connolly, Nicholas J. Higham, and Théo Mary. "Stochastic Rounding and Its Probabilistic Backward Error Analysis". In: *SIAM J. Sci. Comput.* 43.1 (2021), A566–A585.

[24] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett. 2015, pp. 3123–3131.

[25] Francesco Croce and Matthias Hein. "Minimally distorted Adversarial Examples with a Fast Adaptive Boundary Attack". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 2196–2205.

[26] Francesco Croce and Matthias Hein. "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 2206–2216.

[27] Francesco Croce et al. "RobustBench: a standardized adversarial robustness benchmark". In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*. Ed. by Joaquin Vanschoren and Sai-Kit Yeung. 2021.

[28] Matteo Croci, Massimiliano Fasi, Nicholas J Higham, Theo Mary, and Mantas Mikaitis. "Stochastic rounding: implementation, error analysis and applications". In: *Royal Society Open Science* 9.3 (2022), pp. 1–25.

[29] James Weldon Demmel. "On condition numbers and the distance to the nearest ill-posed problem". In: *Numerische Mathematik* 51 (1987), pp. 251–289.

[30] James Weldon Demmel and William Kahan. "Accurate Singular Values of Bidiagonal Matrices". In: *SIAM J. Sci. Comput.* 11.5 (1990), pp. 873–912.

[31] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE signal processing magazine* 29.6 (2012), pp. 141–142.

[32] Steven Diamond and Stephen P. Boyd. "CVXPY: A Python-Embedded Modeling Language for Convex Optimization". In: *J. Mach. Learn. Res.* 17 (2016), 83:1–83:5.

[33] Samuel Fuller Dodge and Lina J. Karam. "Understanding how image quality affects deep neural networks". In: *Eighth International Conference on Quality of Multimedia Experience, QoMEX 2016, Lisbon, Portugal, June 6-8, 2016*. IEEE, 2016, pp. 1–6.

[34] Samuel Fuller Dodge and Lina J. Karam. "A Study and Comparison of Human and Deep Learning Recognition Performance under Visual Distortions". In: *26th International Conference on Computer Communication and Networks, ICCCN 2017, Vancouver, BC, Canada, July 31 - Aug. 3, 2017*. IEEE, 2017, pp. 1–7.

[35] Ivan Dokmanic and Rémi Gribonval. "Beyond Moore-Penrose Part I: Generalized Inverses that Minimize Matrix Norms". In: *CoRR* abs/1706.08349 (2017).

[36] Alexander Domahidi, Eric Chu, and Stephen P. Boyd. "ECOS: An SOCP solver for embedded systems". In: *12th European Control Conference, ECC 2013, Zurich, Switzerland, July 17-19, 2013*. IEEE, 2013, pp. 3071–3076.

[37] John C. Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12 (2011), pp. 2121–2159.

[38] Adam Christopher Earle. "Minimum l_∞ norm solutions to finite dimensional algebraic underdetermined linear systems". PhD thesis. 2015.

[39] Pacôme Eberhart, Julien Brajard, Pierre Fortin, and Fabienne Jézéquel. "High performance numerical validation using stochastic arithmetic". In: *Reliable Computing* 21 (2015), pp. 35–52.

[40] Ivan Evtimov et al. "Robust Physical-World Attacks on Machine Learning Models". In: *CoRR* abs/1707.08945 (2017).

[41] Siddhant Garg, Adarsh Kumar, Vibhor Goel, and Yingyu Liang. "Can Adversarial Weight Perturbations Inject Neural Backdoors". In: *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. Ed. by Mathieu d'Aquin, Stefan Dietze, Claudia Hauff, Edward Curry, and Philippe Cudré-Mauroux. ACM, 2020, pp. 2029–2032.

[42] Philip E. Gill, Walter Murray, and Michael A. Saunders. "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization". In: *SIAM J. Optim.* 12.4 (2002), pp. 979–1006.

[43] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. Ed. by Yee Whye Teh and D. Mike Titterington. Vol. 9. JMLR Proceedings. JMLR.org, 2010, pp. 249–256.

[44] Israel Gohberg and Israel Koltracht. "Mixed, Componentwise, and Structured Condition Numbers". In: *SIAM J. Matrix Anal. Appl.* 14.3 (1993), pp. 688–704.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[46] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[47] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger. 2014, pp. 2672–2680.

[48] Eric Goubault. "Static Analysis by Abstract Interpretation of Numerical Programs and Systems, and FLUCTUAT". In: *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*. Ed. by Francesco Logozzo and Manuel Fähndrich. Vol. 7935. Lecture Notes in Computer Science. Springer, 2013, pp. 1–3.

[49] Stef Graillat, Fabienne Jézéquel, Romain Picot, François Févotte, and Bruno Lathuilière. "Auto-tuning for floating-point precision with Discrete Stochastic Arithmetic". In: *J. Comput. Sci.* 36 (2019).

[50] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep Learning with Limited Numerical Precision". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 1737–1746.

[51] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.

[52] Dan Hendrycks and Thomas G. Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[53] Catherine F. Higham and Desmond J. Higham. "Deep Learning: An Introduction for Applied Mathematicians". In: *SIAM Rev.* 61.4 (2019), pp. 860–891.

[54] Desmond J. Higham and Nicholas J. Higham. "Backward Error and Condition of Structured Linear Systems". In: *SIAM J. Matrix Anal. Appl.* 13.1 (1992), pp. 162–175.

[55] Nicholas J. Higham. "A survey of condition number estimation for triangular matrices". In: *Siam Review* 29.4 (1987), pp. 575–596.

[56] Nicholas J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 1996. ISBN: 978-0-89871-355-8.

[57] Nicholas J. Higham and Théo Mary. "A New Approach to Probabilistic Rounding Error Analysis". In: *SIAM J. Sci. Comput.* 41.5 (2019), A2815–A2835.

[58] Nicholas J. Higham and Théo Mary. "Sharper Probabilistic Backward Error Analysis for Basic Linear Algebra Kernels with Random Data". In: *SIAM J. Sci. Comput.* 42.5 (2020), A3427–A3446.

[59] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[60] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. "Learning with a Strong Adversary". In: *CoRR* abs/1511.03034 (2015).

[61] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations". In: *J. Mach. Learn. Res.* 18 (2017), 187:1–187:30.

[62] John E. Hutchinson. "Fractals and self similarity". In: *Indiana University Mathematics Journal* 30.5 (1981), pp. 713–747.

[63] "IEEE Standard for Binary Floating-Point Arithmetic". In: *ANSI/IEEE Std 754-1985* (1985), pp. 1–20.

[64] Arnault Ioualalen and Matthieu Martel. "Neural Network Precision Tuning". In: *Quantitative Evaluation of Systems, 16th International Conference, QEST 2019, Glasgow, UK, September 10-12, 2019, Proceedings*. Ed. by David Parker and Verena Wolf. Vol. 11785. Lecture Notes in Computer Science. Springer, 2019, pp. 129–143.

[65] Fabienne Jézéquel and Jean-Marie Chesneaux. "CADNA: a library for estimating round-off error propagation". In: *Computer Physics Communications* 178.12 (2008), pp. 933–955.

[66] Kai Jia and Martin C. Rinard. "Exploiting Verified Neural Networks via Floating Point Numerical Error". In: *Static Analysis - 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings*. Ed. by Cezara Dragoi, Suvam Mukherjee, and Kedar S. Namjoshi. Vol. 12913. Lecture Notes in Computer Science. Springer, 2021, pp. 191–205.

[67] Dhiraj D. Kalamkar et al. "A Study of BFLOAT16 for Deep Learning Training". In: *CoRR* abs/1905.12322 (2019).

[68] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 97–117.

[69] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[70] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[71] Miroslav Kubat. *An Introduction to Machine Learning, Third Edition*. Springer, 2021. ISBN: 978-3-030-81934-7.

[72] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[73] Fred C Leone, Lloyd S Nelson, and RB Nottingham. "The folded normal distribution". In: *Technometrics* 3.4 (1961), pp. 543–550.

[74] Ada Lovelace and Luigi Federico Menabrea. "Sketch of the analytical engine invented by Charles Babbage". In: *Scientific memoirs* 3 (1843), pp. 666–731.

[75] Yurii Il'ich Lyubich. "Conditionality in general computational problems". In: *Doklady Akademii Nauk*. Vol. 171. 4. Russian Academy of Sciences. 1966, pp. 791–793.

[76] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[77] Ravi Mangal, Aditya V. Nori, and Alessandro Orso. "Robustness of neural networks: a probabilistic and practical approach". In: *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019, Montreal, QC, Canada, May 29-31, 2019*. Ed. by Anita Sarma and Leonardo Murta. IEEE / ACM, 2019, pp. 93–96.

[78] Matthieu Martel. "An Overview of Semantics for the Validation of Numerical Programs". In: *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings*. Ed. by Radhia Cousot. Vol. 3385. Lecture Notes in Computer Science. Springer, 2005, pp. 59–77.

[79] Warren S. McCulloch and Walter H. Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Philosophy of Artificial Intelligence*. Ed. by Margaret A. Boden. Oxford readings in philosophy. Oxford University Press, 1990, pp. 22–39.

[80] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[81] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013).

[82] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[83] Eliakim H. Moore. "On the reciprocal of the general algebraic matrix". In: *Bulletin of the american mathematical society* 26 (1920), pp. 294–295.

[84] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 2574–2582.

[85] Jean-Michel Muller et al. *Handbook of Floating-Point Arithmetic (2nd Ed.)* Springer, 2018. ISBN: 978-3-319-76525-9.

[86] John Von Neumann and Herman H. Goldstine. "Numerical inverting of matrices of high order". In: *Bulletin of the American Mathematical Society* 53 (1947), pp. 1021–1099.

[87] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 1999. ISBN: 978-0-387-98793-4.

[88] NVIDIA. *CUDA C Programming Guide*. 2019. URL: https://docs.nvidia.com/cuda/archive/10.1/pdf/CUDA_C_Programming_Guide.pdf.

[89] Werner Oettli and William Prager. "Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides". In: *Numerische Mathematik* 6 (1964), pp. 405–409.

[90] Giacomo De Palma, Bobak Toussi Kiani, and Seth Lloyd. "Adversarial Robustness Guarantees for Random Deep Neural Networks". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 2522–2534.

[91] Roger Penrose. "A generalized inverse for matrices". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 51.3 (1955), pp. 406–413.

[92] 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". In: *J. Open Source Softw.* 5.53 (2020), p. 2607.

[93] John R. Rice. "A theory of condition". In: *SIAM Journal on Numerical Analysis* 3.2 (1966), pp. 287–310.

[94] J. L. Rigal and J. Gaches. "On the Compatibility of a Given Solution With the Data of a Linear System". In: *J. ACM* 14.3 (1967), pp. 543–548.

[95] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407.

[96] Jérôme Rony, Eric Granger, Marco Pedersoli, and Ismail Ben Ayed. "Augmented Lagrangian Adversarial Attacks". In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 7718–7727.

[97] Jérôme Rony, Luiz G. Hafemann, Luiz S. Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. "Decoupling Direction and Norm for Efficient Gradient-Based L2 Adversarial Attacks and Defenses". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 4322–4330.

[98] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[99] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach: the intelligent agent book*. Prentice Hall series in artificial intelligence. Prentice Hall, 1995. ISBN: 978-0-13-103805-9.

[100] Dayana Savostianova, Emanuele Zangrando, Gianluca Ceruti, and Francesco Tudisco. "Robust low-rank training via approximate orthonormal constraints". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine. 2023.

[101] Franco Scarselli and Ah Chung Tsoi. "Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results". In: *Neural Networks* 11.1 (1998), pp. 15–37.

[102] Robin M. Schmidt, Frank Schneider, and Philipp Hennig. "Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 9367–9376.

[103] Mathieu Serrurier, Franck Mamalet, Thomas Fel, Louis Béthune, and Thibaut Boissin. "On the explainable properties of 1-Lipschitz Neural Networks: An Optimal Transport Perspective". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine. 2023.

[104] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[105] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. "Fast and Effective Robustness Certification". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 10825–10836.

[106] Robert D. Skeel. "Scaling for Numerical Stability in Gaussian Elimination". In: *J. ACM* 26.3 (1979), pp. 494–526.

[107] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen P. Boyd. "OSQP: an operator splitting solver for quadratic programs". In: *Math. Program. Comput.* 12.4 (2020), pp. 637–672.

[108] Christian Szegedy et al. "Intriguing properties of neural networks". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014.

[109] Wai Sun Tang, Jun Wang, and Yangsheng Xu. "Infinity-norm torque minimization for redundant manipulators using a recurrent neural network". In: *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No. 99CH36304)*. Vol. 3. IEEE. 1999, pp. 2168–2173.

[110] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. "Neural network studies, 1. Comparison of overfitting and overtraining". In: *J. Chem. Inf. Comput. Sci.* 35.5 (1995), pp. 826–833.

[111] Tijmen Tieleman. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), p. 26.

[112] Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. "Evaluating Robustness of Neural Networks with Mixed Integer Programming". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[113] Lloyd N. Trefethen and David Bau. *Numerical linear algebra*. SIAM, 1997. ISBN: 978-0-89871-361-9.

[114] Alan M. Turing. "Rounding-off errors in matrix processes". In: *The Quarterly Journal of Mechanics and Applied Mathematics* 1.1 (1948), pp. 287–308.

[115] Ivan Yu. Tyukin, Desmond J. Higham, and Alexander N. Gorban. "On Adversarial Examples and Stealth Attacks in Artificial Intelligence Systems". In: *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–6.

[116] Ivan Yu. Tyukin, Desmond J. Higham, Eliyas Woldegeorgis, and Alexander N. Gorban. "The Feasibility and Inevitability of Stealth Attacks". In: *CoRR* abs/2106.13997 (2021).

[117] Aladin Virmaux and Kevin Scaman. "Lipschitz regularity of deep neural networks: analysis and efficient estimation". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 3839–3848.

[118] Guorong Wang, Yimin Wei, Sanzheng Qiao, Peng Lin, and Yuzhuo Chen. *Generalized inverses: theory and computations*. Vol. 53. Springer, 2018.

[119] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. "Training Deep Neural Networks with 8-bit Floating Point Numbers". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 7686–7695.

[120] James H. Wilkinson. "Rounding errors in algebraic processes". In: *Information Processing, Proceedings of the 1st International Conference on Information Processing, UNESCO, Paris 15-20 June 1959*. UNESCO (Paris), 1959, pp. 44–53.

[121] James H. Wilkinson. "Error analysis of floating-point computation". In: *Numerische Mathematik* 2 (1960), pp. 319–340.

[122] James H. Wilkinson. "Modern error analysis". In: *SIAM review* 13.4 (1971), pp. 548–568.

[123] Robert B. Wilson. "A simplicial algorithm for concave programming". In: *Ph. D. Dissertation, Graduate School of Bussiness Administration* (1963).

[124] Bichen Wu, Forrest N. Iandola, Peter H. Jin, and Kurt Keutzer. "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 446–454.

[125] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. "Adversarial Weight Perturbation Helps Robust Generalization". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020.

[126] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. "Training and Inference with Integers in Deep Neural Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[127] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *CoRR* abs/1708.07747 (2017).

[128] Chaofei Yang, Qing Wu, Hai Li, and Yiran Chen. "Generative Poisoning Attack Method Against Neural Networks". In: *CoRR* abs/1703.01340 (2017).

[129] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. "Theoretically Principled Trade-off between Robustness and Accuracy". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7472–7482.

[130] Yang Zhang, Hassan Foroosh, Philip David, and Boqing Gong. "CAMOU: Learning Physical Vehicle Camouflages to Adversarially Attack Detectors in the Wild". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Université de Toulouse

Titre : Analyse inverse des erreurs des réseaux de neurones artificiels avec applications aux calculs en virgule flottante et aux attaques adverses

Mots clés : Réseaux de neurones artificiels, Virgule flottante, Analyse d'erreur, Attaques adverses, Erreurs d'arrondi, Erreur inverse

Résumé : L'utilisation d'intelligences artificielles, dont les implémentations reposent souvent sur des réseaux de neurones artificiels, se démocratise maintenant dans une grande variété de tâches. En effet, ces modèles d'apprentissage profond produisent des résultats bien meilleurs que de nombreux algorithmes spécialisés précédemment utilisés et sont donc amenés à être déployés à grande échelle.
C'est dans ce contexte de développement très rapide que des problématiques liées au stockage de ces modèles émergent, car ils sont parfois très profonds et comprennent donc jusqu'à des milliards de paramètres, ainsi que des problématiques liées à leurs performances en termes de calcul tant d'un point de vue de précision que de coût en temps et en énergie. Pour toutes ces raisons, l'utilisation de précision réduite est de plus en plus indispensable.
D'autre part, il a été noté que les réseaux de neurones souffrent d'un manque d'interprétabilité, étant donné qu'ils sont souvent des modèles très profonds, entraînés sur de vastes quantités de données. Par conséquent, ils sont très sensibles aux perturbations qui peuvent toucher les données qu'ils traitent. Les attaques adverses en sont un exemple ; ces perturbations, souvent imperceptibles à l'œil humain, sont conçues pour tromper un réseau de neurones, le faisant échouer dans le traitement de ce qu'on appelle un exemple adverse.
Le but de cette thèse est donc de fournir des outils pour mieux comprendre, expliquer et prédire la sensibilité des réseaux de neurones artificiels à divers types de perturbations.
À cette fin, nous avons d'abord étendu à des réseaux de neurones artificiels certains concepts bien connus de l'algèbre linéaire numérique, tels que le conditionnement et l'erreur inverse. Nous avons donc établi des formules explicites permettant de calculer ces quantités et trouvé des moyens de les calculer lorsque nous ne pouvions pas obtenir de formule. Ces quantités permettent de mieux comprendre l'impact des perturbations sur une fonction mathématique ou un système, selon les variables qui sont perturbées ou non.
Nous avons ensuite utilisé cette analyse d'erreur inverse pour démontrer comment étendre le principe des attaques adverses au cas où, non seulement les données traitées par les réseaux sont perturbées, mais également leurs propres paramètres. Cela offre une nouvelle perspective sur la robustesse des réseaux neuronaux et permet, par exemple, de mieux contrôler la quantification des paramètres pour ensuite réduire la précision arithmétique utilisée et donc faciliter leur stockage. Nous avons ensuite amélioré cette approche, obtenue par l'analyse d'erreur inverse, pour développer des attaques sur les données des réseaux comparables à l'état de l'art.
Enfin, nous avons étendu les approches d'analyse d'erreurs d'arrondi, qui jusqu'à présent avaient été abordées d'un point de vue pratique ou vérifiées par des logiciels, dans les réseaux de neurones en fournissant une analyse théorique basée sur des travaux existants en algèbre linéaire numérique. Cette analyse permet d'obtenir des bornes sur les erreurs directes et inverses lors de l'utilisation d'arithmétiques flottantes. Ces bornes permettent à la fois d'assurer le bon fonctionnement des réseaux de neurones une fois entraînés, mais également de formuler des recommandations concernant les architectures et les méthodes d'entraînement afin d'améliorer la robustesse des réseaux de neurones.

Title: Backward error analysis of artificial neural networks with applications to floating-point computations and adversarial attacks

Key words: Artificial neural networks, Floating-point, Error analysis, Adversarial attacks, Rounding errors, Backward error

Abstract: The use of artificial intelligence, whose implementations are often based on artificial neural networks, is now becoming widespread across a wide variety of tasks. These deep learning models indeed yield much better results than many specialized algorithms previously used and are therefore being deployed on a large scale.
It is in this context of very rapid development that issues related to the storage of these models emerge, since they are sometimes very deep and therefore comprise up to billions of parameters, as well as issues related to their computational performance, both in terms of accuracy and time- and energy-related costs. For all these reasons, the use of reduced precision is increasingly being considered.
On the other hand, it has been noted that neural networks suffer from a lack of interpretability, given that they are often very deep models trained on vast amounts of data. Consequently, they are highly sensitive to small perturbations in the data they process. Adversarial attacks are an example of this; since these are perturbations often imperceptible to the human eye, constructed to deceive a neural network, causing it to fail in processing the so-called adversarial example.
The aim of this thesis is therefore to provide tools to better understand, explain, and predict the sensitivity of artificial neural networks to various types of perturbations.
To this end, we first extended to artificial neural networks some well-known concepts from numerical linear algebra, such as condition number and backward error. These quantities allow to better understand the impact of perturbations on a mathematical function or system, depending on which variables are perturbed or not.
We then use this backward error analysis to demonstrate how to extend the principle of adversarial attacks to the case where not only the data processed by the networks is perturbed but also their own parameters. This provides a new perspective on neural networks' robustness and allows, for example, to better control quantization to reduce the precision of their storage. We then improved this approach, obtained through backward error analysis, to develop attacks on network input comparable to state-of-the-art methods.
Finally, we extended approaches of round-off error analysis, which until now had been approached from a practical standpoint or verified by software, in neural networks by providing a theoretical analysis based on existing work in numerical linear algebra.
This analysis allows for obtaining bounds on forward and backward errors when using floating-point arithmetic. These bounds both ensure the proper functioning of neural networks once trained, and provide recommendations on architectures and training methods to enhance the robustness of neural networks.