Inertial and Second-order Optimization Algorithms for Training Neural Networks

Camille Castera

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Mathématiques Appliquées

Présentée et soutenue par :

M. CAMILLE CASTERA

le lundi 29 novembre 2021

Titre :

Inertial and Second-order Optimization Algorithms for Training Neural Networks

---

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (IRIT)

Directeurs de Thèse :

M. CEDRIC FEVOTTE

M. EDOUARD PAUWELS

Rapporteurs :

MME EMILIE CHOUZENOUX, INRIA

M. PASCAL BIANCHI, TELECOM PARISTECH

Membres du jury :

M. JALAL FADILI, ENSICAEN, Président

M. CEDRIC FEVOTTE, CNRS TOULOUSE, Membre

M. EDOUARD PAUWELS, UNIVERSITE TOULOUSE 3, Membre

M. JÉRÔME BOLTE, UNIVERSITE TOULOUSE 1, Membre

MME CLAIRE BOYER, UNIVERSITE SORBONNE, Membre

“Attendez il faut que ce soit vrai tout ce qu’on dit là ?...”

— Bohort, *Kaamelott*

Remerciements

J'espère par ces quelques lignes parvenir à exprimer amitié et gratitude à ceux qui ont partagé ces trois années avec moi et contribué à la réussite de cette thèse.

Pour commencer, merci évidemment à Cédric, Édouard et Jérôme, pour m'avoir tant apporté scientifiquement tout en m'apprenant à voler de mes propres ailes, ainsi que pour votre sympathie. Ce fut un grand plaisir de vous avoir comme encadrants, je garderai un excellent souvenir de nos nombreuses discussions.

Je tiens à remercier les membres de mon jury de soutenance. Merci à Emilie Chouzenoux et Pascal Bianchi d'avoir rapporté ma thèse, et à Jalal Fadili et Claire Boyer d'avoir été examinateurs. Je suis honoré et très reconnaissant que vous ayez accepté d'évaluer mes travaux.

Puisqu'on écrit rarement des remerciements, j'en profite pour exprimer ma gratitude aux professeurs que j'ai eus tout au long de ma scolarité. Merci aussi à Luciano, Frédéric et Pierre pour ces stages qui m'ont permis d'être prêt pour aborder la thèse. Une pensée aussi pour ceux qui ont contribué scientifiquement à cette thèse au travers de discussions, en particulier merci Emmanuel et Sixin pour vos conseils.

Cette thèse n'aurait pas été aussi joyeuse si je n'avais pas été si bien accueilli par les membres de l'équipe SC : Marie, Nicolas, Thomas, Cédric, Emmanuel, Henrique et Elsa. Merci aussi aux postdocs : Alberto, Dana, Sixin, Paul, Arthur, et Cassio et bien sûr à tous les doctorants qui nous ont montré la voie à suivre : Olivier, Louis, Adrien, Étienne, Mouna, Baha, Serdar et Maxime. Enfin, un grand merci à mes "contemporains" Claire, Vinicius, Asma, Florentin et évidemment Pierre-Hugo, on en aura passé du temps ensemble. Je penserai à vous du haut de mon parapente ! Un grand merci aussi au personnel administratif qui m'a accompagné dans les nombreuses démarches, en particulier Annabelle, Clémentine et Chloé.

À mes amis, grâce à qui je n'ai pas manqué d'occasions de relâcher la pression, les Lyonnais : Thomas, Baptiste, César, Alexis^2, Jules^2, Damien, JC, Marc, Agathe, et les Toulousains : Romain, Thomas, Guillaume, Rama, Etienne, Karl, Nathan, et toute la famille, ainsi que les Tchouss !

Merci à ma famille, ma sœur Gwenaëlle et mon frère Rémi (à ton tour maintenant). Pour mes parents, vous qui m'avez toujours donné de (très) longues explications à mes questions étant petit, voilà le résultat !

Enfin, à Soizick, merci d'avoir été à mes côtés pour cette aventure. Quelle histoire ça aussi...

Abstract

Neural network models became highly popular during the last decade due to their efficiency in various applications. These are very large parametric models whose parameters must be set for each specific task. This crucial process of choosing the parameters, known as training, is done using large datasets. Due to the large amount of data and the size of the neural networks, the training phase is very expensive in terms of computational time and resources.

From a mathematical point of view, training a neural network means solving a large-scale optimization problem. More specifically it involves the minimization of a sum of functions. The large-scale nature of the optimization problem highly restrains the types of algorithms available to minimize this sum of functions. In this context, standard algorithms almost exclusively rely on inexact gradients through the backpropagation method and mini-batch sub-sampling. As a result, first-order methods such as stochastic gradient descent (SGD) remain the most used ones to train neural networks. Additionally, the function to minimize is non-convex and possibly non-differentiable, resulting in limited convergence guarantees for these methods.

In this thesis, we focus on building new algorithms exploiting second-order information only by means of noisy first-order automatic differentiation. Starting from a dynamical system (an ordinary differential equation), we build INNA, an inertial and Newtonian algorithm. By analyzing together the dynamical system and INNA, we prove the convergence of the algorithm to the critical points of the function to minimize. Then, we show that the limit is actually a local minimum with overwhelming probability. Finally, we introduce Step-Tuned SGD that automatically adjusts the step-sizes of SGD. It does so by cleverly modifying the mini-batch sub-sampling allowing for an efficient discretization of second-order information. We prove the almost sure convergence of Step-Tuned SGD to critical points and provide rates of convergence. All the theoretical results are backed by promising numerical experiments on deep learning problems.

Résumé

Les modèles de réseaux de neurones sont devenus extrêmement répandus ces dernières années en raison de leur efficacité pour de nombreuses applications. Ce sont des modèles paramétriques de très grande dimension et dont les paramètres doivent être réglés spécifiquement pour chaque tâche. Cette procédure essentielle de réglage, connue sous le nom de phase d'entraînement, se fait à l'aide de grands jeux de données. En raison du nombre de données ainsi que de la taille des réseaux de neurones, l'entraînement s'avère extrêmement coûteux en temps de calcul et en ressources informatiques. D'un point de vue mathématique, l'entraînement se traduit sous la forme d'un problème d'optimisation en très grande dimension impliquant la minimisation d'une somme de fonctions. Les dimensions de ce problème d'optimisation limitent fortement les possibilités algorithmiques pour minimiser une telle fonction. Dans ce contexte, les algorithmes standards s'appuient presque exclusivement sur des approximations de gradients via la méthode de rétro-propagation et le sous-échantillonnage par mini-lots. Pour ces raisons, les méthodes du premier ordre de type gradient stochastique (SGD) restent les plus répandues pour résoudre ces problèmes. De plus, la fonction à minimiser est non-convexe et potentiellement non-différentiable, limitant ainsi grandement les garanties théoriques de ces méthodes.

Dans cette thèse, nous nous intéressons à construire de nouveaux algorithmes exploitant de l'information de second ordre tout en ne nécessitant que de l'information bruitée du premier ordre, calculée par différentiation automatique. Partant d'un système dynamique (une équation différentielle ordinaire), nous introduisons INNA, un algorithme inertiel et Newtonien. En analysant conjointement le système dynamique et l'algorithme, nous prouvons la convergence de ce dernier vers les points critiques de la fonction à minimiser. Nous montrons ensuite que cette convergence se fait en réalité vers des minimums locaux avec très grande probabilité. Enfin, nous introduisons Step-Tuned SGD, qui, à partir d'une utilisation astucieuse des mini-lots, discrétise efficacement de l'information du second-ordre afin de régler finement les pas de SGD. Nous prouvons la convergence presque sûre de SGD vers les points critiques et explicitons des vitesses de convergence. Tous les résultats s'accompagnent d'expériences encourageantes sur des problèmes d'apprentissage profond (ou deep learning).

Contents

Notations and Acronyms

Below are some notations used in this manuscript. These are general guidelines, yet, some notations are re-defined throughout the chapters.

Basic integers

M	Input size of a neural network
D	Output size of a neural network
P	Number of parameters of a neural network
L	Number of Layers of a neural network
N	Size of a training dataset

Basic variables

x	Input of a neural network
y	Ground-truth output
$\hat{y}$	Output of a neural network
θ	Parameter of a neural network
$(x_n, y_n)_{n\in\{1,\dots,N\}}$	Training dataset
$(\theta_k)_{k\in\mathbb{N}}$	Iterates of an algorithm
$(\xi_k)_{k\in\mathbb{N}}$	Noise produced by mini-batch sub-sampling
$\alpha, \beta, \mu, \nu, \delta, \varepsilon$	Hyper-parameters of algorithms
γ	Step-size
$(\gamma_k)_{k\in\mathbb{N}}$	Sequence of step-sizes

Fonctions

f	Neural network
f_{truth}	Ground-truth function
ℓ	Dissimilarity measure
$\mathcal{J}$	Loss function
$\mathcal{J}_{\mathsf{B}}$	Approximation of $\mathcal{J}$ on a mini-batch B
$(f_l)_{l\in\{1,\dots,L\}}$	Layers of a neural network
$(g_l)_{l\in\{1,\dots,L\}}$	Activation functions of a neural network
$(W_l)_{l\in\{1,\dots,L\}}$	Matrices of dense layers
g, h, F, G	Generic functions

Sets

Sets are usually denoted with sans-serif or blackboard fonts

$\mathbb{N}$	Integer numbers
$\mathbb{N}_{>0}$	Positive integer numbers
$\mathbb{R}$	Real numbers
$\mathbb{R}_+$	Non-negative real numbers
S	Set of stationary points
A	Random subset
B	Mini-batch
Ω	Open subset
K	Compact subset
$[a, b]$	Closed interval
(a, b)	Open interval

Operators

$\lvert \cdot \rvert$	Absolute value or cardinal of a set
$\lVert \cdot \rVert$	Norm
$\langle \cdot, \cdot \rangle$	Scalar product
dist	Distance
conv	Convex hull
A^{-1}	Inverse of the matrix A
A^T	Transposition of the matrix A
det	Determinant of a matrix

Calculus

$\dot{\theta}$	Time derivative of θ (when θ is a trajectory)
$\frac{\mathrm{d}}{\mathrm{d}t}$	Time derivative of a function
$\nabla \mathcal{J}$	Gradient of $\mathcal{J}$
$\nabla^2 \mathcal{J}$	Hessian matrix of $\mathcal{J}$
$\partial \mathcal{J}$	Clarke sub-differential of $\mathcal{J}$
$D\mathcal{J}$	New differential operator for non-smooth tame functions
Jac_G	Jacobian matrix of G
$\sum$	Summation symbol
$\int$	Integration symbol

Probabilistic Notations

$\mathbb{E}[X]$	Expectation of X
$\mathbb{E}[X \lvert Y]$	Expectation of X conditionally to Y

Acronyms

NN	*Neural network*
DNN	*Deep neural network*
DL	*Deep learning*
ReLU	*Rectified linear unit*
ODE	*Ordinary differential equation*
GD	*Gradient descent*
SGD	*Stochastic gradient descent*
HBF	*Heavy-ball with friction*
INNA	*Inertial Newton algorithm*
K-FAC	*Kronecker-factored approximate curvature*
IPAHD	*Inertial proximal algorithm with Hessian damping*
HNAG	*Hessian-driven Nesterov accelerated gradient*
BB	*Barzilai-Borwein*
KL	*Kurdyka-Łojasiewicz*
GV	*Gradient variation*
e.g.	*exempli gratia* (for example)
i.e.	*id est* (that is)
a.s.	*almost surely*
a.e.	*almost every/everywhere*

Chapter 1

Introduction: Challenges in Optimization for Training Neural Networks

Contents

Introduced decades ago (Hebb, 1949; Rosenblatt, 1958), neural networks remained ignored by a large part of the computer science community for a long time. They made a come back in the 1980s (LeCun et al., 1989) and became highly used only in the late 2000s (G. E. Hinton et al., 2006; Bengio, LeCun, et al., 2007). This upsurge of interest is mainly due to recent breakthroughs achieved by neural networks in a wide spectrum of applications including computer vision (Krizhevsky et al., 2012), linguistic (Mikolov et al., 2013), biology, physics, etc. From this variety of applications was born deep learning, the branch of machine learning that gathers the methods revolving around neural networks. The recent success of deep learning is due to the combination of several favorable factors: the progress of computers, the ease of accessing large datasets (Deng et al., 2009), open-source software (Rossum, 1995; Abadi et al., 2016; Paszke et al., 2019), etc., nonetheless, overcoming the issue of training neural networks significantly contributed to this success. Indeed, neural networks are large-scale parameterized functions that must be set up for each specific application. For determining the parameters, one must minimize a very large-scale sum of functions, the so-called loss function. This loss function is non-convex and may also be non-smooth. The resulting unconstrained optimization problem is tackled numerically, but the computational cost is extremely high due to the size of the problem. To this day, the training relies on the backpropagation (Rumelhart and G. E. Hinton, 1986), an optimized manner of computing the gradient of the loss function. As a result, most algorithms available to train neural networks belong to the class of stochastic first-order methods (Bottou and Bousquet, 2008), of which the stochastic gradient descent (Robbins and Monro, 1951) is the prototypical example.

While we are now able to train neural networks for many applications, the training remains among the biggest challenges in deep learning (Bottou et al., 2018) since it often takes long and requires significant investments in computational resources and energy. Taking an (extreme) example, the GPT-3 model (Brown et al., 2020) cost several millions of dollars to be trained just once. This shows how important it is to find more efficient algorithms for training neural networks. Most of the widely used algorithms are originally designed for convex optimization, e.g., ADAGRAD (J. Duchi et al., 2011), and few of them specifically exploit the non-convex landscapes of deep learning loss functions. Taking the specific aspects of deep learning into account would however help to design faster methods. Yet, this usually requires the computation of second-order derivatives, which is prohibitively computationally expensive due to the large-scale dimension of the optimization problem. Similarly,

while the convergence of algorithms is well understood for convex functions (Moulines and Bach, 2011), it reveals to be harder in deep learning. The convergence of several algorithms has been shown for smooth networks typically by using Lipschitz gradient continuity assumptions (Ghadimi and Lan, 2013; Li and Orabona, 2019), but guarantees are rare for non-smooth neural networks, with some exceptions (D. Davis et al., 2020). The present thesis aims to design new algorithms specifically exploiting second-order information to alleviate the training of neural networks and to provide convergence guarantees under assumptions that hold for most deep learning loss functions.

In this introduction chapter, we first present the basic principles of deep learning before detailing the problem of training neural networks from an optimization perspective. We then discuss more precisely the motivations for using second-order information, along with the associated challenges. Finally, we specify the main problems that we address in this thesis.

1.1 Deep learning: prerequisites

We first formulate deep learning as a supervised learning problem.

1.1.1 General supervised learning

A foundational interpretation of machine learning consists in assuming that there exists an unknown *ground-truth* function (or prediction function) f_{truth} representing for example a physical of biological reality. Following the formalism of Hastie et al. (2009, Chapter 2), this ground-truth is expressed through the relation

$$y = f_{\text{truth}}(x), \tag{1.1}$$

where $x \in \mathbb{R}^M$ and $y \in \mathbb{R}^D$ are respectively called *input* and *output* data, for some positive integers M and D. Supervised machine learning aims to estimate this unknown function f_{truth}. In other words f_{truth} represents a possibly very complex reality and one seeks a model that behaves like this function. Machine learning models are a way of building such a surrogate for f_{truth}, the models take the form of parameterized functions,

$$f : (x, \theta) \in \mathbb{R}^M \times \mathbb{R}^P \mapsto \hat{y} \in \mathbb{R}^D, \tag{1.2}$$

where again $x \in \mathbb{R}^M$ and $\hat{y} \in \mathbb{R}^D$. The variable $\theta \in \mathbb{R}^P$ for $P \in \mathbb{N}_{>0}$ represents the parameters of the model f. The objective is roughly to choose the parameter θ such that for a given input x, the output $\hat{y} = f(x, \theta)$ predicted by the model is as "close" as possible to the correct ground-truth output $y = f_{\text{truth}}(x)$.

Example. A famous example to illustrate this mathematical formalism is image classification where $x \in \mathbb{R}^M$ is an image made of M pixels. Assume that we want to build a model to state whether a cat is present in the image. The data y is thus a scalar associated to the presence of a cat in the image x: it takes the value 1 if there is a cat in the image and 0 if not. In this example, f_{truth} would be a function encoding everything necessary to tell whether a cat is present in the image. Determining exactly f_{truth} is out of reach, but it can be approximated by a machine learning model f whose parameters have to be well chosen to achieve the same task as f_{truth}.

1.1.2 Neural networks: concept

In this work, we will focus on a specific type of machine learning models: neural networks.

1.1.2.1 Mathematical representation

Neural networks (NNs) refers to a specific class of supervised machine learning models (1.2). In this thesis we consider *feedforward* NNs, in which the model f in (1.2) has a compositional structure. Let $L \in \mathbb{N}_{>0}$, the model reads,

$$f(\cdot, \theta) = f_L\left(f_{L-1}\left(\dots\left(f_1(\cdot, \theta^{(1)})\right), \theta^{(L-1)}\right), \theta^{(L)}\right), \tag{1.3}$$

where for each $l \in \{1, \dots, L\}$, $f(\cdot, \theta^{(l)})$ is a function from $\mathbb{R}^{D_{l-1}}$ to $\mathbb{R}^{D_l}$, parameterized by $\theta^{(l)} \in \mathbb{R}^{P_l}$, and where $\sum_{l=1}^{L} P_l = P$, (θ in (1.2) is the concatenation of all the $\theta^{(l)}$) and the dimensions $(D_l)_{l \in \{0,\dots,L\}}$ can be chosen freely as long as $D_0 = M$ and $D_L = D$. The functions $(f_l)_{l \in \{1,\dots,L\}}$ are called *layers* and L is thus the *number of layers*. The input $x \in \mathbb{R}^M$ is often called *input layer*, while the function f_L is the *output layer* and the other functions $(f_l)_{l \in \{1,\dots,L-1\}}$ are called *hidden layers*.

The prototypical example of these models is the Multi-Layer Perceptron (MLP) based on the perceptron introduced by Rosenblatt (1958). In this model, for each $l \in \{1, \dots, L\}$, the layer f_l consists in multiplying a matrix $W_l \in \mathbb{R}^{D_{l-1} \times D_l}$ with the previous layer, then adding a *bias* vector $b_l \in \mathbb{R}^{D_l}$, and finally composing the result with a function g_l, typically

applied coordinate-wise and called *activation function*. Given an input $x \in \mathbb{R}^M$, the output $\hat{y} \in \mathbb{R}^D$ of the NN is thus obtained by the relation,

$$\hat{y} = g_L\left(W_L\, g_{L-1}\left(W_{L-1}\, g_{L-2}\left(\dots g_2\left(W_2\, g_1(W_1 x + b_1) + b_2\right) \dots + b_{L-2}\right) + b_{L-1}\right) + b_L\right). \tag{1.4}$$

The type of layers described above are called *dense* layers. Variations of the MLP can be considered, in particular the dimensions of the matrices and vectors can be freely chosen as long as they agree with the rules of matrix-vector products, the same holds for the domain and codomain of the activation functions. Nowadays, the performances of computers allow building NNs with a large number L of layers. In the literature and in the sequel, the term Deep Neural Networks (DNNs) is thus often used, in comparison to "shallow" NNs which refer typically to networks with only one or two layers. In (1.4), the variable θ introduced in (1.2) corresponds to the vector concatenation of the coefficients of the matrices W_1 to W_L and the coefficients of the vectors b_1 to b_L. Thus P corresponds to the total number of coefficients of these matrices and vectors. We will specify in Section 1.1.4.1 how the coefficients of θ are determined.

1.1.2.2 Graphical representation

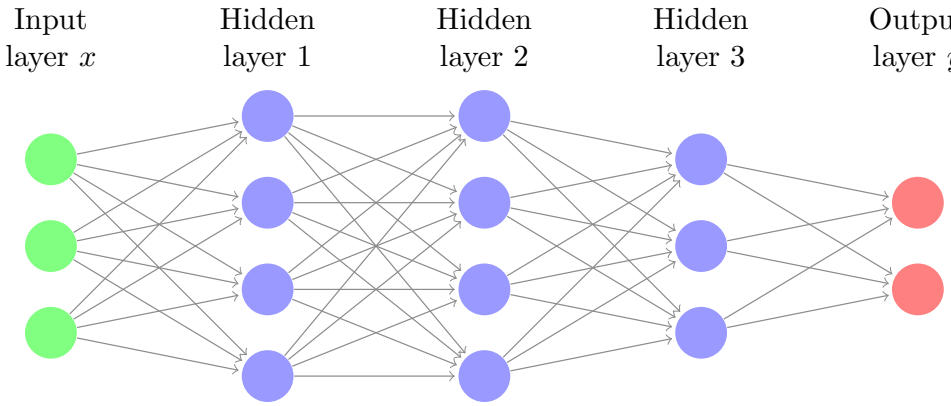


Figure 1.1: Graph representation of a MLP. Illustration made with the package of Mark (2017).

In addition to the mathematical definition, it is sometimes useful to represent DNNs as acyclic graphs like in Figure 1.1. This figure represents a MLP, the input layer is made of M green nodes, each one corresponding to a coefficient of the input x. The arrows between the

green nodes and the blue ones represent the operations yielding $h_1 = g_1(W_1 x + b_1)$. The next layer is then obtained by computing $h_2 = g_2(W_2 h_1 + b_2)$ and so on until obtaining the output $\hat{y}$. We will sometimes use this representation since some concepts are easier to understand using graphs, in particular backpropagation (see Section 1.3.3.1). The term "feedforward" also refers to the acyclic nature of the graph: going from the input on the left to the output on the right. More generally, the term *neural network* comes from the analogy once made between this graphical representation and the human brain (Hebb, 1949), although NNs remain actually essentially a class of non-linear machine learning models.

1.1.3 Some popular architectures and activation functions

Neural networks form a broad class of models that goes far beyond MLPs, here we briefly present some other NN architectures that use other types of layers than dense ones. We also discuss popular activation functions that will be used in the numerical experiments of this manuscript.

1.1.3.1 Some NN architectures

There is an ever-growing number of new NN architectures, here we introduce some of the most popular ones.

- **Residual neural networks.** Residual neural networks also known as ResNets (He et al., 2016) are a quite simple but efficient modification of classical MLPs. A common flaw of MLPs is that usually, if a MLP with L layers performs well for a given task, then adding one or more layers may reduce its performance. This means that the layers of the MLP fail to represent the identity mapping (otherwise additional layers could simply behave like identity layers, not affecting the output). Residual networks address this issue by adding mappings, called *skip-connections*, between a layer $l \in \{1, \dots, L-2\}$ and the layer $l+2$, skipping the layer $l+1$ as represented in Figure 1.2.
- **Convolutional Layers.** Convolutional Neural Networks (CNNs, Fukushima and Miyake 1982; LeCun et al. 1989) highly contributed to the success of NNs. These NNs, inspired from the signal processing theory, were originally designed to specifically exploit structures in images. To this aim, the input vector x is usually reshaped in a matrix form, corresponding to the pixels of the image. Then, several small *filters* (small matrices) are convolved with the input x, yielding several images (one per filter). One then typically reduces the dimensions of these images using *pooling*

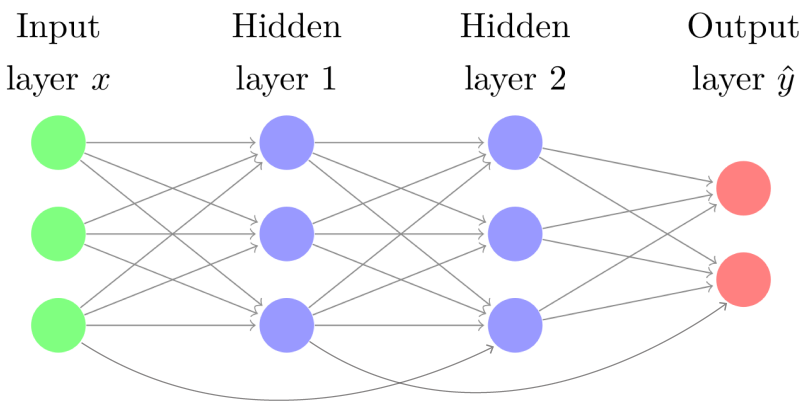


Figure 1.2: Graphical representation of a residual network. The arrows "jumping over layers" are called *skip-connections* and represent a mapping between a layer $l \in \{1, \dots, L-2\}$ and the layer $l+2$ "skipping" the layer $l+1$. Such arrows could be used to connect all the neurons of layers l and $l+2$ but we draw only one of them so that the figure can be red more easily.

activation functions applied to patches of pixels: e.g., keeping only the maximum or mean value among some neighbor pixels. This process forms a *convolutional layer* where the coefficients of each filter are parameters of the NN. CNNs are made of a composition of convolutional layers, and one dense layer is often used as last layer so that the dimensions of the output $\hat{y}$ of the network matches the dimension of the expected output y, as illustrated on Figure 1.3.

Note finally that the definitions above may intersect each other. Different types of layers (dense, residual, convolutional, etc.) can be composed to create variations of the NNs described above. For example, some popular NNs are both residual and convolutional (He et al., 2016).

1.1.3.2 Activation functions and universal approximation

We just detailed some typical NN architectures, we did not however discuss precisely the role of the activation functions $(g_l)_{l\in\{1,\dots,L\}}$. As previously stated, these are functions to apply to each layer of the NN, for example in (1.4), $g_1(W_1 x)$ denotes the operation of applying g_1 to each coefficient of the vector $W_1 x$. The definition is extremely general, almost any real-valued (or even complex-valued) function can be considered, although to ease the training (see Section 1.1.4), it is preferable to use "well-structured" ones. In addition to the pooling activation functions discussed with the CNNs, we now present some popular

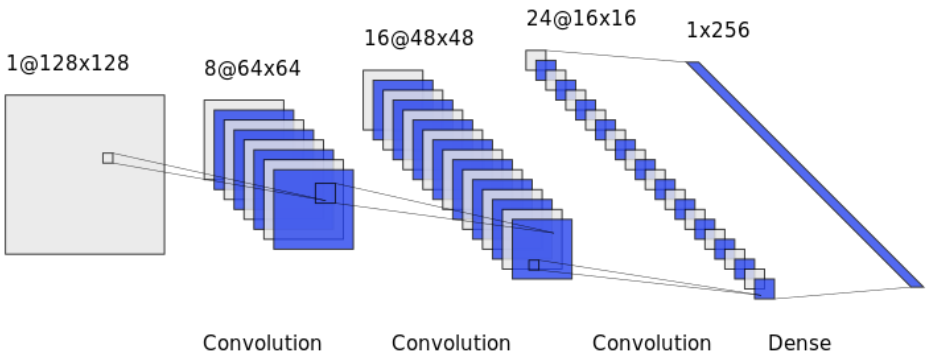


Figure 1.3: Scheme of a CNN. An input matrix x is transformed into several small matrices by a convolutional layer. These matrices are then again transformed into many smaller matrices with the same process. Eventually, a dense layer turns these matrices into a vector ouput $\hat{y}$. Here the parameter θ corresponds to the coefficients of the filters of the convolutional layers and the coefficients of the dense layer. Drawn with http://alexlenail.me/NN-SVG/LeNet.html.

univariate activation functions applied element-wise. The *sigmoid* (or logistic) function $t \in \mathbb{R} \mapsto 1/(1+e^{-t})$ is among the most popular choices, it is a smooth increasing function whose limits are 0 and 1 at $-\infty$ and $+\infty$ respectively. Nowadays, the *Rectified Linear Unit* (ReLU) activation function $t \in \mathbb{R} \mapsto \max(0, t)$ has also become widely used in fields like computer vision. Unlike the sigmoid, ReLU is unbounded as $t \to +\infty$. Both functions are displayed on Figure 1.4.

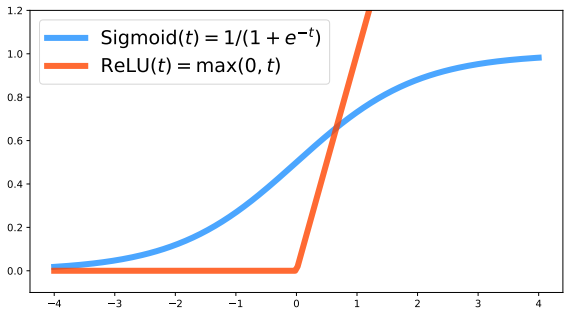


Figure 1.4: The two most popular activation functions: the sigmoid and the ReLU functions.

There is no general rule for choosing which activation functions to use, yet, a fundamental aspect is to choose non-linear ones. Indeed, if they were all linear then observe that, in (1.4), the network f would be linear (or affine) as well and hence may struggle to accurately

approximate some non-linear functions f_{truth}, see for example the XOR function (Goodfellow et al., 2016, Part II, Chapter 6). On the contrary, it has been proved that non-linear NNs are universal approximators. More precisely, several works among which Cybenko (1989), Hornik et al. (1989), and Leshno et al. (1993) proved that even simple two-layers NNs with Sigmoid or ReLU activation functions can approximate arbitrarily-well any measurable function on a compact set when the width of the hidden layer increases (i.e. when the dimensions of W_1 increase).

1.1.4 Supervised deep learning

So far we presented NNs as parameterized functions of $\theta \in \mathbb{R}^P$ used to approximate some ground-truth function, we did not however specify how the parameter $\theta \in \mathbb{R}^P$ was chosen. This question revolves around the aforementioned *training* procedure, which we first detail, before connecting it to the problem of *generalization*. All these aspects of building, training, and using NNs presented in this section form the *deep learning* (DL).

1.1.4.1 Training a neural network

The task of training DNNs is the main topic of this thesis. As we already explained, a NN f is a function of both an input $x \in \mathbb{R}^M$ and a parameter $\theta \in \mathbb{R}^P$ which produces an output $\hat{y} = f(x, \theta) \in \mathbb{R}^D$. Consider such a network f, made of activation functions and layers as presented in Section 1.1.2.

To approximate a ground-truth function f_{truth}, one needs a dataset $(x_n, y_n)_{n\in\{1,\dots,N\}}$ made of $N \in \mathbb{N}_{>0}$ pairs of elements $(x_n, y_n) \in \mathbb{R}^M \times \mathbb{R}^D$, linked for all $n \in \{1, \dots, N\}$ by the relation $y_n = f_{\text{truth}}(x_n) + \zeta_n$ where $\zeta_n \in \mathbb{R}^D$ represents a noise (for example induced by imprecise measurements of f_{truth}). This dataset is called *training set*. In the example of Section 1.1.1, one would need a collection of images $(x_n)_{n\in\{1,\dots,N\}}$, some that contain cats and others that do not, and scalars $(y_n)_{n\in\{1,\dots,N\}}$ indicating the presence or absence of cats in the images.

Using a dissimilarity measure $\ell : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$, one then compares for each $n \in \{1, \dots, N\}$ the output $\hat{y}_n = f(x_n, \theta)$ of the NN with the correct output y_n. The canonical example for ℓ is the squared Euclidean distance on $\mathbb{R}^D$: $\ell(y_n, \hat{y}_n) = \|y_n - \hat{y}_n\|_2^2$. This dissimilarity measure must be thought of as an error between the expected output and the actual output of the NN. The training becomes the problem of finding a $\theta \in \mathbb{R}^P$ which minimizes $\ell(y_n, \hat{y}_n)$ for each $n \in \{1, \dots, N\}$. The canonical way to do so is to find $\theta \in \mathbb{R}^P$ which minimizes the

sum of all the errors computed on the full dataset, namely,

$$\mathcal{J}(\theta) \stackrel{\text{def}}{=} \sum_{n=1}^{N} \ell\left(y_n, f(x_n, \theta)\right), \tag{1.5}$$

this function $\mathcal{J} : \mathbb{R}^P \to \mathbb{R}$ is called *loss function*. We will sometimes denote $\mathcal{J}_n$ the n-th term of the sum in (1.5), i.e., for all $\theta \in \mathbb{R}^P$, $\mathcal{J}_n(\theta) \stackrel{\text{def}}{=} \ell(f(x_n, \theta), y_n)$. Choosing the parameters of the NN is thus formulated as the following optimization problem,

$$\min_{\theta \in \mathbb{R}^P} \quad \mathcal{J}(\theta). \tag{1.6}$$

The *training* phase refers to the process of progressively minimizing $\mathcal{J}$ using iterative optimization algorithms. We will specify how this is done later in Section 1.3, the main objective of this work is to build new algorithm to tackle (1.6). Before that, we introduce the notion of generalization.

Remark 1.1. *We presented the training of DNNs as a supervised learning problem. DNNs can also be used for unsupervised tasks, i.e., using only input data but no output data. Unsupervised deep learning can often be formulated as an optimization problem similar to (1.6): for example in the case of auto-encoders (G. E. Hinton and Salakhutdinov, 2006), each term in (1.5) would be replaced by $\ell\left(x_n, f(x_n, \theta)\right)$, for $n \in \{1, \dots, N\}$. Despite the similarities, we will mostly consider supervised DL problems in this work. We refer to Hastie et al. (2009) for further discussions on supervised and unsupervised learning.*

1.1.4.2 Generalization performances

It is important to keep in mind that the objective when training a DNN on a dataset is to build a model to approximate f_{truth}. For this purpose we shall not necessarily minimize exactly the loss function $\mathcal{J}$ in (1.6). Indeed, the training set $(x_n, y_n)_{n \in \{1,\dots,N\}}$ only contains a finite number of (possibly imprecise) measurements of f_{truth} and hence there may exists many candidate functions $\tilde{f} \neq f_{\text{truth}}$ such that for any $n \in \{1, \dots, N\}$, $y_n = \tilde{f}(x_n) + \zeta_n$. For this reason, finding the lowest value of $\mathcal{J}$ may result in a network f making very accurate predictions on the training set but very poor predictions on other data not used during the training phase. This issue is called *overfitting*, we say that the model overfits the training set, whereas we would like it to *generalize* well on other data. We refer to Hastie et al. (2009, Chapter 7) for a general discussion on the problem of generalization.

To evaluate the generalization performance of the model the standard approach is to

use another dataset $(\tilde{x}_n, \tilde{y}_n)_{n\in\{1,\dots,\tilde{N}\}}$ of size $\tilde{N} \in \mathbb{N}_{>0}$ and called *test set*. This test set is assumed to be sampled from the same function f_{truth}: $\tilde{y}_n = f_{\text{truth}}(\tilde{x}_n) + \tilde{\zeta}_n$, where again $\tilde{\zeta}_n$ represents a perturbation. It is not to be used while training the DNN but only after to ensure that the model performs well on these data which were not used to choose the parameter θ of the NN. In practice one tracks a *test error* during the training to check that the generalization performance is improving. The test error is a metric computed on the test set $(\tilde{x}_n, \tilde{y}_n)_{n\in\{1,\dots,\tilde{N}\}}$ to ensure that the NN f performs well on this set. For example, one could simply consider the quantity

$$\sum_{n=1}^{\tilde{N}} \ell\left(\tilde{y}_n, f(\tilde{x}_n, \theta)\right), \tag{1.7}$$

which is similar to (1.5) but evaluated on the test set. We illustrate the question of generalization on the top row of Figure 1.5. The left figure shows a model *underfitting*: It achieves quite a large error both on training and test sets. In contrast, the model on the right figure performs very well on the training set but not at all on the test set, it overfits. The model on the middle figure is more balanced than the other two, it works quite well on the training set but the test error is lower than for other models. The emergence of the overfitting issue can also be observed during the training phase as illustrated on the bottom of Figure 1.5. When using algorithms in order to decrease $\mathcal{J}$ (during the training), a current phenomenon occurs: at first the error on the test set is quite large but progressively decreases; then after some time, this error stagnates or may even start to increase whereas the value of $\mathcal{J}$ keeps decreasing. This indicates that minimizing exactly $\mathcal{J}$ is not always the best choice in order for the model f to generalize well, it is sometimes preferable to stop the training before reaching the minimum of $\mathcal{J}$ (Bengio, 2012), this is called *early stopping*.

There exists several approaches to overcome or at least mitigate the overfitting issue. For example, assuming additional properties for f_{truth} such as sparsity, bounding the norm of θ in (1.6) or using techniques like weight-decay regularization (Krogh and Hertz, 1992). More generally, many factors impact generalization: f_{truth} and the training set, the structure of the DNN, the algorithm used to tackle (1.6), etc. The effect of these elements on generalization is an active topic of research. In this work we will focus on minimizing (1.5) and hope that this produces good generalization performances. To check that this is the case, we will at least track a test error while doing DNN training experiments.

Now that we presented the main aspects of DL we will focus on the training of DNNs and in particular on how one theoretically studies problems like (1.6) and tackles them numerically.

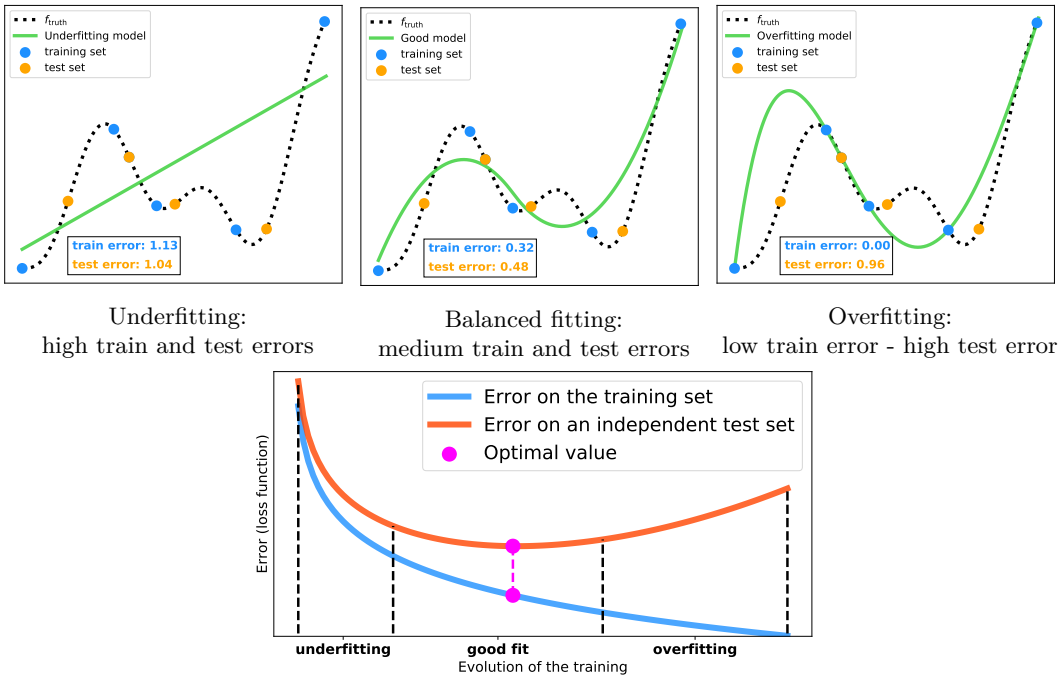


Figure 1.5: Top figures illustrate the issues of overfitting and underfitting on a toy regression problem. It is made on an example with synthethic data sampled from the function $f_{\text{truth}} : t \in [0, 2] \mapsto 0.5t^2 + 2\sin(3t)^{3/2}$. Bottom figure illustrates the same issue but from an optimization point of view. On this figure, the blue curve represents the evolution of a loss function $\mathcal{J}$ during the training of the NN. The red curve represents the evolution of a test error on a test set, different from the training set. In this example the test error increases at late stages, in some training experiments it simply stagnates.

1.2 Properties of the minimization problem

Let $N \in \mathbb{N}_{>0}$ and $P \in \mathbb{N}_{>0}$, throughout this chapter and according to Section 1.1, we denote $(x_n, y_n)_{n\in\{1,\dots,N\}}$ a training dataset, f is a NN and ℓ is a dissimilarity measure. We thus consider a loss function $\mathcal{J}$ that takes the form (1.5).

1.2.1 Basic definitions and assumptions

As explained in Section 1.1.4, training the DNN f amounts (at least approximately) to find a minimizer of $\mathcal{J}$, notion which we now define precisely.

Definition 1.1. *Let $g : \mathbb{R}^P \to \mathbb{R}$, we say that $\theta^\star \in \mathbb{R}^P$ is a local minimizer of g if there*

exists a neighborhood Ω *of* $\theta^\star$ *such that, for all* $\theta \in \Omega$,

$$g(\theta^\star) \leq g(\theta). \tag{1.8}$$

The definition of a *global minimizer* is very similar except that it requires (1.8) to hold for all $\theta \in \mathbb{R}^P$. The definition of local and global maximizers are the same with a reverse inequality in (1.8). It is worth precising that if $\theta^\star$ is a local minimizer, then $\mathcal{J}(\theta^\star)$ is called a local minimum. However, in the literature it is common to make an abuse of terminology by saying that "$\theta^\star$ is a local minimum", we sometimes do the same in the sequel. In order for the minimization problem to be well posed we make the following assumptions.

Assumption 1.1. *The loss function* $\mathcal{J}$ *is lower-bounded,*

$$-\infty < \inf_{\theta \in \mathbb{R}^P} \mathcal{J}(\theta), \tag{1.9}$$

and,

$$\operatorname{dom} \mathcal{J} \stackrel{\text{def}}{=} \left\{\theta \in \mathbb{R}^P | \mathcal{J}(\theta) < +\infty\right\} = \mathbb{R}^P. \tag{1.10}$$

Lower boundedness is a very natural assumption. Indeed, to minimize $\mathcal{J}$ we expect that it has at least a finite infimum. Actually, since $\mathcal{J}$ represents a sum of "errors" (see Equation 1.5), Assumption 1.1 is often guaranteed by construction. For example, $\mathcal{J}$ is greater or equal to 0 when ℓ is the squared Euclidean distance. The second part of the assumption states that $\mathcal{J}$ does not take infinite values on $\mathbb{R}^P$.

When $\mathcal{J}$ is smooth enough, we will denote $\nabla \mathcal{J}(\theta) \in \mathbb{R}^P$ and $\nabla^2 \mathcal{J}(\theta) \in \mathbb{R}^{P \times P}$ the gradient and the Hessian of $\mathcal{J}$ at $\theta \in \mathbb{R}^P$ respectively. For now, we assume that $\mathcal{J}$ is differentiable. To find a point where $\mathcal{J}$ achieves a minimum, a key strategy is to seek *critical points*, defined next.

Definition 1.2. *Let* $g : \mathbb{R}^P \to \mathbb{R}$ *be a differentiable function. A point* $\theta^\star \in \mathbb{R}^P$ *is called a critical point of* g *if* $\nabla g(\theta^\star) = 0$.

The interest of critical points lies in one of Fermat's theorems.

Theorem 1.2 (Fermat's theorem for critical points)**.** *Let* $g : \mathbb{R}^P \to \mathbb{R}$ *be a differentiable function and let* $\theta^\star \in \mathbb{R}^P$ *be a local minimum or a local maximum of* g*, then,* $\nabla g(\theta^\star) = 0$.

Therefore, a necessary condition for a point to be a minimum of a differentiable function is that it is a critical point of this function. We now recall the definition of a convex function.

Definition 1.3. *A function $g : \mathbb{R}^P \to \mathbb{R}$ is convex if for any $\theta_1, \theta_2 \in \mathbb{R}^P$ and $\forall \lambda \in (0,1)$,*

$$g(\lambda\theta_1 + (1-\lambda)\theta_2) \leq \lambda g(\theta_1) + (1-\lambda) g(\theta_2). \tag{1.11}$$

A function is strictly convex if (1.11) holds with strict inequality.

Convexity arises in many optimization problems and is thus a very active field of research, it has many benefits. This property can be used to design fast algorithms, and inequalities like (1.11) are useful to analyze the convergence of algorithms, see for example Nesterov, Y. (2003). In particular the necessary condition from Theorem 1.2 becomes sufficient for differentiable functions: under the convexity assumption any critical point of $\mathcal{J}$ is a global minimum. Thus, for smooth convex functions, solving (1.6) boils down to finding a point $\theta^\star \in \mathbb{R}^P$ such that $\nabla \mathcal{J}(\theta^\star) = 0$.

In practice, many widespread optimization algorithms are originally designed to minimize convex functions, some of them are used in DL (SGD, ADAGRAD, etc.). Yet, this may be problematic since DL loss functions are non-convex in general as we now explain.

1.2.2 Non-convex optimization

We now focus on the theoretical properties of the loss functions met in DL. In view of (1.5), one can see that the analytical properties of $\mathcal{J}$ depend on those of ℓ and the network f. Observe in particular that the loss function $\mathcal{J}$ is non-convex in general. Indeed, while the dissimilarity measure ℓ can be chosen to be smooth and convex, like in the example of Section 1.1.4.1, $(y, \hat{y}) \in \mathbb{R}^D \times \mathbb{R}^D \mapsto \|\hat{y} - y\|_2^2$, the structure of the network f in (1.4) makes $\mathcal{J}$ a non-convex function of $\mathbb{R}^P$ in general. The reason is twofold, first the composition of convex functions need not be convex.[1] In addition, the activation functions may be neither convex nor concave, see for example the sigmoid function presented on Figure 1.4.

For now we assume that $\mathcal{J}$ is differentiable. The lack of convexity makes the minimization of $\mathcal{J}$ harder since Fermat's condition becomes insufficient for finding a global minimum. Fermat's condition remains sufficient for some sub-classes of non-convex functions (e.g., quasi-convex functions), unfortunately, DL loss functions do not belong to any of these classes in general. As a result, the loss function $\mathcal{J}$ may have local minima that are not global, but also maxima or even critical points that are neither minima nor maxima. We illustrate this on Figure 1.6 where a non-convex function has various types of critical points

[1] Sufficient conditions for the composition of convex functions to be convex typically requires monotonicity assumptions.

whereas the convex one has a unique global minimum.

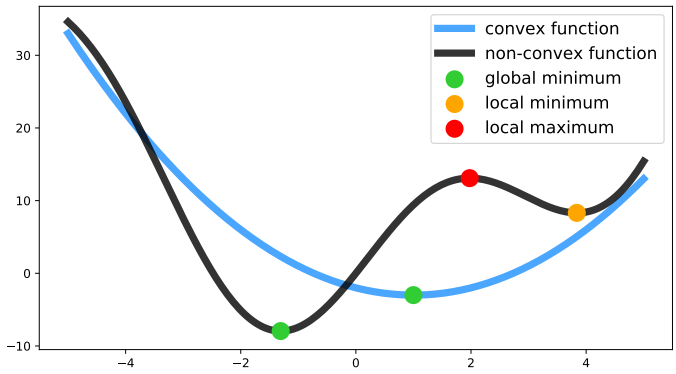


Figure 1.6: Illustration of several types of critical points of non-convex functions compared to convex ones. The function in blue is $t \in \mathbb{R} \mapsto t^2 - 2t - 2$ and the one in black is $t \in \mathbb{R} \mapsto t^2 + 10\sin(t)$.

The existence of spurious critical points complicates the theoretical analysis of the problem (1.6). However, practitioners seem to often tackle such problems "almost as if there were convex": they mostly use algorithms made to progressively decrease $\mathcal{J}$ but not specifically designed to avoid local minima or strict saddle points (critical points where the Hessian has negative eigenvalues). The main reason for this is that such methods produce satisfying results for many applications in practice: they often achieve a low value of $\mathcal{J}$ and yield good generalization performances. Explaining this phenomenon is an active topic of research, but some results suggests that the value of $\mathcal{J}$ at most local minima is likely to be small (close to the global minimum) for DNNs with "sufficiently" wide layers (Choromanska et al., 2015). Some works also argue that most critical points are actually strict saddle points and that the main challenge is to escape them quickly as they may significantly slow down the training (Dauphin et al., 2014).

There are many other aspects making general non-convex optimization harder than the special case of convex minimization. In particular, without convexity, (1.6) is difficult to tackle or even define for non-smooth loss functions, which again, may occur in DL as we now explain.

1.2.3 Non-smooth optimization

In addition to non-convexity, the loss functions in DL can also be non-differentiable. Unlike convexity, differentiability is preserved by composing functions, here the non-smoothness

comes from the use of some popular activation functions, with in particular the ReLU function (see Figure 1.4) which is non-differentiable at 0.

Despite the non-smoothness, DL loss functions have some structural properties. Indeed, they are usually made of a composition of linear functions (e.g., dense layers), piece-wise polynomial (ReLU, squared Euclidean distance), or exponentials (sigmoid activation) and logarithms (cross-entropy dissimilarity), etc. As a result, most of them are *tame* (D. Davis et al., 2020). This notion will be defined precisely in Chapter 2, but formally, it means that the graph of the loss functions can be split into a finite number of pieces, each of which can be described with inequalities involving polynomials, exponentials, etc. Thus, the study of tame functions can be split into pieces on which they behave "well". Additionally, while some loss functions are non-differentiable, their structures make them locally Lipschitz continuous in general.

Definition 1.4. *A function $g : \mathbb{R}^P \to \mathbb{R}$ is* locally Lipschitz continuous*, if for any $\theta \in \mathbb{R}^P$, there exists a neighborhood Ω of θ and a constant $L_\Omega > 0$ (depending on Ω) such that for any $\theta_1, \theta_2 \in \Omega$,*

$$|g(\theta_1) - g(\theta_2)| \leq L_\Omega \|\theta_1 - \theta_2\|, \tag{1.12}$$

where $\|\cdot\|$ is a given norm on $\mathbb{R}^P$, and L_Ω is referred to as a Lipschitz constant *of g on Ω. A function $g : \mathbb{R}^P \to \mathbb{R}^D$ is locally Lipschitz continuous if each of its coordinates is locally Lipschitz continuous. A function $g : \mathbb{R}^P \to \mathbb{R}$ is said to be (globally) Lipschitz continuous if there exists a Lipschitz constant such that (1.12) holds for any $\theta_1, \theta_2 \in \mathbb{R}^P$.*

Throughout this thesis we will always study NNs, dissimilarity measures and loss functions that are locally Lipschitz continuous. By Rademacher's theorem (Heinonen, 2005), locally Lipschitz continuous functions are differentiable almost everywhere on $\mathbb{R}^P$, hence their gradient is well-defined at almost any point of $\mathbb{R}^P$. The local Lipschitz continuity will play an important role in the convergence analysis of the new algorithms that we will introduce, even though this property itself is not sufficient in general to obtain convergence results for general non-smooth non-convex functions (Daniilidis and Drusvyatskiy, 2020; Rios-Zertuche, 2020).

While, DL loss functions are differentiable almost everywhere, non-smoothness cannot be ignored. Indeed, algorithms may encounter points of non-differentiability and such points may even be minima. The canonical example illustrating this is the function $t \in \mathbb{R} \mapsto |t|$. The latter is differentiable everywhere except at $t = 0$, but this point is exactly the unique global minimum. This function illustrates another hardship: the norm of the gradient is not a sign of closeness to critical points. Indeed, although the gradient of $|\cdot|$ is well-defined on

$\mathbb{R} \setminus \{0\}$, it is always equal to ± 1 and thus it never tends to 0 regardless how close we may be to the minimum $t = 0$. Of course similar issues also appear when considering second-order derivatives (see later in Section 1.4.1).

A standard approach to deal with non-smoothness in optimization is to introduce a notion of subgradient, generalizing the gradient for non-differentiable functions. For example, for a differentiable convex function $g : \mathbb{R}^P \to \mathbb{R}$ and for any $\theta \in \mathbb{R}^P$, the gradient $\nabla g(\theta)$ is the only element of $\mathbb{R}^P$ such that,

$$\forall\, \psi \in \mathbb{R}^P,\ g(\psi) \geq g(\theta) + \langle \nabla g(\theta), \psi - \theta \rangle, \tag{1.13}$$

see Nesterov, Y. (2003, Chapter 2). Therefore, the standard definition of the subdifferential of a convex but non-differentiable function g at $\theta \in \mathbb{R}^P$ is simply the set,

$$\left\{ v \in \mathbb{R}^P \middle| \forall\, \psi \in \mathbb{R}^P,\ g(\psi) \geq g(\theta) + \langle v, \psi - \theta \rangle \right\}, \tag{1.14}$$

and the elements of this set are called subgradients (see Rockafellar 1996). The subdifferential is thus a set-valued operator, and as we said, whenever g is differentiable at θ, this subdifferential simply reduces to $\{\nabla g(\theta)\}$. Unfortunately, this definition of the subdifferential only suits convex functions. For non-convex functions there may be no vector satisfying the right hand-side inequality in the set (1.14). For example, the function $t \in \mathbb{R} \mapsto -|t|$ is a simple example of a non-convex function for which the set (1.14) is empty at $t = 0$. In Chapter 2 we will address this issue using the Clarke subdifferential (Clarke, 1990) which is well-defined for locally Lipschitz continuous DL functions, but is however not fully compatible with calculus which generates other complications.

1.3 Large-scale optimization framework

As said at the beginning, DL had an irregular development. It was introduced quite a long time ago (McCulloch and Pitts, 1943; Rosenblatt, 1958), was further developed thirty years later (Rumelhart and G. E. Hinton, 1986; LeCun et al., 1989) but only became highly used in the early 2010s after producing state-of-the-art results on problems such as the ImageNet classification challenge (Deng et al., 2009) with the AlexNet DNN (Krizhevsky et al., 2012). The main reason for this late success originates from in the difficulty of training DNNs. Indeed, assuming for now that $\mathcal{J}$ is differentiable, in Section 1.2.1 we saw that seeking critical points was an appropriate strategy to find candidates for being minimizers. However, finding a closed-form solution $\theta^\star \in \mathbb{R}^P$ to the equation $\nabla \mathcal{J}(\theta^\star) = 0$

is not possible in general this is a system of P non-linear equations. Hence, one must resort to finding approximate solutions numerically, usually using iterative algorithms. We first present gradient descent, the most classical of these algorithms. Then, we specify the computational limitations that made the training of DNNs unpractical for a long time and finally introduce fundamental tools for overcoming these limitations.

1.3.1 Gradient descent

Gradient descent (GD) is the archetypal method for minimizing a function and is used in many applications. Its introduction goes back at least to Cauchy in 1847 according to Lemaréchal (2012). Consider an initialization $\theta_0 \in \mathbb{R}^P$ and a number $\gamma > 0$ called *step-size* (or *learning rate* in the DL literature), for all $k \in \mathbb{N}$ the algorithm simply reads,

$$\theta_{k+1} = \theta_k - \gamma \nabla \mathcal{J}(\theta_k). \tag{1.15}$$

GD follows the idea of "steepest descent": the vector $-\nabla \mathcal{J}(\theta_k)$ in (1.15) indicates the direction in which an infinitesimal variation of θ_k decreases $\mathcal{J}$ the most. Indeed, let $\theta \in \mathbb{R}^P$, $\gamma > 0$ small, $\|\cdot\|$ be a norm on $\mathbb{R}^P$ and let $d \in \mathbb{R}^P$ such that $\|d\| = 1$. By a Taylor expansion and the Cauchy-Schwarz inequality, it holds,

$$\begin{aligned} \mathcal{J}(\theta + \gamma d) &= \mathcal{J}(\theta) + \langle \nabla \mathcal{J}(\theta), \gamma d \rangle + o(\gamma) \\ &\geq \mathcal{J}(\theta) - \gamma \|\nabla \mathcal{J}(\theta)\| \, \|d\| + o(\gamma) = \mathcal{J}(\theta) - \gamma \|\nabla \mathcal{J}(\theta)\| + o(\gamma). \end{aligned} \tag{1.16}$$

Thus, observe that at first-order approximation, taking $d = -\nabla \mathcal{J}(\theta)/\|\nabla \mathcal{J}(\theta)\|$ is the optimal choice to minimize the left-hand side of (1.16), hence the term "steepest descent" (see Bertsekas 1999 for further details). This justification for using $-\nabla \mathcal{J}(\theta)$ as update direction is based on arguments that hold infinitesimally, they are not valid anymore when using too-large step-sizes $\gamma > 0$. However, accurately choosing the step-size $\gamma > 0$ is not easy in general as we will convey in Section 1.4.1. Despite its simplicity, GD can be very powerful, in particular remark that (1.15) stabilizes if and only if $\nabla \mathcal{J}(\theta_k) = 0$, i.e., if and only if one has found a critical point.

Apart from the choice of the step-size (which can be difficult), GD simply requires one to be able to evaluate the gradient $\nabla \mathcal{J}$ of the loss function. Even though this may seem to be a very mild requirement, computing gradients is computationally expensive in DL for reasons that we now make precise.

1.3.2 The high computational cost of deep learning

The main challenge for training DNNs is to deal with the computational cost of DL. It has two major causes.

1.3.2.1 DNNs are trained on large datasets

The ground-truth f_{truth} (from Section 1.1.1) that one wishes to approximate is possibly very complex. Thus, in order for NNs to generalize well (as discussed in Section 1.1.4.2) one must often train them using very large datasets. These days, such datasets can be accessed fairly easily, for example very large datasets of images (Deng et al., 2009) or texts (Marcus et al., 1993) are publicly available online. This large amount of data available is the reason for the famous term *big data*, it allows DNNs to achieve state-of-the-art performances on many tasks. While large datasets are used for generalization purposes, they make the training phase computationally expensive. Indeed, from a mathematical point of view, the size of the dataset is the number N of elements in the sum structure of $\mathcal{J}$ (see Formula 1.5), so, to evaluate $\mathcal{J}$ at $\theta \in \mathbb{R}^P$, one has to pass N data through the NN. This is true for the gradient $\nabla\mathcal{J}$ as well as explained next in Section 1.3.3.1.

1.3.2.2 DNNs have many parameters

Due to the training set being very large and the complexity of f_{truth}, DNNs may also need a very large number P of parameters to fit the training set well. This typically results in $P \geq 10^6$, but some DNNs even have billions of parameters (Brown et al., 2020). The resulting computational and storage costs (due both to the values of N and P) made DL unpractical for a long time. Nowadays, the ever-growing performances of computers allow one to process and store gigabytes of data within a reasonable time. A notable improvement is the use of graphics processing units (GPUs) in addition to the classical microprocessors (also known as CPUs) (Owens et al., 2008).

Although the technical limitations that we presented are partially addressed by the availability of large datasets and the performances of computers, there is a crucial tool that eases the training of DNNs: the *backpropagation* algorithm.

1.3.3 Differentiation of smooth DL loss functions

In this section we assume that the loss function $\mathcal{J}$, the dissimilarity measure ℓ and all the activation functions of the DNN considered are differentiable. As we said, training DNNs relies on iterative algorithms like GD which are based on the ability of evaluating $\nabla\mathcal{J}$, which is very expensive due to the large values of P and N. The *backpropagation* algorithm (Rumelhart and G. E. Hinton, 1986) is an efficient technique for differentiating NNs, it is the keystone of the training.

1.3.3.1 Backpropagation: concept

We begin with an example, let g_1, g_2, g_3 be three differentiable functions from $\mathbb{R}$ to $\mathbb{R}$. For any $x \in \mathbb{R}$, the intuitive way of computing $g_3 \circ g_2 \circ g_1(x)$ is to do successive compositions "forwardly": $h_1(x) = g_1(x)$, then $h_2(x) = g_2(h_1(x))$ and finally $h_3(x) = g_3(h_2(x))$. With these notations, assume that one wants to compute the effect of an infinitesimal variation of x on h_3. The chain rule states that since the functions g_1, g_2, g_3 are differentiable, then, $h_1 = g_1$, $h_2 = g_2 \circ g_1$ and $h_3 = g_3 \circ g_2 \circ g_1$ are differentiable functions of $x \in \mathbb{R}$ as well and for all $x \in \mathbb{R}$,

$$\frac{\mathrm{d}h_3}{\mathrm{d}x}(x) = \frac{\mathrm{d}h_3}{\mathrm{d}h_2}(x)\frac{\mathrm{d}h_2}{\mathrm{d}h_1}(x)\frac{\mathrm{d}h_1}{\mathrm{d}x}(x) = g_3'\left(g_2\left(g_1(x)\right)\right) g_2'\left(g_1(x)\right) g_1'(x). \tag{1.17}$$

Observe that to evaluate $\frac{\mathrm{d}h_3}{\mathrm{d}x}(x)$ we use twice the value of $h_1 = g_1(x)$ and once the value of $h_2 = g_2(h_1)$ which are both computed when evaluating $g_3 \circ g_2 \circ g_1(x)$. This suggests that these values should be stored to avoid recomputing them, but also that one may prefer a recursive implementation to numerically evaluate (1.17).

This illustrative example can be extended to DNNs due to their compositional structure. For a MLP like (1.4) this means in particular that to evaluate the value of $\mathcal{J}$ and its partial derivatives with respect to the coefficients of each matrix $W_1, \ldots, W_L$, the most efficient way is to first evaluate $\mathcal{J}$ and store the intermediate compositions, then recursively compute the derivatives first with respect to W_L, and finish with W_1. For a more detailed introduction to backpropagation we refer to Goodfellow et al. (2016, Part II, Chapter 6), but overall it is a technique exploiting the compositional structure (1.3) in order to evaluate the gradient $\nabla\mathcal{J}$ for a cost similar to the cost of evaluating the function $\mathcal{J}$ itself. Since N and P are extremely large, this algorithm is crucial for training DNNs. The computational complexity depends of course on the architecture of the network and the activation functions. Nonetheless, the complexity of a forward pass (for a single data) of a MLP like (1.4) is approximately $O(P)$

(linear in the number of parameters of the network) and backpropagation has also a roughly linear complexity. However, these complexities are stated with a O sign and hence involving multiplicative constants which usually make the evaluations of gradients a little slower than those of the loss function in practice (but of the same order of magnitude). Finally, note that the word *backpropagation* comes from the graphical representation of DNNs (Figure 1.1): for $\theta \in \mathbb{R}^P$, $\mathcal{J}(\theta)$ is obtained by composition, starting from the input layer to the output in a "forward" way, while the gradient $\nabla\mathcal{J}(\theta)$ is obtained starting from the last layer, hence "backward".

1.3.3.2 Automatic differentiation and software implementation

Since backpropagation is a general method to differentiate compositions of functions, it is also known as *automatic differentiation*. It is implemented in the two most popular DL libraries `tensorflow` (Abadi et al., 2016) and `pytorch` (Paszke et al., 2019). Given the source code for evaluating a loss function, these libraries use automatic differentiation and yield an efficient implementation for evaluating the gradient of this loss function. Altogether the high-level programming languages of these DL libraries provide flexibility of use and portability: the code for training a DNN is very similar on a small laptop and on a large cluster. They ease an efficient use of each type of computer, taking the most of CPUs, GPUs, and memory resources. For these reasons, they play a crucial role in the fast development of DL. These libraries have become so useful that nowadays it may even appear irrelevant to design algorithms for training DNNs if those are not implementable in such libraries. Hence, almost all algorithms used for training DNNs are based on automatic differentiation and are implementable in `pytorch` and `tensorflow`.

1.3.4 Mini-batch sub-sampling and stochastic algorithms

In this section, we consider a loss functions $\mathcal{J}$ which is differentiable, but the following discussion can be extended to non-differentiable locally-Lipschitz continuous functions (see Chapter 2). Thanks to the backpropagation algorithm we have a convenient way of evaluating the gradient $\nabla\mathcal{J}$ and thus we could use GD to train DNNs. Unfortunately, even-though backpropagation is an efficient method for computing gradients, the size N of the dataset and the number P of parameters still make the evaluation of the gradient expensive. For example, in the numerical experiments considered in Chapters 2 and 4, it would typically take a few minutes to evaluate $\nabla\mathcal{J}$ exactly at one single point $\theta \in \mathbb{R}^P$ using a standard GPU. It could even take hours for larger datasets and networks! While a few minutes for

evaluating $\nabla \mathcal{J}$ may seem reasonable, the main issue is that we may have to run thousands of iterations of GD so that the NN performs well. So we could theoretically compute exact gradients but practitioners actually rely on an alternative strategy.

In DL, the classical approach is to use approximate values of $\nabla \mathcal{J}$ via what is called mini-batch sub-sampling. It consists in approximating $\mathcal{J}$ and $\nabla \mathcal{J}$ by computing only a few terms of the sum in (1.5). More precisely, let $\mathsf{B} \subset \{1, \dots, N\}$ a subset of indices corresponding to a sub-sample of the full dataset. For any $\theta \in \mathbb{R}^P$ consider the following quantities,

$$\mathcal{J}_{\mathsf{B}}(\theta) \overset{\text{def}}{=} \frac{1}{|\mathsf{B}|} \sum_{n \in \mathsf{B}} \mathcal{J}_n(\theta), \quad \text{and,} \quad \nabla \mathcal{J}_{\mathsf{B}}(\theta) \overset{\text{def}}{=} \frac{1}{|\mathsf{B}|} \sum_{n \in \mathsf{B}} \nabla \mathcal{J}_n(\theta), \tag{1.18}$$

where $|\mathsf{B}|$ denotes the number of elements belonging to the set B. Subsets of indices such as B are called *mini-batches*. Remark that if $\mathsf{B} \equiv \{1, \dots, N\}$, i.e., if B corresponds to the full dataset, then we recover $\mathcal{J}_{\mathsf{B}} = \mathcal{J}$.

Such approximations of $\nabla \mathcal{J}$ yield a modified version of GD: *stochastic gradient descent* (SGD), it formally consists in replacing the gradient in the iterative process (1.15) by an approximation as in (1.18). More precisely consider a random subset $\mathsf{A} \in \{1, \dots, N\}$ such that for any $\theta \in \mathbb{R}^P$, $\mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{A}}(\theta)\right] = \nabla \mathcal{J}(\theta)$, where the expectation is taken over the realizations of the random subset A. The most standard choice for this to hold is to take A uniformly distributed over all the subsets of $\{1, \dots, N\}$ of a given number of elements (called *batch-size*). For example, consider mini-batches of size one (made of a single data point). For such a choice we have indeed,

$$\begin{aligned} \mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{A}}(\theta)\right] = \sum_{\substack{\mathsf{B} \\ \text{s.t. } |\mathsf{B}|=1}} \mathbb{P}\left(A = B\right) \nabla \mathcal{J}_{\mathsf{B}}(\theta) = \sum_{\substack{\mathsf{B} \\ \text{s.t. } |\mathsf{B}|=1}} \frac{1}{N} \nabla \mathcal{J}_{\mathsf{B}}(\theta) \\ = \frac{1}{N} \sum_{\substack{\mathsf{B} \\ \text{s.t. } |\mathsf{B}|=1}} \sum_{n \in \mathsf{B}} \nabla \mathcal{J}_n(\theta) = \frac{1}{N} \sum_{n=1}^{N} \nabla \mathcal{J}_n(\theta) = \nabla \mathcal{J}(\theta). \end{aligned} \tag{1.19}$$

The general case of mini-batches of an arbitrary (fixed) size can be obtained with similar computations and enumeration arguments.

Now, consider a sequence $(\mathsf{B}_k)_{k \in \mathbb{N}}$ of realizations of independent copies of A and a sequence $(\gamma_k)_{k \in \mathbb{N}}$ of step-sizes, then SGD boils down to the iterative process,

$$\theta_{k+1} = \theta_k - \gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k). \tag{1.20}$$

A convenient way of understanding SGD is to view it as a *noisy* version of GD by writing the following equivalent formulation of (1.20),

$$\theta_{k+1} = \theta_k - \gamma_k \left(\nabla \mathcal{J}(\theta_k) + \xi_k \right). \tag{1.21}$$

Here, $\xi_k = \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) - \nabla \mathcal{J}(\theta_k)$ compensates for the part of the gradient that is not considered in the update of SGD compared to GD. Since B_k is sampled from a random subset, ξ_k can be seen as a random variable called *noise*. Additionally, since we chose B_k such that $\mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{B}_k}(\theta)\right] = \nabla \mathcal{J}(\theta)$, at iteration k, the conditional expectation of ξ_k with respect to the current iterate θ_k is thus,

$$\mathbb{E}\left[\xi_k | \theta_k\right] = \mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) - \nabla \mathcal{J}(\theta_k) | \theta_k\right] = \mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) | \theta_k\right] - \nabla \mathcal{J}(\theta_k) = 0.$$

So $(\xi_k)_{k \in \mathbb{N}}$ is actually a sequence of zero-mean martingales (adapted to the filtration induced by the random mini-batches up to iteration k). This means that at every iteration the update of SGD does not follow the steepest descent direction (unlike GD) but at least it does in expectation.

For the sake of precision, SGD is a class of stochastic algorithms introduced by Robbins and Monro (1951), and (1.20) corresponds to the mini-batch version of SGD. Nowadays, mini-batch SGD remains the central algorithm for many applications and in particular for training DNNs due to its limited computational cost. Indeed, mini-batch sub-sampling allows one to iterate SGD much faster than GD when using small mini-batches. The reason is that backpropagation is highly compatible with mini-batch sub-sampling since each gradient $\nabla \mathcal{J}_n$, for $n \in \{1, \dots, N\}$ is evaluated independently of the others (by passing each data point of the training set into the network and backpropagating). As a result, computing $\nabla \mathcal{J}_{\mathsf{B}}$ for $\mathsf{B} \subset \{1, \dots, N\}$ is simply $N/|\mathsf{B}|$ times faster than computing $\nabla \mathcal{J}$. To compare the speed of methods when using mini-batches, it is often convenient to count the *epochs* rather than the iterations. An epoch corresponds to N backpropagations (each for a single data), this way, performing one iteration of GD or $N/|\mathsf{B}|$ iterations of SGD both correspond to one epoch. So, SGD can be iterated faster than GD, but it uses imprecise update directions (compared to GD), thus there are no guarantees in general that SGD will solve (1.6) faster than GD. Nonetheless, in large-scale optimization problems (and in particular in DL) SGD is often reported being significantly faster empirically and is thus preferred to GD (Bottou and Bousquet, 2008).

However, using approximations of the gradient $\nabla \mathcal{J}$ brings new issues. In particular, crit-

ical points may not be stationary points[2] of SGD (unlike GD): their may be an iteration $k \in \mathbb{N}$ of SGD such that $\nabla \mathcal{J}(\theta_k) = 0$ but $\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \neq 0$ and hence the algorithms escapes the critical point θ_k. To overcome this issue the standard approach is to use a sequence of vanishing step-sizes $(\gamma_k)_{k\in\mathbb{N}}$ that meets (for example) the Robbins-Monro condition (Robbins and Monro, 1951),

$$\sum_{k=1}^{+\infty} \gamma_k = +\infty, \quad \text{and,} \quad \sum_{k=1}^{+\infty} \gamma_k^2 < +\infty. \tag{1.22}$$

Under appropriate assumptions (such as the uniform boundedness of the noise $(\xi_k)_{k\in\mathbb{N}}$ in (1.21)), this condition is sufficient to ensure the convergence of SGD to critical points as we will see in Chapter 4. We give a glimpse of this phenomenon on Figure 1.7 where we see that SGD behaves indeed like a perturbed version of GD, and that it fails to converge asymptotically to a critical point when used with non-vanishing step-sizes. Nonetheless, taking a sequence of step-sizes that vanishes too quickly may considerably slow down SGD, hence the choice of this sequence is critical in practice and often reveals to be a challenging task. Some methods such as back-tracking line-search (Armijo, 1966) ease the choice of the step-sizes. Unfortunately, most of these techniques require computing exact values of $\mathcal{J}$ which, in DL, is almost as expensive as computing exact gradients $\nabla \mathcal{J}$ and hence is not suited for training DNNs.

1.4 Using second-order information for training DNNs

We just presented how training DNNs became achievable in many applications. However, the training remains complicated, takes a long time, and requires a lot of computational resources. In this context, we now explain why second-order information is promising to alleviate these issues.

1.4.1 Motivations

For now, assume that $\mathcal{J}$ is twice continuously differentiable, there are several motivations for exploiting second-order derivatives of $\mathcal{J}$ to train DNNs. First Newton's method can be very efficient on some large-scale machine learning problems (Berahas et al., 2020; Xu et al., 2020). For example, it converges in a single iteration on the least-square problem: let

[2] A stationary point of an algorithm is a point where the algorithm remains stuck if it reaches it.

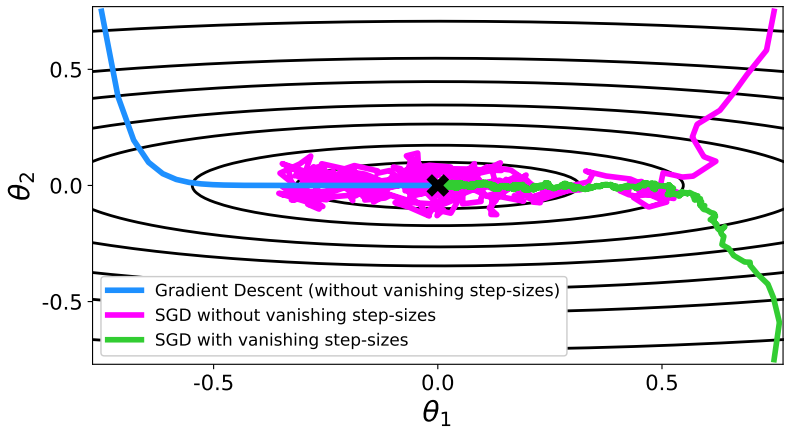


Figure 1.7: Comparison of GD and SGD with and without vanishing step-sizes for minimizing $\mathcal{J}(\theta_1, \theta_2) = \theta_1^2 + 10\theta_2^2$ whose global minimum $(0, 0)$ is represented by the black cross in the middle. The need for using vanishing step-sizes is highlighted by the pink curve which seems to bounce "randomly" around the minimum, unlike the green curve.

$A \in \mathbb{R}^{P \times P}$ be a positive definite matrix and $b \in \mathbb{R}^P$, consider the function

$$\mathcal{Q} : \theta \in \mathbb{R}^P \mapsto \frac{1}{2}\|A\theta - b\|_2^2. \tag{1.23}$$

One can easily check that this function has a unique minimum at $\theta = A^{-1}b$. The function $\mathcal{Q}$ is twice-differentiable and an iteration $k \in \mathbb{N}$ of Newton's method applied to $\mathcal{Q}$ reads,

$$\theta_{k+1} = \theta_k - \nabla^2\mathcal{Q}(\theta_k)^{-1}\nabla\mathcal{Q}(\theta_k) = \theta_k - \left(A^T A\right)^{-1}\left(A^T\left(A\theta_k - b\right)\right) = A^{-1}b. \tag{1.24}$$

So Newton's method does converge in one iteration for any initialization, regardless the condition number of A, while GD can be very slow if A is poorly conditioned, see for example Nocedal and S. Wright (2006, Chapter 2).

Second-order information can also be used to avoid strict saddle points. Indeed, under Assumption 1.1, a necessary condition for a point $\theta^\star \in \mathbb{R}^P$ to be a local minimum is that $\nabla\mathcal{J}(\theta^\star) = 0$ and that $\nabla^2\mathcal{J}(\theta^\star)$ is positive semi-definite (it has only non-negative eigenvalues). This gives for example the idea of building algorithms looking for directions of *negative curvature*. Given a point $\theta \in \mathbb{R}^P$ such methods perform an update $\theta + d$ where $d \in \mathbb{R}^P$ is a direction spawned by the eigenvectors associated to the negative eigenvalues of $\nabla^2\mathcal{J}(\theta)$ if such eigenvalues exists. This way, it is possible to build methods that do not stop until they find a region where $\nabla^2\mathcal{J}$ is non-negative.

Finally, another flavor of second-order information is that it can help to choose the step-size sequence $(\gamma_k)_{k\in\mathbb{N}}$ required by most algorithms. To simplify, consider GD and assume that for any $\theta \in \mathbb{R}^P$, the Hessian $\nabla^2\mathcal{J}(\theta)$ has no eigenvalues with magnitude larger than $L_{\nabla\mathcal{J}} > 0$ (this means that $L_{\nabla\mathcal{J}}$ is a Lipschitz constant of $\nabla\mathcal{J}$). Then consider an iteration $k \in \mathbb{N}$ of GD as in (1.15), for two consecutive iterates θ_k and θ_{k+1}, by the Taylor-Lagrange formula, there exists $\psi \in \mathbb{R}^P$ such that,

$$\mathcal{J}(\theta_{k+1}) = \mathcal{J}(\theta_k) + \langle \nabla\mathcal{J}(\theta_k), \theta_{k+1} - \theta_k\rangle + \frac{1}{2}\langle \nabla^2\mathcal{J}(\psi)(\theta_{k+1} - \theta_k), \theta_{k+1} - \theta_k\rangle \tag{1.25}$$

$$= \mathcal{J}(\theta_k) - \gamma\|\nabla\mathcal{J}(\theta_k)\|^2 + \frac{1}{2}\gamma^2\langle \nabla^2\mathcal{J}(\psi)\nabla\mathcal{J}(\theta_k), \nabla\mathcal{J}(\theta_k)\rangle \tag{1.26}$$

$$\leq \mathcal{J}(\theta_k) - \gamma(1 - \frac{L_{\nabla\mathcal{J}}\gamma}{2})\|\nabla\mathcal{J}(\theta_k)\|^2. \tag{1.27}$$

So if $L_{\nabla\mathcal{J}}$ is known, we can take $\gamma < 2/L_{\nabla\mathcal{J}}$ and ensure that $\mathcal{J}(\theta_{k+1}) \leq \mathcal{J}(\theta_k)$, the inequality above is called *descent lemma*, see for example Bertsekas (1999, Proposition A.24). Here second-order information would help to estimate $L_{\nabla\mathcal{J}}$ since $L_{\nabla\mathcal{J}}$ is simply the largest eigenvalue of the Hessian $\nabla^2\mathcal{J}$ on $\mathbb{R}^P$. We could also consider to use second-order information to compute local estimations of $L_{\nabla\mathcal{J}}$. This might accelerate the training but would also ease the choice of the step-sizes.

To summarize, second-order information could substantially benefit the training of DNNs in many ways, there are however many obstacles. First computing the Hessian explicitly—a matrix with P^2 elements— or storing it is hardly possible. Worse, inverting it (for Newton's method) or computing its eigenvalues (for the purposes discussed above) is unpractical. There are actually additional drawbacks: for non-smooth function second-order information may be useless, for example the function $t \in \mathbb{R} \mapsto |t|$ is twice-differentiable almost everywhere but with zero second-order derivative. Secondly using only "noisy" quantities (via mini-batch sub-sampling) may significantly weaken the benefits of higher-order derivatives even for smooth functions. For example, the update direction of SGD is inaccurate compared to GD due to the noise $(\xi_k)_{k\in\mathbb{N}}$ discussed in Section 1.3.4 (Equation 1.21). Thus, we may wonder if it is worth well-choosing the step-size γ_k since the direction itself is imprecise. We refer to Bottou et al. (2018) for further discussion on the benefits of second-order information for training DNNs and the associated challenges.

1.4.2 Precise problem statement

To sum up what we presented so far, our objective is to tackle the challenging task of building efficient algorithms for training DNNs. The training amounts to the minimization

of high-dimensional non-convex functions which have possibly many spurious critical points. In some cases we may even have to deal with the minimization of non-smooth loss functions which is significantly harder in non-convex settings. The high-dimensional nature of the problem makes the use of backpropagation hardly avoidable and the storage of gradient estimations is very limited. Furthermore, the loss function takes the form of a very large finite sum, enforcing the need of using mini-batch sub-sampling. In this context SGD remains the fundamental tool for training DNNs due to its compatibility with the noisy first-order optimization framework that we just summarized. However, the exploitation of second-order information seems promising to build faster algorithms or ease the choice of the step-sizes of SGD, but theoretical and technical limitations are manifold. In this work we will thus tackle the following questions,

- *Can one build practical algorithms exploiting second-order information despite all the theoretical and technical limitations of deep learning?*
- *For these algorithms, does second-order information really benefits the training of DNNs in the presence of mini-batch sub-sampling and non-smoothness?*
- *What convergence guarantees and rates can one derive for these algorithms in such a general theoretical framework where convergence may seem unlikely to occur?*

1.4.3 Overview of existing methods

We review some general ideas to tackle the problems stated above and then review existing methods some of which exploit these ideas.

1.4.3.1 Second-order information via first-order oracles

Throughout this section, let $\theta \in \mathbb{R}^P$ be a point of $\mathbb{R}^P$ and $\mathsf{B} \subset \{1, \dots, N\}$ a mini-batch. A first idea to build second-order methods for DL is to find more affordable Newton-like methods. Adaptations of Newton's method have been proposed for many large-scale machine learning applications (Martens, 2010; Byrd et al., 2011; Boyer and Godichon-Baggioni, 2020). Yet, few of these methods can be successfully applied to DL due to the computational cost and the mini-batch sub-sampling. Indeed, according to what we explained in Section 1.4.1, assuming that $\mathcal{J}$ is twice-differentiable, Newton's method is an iterative process whose updates take the form $(\nabla^2 \mathcal{J}(\theta))^{-1} \nabla \mathcal{J}(\theta)$, and thus are very expensive to compute. Some DL algorithms however manage to adapt Newton's method (ADAGRAD, K-FAC, etc., which are introduced hereafter). They use a mini-batch estimation $\nabla \mathcal{J}_{\mathsf{B}}(\theta)$

of the gradient and a surrogate matrix, cheaper to compute and to invert than $\nabla^2\mathcal{J}(\theta)$. With these elements they build a stochastic quasi-Newton update. Most methods presented below somehow rely on this idea.

Regarding the use of fine-tuned step-sizes discussed in Section 1.4.1. Unlike computing the eigenvalues of $\nabla^2\mathcal{J}(\theta)$, it is however possible to evaluate quantities such as the term $\nabla^2\mathcal{J}(\theta)\nabla\mathcal{J}(\theta)$ which appears in (1.26). Indeed, backpropagation can be adapted to evaluate "Hessian times vector" products within quite a reasonable time (Pearlmutter, 1994). It is however more expensive than computing gradients and its combination with mini-batch sub-sampling yields poor performances in general (Martens et al., 2012).

Similarly, terms of the form $\nabla^2\mathcal{J}(\theta)\nabla\mathcal{J}(\theta)$ can be approximated through finite differences. For example, a Taylor approximation with small $\gamma > 0$ yields, $\nabla\mathcal{J}(\theta - \gamma\nabla\mathcal{J}(\theta)) \simeq \nabla\mathcal{J}(\theta) - \gamma\nabla^2\mathcal{J}(\theta)\nabla\mathcal{J}(\theta)$. Discretization is also used in quasi-Newton methods (see for example Broyden 1967; Byrd et al. 2016). Yet, similarly to the idea above, finite differences must be used with care to be efficient in presence of noisy gradients (Schraudolph et al., 2007). We will investigate this later in Chapter 4.

1.4.3.2 Standard algorithms for training DNNs

Due to its high compatibility with the backpropagation and its relative efficiency in practice, SGD remains the fundamental method for training DNNs. We present other popular methods, some of which are direct extensions of SGD and some that already exploit second-order derivatives. Throughout this section we assume that $\mathcal{J}$ is differentiable, $(\gamma_k)_{k\in\mathbb{N}}$ denotes again a sequence of step-sizes, $(\mathsf{B}_k)_{k\in\mathbb{N}}$ are mini-batches and $\theta_k \in \mathbb{R}^P$, where $k \in \mathbb{N}_{>0}$ is the index of the current algorithm iteration. We present the mini-batch version of each method.

Momentum methods. Momentum –or inertia– refers to a class of methods inherited from the Heavy-Ball with Friction (HBF) method (Polyak, 1964). One iteration can be described by the iterative process,

$$\begin{cases} m_k &= \mu m_{k-1} - \gamma_k \nabla\mathcal{J}_{\mathsf{B}_k}(\theta_k) \\ \theta_{k+1} &= \theta_k + m_k \end{cases}, \tag{1.28}$$

where $\mu \in (0, 1)$. To understand the idea behind HBF, think of the graph of the loss function—the set $\left\{(\theta, \mathcal{J}(\theta)) \,\middle|\, \theta \in \mathbb{R}^P\right\}$—as the *landscape* of a mountain (the local minima lie at the bottom of the valleys of this landscape). In this formalism, the sequence of

iterates and values $(\theta_k, \mathcal{J}(\theta_k))_{k\in\mathbb{N}}$ of HBF represents the successive positions of the ball on the landscape. This ball is subject to gravity (represented by $-\nabla\mathcal{J}$ or its approximation $-\nabla\mathcal{J}_{\mathsf{B}_k}$) which pulls it down, it progressively accumulates speed (through the term m_k in Equation 1.28), this may produce an acceleration, resulting in a faster decrease of the loss function. In the heavy-ball model, the ball is also slowed down by friction effects which generate energy dissipation, so that the ball will eventually reach a rest point that we would like to be a minimum. We will discuss the heavy-ball formalism further in Chapter 2.

There exists different non-equivalent variations to HBF, the one presented here follows the formulation of Sutskever et al. (2013). A popular variation is the Nesterov accelerated gradient (NAG) method (Nesterov, 1983), for which an iteration is,

$$\begin{cases} m_k &= \mu_{k-1} m_{k-1} - \gamma_k \nabla\mathcal{J}_{\mathsf{B}_k}(\theta_k + \mu_{k-1} m_{k-1}) \\ \theta_{k+1} &= \theta_k + m_k \end{cases} . \tag{1.29}$$

It is similar to (1.28) with the exception that the gradient of $\mathcal{J}_{\mathsf{B}_k}$ is not evaluated at θ_k but rather extrapolated to the point $\theta_k + \mu m_{k-1}$ and that μ_k may vary over the iterations (originally, $\mu_k = k/k+3$).

ADAGRAD and RMSprop. The ADAGRAD method (J. Duchi et al., 2011) was originally proposed as a method for convex online learning but became very popular in DL due to its efficiency. It is a non-momentum method which basically reads,

$$\begin{cases} \Gamma_k &= \sqrt{\sum_{k=0}^{k} \nabla\mathcal{J}_{\mathsf{B}_k}(\theta_k) \odot \nabla\mathcal{J}_{\mathsf{B}_k}(\theta_k)} + \varepsilon \\ \theta_{k+1} &= \theta_k - \alpha \ \mathrm{diag}\,(\Gamma_k)^{-1} \nabla\mathcal{J}_{\mathsf{B}_k}(\theta_k) \end{cases} , \tag{1.30}$$

where $\alpha > 0$, $\varepsilon > 0$ is small, $\odot$ is the element-wise product between vectors, the square-root is applied element-wise, and diag is the operator turning vectors into diagonal matrices. ADAGRAD belongs to the family of *adaptive methods* which aims to adapt the step-size to each iteration using first (or higher-order) information. As we said above, ADAGRAD can be seen as a preconditioner method, replacing the Hessian in Newton's method, by $\alpha\,\mathrm{diag}\,(\Gamma_k)^{-1}$. From (1.30), each coefficient of the vector step-size $\alpha\Gamma_k^{-1}$ is decreasing as k increases. This may result in the step-size becoming too small too quickly. Some variants of ADAGRAD address this issue, for example RMSprop (Tieleman and G. Hinton, 2012)

which uses an exponential moving average,

$$\begin{cases} v_k & = \beta v_{k-1} + (1-\beta)\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \odot \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \\ \Gamma_k & = \sqrt{v_k} + \varepsilon \\ \theta_{k+1} & = \theta_k - \alpha \ \mathrm{diag}\,(\Gamma_k)^{-1} \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \end{cases}, \tag{1.31}$$

here $\beta \in (0,1)$ and again $\alpha > 0$, $\varepsilon > 0$.

The ADAM algorithm. ADAM is essentially a combination of RMSprop and momentum, it reads,

$$\begin{cases} m_k & = \beta_1 m_{k-1} + (1-\beta_1)\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \\ v_k & = \beta_2 v_{k-1} + (1-\beta_2)\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \odot \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \\ \Gamma_k & = \sqrt{v_k} + \varepsilon \\ \theta_{k+1} & = \theta_k - \alpha \ \mathrm{diag}\,(\Gamma_k)^{-1} m_k \end{cases}, \tag{1.32}$$

Again, $\beta_1, \beta_2 \in (0,1)$ and $\alpha > 0$, $\varepsilon > 0$. This algorithm was proposed by Kingma and Ba (2015) and is probably the most-used algorithm for training DNNs apart from SGD. The main reason for this is that ADAM is known to be more robust to the choice of its hyper-parameters α, β_1, β_2 and ε in practice than SGD is for the choice of its sequence of step-sizes $(\gamma_k)_{k\in\mathbb{N}}$. In particular, ADAM is known to provide satisfying results in many cases even when using the default values for the hyper-parameters (provided in the original paper).[3] This is an important advantage since tuning hyper-parameters is a highly time-consuming task (Asi and J. C. Duchi, 2019).

Full matrix preconditioners. We now review some methods using this time a non-diagonal matrix as preconditioner (in comparison to ADAGRAD). Again, these methods recall of course Newton's method which uses the inverse of the Hessian $\left(\nabla^2 \mathcal{J}(\theta_k)\right)^{-1}$ but replace it by more computationally affordable matrices.

Among these methods, Natural Gradient (NG) (Amari, 1998), full-matrix ADAGRAD (Agarwal et al., 2019) and similar methods revolve around replacing the Hessian by the Gram (or Gauss-Newton) matrix $\nabla \mathcal{J}(\theta_k) \nabla \mathcal{J}(\theta_k)^T$, or similar matrices such as the Fisher information matrix. While being affordable, their computational cost is higher than those of the methods presented above and is not compensated by significant improvements in

[3] ADAGRAD and other adaptive methods seem to benefit from similar qualities but this holds in particular for ADAM.

general (in particular for NG). The K-FAC algorithm (Martens and Grosse, 2015) speeds up NG by first making a block-approximation of the Fisher information matrix (one block per layer of the NN) and then computing a Kronecker approximation of each block to exploit the fact that inverting a Kronecker product is straightforward. K-FAC is reported being efficient on several problems and is among the most popular algorithms although ADAM and SGD remain the most used one by far. A block-matrix approach was also followed by Dudar et al. (2017) which designed a sub-space trust-region method. Their algorithm consists in building one update direction and one learning rate per layer of the NN. They report competitive results for their method on dense networks.

Seeking directions of negative curvature. In Section 1.4.1 we previously discussed the interest of looking for directions of negative curvature (eigenvectors associated to negative eigenvalues of the Hessian of $\mathcal{J}$). This idea has been present in the literature for a few years (Mizutani and Dreyfus, 2008), but remains quite inefficient and unpractical in DL, again, for computational reasons. Some progress has been made, for example Carmon et al. (2017) proposed a method for detecting locally non-convex regions. Their method requires however exact gradient evaluations, making it still unpractical for large NNs and datasets. Note that the matrix $\nabla\mathcal{J}(\theta_k)\nabla\mathcal{J}(\theta_k)^T$ used in NG, K-FAC, etc., is always symmetric positive and as such neglects the non-convex nature of $\mathcal{J}$.

Some other methods. We presented the most famous methods for training DNNs, it goes without saying that many variations of these methods have been proposed. We conclude this section by giving a non-exhaustive list: there are several methods similar to ADAM and ADAGRAD: ADAMW (Loshchilov and Hutter, 2019), Adabelief (Zhuang et al., 2020), Adadelta (Zeiler, 2012), AMSgrad (Reddi et al., 2018), etc., and some other methods such as the stochastic Barzilai-Borwein methods (Tan et al., 2016; Liang et al., 2019) or the Lookahead algorithm (M. Zhang et al., 2019).

1.5 Organization of the manuscript

To tackle the problems presented above, the manuscript is organized as follows.

Chapter 2 introduces a new algorithm called INNA which stands for **In**ertial **N**ewton **A**lgorithm. The starting point is a second-order continuous-time autonomous ordinary differential equation (ODE) called DIN (Alvarez et al., 2002) and inspired by Newton's second law of dynamics. The latter models a mix between accelerated gradient descent

(see HBF in Section 1.4.3.2) and Newton's method via a term involving the Hessian of the loss function. DIN is thus a second-order ODE both in "time" and "space". However, when using a first-order reformulation in time of this ODE, the Hessian terms vanishes as well, making second-order derivatives implicit. Using the notion of Clarke's subdifferential we extend the ODE to non-smooth non-convex functions. We then evidence that mini-batch sub-sampling combined with the Clarke subdifferential generates spurious non-critical stationary points. To cope with this we introduce a new notion of steady states and obtain a new differential inclusion which we discretize to obtain the INNA algorithm. We then prove the almost-sure convergence of sub-sequences of iterations of INNA to critical points using a Lyapunov analysis combined with the results of Benaïm et al. (2005) on perturbed solutions of differential inclusions. We additionally derive rates of convergence for the solutions of the differential inclusion and investigate the empirical performances of INNA.

Chapter 3 studies the asymptotic behavior of INNA and the solutions of the DIN system introduced in Chapter 2. While we previously showed that INNA almost surely converges to critical points, we now focus on the nature of such limit points (minima, strict saddle points etc.). Although INNA is a mix between inertial gradient descent—which is likely to avoid strict saddle points (Lee et al., 2016; O'Neill and S. J. Wright, 2019)—and Newton's method—which may converge to any type of critical point—we show, for smooth functions, that a full-batch (i.e., without mini-batches) version of INNA is likely to avoid strict saddle points for most initializations. The results rely on the stable manifold theorem and the Hartman-Grobman theorem. Some numerical illustrations are provided.

Chapter 4 is dedicated to using second-order information for fine-tuning the sequence of step-sizes of SGD in non-convex settings. Using a simple variational model we retrieve a step-size first-proposed by Alvarez and Cabot (2004), the latter being too computationally expensive, we approximate it and make a connection with the step-size of Barzilai and Borwein (1988). We then use the link between these step-sizes as well as empirical and theoretical observations to modify the mini-batch sub-sampling in order to efficiently approximate second-order information through discretization with noisy gradients. We prove the almost sure convergence of the resulting algorithm, called Step-Tuned SGD, and derive rates of convergence for the sequence of values of the loss function. We conclude with numerical experiments suggesting that second-order information can indeed be efficiently used to fine-tune the step-sizes of SGD when training DNNs.

The concluding chapter summarizes the results presented in this thesis. We then discuss remaining open questions and draw some perspectives for future work.

List of Publications

Accepted journal papers

Castera et al. (2021a)
C. Castera , J. Bolte, C. Févotte, and E. Pauwels (2021). An Inertial Newton Algorithm for Deep Learning. *In Journal of Machine Learning Research (JMLR)* 22.134, pp. 1–31.
Available at: https://jmlr.csail.mit.edu/papers/v22/19-1024.html

Castera et al. (2021b)
C. Castera , J. Bolte, C. Févotte, and E. Pauwels (2021). Second-order Step-size Tuning of SGD for Non-convex Optimization. To appear in *Neural Processing Letters*.
Available on arXiv: https://arxiv.org/abs/2103.03570

Submitted journal papers

Castera (2021)
C. Castera (coming in 2021). Inertial Newton Algorithms Avoiding Strict Saddle Points.
Available on arXiv: https://arxiv.org/abs/2111.04596

International conferences

Castera et al. (2019a)
C. Castera , J. Bolte, C. Févotte, and E. Pauwels (2019). An Inertial Newton Algorithm for Deep Learning. Poster and paper presented at the *NeurIPS Workshop: Beyond First-order Methods in Machine Learning*.
Poster available at: https://camcastera.github.io
Proceedings available at: https://sites.google.com/site/optneurips19

Chapter 2

INNA: An Inertial Newton Algorithm for Deep Learning

This chapter is adapted from Castera et al. (2021a).

Contents

2.1 Introduction

We focus on building a new algorithm for training DNNs featuring inertia and Newtonian behavior while requiring only noisy first-order information. In this chapter we tackle the problems introduced in Section 1.4.2 in their more general form: we consider loss functions that are non-convex but also non-smooth. However, most of the fundamental notions and tools (backpropagation, SGD, etc.) discussed in the introduction chapter were defined for smooth functions. Here we will thus introduce the Clarke subdifferential (Clarke, 1990) as a surrogate for gradients. We will also discuss the incompatibility of the combination of vanilla backpropagation and mini-batch sub-sampling with the Clarke subgradient of non-smooth non-convex loss functions, a problem ignored by most DL practitioners.

In such a general framework we will rely on the Ordinary Differential Equations (ODE) approach introduced in Ljung (1977), and then developed by Benaïm (1999), Kushner and Yin (2003), Benaïm et al. (2005), and Borkar (2009). It is useful to analyze optimization algorithms. For example, assume temporarily that the loss function $\mathcal{J}$ is differentiable, recall from (1.15) that two consecutive iterations $\theta_k \in \mathbb{R}^P$ and $\theta_{k+1} \in \mathbb{R}^P$ of GD with step-size $\gamma > 0$ are linked by the relation $\theta_{k+1} = \theta_k - \gamma \nabla \mathcal{J}(\theta_k)$, or equivalently, $(\theta_{k+1} - \theta_k)/\gamma + \nabla \mathcal{J}(\theta_k) = 0$. Formally, when γ is small (in a sense that can be made precise), we see that $(\theta_{k+1} - \theta_k)/\gamma$ looks like a discretization of the derivative of some continuous-time differentiable function $\theta : \mathbb{R}_+ \to \mathbb{R}^P$. Denoting by $\dot{\theta}$ the derivative of θ, the following ODE:

$$\dot{\theta}(t) + \nabla \mathcal{J}(\theta(t)), \quad \text{for all } t > 0, \tag{2.1}$$

is a continuous-time model of the discrete GD algorithm, here the time parameter t acts as a continuous "iteration counter". For a given initial condition $\theta_0 \in \mathbb{R}^P$, a differentiable function $\theta : \mathbb{R}_+ \to \mathbb{R}^P$ is called *solution* or *trajectory* of the ODE if $\theta(0) = \theta_0$ and (2.1) holds for all $t > 0$.

The aforementioned literature gives a precise characterization of the link between contin-

uous dynamics and discrete algorithms. In particular, it connects the asymptotic behavior of discrete algorithms (as the iteration index k tends to $+\infty$) and the limit (as time t tends to $+\infty$) of the solutions of their underlying ODEs. In particular, when the solutions of an ODE converge—which means that they reach a limit as t tends to $+\infty$—if this limit is a critical point of the loss function $\mathcal{J}$, then under some conditions one may be able to conclude that the corresponding discrete algorithm also converges to critical points of $\mathcal{J}$.

This approach is relevant for many algorithms, in particular it is connected to the heavy-ball interpretation of momentum methods discussed in Section 1.4.3.2. Indeed, let $\mathcal{J} : \mathbb{R}^P \to \mathbb{R}$ be a differentiable loss function, the interpretation of a ball evolving on the graph of $\mathcal{J}$ can be described by the following ODE,

$$\underbrace{\ddot{\theta}(t)}_{\text{Inertial term}} + \underbrace{\alpha\,\dot{\theta}(t)}_{\text{Friction term}} + \underbrace{\nabla\mathcal{J}(\theta(t))}_{\text{Gravity effect}} = 0, \quad \text{for } t \in [0, +\infty), \tag{2.2}$$

where $\alpha > 0$, and $\theta : \mathbb{R}_+ \to \mathbb{R}^P$ is a twice-differentiable function which represents the position of a ball on the graph of $\mathcal{J}$. Similarly to (2.1), $\dot{\theta}$ and $\ddot{\theta}$ denote the first and second-order time derivatives of θ respectively. This ODE point of view echos Newton's law of dynamics (the acceleration $\ddot{\theta}$ is equal to a sum of "forces"). HBF is thus sometimes referred to as an *accelerated* version of gradient descent since (2.2) takes inertia into account. The ODE paradigm was recently used in several works (Adil, 2018; D. Davis et al., 2020; Barakat and Bianchi, 2021). Inertial first-order methods like (2.2) remain however hard to study when adapted to non-differentiable functions since non-smoothness causes "shocks": the landscape of a non-differentiable loss function has "corners" and "walls" which generate a discontinuity of the velocity $\dot{\theta}$.

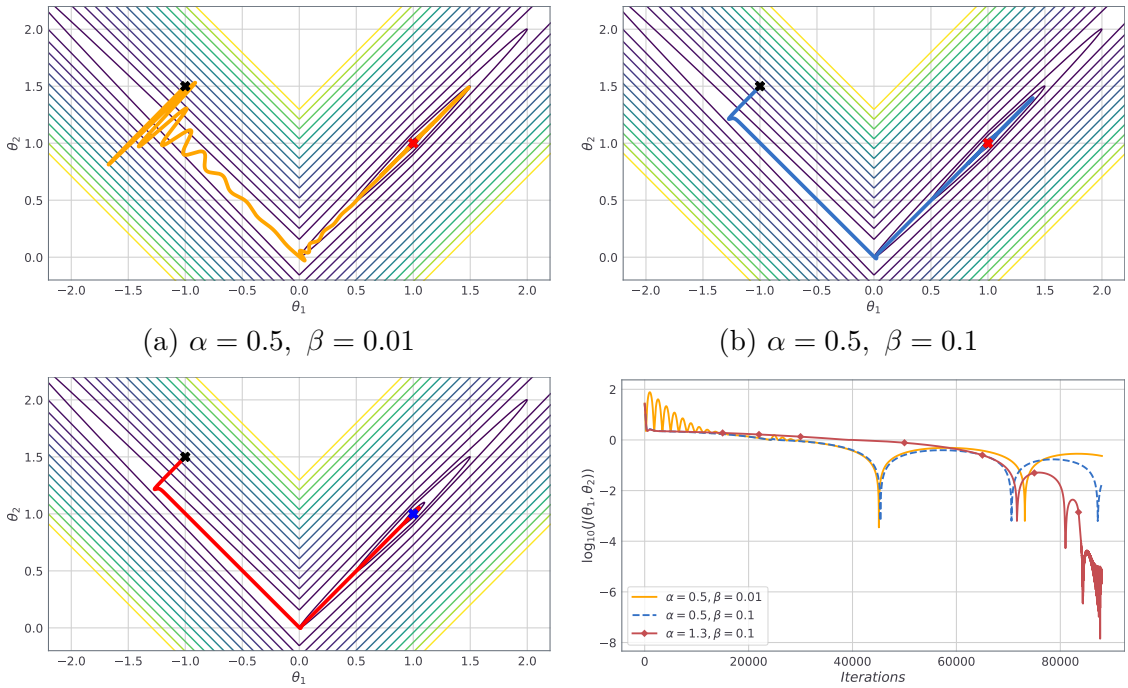
Considering another ODE, Attouch and Redont (2001) showed that adding inertia to continuous-time Newton's dynamics has a regularization effect. This later motivated the combination of HBF and inertial Newton's method and led to the introduction of the following continuous-time dynamical system (or ODE) introduced in Alvarez et al. (2002) and referred to as DIN (standing for "dynamical inertial Newton"). Just for now, let $\mathcal{J} : \mathbb{R}^P \to \mathbb{R}$ be a twice continuously differentiable loss function, DIN reads,

$$\underbrace{\ddot{\theta}(t)}_{\text{Inertial term}} + \underbrace{\alpha\,\dot{\theta}(t)}_{\text{Friction term}} + \underbrace{\beta\,\nabla^2\mathcal{J}(\theta(t))\dot{\theta}(t)}_{\text{Newtonian effects}} + \underbrace{\nabla\mathcal{J}(\theta(t))}_{\text{Gravity effect}} = 0, \quad \text{for } t \in [0, +\infty), \tag{2.3}$$

where the notations are the same as in (2.2). It can be shown that when the solutions of (2.3) converge, they converge to critical points of $\mathcal{J}$ (Alvarez et al., 2002). Hence, rather

than using ODEs to analyze discrete algorithms, we will instead discretize a version of DIN adapted to non-smooth functions in order to obtain an optimization algorithm which possesses inertial and Newtonian properties and that is well suited for minimizing $\mathcal{J}$. Our resulting second-order algorithm is called INNA.

Before going into the details, we illustrate one interest of mixing HBF and Newton's method on Figure 2.1 for a simple non-smooth and non-convex function of $\mathbb{R}^2$. This figure shows in particular how the additional term in (2.3), compared to (2.2), helps to reduce parasitic "transverse" oscillations.



(a) $\alpha = 0.5,\ \beta = 0.01$ (b) $\alpha = 0.5,\ \beta = 0.1$

(c) $\alpha = 1.3,\ \beta = 0.1$ (d) Objective function $\mathcal{J}$ (in log-scale)

Figure 2.1: Illustration of the role of the hyper-parameters of DIN on the non-smooth function $\mathcal{J}(\theta_1, \theta_2) = 100(\theta_2 - |\theta_1|)^2 + |1 - \theta_1|$. The results are simulated using a full-batch version of the algorithm INNA introduced later (see Algorithm 1). Subplots (a-c) represent the trajectories of the parameters θ_1 and θ_2 in $\mathbb{R}^2$ for three choices of hyper-parameters α and β, see (2.3) for an intuitive explanation. Subplot (d) displays the values of the objective function $\mathcal{J}(\theta_1, \theta_2)$ for the three settings considered. The details of this experiment and further discussions are provided in Section 2.6.1.

The previous discussions hold only for smooth functions. In the rest of this chapter, we will adapt DIN to DL by overcoming the computational and conceptual difficulties raised

by the second-order objects $\ddot{\theta}$ and $\nabla^2\mathcal{J}(\theta)$ appearing in (2.3). This is done by combining a phase-space lifting method (a first-order reformulation of DIN) introduced in Alvarez et al. (2002) with the use of the Clarke subdifferential $\partial\mathcal{J}$ (see Definition 2.2). We then evidence a sum rule failure for non-smooth non-convex functions, whereas the sum-rule is crucial for the theoretical convergence analysis of mini-batch algorithms. Yet, many DL studies ignore the failure of the sum rule: practitioners sum sub-gradients when using optimization algorithms for DL, but only analyze the methods under simplifying assumptions such as smoothness or convexity. We tackle this difficulty as is, and show that the sum rule failure creates additional spurious stationary points that are not (Clarke) critical. To address this question, we introduce the notion of D-criticality. We finally obtain a new first-order differential inclusion that we discretize to obtain INNA. We then show the convergence of INNA to such D-critical points.

The rest of this chapter is thus organized as follows. The step by step introduction of INNA is described in Section 2.3. Its convergence is proved in Section 2.4, and we then provide convergence rates for the solutions of the underlying continuous-time differential inclusion (our non-smooth adaptation of DIN) in Section 2.5. Finally, Section 2.6 provides some experimental results on DL benchmark problems using standard datasets (MNIST, CIFAR-10, CIFAR-100). First, we discuss the essential properties of the loss function $\mathcal{J}$ mentioned in the introduction chapter: local Lipschitz continuity and tameness.

2.2 A functional framework for non-smooth non-convex optimization

We first recall some notations and discuss local Lipschitz continuity.

2.2.1 Locally Lipschitz continuous neural network and loss function

We keep the notations of the introduction and still consider a general type of DNN $f : (x, \theta) \in \mathbb{R}^M \times \mathbb{R}^P \mapsto y \in \mathbb{R}^D$ that is *locally Lipschitz continuous* in its parameter θ (see Definition 1.4). This includes in particular the networks and activation functions introduced in Section 1.1.3. As before, throughout this chapter we consider a training dataset $(x_n, y_n)_{n\in\{1,\dots,N\}}$ and we recall that the loss function $\mathcal{J}$ takes the following form,

$$\mathcal{J}(\theta) = \sum_{n=1}^{N} \ell(f(x_n, \theta), y_n), \tag{2.4}$$

for a given dissimilarity measure $\ell : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ that we assume to be locally Lipschitz continuous, so that $\mathcal{J}$ and each $\mathcal{J}_n = \ell(f(x_n, .), y_n)$ are locally Lipschitz continuous on $\mathbb{R}^P$. Despite non-smoothness and non-convexity, the loss function possesses a very strong property called tameness. We now introduce this notion which is essential for the theoretical analysis of Section 2.4.

2.2.2 Neural networks are tame functions

Tameness refers to a geometrical property shared by many functions and sets. It holds in many finite-dimensional optimization problems met in practice. Prominent classes of tame objects are piecewise-linear or piecewise-polynomial objects (with finitely many pieces), and more generally, semi-algebraic objects. However, the notion is much more general, as we intend to convey below. A rigorous definition is given at the end of this section (Definition 2.1).

As mentioned in the introduction (Section 1.2.3), sets or functions are called tame when they can be described by a finite number of basic formulas, inequalities, or Boolean operations involving standard functions such as polynomials, exponentials, or max functions. We refer to Attouch et al. (2010) for illustrations, recipes and examples within a general optimization setting or D. Davis et al. (2020) for a link with NNs. The reader is referred to Van den Dries (1998), Coste (2000), and Shiota (2012) for foundational work on tameness. Again, this property is powerful as it allows splitting the study of non-smooth objects on an union of smooth pieces. This is the so-called *stratification* property of tame sets and functions. In a non-convex optimization settings, the stratification property is crucial to generalize qualitative algorithmic results to non-smooth objects.

In most finite-dimensional DL problems, the loss functions $\mathcal{J}$ is tame. To understand this assertion and illustrate the wide scope of the tameness assumption, let us provide concrete examples (see also D. Davis et al. 2020). The loss $\mathcal{J}$ is tame for any NN built from the following traditional components:

- The NN f must have a fixed but arbitrary large number of layers of arbitrary dimensions and must be feedforward, meaning that it can be represented by a directed acyclic graph as in Section 1.1.2.2.
- The activation functions must be among classical ones: ReLU, sigmoid, tanh, soft plus, etc. including multivariate activation functions (norm, sorting, pooling), or functions defined piecewise with polynomials, exponentials and logarithms.

– The dissimilarity measure ℓ can be built from norms, logistic loss or cross-entropy, or more generally functions defined piecewise using polynomials, exponentials and logarithms.

Tameness is thus shared by many loss functions in DL, in addition, we can assume that each term $\mathcal{J}_n = \ell(f(x_n, .), y_n)$ in (2.4) is tame as well. The results above are obtained by quantifier elimination arguments using property (iii) below. For the sake of completeness, we provide the precise definitions of tameness and o-minimality.

Definition 2.1. ***[o-minimal structure]*** *(Coste, 2000, Definition 1.5) An o-minimal structure on $(\mathbb{R}, +, .)$ is a countable collection of sets $\mathcal{O} = \{\mathcal{O}_q\}_{q\geq 1}$ where each $\mathcal{O}_q$ is itself a collection of subsets of $\mathbb{R}^q$, called* definable *subsets. They must have the following properties, for each $q \geq 1$:*

(i) (Boolean properties) $\mathcal{O}_q$ contains the empty set, is stable by finite union, finite intersection and complementation;

(ii) (Lifting property) if A belongs to $\mathcal{O}_q$, then $A \times \mathbb{R}$ and $\mathbb{R} \times A$ belong to $\mathcal{O}_{q+1}$.

(iii) (Projection or quantifier elimination property) if $\mathcal{P} : \mathbb{R}^{q+1} \to \mathbb{R}^q$ is the canonical projection onto $\mathbb{R}^q$ then for any A in $\mathcal{O}_{q+1}$, the set $\mathcal{P}(A)$ belongs to $\mathcal{O}_q$.

(iv) (Semi-algebraicity) $\mathcal{O}_q$ contains the family of algebraic subsets of $\mathbb{R}^q$, that is, every set of the form

$$\{\theta \in \mathbb{R}^q \mid \mathcal{Q}(\theta) = 0\},$$

where $\mathcal{Q} : \mathbb{R}^q \to \mathbb{R}$ is a polynomial function.

(v) (Minimality property), the elements of $\mathcal{O}_1$ are exactly the finite unions of intervals and points.

A function is said to be *definable* in an o-minimal structure $\mathcal{O}$ if its graph can be defined[1] in $\mathcal{O}$.

From now on we fix an o-minimal structure $\mathcal{O}$. A function or a set will be called *tame* if it is definable in this o-minimal structure $\mathcal{O}$.

2.3 From DIN to INNA: an inertial Newton algorithm

We describe the construction of our proposed algorithm INNA from the discretization of the second-order ODE (2.3).

[1] This means that the graph can be described using first-order logic (formally, using quantifiers on variables).

2.3.1 Handling non-smoothness and non-convexity

We first generalize (2.3) to the non-smooth non-convex setting. Recall that the dynamical system DIN is,

$$\ddot{\theta}(t) + \alpha\dot{\theta}(t) + \beta\nabla^2\mathcal{J}(\theta(t))\dot{\theta}(t) + \nabla\mathcal{J}(\theta(t)) = 0, \tag{2.5}$$

where $\mathcal{J}$ is (for now) a twice-differentiable function, $\alpha \geq 0$, $\beta > 0$ are two hyper-parameters and $\theta : \mathbb{R}_+ \to \mathbb{R}^P$ is a twice-differentiable function of $\mathbb{R}_+$. We cannot exploit (2.5) directly since in most DL applications $\mathcal{J}$ is not twice differentiable (and even not differentiable at all). We first overcome the explicit use of the Hessian matrix $\nabla^2\mathcal{J}$ by introducing an auxiliary variable $\psi : \mathbb{R}_+ \to \mathbb{R}^P$ like in Alvarez et al. (2002). Consider the following dynamical system,

$$\begin{cases} \dot{\theta}(t) + \beta\nabla\mathcal{J}(\theta(t)) & +(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t) = 0 \\ \dot{\psi}(t) & +(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t) = 0 \end{cases}, \quad \text{for a.e. } t \in (0, +\infty). \tag{2.6}$$

This system is well defined even if $\mathcal{J}$ is only once differentiable. Additionally, as explained in Alvarez et al. (2002), (2.5) is equivalent to (2.6) when $\mathcal{J}$ is twice differentiable. Indeed, one can rewrite (2.5) into (2.6) by introducing $\psi = -\beta\dot{\theta} - \beta^2\nabla\mathcal{J}(\theta) - (\alpha\beta - 1)\theta$. Conversely, one can substitute the first line of (2.6) into the second one to retrieve (2.5). Since (2.6) does not require the existence of second-order derivatives, it is a first-order generalization of DIN.

Let us now introduce a new version of (2.6) for non-differentiable functions. According to Rademacher's theorem, locally Lipschitz continuous functions are differentiable almost everywhere. Denote R the set of points where $\mathcal{J}$ is differentiable. Then, $\mathbb{R}^P \setminus \mathsf{R}$ has zero Lebesgue measure. It follows that for any $\theta^\star \in \mathbb{R}^P \setminus \mathsf{R}$, there exists a sequence of points in R whose limit is $\theta^\star$. Such limit sequences are useful for introducing the Clarke subdifferential (Clarke, 1990), defined next.

Definition 2.2 (Clarke subdifferential of locally Lipschitz functions)**.** *Let $g : \mathbb{R}^P \to \mathbb{R}$ be a locally Lipschitz continuous function. Since g is differentiable almost everywhere, denote $\mathsf{R} \subset \mathbb{R}^P$ the set of points where differentiability holds. The Clarke subdifferential of g at $\theta \in \mathbb{R}^P$, denoted by $\partial g(\theta)$, is the set defined by,*

$$\partial g(\theta) = \operatorname{conv}\left\{ v \in \mathbb{R}^P \mid \exists (\theta_k)_{k\in\mathbb{N}} \in \mathsf{R}^{\mathbb{N}}, \text{ such that } \theta_k \xrightarrow[k\to\infty]{} \theta \text{ and } \nabla g(\theta_k) \xrightarrow[k\to\infty]{} v \right\}, \tag{2.7}$$

where conv *denotes the convex hull operator. The elements of the Clarke subdifferential are*

called Clarke subgradients.

The Clarke subdifferential is a nonempty compact convex set. It coincides with the gradient for smooth functions and with the traditional subdifferential (1.14) for non-smooth convex functions. As previously mentioned, and contrarily to the gradient and (1.14), it does not enjoy a sum rule: the sum of the Clarke subdifferentials of functions is not equal to the Clarke subdifferential of the sum of functions in general (the latter is included in the former).

Thanks to Definition 2.2, we can extend (2.6) to non-differentiable functions. Since $\partial\mathcal{J}(\theta)$ is a set, we no longer study a differential equation but rather a *differential inclusion*, namely,

$$\begin{cases} \dot{\theta}(t) + \beta\partial\mathcal{J}(\theta(t)) & +(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t) \ni 0 \\ \dot{\psi}(t) & +(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t) \ni 0 \end{cases}, \quad \text{for a.e. } t \in (0, +\infty). \tag{2.8}$$

For a given initial condition $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P$, we call *solution* (or *trajectory*) of this system any absolutely continuous curve (θ, ψ) from $\mathbb{R}_+$ to $\mathbb{R}^P \times \mathbb{R}^P$ for which $(\theta(0), \psi(0)) = (\theta_0, \psi_0)$ and (2.8) holds. Absolute continuity amounts to the fact that θ is differentiable almost everywhere with integrable derivative and,

$$\theta(t) - \theta(0) = \int_0^t \dot{\theta}(s)\,\mathrm{d}s, \text{ for } t \in [0, +\infty).$$

Due to the properties of the Clarke subdifferential, existence of a solution to differential inclusions such as (2.8) is ensured, see Aubin and Cellina (2012); note however that uniqueness of the solution does not hold in general due to the set-valued nature of (2.8). We will now use the structure of (2.8) to build a new algorithm to train DNNs.

2.3.2 Discretization of the differential inclusion

To obtain the basic form of our algorithm, we discretize (2.8) according to the classical explicit Euler method. Given (θ, ψ) a solution of (2.8) and any time $t_k \geq 0$, set $\theta_k = \theta(t_k)$ and $\psi_k = \psi(t_k)$. Then, at time $t_{k+1} = t_k + \gamma_k$ with γ_k positive small, one can approximate $\dot{\theta}(t_{k+1})$ and $\dot{\psi}(t_{k+1})$ by

$$\dot{\theta}(t_{k+1}) \simeq \frac{\theta_{k+1} - \theta_k}{\gamma_k}, \qquad \dot{\psi}(t_{k+1}) \simeq \frac{\psi_{k+1} - \psi_k}{\gamma_k}.$$

For an initialization $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P$, this discretization yields the following algorithm, for all $k \in \mathbb{N}$,

$$\begin{cases} v_k & \in \partial \mathcal{J}(\theta_k) \\ \theta_{k+1} & = \theta_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k - \beta v_k \right) \\ \psi_{k+1} & = \psi_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k \right) \end{cases} \tag{2.9}$$

The algorithm above is well defined for non-smooth non-convex loss functions like $\mathcal{J}$, yet, it is not suited to train DNNs. First, numerically evaluating $\partial \mathcal{J}(\theta_k)$ is not possible in general since there is no operational calculus for the Clarke subdifferential; secondly, to cope with the computational cost of DL we need to introduce a mini-batch sub-sampling strategy similarly to the one for smooth functions (Section 1.3.4). This makes the absence of sum rule even more critical. The next section is meant to address these issues and to eventually design a practical algorithm, INNA.

2.3.3 INNA and a new notion of steady states

We consider again mini-batch sub-sampling, let $\mathsf{B} \subset \{1, \dots, N\}$, we recall the definition of $\mathcal{J}_\mathsf{B}$,

$$\mathcal{J}_\mathsf{B} \colon \theta \mapsto \sum_{n \in \mathsf{B}} \ell(f(x_n, \theta), y_n), \tag{2.10}$$

and we denote $\partial \mathcal{J}_\mathsf{B}$ the Clarke subdifferential of $\mathcal{J}_\mathsf{B}$. However, as we already said, unlike in the differentiable case, subdifferentials do not sum up to a subdifferential of the sum for non-convex non-smooth functions in general, that is

$$\partial \mathcal{J}_\mathsf{B}(\theta) \neq \sum_{n \in \mathsf{B}} \partial \ell(f(x_n, \theta), y_n).$$

A simple example is $t \in \mathbb{R} \mapsto |t| - |t| = 0$. The Clarke subdifferential of this function at $t = 0$ is $\{0\}$, whereas $\partial(|0|) + \partial(-|0|) = [-1, 1] + [-1, 1] = [-2, 2] \neq \{0\}$.

At this point we shall make an important precision. While the Clarke subdifferential does not enjoy a sum-rule, the automatic differentiation libraries mentioned in Section 1.3.3.2 implement backpropagation using smooth calculus rules even for non-smooth and non-convex loss functions. As a result, these implementations of backpropagation return objects that are not Clarke subgradients in general. Despite the lack of mathematical justifications, such practice provides satisfying results in many cases. Hence, in order to match this practice,

we introduce a notion of steady states. They correspond to the stationary points generated by the combination of mini-batch sub-sampling with the sum-rule failure of the Clarke subdifferential. As we shall see, this allows both for practical applications and convergence analysis. Doing so, we capture the real stationary points met in practice.

To this aim, and similarly to (1.18), we introduce the following objects, for any $\mathsf{B} \subset \{1, \dots, N\}$,

$$D\mathcal{J}_{\mathsf{B}} = \sum_{n \in \mathsf{B}} \partial \left[\ell(f(x_n, \cdot), y_n) \right], \quad D\mathcal{J} = \sum_{n=1}^{N} \partial \left[\ell(f(x_n, \cdot), y_n) \right]. \tag{2.11}$$

Observe that, for each B, we have $D\mathcal{J}_{\mathsf{B}} \supset \partial \mathcal{J}_{\mathsf{B}}$ and that $\mathcal{J}_{\mathsf{B}}$ is differentiable almost everywhere (again by Rademacher's theorem) with $D\mathcal{J}_{\mathsf{B}} = \partial \mathcal{J}_{\mathsf{B}} = \{\nabla \mathcal{J}_{\mathsf{B}}\}$, see Clarke (1990). This means in particular that $D\mathcal{J} = \partial \mathcal{J}$ almost everywhere so the set of points where our operator D does not coincide with the Clarke subgradient has zero measure.

A point satisfying $0 \in D\mathcal{J}(\theta)$ will be called *D-critical*. Note that Clarke-critical points ($0 \in \partial \mathcal{J}$) are D-critical points but that the converse is not true. This new operator D enjoys favorable properties: sum and chain rules hold along continuous curves (see Lemmas 2.2 and 2.3 below). Additionally, we will prove the existence of a tame Sard's lemma (see Lemma 2.4). To our knowledge, this notion of D-critical points (or steady states) has not been used previously in the literature and a direct approach modeling the mini-batch practice has never been considered before.[2] While this notion is needed for the theoretical analysis, one should keep in mind that the introduction of $D\mathcal{J}$ does not change the implementation of algorithms in practice provided that the automatic differentiation library returns Clarke subgradients of the functions $\mathcal{J}_n$, for $n \in \{1, \dots, N\}$.

Ultimately, we can now rewrite the differential inclusion (2.8) by replacing $\partial \mathcal{J}$ with $D\mathcal{J}$. This yields a differential inclusion adapted to study mini-batch approximations of non-smooth loss functions. It simply reads,

$$\begin{cases} \dot{\theta}(t) + \beta D\mathcal{J}(\theta(t)) & +(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t) \ni 0 \\ \dot{\psi}(t) & +(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t) \ni 0 \end{cases}, \quad \text{for a.e. } t \in (0, +\infty). \tag{2.12}$$

Discretizing this system gives a practical version of INNA. Let us consider a sequence $(\mathsf{B}_k)_{k \in \mathbb{N}}$ of nonempty subsets of $\{1, \dots, N\}$, chosen independently and uniformly at random

[2] Following a first version of the publication associated to this chapter (Castera et al., 2019b), Bolte and Pauwels (2020b) have further developed the present ideas and in particular the connection to the back-propagation algorithm.

with replacement (like for smooth functions), and a sequence of positive step-sizes $(\gamma_k)_{k\in\mathbb{N}}$. For a given initialization $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P$, at iteration $k \geq 0$, our algorithm reads,

$$\text{(INNA)} \quad \begin{cases} v_k & \in D\mathcal{J}_{\mathsf{B}_k}(\theta_k) \\ \theta_{k+1} & = \theta_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k - \beta v_k \right) \\ \psi_{k+1} & = \psi_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k \right) \end{cases} \tag{2.13}$$

Here again $\alpha \geq 0$ and $\beta > 0$ are hyper-parameters of the algorithm. INNA in its practical form is summarized in Algorithm 1.

Algorithm 1 INNA: an Inertial Newton Algorithm for deep learning

Objective function: $\mathcal{J} = \sum_{n=1}^{N} \mathcal{J}_n$, with $\mathcal{J}_n : \mathbb{R}^P \to \mathbb{R}$ locally Lipschitz continuous.
Input: $\alpha \geq 0$, $\beta > 0$, a sequence of step-sizes $(\gamma_k)_{k\in\mathbb{N}}$.
Input: nonempty mini-batches $(\mathsf{B}_k)_{k\in\mathbb{N}}$.
Initialize $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P$,
for $k = 0, \ldots$ **do**
  $v_k \in \sum_{n\in\mathsf{B}_k} \partial\left[\mathcal{J}_n(\theta_k)\right]$
  $\theta_{k+1} = \theta_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k - \beta v_k \right)$
  $\psi_{k+1} = \psi_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k \right)$
end for

Note finally that similarly to what we explained regarding SGD in Section 1.3.4, the mini-batch sub-sampling in INNA yields a stochastic approximation of the full-batch algorithm that one would obtain by choosing for all $k \in \mathbb{N}$, $\mathsf{B}_k = \{1, \ldots, N\}$, i.e., when $\mathcal{J}_{\mathsf{B}_k} = \mathcal{J}$. Indeed, the vectors v_k in (2.13) may be written as $v_k = \tilde{v}_k + \eta_k$, where $\tilde{v}_k \in D\mathcal{J}(\theta_k)$ and η_k compensates for the missing subgradients and can be seen as a zero-mean noise. Hence, INNA admits the following general abstract stochastic formulation,

$$\begin{cases} w_k & \in D\mathcal{J}(\theta_k) \\ \theta_{k+1} & = \theta_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k - \beta w_k + \xi_k \right) \\ \psi_{k+1} & = \psi_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k \right) \end{cases} \tag{2.14}$$

where $(\xi_k)_{k\in\mathbb{N}}$ is a sequence of random variables representing the noise like in Section 1.3.4. While (2.13) is the version implemented in practice, this equivalent form (2.14) is more convenient for the convergence analysis of the next section. We also point out that the

equivalence between (2.13) and (2.14) holds thanks to the use of $D\mathcal{J}$ and would not hold with $\partial\mathcal{J}$ as in (2.9).

2.4 Convergence results for INNA

We first state our main result regarding the convergence of INNA in DL.

2.4.1 Main result: accumulation points of INNA are critical

We now study the convergence of INNA. The main idea of the proof is to use the ODE argument to state that the discrete algorithm (2.13) asymptotically behaves like the solutions of the continuous differential inclusion (2.12). In addition to tameness and local Lipschitz continuity, we make the following assumptions, which can be easily ensured by practitioners.

Assumption 2.1 (Stochastic approximation)**.** *The sets* $(\mathsf{B}_k)_{k\in\mathbb{N}}$ *are taken independently uniformly at random with replacement. The step-size sequence* $(\gamma_k)_{k\in\mathbb{N}}$ *is positive with* $\sum_k \gamma_k = +\infty$ *and satisfies* $\gamma_k = o\left(\frac{1}{\log k}\right)$*, that is* $\limsup_{k\to+\infty} |\gamma_k \log k| = 0$.

Typical admissible choices are $\gamma_k = C(k+1)^{-a}$ with $a \in (0,1]$, $C > 0$. The main theoretical result for INNA follows.

Theorem 2.1 (INNA converges to the set of D-critical points of $\mathcal{J}$)**.** *Assume that for* $n \in \{1,\dots,N\}$*, each* $\mathcal{J}_n$ *is locally Lipschitz continuous, tame and that the step-sizes and mini-batches satisfy Assumption 2.1. Set an initial condition* (θ_0,ψ_0) *and assume that there exists* $C > 0$ *such that* $\sup_{k\geq 0} \|(\theta_k,\psi_k)\| \leq C$ *almost surely, where* $(\theta_k,\psi_k)_{k\in\mathbb{N}}$ *are generated by INNA. Then, almost surely, any accumulation point* $\bar{\theta}$ *of the sequence* $(\theta_k)_{k\in\mathbb{N}}$ *satisfies* $D\mathcal{J}(\bar{\theta}) \ni 0$*. In addition* $(\mathcal{J}(\theta_k))_{k\in\mathbb{N}}$ *converges.*

Before proving Theorem 2.1, we make some important comments and illustrate this result.

2.4.2 Comments on the results of Theorem 2.1

– **On the step-sizes.** First, Assumption 2.1 offers much more flexibility on the choice of the step-sizes than the usual Robbins-Monro condition (1.22) commonly used for SGD. This is due to the finite-sum structure of $\mathcal{J}$, the boundedness assumption and the local Lipschitz continuity which make the noise $(\xi_k)_{k\in\mathbb{N}}$ uniformly bounded and hence sub-Gaussian. Using Benaïm et al. (2005, Remark 1.5), this allows for much

larger step-sizes than in the more common "bounded second moment setting". This has an interest in practice as highlighted in Figure 2.4 of the experimental section.

– **On the scope of the theorem.** The results above are actually more general than for DL loss functions. They hold for any locally Lipschitz continuous tame function with finite-sum structure and for the general stochastic process (2.14). We do not use any other specific structure of DL loss functions. These results could be adapted for other assumptions on the noise, in view of Benaïm et al. (2005).

– **On D-criticality.** The result of Theorem 2.1 states that the bounded discrete trajectories of INNA are attracted by the D-critical points. The D-critical points include local minima, and we will actually prove that INNA is likely to avoid strict saddle points in the next chapter. This is also corroborated by the empirical observations of Section 2.6.2. Although $D\mathcal{J}$ coincides with $\partial\mathcal{J}$ almost everywhere, the notion of D-criticality cannot be ignored. Indeed, if the algorithm was initialized on the D-critical set, the algorithm would be stationary *even if the initialization is non-Clarke critical*. Hopefully, in practice one can expect to avoid such points with overwhelming probability. Indeed, following our introduction of D-critical points, Bolte and Pauwels (2020a) proved that SGD almost surely converges to the set of Clarke-critical points in practice. In other words, D-critical points that are not Clarke-critical are likely to be avoided by SGD (see also Bianchi et al. 2020). The same result can be hoped for INNA but have not been proved to this day.

– **On the boundedness assumption.** The boundedness assumption on the iterates is a classical assumption for first or second-order algorithms, see for instance D. Davis et al. (2020) and J. C. Duchi and Ruan (2018). When using deterministic algorithms (i.e., without mini-batch approximations), properties such as the coercivity[3] of $\mathcal{J}$ can be sufficient to remove the boundedness assumption for descent algorithms. This does not remain true when dealing with mini-batch sub-sampling. Yet, in the case of INNA, the coercivity of $\mathcal{J}$ would at least guarantee that the solutions of the continuous underlying differential inclusion (2.12) remain bounded. Indeed, we will prove in Section 2.4.4 that for any solution (θ, ψ) of (2.12), the function $t \geq 0 \mapsto 2(1 + \alpha\beta)\mathcal{J}(\theta(t)) + \left\|(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t)\right\|^2$ is decreasing in time (see Lemma 2.6 hereafter). As a consequence, we cannot have $\mathcal{J}(\theta(t)) \xrightarrow[t\to\infty]{} \infty$ so the coercivity of $\mathcal{J}$ would guarantee that $\|\theta(t)\| \not\to \infty$. Similarly, we would have $\|\psi(t)\| \not\to \infty$ as well. However, DL loss functions are not coercive in general and ensuring the boundedness assumption in DL or even for non-convex semi-algebraic optimization problems is far beyond the

[3] $\mathcal{J}$ is said to be coercive if $\lim_{\|\theta\|\to\infty} \mathcal{J}(\theta) = +\infty$.

scope of this thesis. Alternatively, we could have projected the iterates on a possibly very large compact ball to ensure boundedness. Adding such a projection would however imply to adapt the proof of convergence substantially, in particular the results of Benaïm et al. (2005).

2.4.3 Preliminary variational results

Prior to proving Theorem 3, we extend some results known for the Clarke subdifferential of tame functions to the operator D that we previously introduced. First, we recall a useful result of D. Davis et al. (2020) which follows from the projection formula in Bolte et al. (2007b).

Lemma 2.2 (Chain rule for the Clarke subdifferential)**.** *Let $\mathcal{J} : \mathbb{R}^P \to \mathbb{R}$ be a locally Lipschitz continuous tame function, then $\mathcal{J}$ admits a chain rule, meaning that for any absolutely continuous curve $\theta : \mathbb{R}_+ \to \mathbb{R}^P$, $\mathcal{J} \circ \theta$ is differentiable a.e. and for a.e. $t \geq 0$,*

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}t}(\theta(t)) = \langle \dot{\theta}(t), \partial\mathcal{J}(\theta(t)) \rangle = \langle \dot{\theta}(t), v \rangle, \quad \forall v \in \partial\mathcal{J}(\theta(t)). \tag{2.15}$$

Note that the function $t \geq 0 \mapsto \mathcal{J}(\theta(t))$ is differentiable for a.e. $t > 0$, despite the non-differentiability of $\mathcal{J}$ (and possibly θ). This holds thanks to the absolute continuity of θ and the chain rule above. Additionally, notice that the value of $\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}t}(\theta(t))$ in (2.15) does not depend on the choice of the element v taken in $\partial\mathcal{J}(\theta(t))$, this justifies the notation $\langle \dot{\theta}(t), \partial\mathcal{J}(\theta(t)) \rangle$.

We now prove a very similar chain-rule for the operator D. Consider again a function with an additive finite-sum structure, $\mathcal{J} = \sum_{n=1}^N \mathcal{J}_n$, where again each $\mathcal{J}_n : \mathbb{R}^P \to \mathbb{R}$ is locally Lipschitz continuous and tame, and recall that for any $\theta \in \mathbb{R}^P$, we have the following: $D\mathcal{J}(\theta) = \sum_{n=1}^N \partial\mathcal{J}_n(\theta)$. The following lemma is a direct generalization of the above chain rule.

Lemma 2.3 (Chain rule for $D\mathcal{J}$)**.** *Let $\mathcal{J}$ be a sum of tame functions as described above. Let $\Gamma : [0,1] \to \mathbb{R}^P$ be an absolutely continuous curve so that $t \mapsto \mathcal{J}(\Gamma(t))$ is differentiable almost everywhere. For a.e. $t \in [0,1]$, and for all $v \in D\mathcal{J}(\Gamma(t))$,*

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{J}(\Gamma(t)) = \left\langle v, \dot{\Gamma}(t) \right\rangle.$$

Proof. By local Lipschitz continuity and absolute continuity, each $\mathcal{J}_n$ is differentiable almost everywhere and Lemma 2.2 can be applied:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{J}_n(\Gamma(t)) = \left\langle v_n, \dot{\Gamma}(t) \right\rangle, \text{for all } v_n \in \partial\mathcal{J}_n(\Gamma(t)) \text{ and for a.e. } t \geq 0.$$

Thus

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{J}(\Gamma(t)) = \sum_{n=1}^{N} \frac{\mathrm{d}}{\mathrm{d}t}\mathcal{J}_n(\Gamma(t)) = \sum_{n=1}^{N} \left\langle v_n, \dot{\Gamma}(t) \right\rangle,$$

for any $v_n \in \partial\mathcal{J}_n(\Gamma(t))$, for all $n = \{1, \ldots, N\}$, and for a.e. $t \geq 0$. This proves the desired result. □

We finish this section with a Sard lemma for D-critical values, which is an adaptation of the Sard lemma of Bolte et al. (2007b) for the Clarke subdifferential.

Lemma 2.4 (A Sard's theorem for tame D-critical values)**.** *Let,*

$$D\text{-}crit \stackrel{\text{def}}{=} \left\{\theta \in \mathbb{R}^P \mid D\mathcal{J}(\theta) \ni 0\right\},$$

then $\mathcal{J}(D\text{-}crit)$ *is finite.*

Proof. The set D-crit is tame and hence it has a finite number of connected components. It is sufficient to prove that $\mathcal{J}$ is constant on each connected component of D-crit. Hence, without loss of generality, assume that D-crit is connected and consider $\theta_0, \theta_1 \in D$-crit. By Whitney regularity (Van den Dries, 1998, Chapter 3), there exists a tame continuous path Γ joining θ_0 to θ_1. Since Γ is tame, the monotonicity lemma (see for example Kurdyka 1998, Lemma 2) states the existence of a finite collection of real numbers $0 = a_0 < a_1 < \ldots < a_q = 1$, such that Γ is C^1 on each segment (a_{j-1}, a_j), $j = 1, \ldots, q$. We can then apply Lemma 2.3 to each part $\Gamma_{|(a_i, a_{i+1})}$ of the path, and since the image of Γ is included in D-crit, the derivative $\frac{\mathrm{d}}{\mathrm{d}t}(\mathcal{J} \circ \Gamma)$ is zero on each (a_i, a_{i+1}). So, $\mathcal{J} \circ \Gamma$ is constant except perhaps on the finite set of points $(a_j)_{j \in \{1,\ldots,q\}}$, so it is constant by continuity. Hence, $\mathcal{J}(\theta_0) = \mathcal{J}(\Gamma(0)) = \mathcal{J}(\Gamma(1)) = \mathcal{J}(\theta_1)$. □

2.4.4 Proof of convergence for INNA

To prove the convergence of INNA, we follow the stochastic method for differential inclusions developed in Benaïm et al. (2005) which relies on the analysis of the differential system

(2.12). The steady states (or stationary points) of (2.12) are given by,

$$\mathsf{S} = \left\{(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P \mid 0 \in D\mathcal{J}(\theta), \psi = (1 - \alpha\beta)\theta\right\}. \tag{2.16}$$

These points are initialization values for which the system does not evolve and remains constant. Observe that the first coordinates of these points are D-critical for $\mathcal{J}$ and that conversely any D-critical point of $\mathcal{J}$ corresponds to a unique steady state in S. We now define an important tool for studying differential inclusions.

Definition 2.3 (Lyapunov function)**.** *Let V be a subset of $\mathbb{R}^P \times \mathbb{R}^P$, we say that $E : \mathbb{R}^P \times \mathbb{R}^P \to \mathbb{R}$ is a Lyapunov function for the set V and the dynamics (2.12) if,*

(i) For any solution (θ, ψ) of (2.12) with initial condition $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P$, we have: $E(\theta(t), \psi(t)) \leq E(\theta_0, \psi_0)$ for a.e. $t \geq 0$.

(ii) For any solution (θ, ψ) of (2.12) with initial condition $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P \setminus \mathsf{V}$, we have:
$E(\theta(t), \psi(t)) < E(\theta_0, \psi_0)$ for a.e. $t \geq 0$.

In the mechanical formalism discussed in the introduction, a Lyapunov function represents an energy that does not increase over time. Formally, the ball evolving on the landscape of $\mathcal{J}$ loses energy and hence must eventually slow down. In practice, to establish that a function is a Lyapunov, one proves that it is decreasing by differentiating with the chain rules stated above. In the context of INNA, we will use Lemma 2.3. In order to build a Lyapunov function for the dynamics (2.12) and the set S, for (θ, ψ) solution of (2.12), consider the two following energy-like functions,

$$\begin{cases} E_{\min}(\theta(t), \psi(t)) &= (1 - \sqrt{\alpha\beta})^2 \mathcal{J}(\theta(t)) + \frac{1}{2}\left\|(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t)\right\|^2 \\ E_{\max}(\theta(t), \psi(t)) &= (1 + \sqrt{\alpha\beta})^2 \mathcal{J}(\theta(t)) + \frac{1}{2}\left\|(\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t)\right\|^2. \end{cases} \tag{2.17}$$

Then the following lemma applies.

Lemma 2.5 (Differentiation along DIN trajectories)**.** *Let (θ, ψ) be a solution of (2.12) with initial condition $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P$. For a.e. $t > 0$, θ and ψ are differentiable at t, (2.12) holds, $\frac{\dot\theta(t) - \dot\psi(t)}{\beta} \in D\mathcal{J}(\theta(t))$ and*

$$\begin{aligned} \frac{\mathrm{d}E_{\min}}{\mathrm{d}t}(\theta(t), \psi(t)) &= -\left\|\sqrt{\alpha}\dot\theta(t) - \frac{1}{\sqrt{\beta}}\left(\dot\psi(t) - \dot\theta(t)\right)\right\|^2 \\ \frac{\mathrm{d}E_{\max}}{\mathrm{d}t}(\theta(t), \psi(t)) &= -\left\|\sqrt{\alpha}\dot\theta(t) + \frac{1}{\sqrt{\beta}}\left(\dot\psi(t) - \dot\theta(t)\right)\right\|^2 \end{aligned}$$

Proof. Define $E_\lambda(\theta,\psi) = \lambda\mathcal{J}(\theta) + \frac{1}{2}\left\|(\alpha-\frac{1}{\beta})\theta + \frac{1}{\beta}\psi\right\|^2$. We aim to choose λ so that E_λ is a Lyapunov function. Because $\mathcal{J}$ is tame and locally Lipschitz continuous, using Lemma 2.3 we know that for any absolutely continuous trajectory $\theta : \mathbb{R}_+ \to \mathbb{R}^P$ and for a.e. $t > 0$,

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}t}(\theta(t)) = \langle \dot\theta(t), D\mathcal{J}(\theta(t))\rangle = \langle \dot\theta(t), v(t)\rangle, \quad \forall v(t) \in D\mathcal{J}(\theta(t)). \tag{2.18}$$

Let θ and ψ be solutions of (2.12). For a.e. $t \geq 0$, we can differentiate $E_\lambda(\theta,\psi)$ to obtain

$$\begin{aligned}\frac{\mathrm{d}E_\lambda}{\mathrm{d}t}(\theta(t),\psi(t)) =& \lambda\langle\dot\theta(t), v(t)\rangle + (\alpha-\frac{1}{\beta})\langle\dot\theta(t), (\alpha-\frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t)\rangle \\ &+ \frac{1}{\beta}\langle\dot\psi(t), (\alpha-\frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t)\rangle,\end{aligned} \tag{2.19}$$

for all $v(t) \in D\mathcal{J}(\theta(t))$. Using (2.12), we get $\frac{1}{\beta}(\dot\theta(t) - \dot\psi(t)) \in D\mathcal{J}(\theta(t))$ and $-\dot\psi(t) = (\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t)$ a.e. Choosing $v(t) = \frac{1}{\beta}(\dot\theta(t) - \dot\psi(t))$ yields:

$$\frac{\mathrm{d}E_\lambda}{\mathrm{d}t}(\theta(t),\psi(t)) = \lambda\left\langle\dot\theta(t), \frac{\dot\theta(t)-\dot\psi(t)}{\beta}\right\rangle - (\alpha-\frac{1}{\beta})\left\langle\dot\theta(t),\dot\psi(t)\right\rangle - \frac{1}{\beta}\left\langle\dot\psi(t),\dot\psi(t)\right\rangle.$$

Then, expressing everything as a function of $\dot\theta$ and $\frac{1}{\beta}(\psi-\theta)$, one can show that a.e. on $\mathbb{R}_+$:

$$\begin{aligned}\frac{\mathrm{d}E_\lambda}{\mathrm{d}t}(\theta(t),\psi(t)) &= -\alpha\|\dot\theta(t)\|^2 - \beta\left\|\frac{\dot\theta(t)-\dot\psi(t)}{\beta}\right\|^2 + (\lambda-\alpha\beta-1)\,\langle\dot\theta(t), \frac{\dot\theta(t)-\dot\psi(t)}{\beta}\rangle \\ &= -\left\|\sqrt{\alpha}\dot\theta(t) + \frac{\alpha\beta+1-\lambda}{2\sqrt{\alpha}}\frac{\dot\theta(t)-\dot\psi(t)}{\beta}\right\|^2 - \left(\beta - \frac{(\alpha\beta+1-\lambda)^2}{4\alpha}\right)\left\|\frac{\dot\theta(t)-\dot\psi(t)}{\beta}\right\|^2.\end{aligned}$$

We aim to choose λ so that E_λ is decreasing that is $\left(\beta - \frac{(\alpha\beta+1-\lambda)^2}{4\alpha}\right) > 0$. This holds whenever $\lambda \in [(1-\sqrt{\alpha\beta})^2, (1+\sqrt{\alpha\beta})^2]$. We choose $\lambda_{\min} = (1-\sqrt{\alpha\beta})^2$, and $\lambda_{\max} = (1+\sqrt{\alpha\beta})^2$, for these two values we obtain for a.e. $t > 0$,

$$\begin{cases}\dot E_{\lambda_{\min}}(\theta(t),\psi(t)) &= -\left\|\sqrt{\alpha}\dot\theta(t) + \frac{1}{\sqrt{\beta}}\left(\dot\theta(t)-\dot\psi(t)\right)\right\|^2 \\ \dot E_{\lambda_{\max}}(\theta(t),\psi(t)) &= -\left\|\sqrt{\alpha}\dot\theta(t) - \frac{1}{\sqrt{\beta}}\left(\dot\theta(t)-\dot\psi(t)\right)\right\|^2\end{cases} \tag{2.20}$$

Remark finally that by definition $E_{\min} = E_{\lambda_{\min}}$ and $E_{\max} = E_{\lambda_{\max}}$. □

Recall that $\mathsf{S} = \left\{(\theta,\psi) \in \mathbb{R}^P \times \mathbb{R}^P \mid 0 \in D\mathcal{J}(\theta), \psi = (1-\alpha\beta)\theta\right\}$ and define $E = E_{\min} +$

$E_{\max}$. By a direct integration argument, we obtain the following lemma.

Lemma 2.6 (E is Lyapunov function for INNA with respect to S)**.** *For all $(\theta_0, \psi_0) \notin \mathsf{S}$ and for any solution (θ, ψ) with initial condition $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P$,*

$$E(\theta(t), \psi(t)) < E(\theta_0, \psi_0), \text{ for a.e. } t > 0. \tag{2.21}$$

We are now in position to provide the desired proof.

Proof of Theorem 2.1. Lemmas 2.5 and 2.6 state that E is a Lyapunov function for the set S and the dynamics (2.12). Let $\mathsf{S}_1 = \{\theta \in \mathbb{R}^P \mid (\theta, \psi) \in \mathsf{S}\}$ which is exactly the set of D-critical points of $\mathcal{J}$. Using Lemma 2.4 of Section 2.4.3, $\mathcal{J}(\mathsf{S}_1)$ is finite. Moreover, since $E(\theta, \psi) = 2(1 + \alpha\beta)\mathcal{J}(\theta)$ for all $(\theta, \psi) \in \mathsf{S}$, $E(\mathsf{S})$ is finite as well, so in particular, $E(\mathsf{S})$ has empty interior.

Denote by L the set of accumulation points of the sequences $((\theta_k, \psi_k))_{k\in\mathbb{N}}$ produced by (2.13) starting at (θ_0, ψ_0) and denote L_1 the projection of L on $\mathbb{R}^P \times \{0\}$. We have the 3 following properties:

- By assumption, we have $\|(\theta_k, \psi_k)\| \leq C$ almost surely, for all $k \in \mathbb{N}$.
- By local Lipschitz continuity, for any $\|\theta\| \leq C$ and any $\mathsf{B} \subset \{1, \ldots, N\}$, $D\mathcal{J}_{\mathsf{B}}(\theta)$ is uniformly bounded , hence the centered noise $(\xi_k)_{k\in\mathbb{N}}$ is a uniformly bounded martingale difference sequence.
- By Assumption 2.1, the sequence $(\gamma_k)_{k\in\mathbb{N}}$ is chosen such that $\gamma_k = o(\frac{1}{\log k})$ (see Section 2.4.2).

Then the sufficient conditions provided in Remark 1.5 of Benaïm et al. (2005) state that the discrete process $(\theta_k, \psi_k)_{k\in\mathbb{N}}$ asymptotically behaves like the solutions of (2.12). We can then combine Proposition 3.27 and Theorem 3.6 of Benaïm et al. (2005), to deduce that the limit set L of the discrete process is contained in the set S where the Lyapunov function E has vanishing derivatives. Thus, the set L_1 (the set of the first coordinates of all accumulation points) contains only D-critical points of $\mathcal{J}$. In addition, $E(\mathsf{L})$ is a singleton, and for all $(\theta, \psi) \in \mathsf{S}$, we have $E(\theta, \psi) = 2(1+\alpha\beta)\mathcal{J}(\theta)$, so $\mathcal{J}(\mathsf{L}_1)$ is also a singleton and the theorem follows.

□

We now have a practical new second-order algorithm for training DNNs with convergence guarantees both for the iterates and the sequence of values of the loss function. Before presenting numerical experiments for INNA, we derive rates of convergence for the solutions

of the differential inclusion from which INNA is built.

2.5 Towards convergence rates for INNA

In the previous section, connecting INNA to the asymptotic behavior of the solutions of (2.12) was one of the keys to prove the convergence of the discrete algorithm. We now turn our attention to the continuous dynamical system: we focus on (2.8)—we no longer use (2.12) and $D\mathcal{J}$ although this would be possible but would require more technical proofs. In this section and in this section only, we pertain to loss functions $\mathcal{J}$ that are real semi-algebraic.[4] Semi-algebraic functions are a particular type of tame functions: a set is called semi-algebraic if it is a finite union of sets of the form,

$$\{\theta \in \mathbb{R}^P \mid \mathcal{Q}(\theta) = 0, \mathcal{Q}_i(\theta) < 0\}$$

where $\mathcal{Q}, \mathcal{Q}_i$ are real polynomial functions. A function is called semi-algebraic if its graph is semi-algebraic.

We will characterize the convergence rate of the solutions of the continuous-time system (2.8) to critical points. Let us first introduce an essential mechanism to obtain such convergence rates: the Kurdyka-Łojasiewicz (KL) property.

2.5.1 The non-smooth Kurdyka-Łojasiewicz property for the Clarke subdifferential

The non-smooth Kurdyka-Łojasiewicz (KL) property, as introduced in (Bolte et al., 2010), is a measure of "amenability to sharpness" (as illustrated at the end of Section 2.5.3). Here we state a uniform version for the Clarke subdifferential of semi-algebraic functions following Bolte et al. (2007b) and Bolte et al. (2014). In the sequel we denote by "dist" any given distance on $\mathbb{R}^P$.

Lemma 2.7 (Uniform non-smooth KL property for the Clarke subdifferential)**.** *Let* K *be a nonempty compact set and let* $G : \mathbb{R}^P \to \mathbb{R}$ *be a semi-algebraic locally Lipschitz continuous function. Assume that* G *is constant on* K*, with value* $G^\star$*. Then there exist* $\varepsilon > 0$*,* $\delta > 0$*,*

[4] We could extend the results of this section to more general objects including analytic functions on bounded sets. The semi-algebraicity assumption is made here for the sake of clarity.

$a \in (0,1)$ and $\rho > 0$ such that, for all

$$v \in \left\{ v \in \mathbb{R}^P \mid \operatorname{dist}(v, \mathsf{K}) < \varepsilon \right\} \cap \left\{ v \in \mathbb{R}^P \mid G^\star < G(v) < G^\star + \delta \right\},$$

it holds that,

$$\rho(1-a)\left(G(v) - G^\star\right)^{-a} \operatorname{dist}\left(0, \partial G(v)\right) > 1. \tag{2.22}$$

In the sequel, we make an abuse of notation by writing $\|\partial \mathcal{J}(\cdot)\| \stackrel{\text{def}}{=} \operatorname{dist}(0, \partial \mathcal{J}(\cdot))$. To obtain a convergence rate, we will use inequality (2.22) on the Lyapunov function E used to prove the convergence of INNA. Before doing so we prove a general result of convergence that is built around the KL property and that can be applied not only to (2.8) but also to other dynamical systems.

2.5.2 A general asymptotic rate

We state a general theorem that leads to the existence of a convergence rate. This theorem will hold in particular for (2.8). We start with the result.

Theorem 2.8. *Let $X : [0, +\infty) \to \mathbb{R}^P$ be a bounded absolutely continuous trajectory and let $G : \mathbb{R}^P \to \mathbb{R}$ be a semi-algebraic locally Lipschitz continuous function. If there exists $c_1 > 0$ such that for a.e. $t > 0$,*

$$\frac{\mathrm{d}G}{\mathrm{d}t}(X(t)) \leq -c_1 \|(\partial G)(X(t))\|^2, \tag{i}$$

then $G(X(t))$ converges to a limit value $G^\star$ and,

$$|G(X(t)) - G^\star| = O\left(\frac{1}{t}\right).$$

If in addition there exists $c_2 > 0$ such that for a.e. $t > 0$,

$$c_2 \|\dot{X}(t)\| \leq \|(\partial G)(X(t))\|, \tag{ii}$$

then, X converges to a critical point of G with a rate[5] of the form $O(1/t^b)$ with $b > 0$.

Proof. We first prove the convergence of $t \geq 0 \mapsto G(X(t))$. Suppose that (i) holds. Since X is bounded and G is continuous, $G(X(\cdot))$ is bounded. Moreover, from (i), $G(X(\cdot))$ is

[5] In some cases we even have linear rates or finite-time convergence as detailed in the proof.

decreasing, so it converges to some value $G^\star$. To simplify suppose $G \geq 0$ and $G^\star = 0$. Define,

$$\mathsf{I} = \left\{ x \in \mathbb{R}^P \;\middle|\; G(x) = 0 \right\}.$$

Suppose first that there exists $s \geq 0$, such that $X(s) \in \mathsf{I}$. Since $G(X(\cdot))$ is decreasing with limit 0, then for all $t \geq s$, $G(X(t)) = 0$ and the convergence rate holds true.

Let us thus assume that for all $t \geq 0$, $G(X(t)) > 0$. The trajectory X is bounded in $\mathbb{R}^P$, hence there exists a compact set $\mathsf{C} \subset \mathbb{R}^P$ such that $X(t) \in \mathsf{C}$ for all $t \geq 0$. Define $\mathsf{K} = \mathsf{I} \cap \mathsf{C}$. It is a compact set since I is closed (by continuity of G) and C is compact. Moreover, G is constant on K. As such by Lemma 2.7, there exist $\varepsilon > 0$, $\delta > 0$, $a \in (0, 1)$ and a constant $\rho > 0$ such that for all

$$v \in \left\{ v \in \mathbb{R}^P, \operatorname{dist}(v, \mathsf{K}) < \varepsilon \right\} \cap \left\{ 0 < G(v) < \delta \right\},$$

it holds that

$$\rho(1-a)\left(G(v)\right)^{-a} \operatorname{dist}\left(0, \partial G(v)\right) > 1.$$

We have $G(X(t)) \to 0$ so there exists $t_0 \geq 0$ such that for all $t \geq t_0$, $0 < G(X(t)) < \delta$. Without loss of generality, we assume $t_0 = 0$. Similarly, we have $\operatorname{dist}(X(t), K) \to 0$, so we may assume that for all $t \geq 0$, $\operatorname{dist}\left(X(t), \mathsf{K}\right) < \varepsilon$. Thus, for all $t \geq 0$,

$$\rho(1-a) G(X(t))^{-a} \|\partial G(X(t))\| > 1.$$

Going back to assumption (i), for a.e. $t > 0$, it holds that

$$\frac{\mathrm{d}G}{\mathrm{d}t}(X(t)) \leq -c_1 \|(\partial G)(X(t))\|^2,$$

but the KL property implies that for a.e. $t > 0$,

$$-\|\partial G(X(t))\|^2 < -\frac{1}{\rho^2(1-a)^2} G(X(t))^{2a}.$$

Therefore,

$$\frac{\mathrm{d}G}{\mathrm{d}t}(X(t)) < -\frac{c_1}{\rho^2(1-a)^2} G(X(t))^{2a}, \tag{2.23}$$

this is a differential inequality with respect to the function $G(X(\cdot))$. We consider two cases depending on the value of a. If $0 < a \leq 1/2$, then for large-enough $t \geq 0$, it holds

$G(X(t)) < 1$ so $-G(X(t))^{2a} < -G(X(t))$ and hence,

$$\frac{\mathrm{d}G}{\mathrm{d}t}(X(t)) < -\frac{c_1}{\rho^2(1-a)^2}G(X(t)),$$

so we obtain a linear rate. When $1/2 < a < 1$, we go back to (2.23), remark that for a.e. $t > 0$,

$$G(X(t))^{-2a}\frac{\mathrm{d}}{\mathrm{d}t}G(X(t)) = \frac{1}{1-2a}\frac{\mathrm{d}}{\mathrm{d}t}G(X(t))^{1-2a} < -\frac{c_1}{\rho^2(1-a)^2}, \tag{2.24}$$

with $1-2a < 0$. We can integrate (2.24) from 0 to $t > 0$:

$$G(X(t))^{1-2a} > \frac{(2a-1)c_1}{\rho^2(1-a)^2}t + G(X(0))^{1-2a} > \frac{(2a-1)c_1}{\rho^2(1-a)^2}t.$$

Since $\frac{1}{1-2a} < -1$, one obtains a convergence rate of the form $O\left(t^{\frac{1}{1-2a}}\right)$. In both cases the rate is at least $O\left(\frac{1}{t}\right)$.

We assume now that both (i) and (ii) hold and prove the convergence of the trajectory X with a convergence rate. Let $t > s > 0$, by the fundamental theorem of calculus (provided by the absolute continuity of X) and the triangular inequality,

$$\|X(t) - X(s)\| \leq \left\|\int_s^t \dot{X}(\tau)\,\mathrm{d}\tau\right\| \leq \int_s^t \|\dot{X}(\tau)\|\,\mathrm{d}\tau. \tag{2.25}$$

We wish to bound $\|\dot{X}\|$ using G. Using the chain rule (Lemma 2.2 of Section 2.4.3), for a.e. $\tau > 0$,

$$\frac{\mathrm{d}}{\mathrm{d}\tau}G(X(\tau))^{1-a} = (1-a)G(X(\tau))^{-a}\langle \dot{X}(\tau), (\partial G)(X(\tau))\rangle. \tag{2.26}$$

Then, from (i), we deduce that for a.e. $\tau > 0$,

$$\langle \dot{X}(\tau), (\partial G)(X(\tau))\rangle = \frac{\mathrm{d}G}{\mathrm{d}\tau}(X(\tau)) \leq -c_1\|(\partial G)(X(\tau))\|^2, \tag{2.27}$$

so

$$\frac{\mathrm{d}}{\mathrm{d}\tau}G(X(\tau))^{1-a} \leq -c_1(1-a)G(X(\tau))^{-a}\|(\partial G)(X(\tau))\|^2. \tag{2.28}$$

The KL property (2.22) implies that for a.e. $\tau > 0$,

$$-(1-a)G(X(\tau))^{-a}\|(\partial G)(X(\tau))\| < -\frac{1}{\rho}. \tag{2.29}$$

Putting this in (2.28) and using assumption (ii) we finally obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}G(X(\tau))^{1-a} < -\frac{c_1}{\rho}\|(\partial G)(X(\tau))\| \leq -\frac{c_1 c_2}{\rho}\|\dot{X}(\tau)\|. \tag{2.30}$$

We can use that in (2.25),

$$\begin{aligned}\|X(t) - X(s)\| &\leq -\frac{\rho}{c_1 c_2}\int_s^t \frac{\mathrm{d}}{\mathrm{d}t}G(X(\tau))^{1-a}\,\mathrm{d}\tau \\ &= \frac{\rho}{c_1 c_2}(G(X(s))^{1-a} - G(X(t))^{1-a}).\end{aligned} \tag{2.31}$$

Then, using the convergence rate that we already proved for G, we deduce that the Cauchy criterion holds for X inside the compact (hence complete) subset $\mathsf{C} \subset \mathbb{R}^P$ containing the trajectory, so X converges. Then from (i) and the convergence of X, we have that $\liminf_{t\to+\infty} \|\partial G(X(t))\| = 0$ because ∂G has closed graph. This shows that the limit of X is a critical point of G. Finally, taking the limit in (2.31) and using the convergence rate of G we obtain a rate for X as well. □

Remark 2.9. *Theorem 2.8 takes the form of a general recipe to obtain a convergence rate since it may be applied in many cases, to curves or flows, provided that a convenient Lyapunov function is given. Note also that it is sufficient for assumptions (i) and (ii) to hold only after some time $t_0 > 0$ as in such case, one could simply do a time shift to use the theorem.*

2.5.3 Application to INNA

We now apply Theorem 2.8 to the deterministic continuous dynamical system (2.8) from which INNA is built.

Theorem 2.10 (Convergence rates)**.** *Suppose that $\mathcal{J}$ is semi-algebraic locally Lipschitz continuous and lower bounded. Then, any bounded trajectory (θ, ψ) that solves (2.8) converges to a point $(\bar{\theta}, \bar{\psi}) \in \mathsf{S}$, with a convergence rate of the form $O\left(t^{-b}\right)$ with $b > 0$. Moreover, $\mathcal{J}(\theta(t))$ converges to its limit $\bar{\mathcal{J}}$ with rate $\left|\mathcal{J}(\theta(t)) - \bar{\mathcal{J}}\right| = O\left(\frac{1}{t}\right)$.*

Proof. Let (θ, ψ) be a bounded solution of (2.8). We would like to use Theorem 2.8 with $X = (\theta, \psi)$, and a well-chosen function. Recall the Lyapunov function introduced in the proof of Theorem 2.1: $E(\theta_1, \theta_2) = 2(1 + \alpha\beta)\mathcal{J}(\theta_1) + \left\|(\alpha - \frac{1}{\beta})\theta_1 + \frac{1}{\beta}\theta_2\right\|^2$, for all $(\theta_1, \theta_2) \in \mathbb{R}^P \times \mathbb{R}^P$. We proved a descent property for E along the solutions of (2.12), this holds also

for the solutions of (2.8) since for all $\theta_1 \in \mathbb{R}^P$, $\partial \mathcal{J}(\theta_1) \subset D\mathcal{J}(\theta_1)$. Due to the properties of $\mathcal{J}$, the function E is semi-algebraic and locally Lipschitz continuous, so it remains to prove that (i) and (ii) hold for E along the solution (θ, ψ) of (2.8).

For $t \geq 0$, denote $w(t) = (\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t)$, then according to Lemma 2.5 for a.e. $t > 0$,

$$\begin{aligned}\frac{\mathrm{d}E}{\mathrm{d}t}(\theta(t), \psi(t)) &= -\|\sqrt{\alpha}\dot{\theta}(t) - \frac{1}{\sqrt{\beta}}\left(\dot{\psi}(t) - \dot{\theta}(t)\right)\|^2 - \|\sqrt{\alpha}\dot{\theta}(t) + \frac{1}{\sqrt{\beta}}\left(\dot{\psi}(t) - \dot{\theta}(t)\right)\|^2 \\ &= -2\alpha\|\dot{\theta}(t)\|^2 - \frac{2}{\beta}\|\dot{\psi}(t) - \dot{\theta}(t)\|^2 = -2\alpha\|\dot{\theta}(t)\|^2 - \frac{2}{\beta}\|\beta\partial\mathcal{J}(\theta(t))\|^2 \\ &= -2\alpha\| - \beta\partial\mathcal{J}(\theta(t)) - w(t)\|^2 - 2\beta\|\partial\mathcal{J}(\theta(t))\|^2. \end{aligned} \tag{2.32}$$

On the other hand, by standard results on the sum of differentiable and non-differentiable functions, we have for all $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$,

$$\partial E(\theta, \psi) = 2\begin{pmatrix} (1 + \alpha\beta)\partial\mathcal{J}(\theta) + (\alpha - \frac{1}{\beta})\left((\alpha - \frac{1}{\beta})\theta + \frac{1}{\beta}\psi\right) \\ \frac{1}{\beta}\left((\alpha - \frac{1}{\beta})\theta + \frac{1}{\beta}\psi\right) \end{pmatrix}, \tag{2.33}$$

so for a.e. $t > 0$,

$$\frac{\|\partial E(\theta(t), \psi(t))\|^2}{4} = \left\|(1 + \alpha\beta)\partial\mathcal{J}(\theta(t)) + (\alpha - \frac{1}{\beta})w(t)\right\|^2 + \left\|\frac{1}{\beta}w(t)\right\|^2. \tag{2.34}$$

We wish to find $c_1 > 0$, such that $\frac{1}{2}\frac{\mathrm{d}E}{\mathrm{d}t} + \frac{c_1}{4}\|\partial E\|^2 < 0$. This follows from the following claim.

Claim: let $r_1 > 0$, $r_2 \in \mathbb{R}$, $r_3 > 0$, then there exist C_1 and C_2 two positive constants such that for any $a, b \in \mathbb{R}$,

$$C_1(a^2 + b^2) \leq (r_1 a + r_2 b)^2 + r_3 b^2 \leq C_2(a^2 + b^2). \tag{2.35}$$

Indeed, the function $Q : (a, b) \mapsto (r_1 a + r_2 b)^2 + r_3 b^2$ is a positive definite quadratic form, C_1 and C_2 can be taken to be two eigenvalues of the positive definite matrix which represents Q. Hence, (2.35) holds for all a and b.

Applying the previous claim to (2.32) and (2.34) leads to the existence of $c_1 > 0$ such that for a.e. $t > 0$,

$$\frac{\mathrm{d}E}{\mathrm{d}t}(\theta(t), \psi(t)) \leq -c_1\|\partial E(\theta(t), \psi(t))\|^2,$$

so assumption (i) holds for INNA.

It now remains to show that (ii) of Theorem 2.8 holds i.e., that there exists $c_2 > 0$ such that for the solution (θ, ψ) of (2.8) and for a.e. $t > 0$, $\|\partial E(\theta(t), \psi(t))\|^2 \geq c_2 \left(\|\dot{\theta}(t)\|^2 + \|\dot{\psi}(t)\|^2\right)$. Using (2.8) and (2.34) we obtain:

$$\frac{\|\partial E(\theta(t), \psi(t))\|^2}{4} = \left\| \frac{1}{\beta}(1+\alpha\beta)\dot{\theta}(t) + \left[(\alpha - \frac{1}{\beta}) - \frac{1}{\beta}(1+\alpha\beta)\right]\dot{\psi}(t) \right\|^2 + \frac{1}{\beta^2}\|\dot{\psi}(t)\|^2, \tag{2.36}$$

and applying again the claim above to (2.36) one can show that there exist $c_2 > 0$, such that for a.e. $t > 0$,

$$\|\partial E(\theta(t), \psi(t))\|^2 \geq c_2 \left(\|\dot{\theta}(t)\|^2 + \|\dot{\psi}(t)\|^2\right).$$

So, assumption (ii) holds for (2.8). To conclude, we can apply Theorem 2.8 to (2.8) and the proof is complete. □

Remark 2.11. *(a) Since the discrete algorithm INNA asymptotically resembles DIN (its continuous-time version, see the proof of Theorem 2.1), the results above suggest that similar behaviors and rates could be hoped for INNA itself. Yet, these results remain difficult to obtain in the case of DL, in particular in the mini-batch setting because of the noise $(\xi_k)_{k\in\mathbb{N}}$.*
(b) The proof above is significantly simpler when $\alpha\beta > 1$ since Alvarez et al. (2002) proved that in this case, (2.8) is equivalent to a gradient system, thus assumptions (i) and (ii) of Theorem 2.8 instantly hold.
(c) Theorems 2.8 and 2.10 can be adapted to the case where the Clarke subdifferential is replaced by $D\mathcal{J}$, but we do not state it here for the sake of simplicity.
(d) Theorems 2.8 and 2.10 are actually valid by assuming that $\mathcal{J}$ belongs to a polynomially bounded o-minimal structure. One of the most common instance of such structures is the one given by globally subanalytic sets (as illustrated in an example below). We refer to Bolte et al. (2007a) for a definition and further references.

Let us now comment the results of Theorem 2.10. First, we restrained the study to semi-algebraic loss functions $\mathcal{J}$, which are a subclass of tame loss functions. Most networks, activation functions and dissimilarity measures mentioned in Section 2.2.2 fall into this category. Nonetheless, the loss functions of the DL experiments of Section 2.6.2 are not semi-algebraic. Indeed, the dissimilarity measure ℓ used is the cross-entropy: $\ell(f(x_n, \theta), y_n) = -\sum_{d=1}^{D} \mathbf{1}_{[y_n]_d=1} \log([f(x_n, \theta)]_d)$. Such a function cannot be described by polynomials and presents a singularity whenever $[f(x_n, \theta)]_d = 0$. Fortunately, for inputs restricted to a compact set, due to the numerical precision but also to the "soft-max" functions often used in classification experiments, the outputs of the network f have values in $[\varepsilon, 1]$ for some small

$\varepsilon > 0$. Therefore, the singularity at 0 is harmless and the cross-entropy acts as a globally subanalytic function. As a consequence the non-smooth Łojasiewicz inequality holds, and we could obtain the same rates.

The rate of convergence of the trajectory in Theorem 2.10 is non-explicit in the sense that the exponent $b > 0$ is unknown in general. In the light of the proof of Theorem 2.8, this exponent depends on the KL exponent a of the Lyapunov function, which is itself hard to determine in practice. However, the intuition is that small exponents a may yield faster convergence rates (indeed, when $a \in (0, 1/2)$ we actually have a linear rate). As an example, for the function: $t \in \mathbb{R} \mapsto |t|^c$ with $c > 1$, the exponent at $t = 0$ is $a = 1 - \frac{1}{c}$ and thus, the closer c is to 1, the smaller a is, and the faster the convergence becomes.

2.6 Experiments

In this section we first discuss the role and the influence of the hyper-parameters of INNA as illustrated on the 2D example given in Figure 2.1. We then compare INNA with SGD, ADAGRAD and ADAM on deep learning problems for image recognition.

2.6.1 Understanding the role of the hyper-parameters of INNA

Both hyper-parameters α and β can be seen as damping coefficients from the viewpoint of mechanics as discussed by Alvarez et al. (2002) and sketched in the introduction. Recall that DIN, the second-order model used to build INNA, originally reads,

$$\ddot{\theta}(t) = -\alpha\,\dot{\theta}(t) - \beta\,\nabla^2\mathcal{J}(\theta(t))\dot{\theta}(t) - \nabla\mathcal{J}(\theta(t)).$$

Rewriting DIN as above highlights the mechanical interpretation inspired by Newton's second law of dynamics: the acceleration of a ball evolving on the landscape of $\mathcal{J}$ coincides with a sum of forces applied to the ball. Three forces are at stake: the gravity $-\nabla\mathcal{J}$ and two friction terms. The term $-\alpha\dot{\theta}$ acts as a stabilizer, reducing the speed $\dot{\theta}$. The parameter α thus corresponds to a *viscous damping* intensity similarly to the damping in the HBF method (2.2). On the other hand the parameter β can be seen as a *Newton damping* coefficient which takes into account the geometry of the landscape to brake or accelerate the dynamics in an adaptive anisotropic fashion. Indeed, the term $-\beta\nabla\mathcal{J}^2(\theta)\dot{\theta}$ accounts for the correlation between the speed $\dot{\theta}$ and the Hessian matrix of $-\mathcal{J}(\theta)$ which represents the variations of the gravity term $-\nabla\mathcal{J}(\theta)$, see Alvarez and Pérez (1998) and Alvarez et al.

(2002) for further insights.

We now turn our attention to INNA, and illustrate the versatility of the hyper-parameters α and β in this case. We proceed on a 2D visual non-smooth ill-conditioned example à la Rosenbrock, see Figure 2.1. For this example, we aim to find the minimum of the function $\mathcal{J}(\theta_1, \theta_2) = 100(\theta_2 - |\theta_1|)^2 + |1 - \theta_1|$. This function has a V-shaped valley, and a unique critical point at $(1, 1)$ which is also the global minimum. Starting from the point $(-1, 1.5)$ (the black cross), we apply INNA (without mini-batch sub-sampling) with constant step-sizes $\gamma_k = 10^{-4}$. Figure 2.1 shows that when β is too small, the trajectory presents many transverse oscillations as well as longitudinal ones close to the critical point (subplot a). Then, increasing β significantly reduces transverse oscillations (subplot b). Finally, the longitudinal oscillations are reduced by choosing a higher α (subplot c). In addition, these behaviors are also reflected in the values of the objective function (subplot d). The orange curve (first setting) presents large oscillations. Moreover, looking at the red curve, corresponding to plot (c), there is a short period between $20,000$ and $60,000$ iterations when the decrease is slower than for the other values of α and β, but still it presents fewer oscillations. In the longer term, the third choice ($\alpha = 1.3$, $\beta = 0.1$) provides remarkably good performances.

The choice of these hyper-parameters may come with rates of convergence for convex and strongly convex smooth functions (Attouch et al., 2020). Following this work, one may also consider to make α and β vary in time (for example like the famous Nesterov damping coefficient $\frac{\alpha}{t}$, Su et al. 2014). In our DL experiments we will however keep these parameters constant so that our theorems still hold. Yet, different behaviors depending on (α, β) can also be observed for DL problems as illustrated on Figure 2.2 and described next. Although we did not evidence some universal method to choose (α, β), we used mechanical intuitions to tune these parameters. The coefficient α induces viscous damping, thus one may try to reduce it when convergence appears to be slow. On the other hand, one may want to increase β when large oscillations are observed. Yet, since β affects directly the subgradient effect in (2.13), taking β too large may jeopardize the numerical stability of the algorithm. We will study the role of the hyper-parameters further in Chapter 3. Indeed, as mentioned in Remark 2.11-b, when $\alpha\beta \geq 1$, (2.12) can be shown to be a gradient system. On the other hand, when $\alpha\beta < 1$ the dynamics is of a different type. We will study how this reflects on INNA and the solutions of DIN.

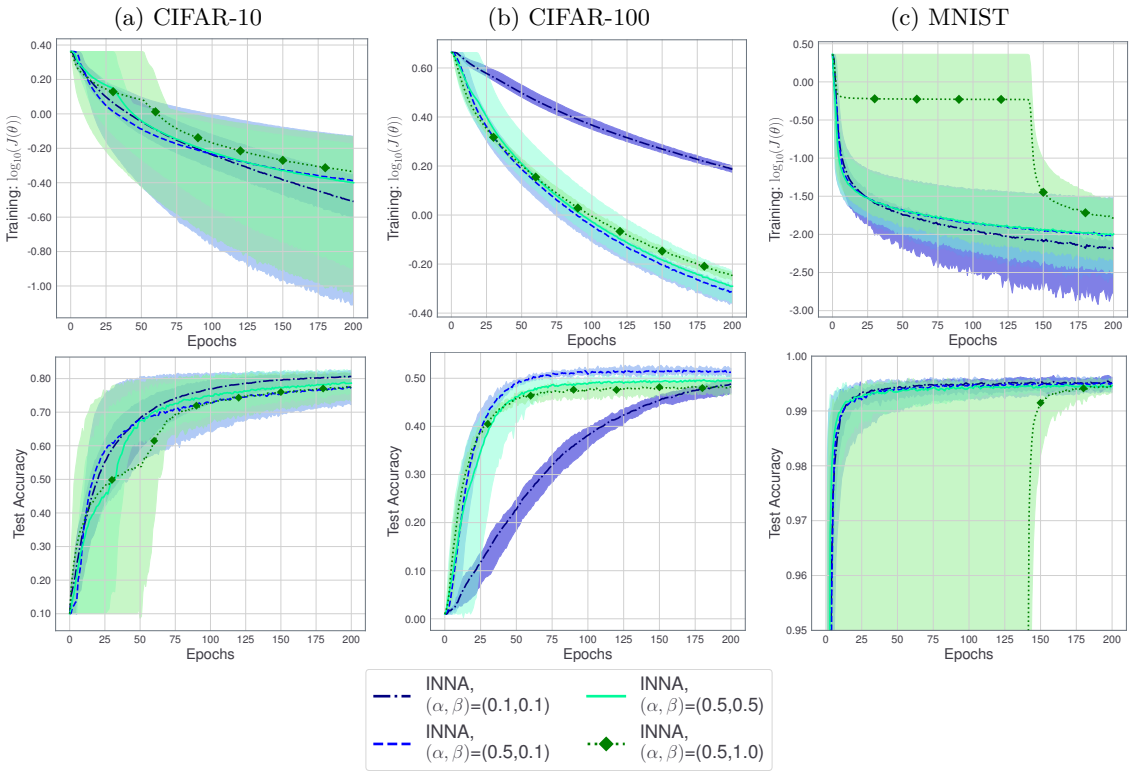


Figure 2.2: Analysis of the sensibility of INNA to the choice of α and β for three different image classification problems. Top: logarithm of the loss function $\mathcal{J}(\theta)$ during the training. Bottom: classification accuracy on the test set.

2.6.2 Training a DNN with INNA

Before comparing INNA to concurrent algorithms in DL, we first describe the methodology that we followed.

2.6.2.1 Methodology

– We train a DNN for classification using the three most common image datasets: MNIST, CIFAR-10, and CIFAR-100 (LeCun et al., 1998; Krizhevsky, 2009). These datasets are composed of $60,000$ small images associated with a label (numbers, objects, animals, etc.). We split the datasets into $50,000$ images for training and $10,000$ for testing.

– Regarding the network, we use a slightly modified version of Network in Network

(NiN) (M. Lin et al., 2014). It is a reasonably large CNN with $P \sim 10^6$ parameters to optimize. We use ReLU activation functions.

– The dissimilarity measure ℓ that is used in the empirical loss $\mathcal{J}$ given by (2.4) is the cross-entropy. The loss function $\mathcal{J}$ is optimized with respect to θ (the weights of the DNN) on the training data. The classification accuracy of the trained DNN is measured using the test data of $10,000$ images. Measuring the accuracy boils down to counting how many of the $10,000$ images were correctly classified (in percentage).

– Based on the results of Section 2.6.1, we run INNA for four different values of (α, β):

$$(\alpha, \beta) \in \{(0.1, 0.1), (0.5, 0.1), (0.5, 0.5), (0.5, 1)\}.$$

Given an initialization of the weights θ_0, we initialize ψ_0 such that the initial velocity is in the direction of $-\nabla\mathcal{J}(\theta_0)$. More precisely, we use $\psi_0 = (1-\alpha\beta)\theta_0 - (\beta^2-\beta)\nabla\mathcal{J}(\theta_0)$.

– We compare our algorithm INNA with several algorithms introduced in Section 1.4.3.2: SGD, ADAGRAD (J. Duchi et al., 2011) and ADAM (Kingma and Ba, 2015). At each iteration $k \in \mathbb{N}$, we compute the approximation of $\partial\mathcal{J}(\theta)$ on a subset $\mathsf{B}_k \subset \{1, \ldots, 50,000\}$ of size 32. The algorithms are initialized with the same random weights (drawn from a normal distribution). Five random initializations are considered for each experiment.

– Regarding the choice of the step-sizes, ADAGRAD and ADAM both use an adaptive procedure based on past gradients, see (1.30) and (1.32). For the other two algorithms (INNA and SGD), we use the classical step-size schedule $\gamma_k = \frac{\gamma_0}{\sqrt{k+1}}$ with $\gamma_0 > 0$, which meets Assumption 2.1. For all four algorithms, choosing the right initial step length ($\gamma_0 > 0$ for INNA and SGD, $\alpha > 0$ for the other methods) is often critical in terms of efficiency. We choose this parameter using a grid-search: for each algorithm we select the initial step-size that most decreases the training error $\mathcal{J}$ after fifteen epochs (recall that one epoch essentially consists in a complete pass over the dataset). Note that we could use more flexible step-size schedules, but we chose a standard schedule for simplicity. Different decay schemes are considered in Figure 2.4.

For these experiments, we used `keras 2.2.4` (Chollet, 2015) with `tensorflow 1.13.1` (Abadi et al., 2016) as backend. The INNA algorithm is available in `pytorch`, `keras` and `tensorflow`: https://github.com/camcastera/Inna-for-DeepLearning/ (Castera, 2019).

2.6.2.2 Results

Figure 2.2 displays the training loss $\mathcal{J}$ and test accuracy with respect to the epochs for INNA in its four hyper-parameter configurations considered and for the three datasets considered. Figure 2.3 displays the performance of INNA with the hyper-parameter configuration that led to the smallest average training error in Figure 2.2, with comparison to SGD, ADAGRAD and ADAM. In these two figures (and also in subsequent Figure 2.4), solid lines represent mean values and pale surfaces represent the best and worst runs in terms of training loss and validation accuracy over five random initializations.

Figure 2.2 suggests that the tuning of the hyper-parameters α and β is not crucial to obtain satisfactory results both for training and testing. The hyper-parameters mostly affect the training speed, so, INNA looks quite stable with respect to these hyper-parameters. Setting $(\alpha, \beta) = (0.5, 0.1)$ appears to be a good default choice[6], nevertheless, tuning these hyper-parameters is of course advised to get the most out of INNA.

Figure 2.3 shows what can be achieved with a moderately large network and coarse grid-search tuning of the initial step-size. In our comparison, INNA and ADAM outperform SGD and ADAGRAD for training. While ADAM seems to be faster in the early training phase, INNA achieves the best accuracy almost every time especially on CIFAR-100 (Figure 2.3(b)). Thus, INNA appears to be competitive in comparison to the other algorithms with the advantage of having solid theoretical foundations and a simple step-size rule as compared to ADAM and ADAGRAD. Additional DL experiments are performed in Section 4.5, where INNA appears again to be efficient for training and seems to possess very good generalization properties, see Remark 4.5.

Finally, let us point out that although ADAM was faster in the experiments of Figure 2.3, INNA can outperform ADAM using the slow step-size decay discussed in Section 2.4.2. Indeed, in the previous experiments we used a standard decreasing step-size of the form $\gamma_0/\sqrt{k+1}$ for simplicity, but Assumption 2.1 allows for step-sizes decreasing much more slowly. As such, we also considered decays of the form $\gamma_0(k+1)^{-q}$ with $q \leq 1/2$. The results are displayed on top of Figure 2.4. Except when q is too small (too slow decay, e.g., $q = 1/16$), these results show that some decays slower than $q = 1/2$ make INNA a little faster than any of the other algorithms. In particular, with a step-size decay proportional to $k^{-1/4}$, INNA outperforms ADAM (bottom of Figure 2.4). This suggests that tuning q can also significantly accelerate the training process.

[6]This observation is confirmed by additional experiments conducted on Section 4.5 of Chapter 4.

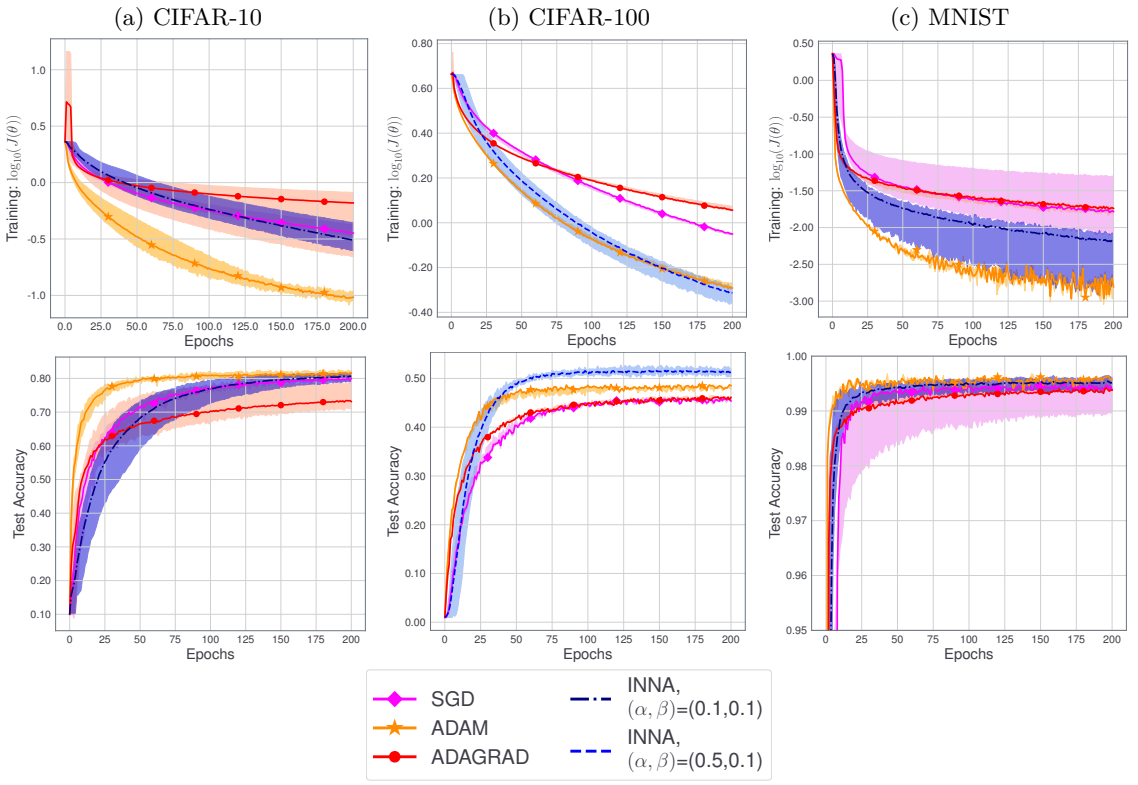


Figure 2.3: Comparison of INNA with concurrent algorithms: SGD, ADAM and ADAGRAD. Top: logarithm of the loss function $\mathcal{J}(\theta)$ during the training. Bottom: classification accuracy on the test set.

2.7 Conclusion

In this chapter we introduced a new second-order method featuring inertial and Newtonian behaviors. In Section 1.4.3.1 of the introduction, we presented the most common strategies to exploit second-order information via first-order oracles. Here, we used a rather orthogonal approach: we exploited the fact that DIN, a second-order ODE in time and space can be rewritten as a first-order system not only in time but also in space. The resulting implicit use of second-order information makes INNA highly compatible with first-order mini-batch subsampling. We provided a powerful algorithmic convergence analysis under weak hypotheses applicable to most DL problems. We also provided new general results to study differential inclusions with the Clarke subdifferential and obtain convergence rates for the continuous-time counterpart of our algorithm, as well as general results for the solutions of a class of differential inclusions. To our knowledge, the paper from which this chapter is adapted

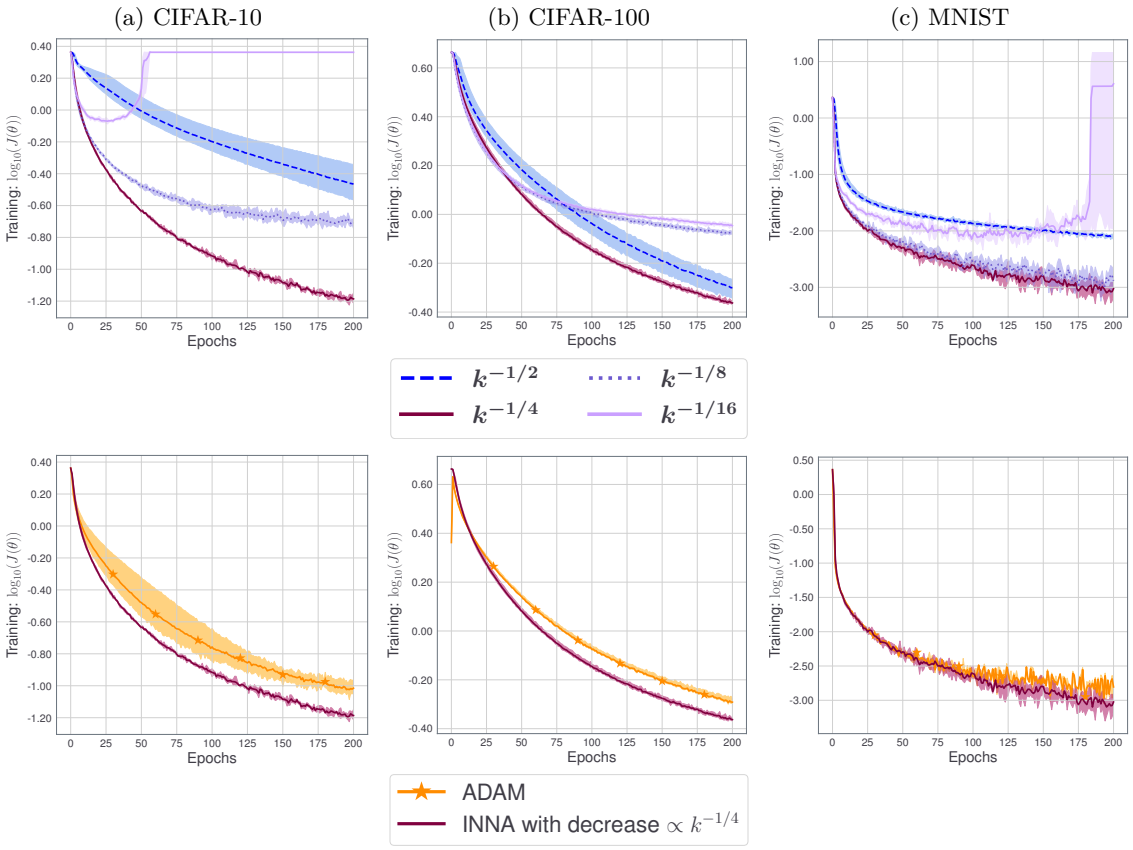


Figure 2.4: On top: Training loss of INNA on three image classification problems with various step-size decays. In the legend, k^{-q} means a step-size decay at iteration k of the form $\gamma_k = \gamma_0 k^{-q}$. The bottom row show the comparison between INNA with a well-chosen step-size decay and ADAM.

was the first one to rigorously handle the analysis of mini-batch sub-sampling for non-smooth DNNs via the introduction of D-critical points. Our experiments show that INNA is very competitive with concurrent algorithms for DL, with the advantage of having simple and explainable hyper-parameters. Finally, the satisfactory performances of the numerical experiments suggest that INNA seems to avoid spurious critical points and converge to minima. Next chapter is devoted to providing a better understanding of this phenomenon.

Chapter 3

Escape of Strict Saddle Points and Asymptotic Behavior of INNA

This chapter is adapted from Castera (2021).

Contents

3.1 Introduction

In Chapter 2, we introduced DIN, a second-order differential equation mixing Newton's method and HBF. From DIN we built INNA, a practical algorithm to tackle (1.6) and train

DNNs. While we proved that the accumulation points of INNA yield D-critical points of $\mathcal{J}$, we now study more precisely the nature of the critical points that INNA is likely to find. Indeed, since the function $\mathcal{J}$ is non-convex, it may thus have spurious critical points (critical points that are not local minima). It has been proved that gradient descent and HBF are likely to avoid *strict* saddles (critical points where $\nabla^2\mathcal{J}$ has negative eigenvalues, Goudou and Munier 2009; Lee et al. 2016; O'Neill and S. J. Wright 2019), however, vanilla Newton's method (with unit step-sizes) is attracted by any type of critical points, not only minima (see e.g., Dauphin et al. 2014), which is problematic when solving minimization problems like (1.6). Since DIN mixes Newton's method and HBF, we would like to answer the following question: *are the solutions of DIN—and the INNA algorithm—likely to avoid strict saddle points?*

In order to get a better understanding of INNA, we consider a less general framework than in Chapter 2. Here we study a loss function $\mathcal{J}\colon \mathbb{R}^P \to \mathbb{R}$ which is twice continuously differentiable on $\mathbb{R}^P$, and study a deterministic (full-batch) version of INNA with fixed step-sizes. In this framework, we answer positively to the above question, both for DIN and INNA, regardless the choice of the hyper-parameter β and for any $\alpha > 0$. Additionally, we shed light on the link between the choice of α and β and the asymptotic behavior of the solutions of DIN, with in particular the emergence of spirals when $\alpha\beta < 1$, this gives a better understanding of the role played by these hyper-parameters. We also provide numerical experiments illustrating the theoretical results.

Organization. The organization of this chapter is the following. We recall essential notions and optimality conditions in Section 3.2, we then prove that the solutions of DIN almost always avoid strict saddle points in Section 3.3.1, and study their qualitative behaviors in Section 3.3.2. Then, Section 3.4 is devoted to prove similar results for the discrete algorithm INNA, some conclusions are finally drawn. We first review the literature specifically related to this chapter.

Related work. As already said, DIN was first introduced by Alvarez et al. (2002). It was then studied by many, in particular Attouch et al. (2014), Attouch et al. (2016), and Shi et al. (2021) considered extensions of DIN where the hyper-parameters α and β vary over time (unlike what we did in Chapter 2). INNA was not the only algorithm based on the first-order equivalent formulation of DIN, this feature was also exploited by L. Chen and Luo (2019) and Attouch et al. (2020) to build algorithms called HNAG and IPAHD respectively. Regarding the effect of the parameters α and β, Attouch et al. (2020) and

Attouch et al. (2021) recently provided a global understanding of the link between these parameters and quantitative properties such as asymptotic rates of convergence for convex and strongly-convex loss functions. Here we rather focus on qualitative properties (such as the existence of spiraling solutions) and consider non-convex functions.

Our analysis mainly relies on results from the theory of dynamical systems, and in particular on the stable manifold theorem (Pliss, 1964; Kelley, 1966). This theorem can be used to prove that optimization algorithms are likely to avoid strict saddle points, it has been used for example by Goudou and Munier (2009) followed by Lee et al. (2016) and O'Neill and S. J. Wright (2019) for gradient descent and HBF. Finally, to analyze the qualitative behavior of DIN, we use the Hartman-Grobman Theorem (Grobman, 1959; Hartman, 1960).

3.2 Preliminary discussions and definitions

Before analyzing asymptotic behavior of optimization methods, we recall some fundamental notions which will be important in what follows. We refer to the notions of minimizers and maximizers introduced in Section 1.2.1 and in particular Definition 1.1. We now recall the following optimality conditions (see for example Nocedal and S. Wright 2006)—even-though we already discussed them a little in Chapter 1—which will be central in this chapter.

Proposition 3.1 (Optimality conditions)**.** *Let $g\colon \mathbb{R}^P \to \mathbb{R}$ be a twice continuously differentiable function and let $\theta^\star \in \mathbb{R}^P$. If $\theta^\star$ is a local minimizer of g, then the following holds:*

- *First-order condition: $\theta^\star$ is a critical point of g, i.e., $\nabla g(\theta^\star) = 0$.*
- *Second-order condition: The Hessian matrix $\nabla^2 g(\theta^\star)$ is positive semidefinite. Equivalently, all the eigenvalues of $\nabla^2 g(\theta^\star)$ are* non-negative.

Similarly, for any $\theta^\star \in \mathbb{R}^P$, if $\nabla g(\theta^\star) = 0$ and $\nabla^2 g(\theta^\star)$ is positive definite (or equivalently, $\nabla^2 g(\theta^\star)$ has only positive *eigenvalues), then $\theta^\star$ is a local minimizer of g.*

Similar results hold for maximizers but with negativity conditions for the Hessian matrix. The link between the eigenvalues of the Hessian matrix of the loss function $\mathcal{J}$ in (1.6), and the nature of critical points plays a crucial role in the sequel. As mentioned in the introduction, some critical points are irrelevant to minimize $\mathcal{J}$, we distinguish three types:

- Those where the Hessian of $\mathcal{J}$ has only positive eigenvalues. From Proposition 3.1, these points are local minima.
- The points where the Hessian matrix has at least one negative eigenvalue, which are

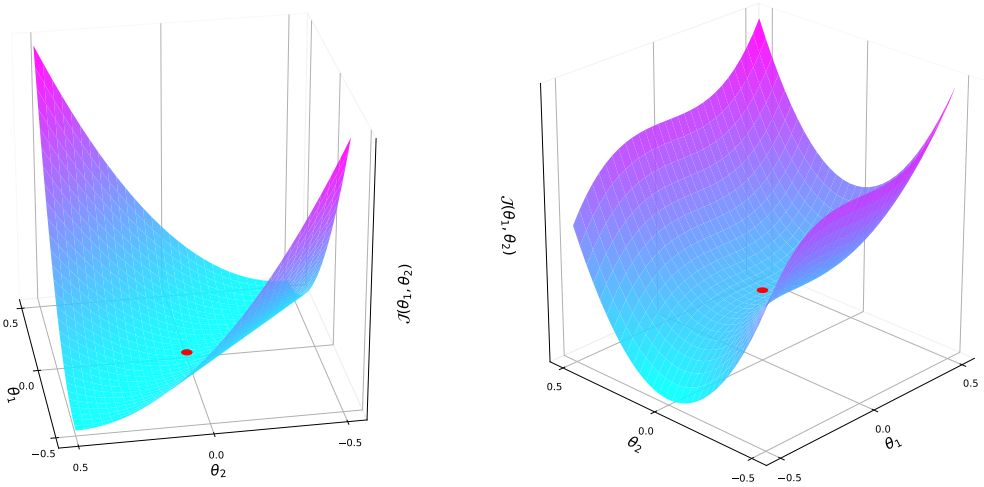


Figure 3.1: Example of two functions whose Hessian matrices are singular at $(0, 0)$. For the function $(\theta_1, \theta_2) \in \mathbb{R}^2 \mapsto \frac{1}{2}\theta_1^2 + \frac{1}{2}\theta_2^2 + \theta_1\theta_2$ (on the left), the critical point $(0, 0)$ (in red) is a minimum. For the function $(\theta_1, \theta_2) \in \mathbb{R}^2 \mapsto \theta_1^3 + \theta_2^2$ (on the right), the critical point $(0, 0)$ is neither a minimum nor a maximum.

referred to as *strict saddle points*. Such a point cannot be a local minimum, it is either a maximum or not an extremum.

– The points where the Hessian matrix has only non-negative eigenvalues and at least one zero eigenvalue, we call them *non-strict* saddle points. Such points may be maximizers, minimizers, or neither of them. For example, consider the functions $(\theta_1, \theta_2) \in \mathbb{R}^2 \mapsto \frac{1}{2}\theta_1^2 + \frac{1}{2}\theta_2^2 + \theta_1\theta_2$ and $(\theta_1, \theta_2) \in \mathbb{R}^2 \mapsto \theta_1^3 + \theta_2^2$. For both functions, $(0, 0)$ is a critical point and the eigenvalues of their Hessian matrices at $(0, 0)$ are 0 and 2. Yet, one can easily check that $(0, 0)$ is a minimizer for the first function and is not an extremum for the second one. These considerations are illustrated on Figure 3.1.

Due to the difficulties raised by the existence of non-strict saddle points, some results of this chapter hold only for Morse functions, defined next.

Definition 3.1. *A twice continuously differentiable function $g\colon \mathbb{R}^P \to \mathbb{R}$ is a Morse function if for any $\theta \in \mathbb{R}^P$ such that $\nabla g(\theta) = 0$, the Hessian $\nabla^2 g(\theta)$ has no zero eigenvalues.*

Morse functions are functions for which all saddles are strict and other critical points are minima. Some of the following results are restricted to Morse functions, others are more general, yet, in every case we will need the following assumption.

Assumption 3.1. *The loss function $\mathcal{J}$ has isolated critical points: for any $\theta^\star \in \mathbb{R}^P$ such that $\nabla \mathcal{J}(\theta^\star) = 0$, there exists a neighborhood $\Omega \subset \mathbb{R}^P$ of $\theta^\star$ such that $\theta^\star$ is the only critical point inside Ω.*

This assumption guarantees in particular that $\mathcal{J}$ has at most a countable (possibly infinite) number of critical points. Note additionally that Assumption 3.1 holds for Morse functions. Let us now move on to the analysis of DIN.

3.3 Continuous case: asymptotic behavior of the solutions of DIN

We recall that in this chapter, $\mathcal{J} : \mathbb{R}^P \to \mathbb{R}$ is a twice continuously differentiable function. Let $\alpha \geq 0$ and $\beta > 0$, we consider DIN for twice differentiable functions in its equivalent first-order form,

$$\begin{cases} \dot{\theta}(t) &= -\left(\alpha - \frac{1}{\beta}\right)\theta(t) - \frac{1}{\beta}\psi(t) - \beta\nabla\mathcal{J}(\theta(t)) \\ \dot{\psi}(t) &= -\left(\alpha - \frac{1}{\beta}\right)\theta(t) - \frac{1}{\beta}\psi(t) \end{cases}, \quad \text{for all } t > 0, \tag{3.1}$$

where $(\theta, \psi) \colon \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}^P \times \mathbb{R}^P$ is differentiable for all $t > 0$. Since $\mathcal{J}$ is twice continuously differentiable, the existence and uniqueness (with respect to initial conditions) of the solutions of (3.1) are granted by the Cauchy-Lipschitz theorem, see Alvarez et al. (2002). Let us focus on the asymptotic behavior of the solutions with respect to initial conditions.

Necessary condition for being a stationary point. A key element in the proof of convergence of INNA in Chapter 2 was that the stationary points of the solutions of DIN yield D-critical points of $\mathcal{J}$—which are simply critical points for smooth functions—see (2.16). Indeed, since $\mathcal{J}$ is differentiable, the set of stationary points of the solutions of (3.1) is,

$$\mathsf{S} = \left\{ (\theta^\star, \psi^\star) \in \mathbb{R}^P \times \mathbb{R}^P \middle| \nabla\mathcal{J}(\theta^\star) = 0,\ \psi^\star = (1 - \alpha\beta)\theta^\star \right\},$$

and that a bounded solution (θ, ψ) of (3.1) converges to a point of S hence the first coordinate θ of a bounded solution converges to a critical point of $\mathcal{J}$. We will study the type of points of S which the solutions of (3.1) are likely to converge to, and then study the qualitative asymptotic behavior of these solutions.

3.3.1 DIN is likely to avoid strict saddle points

We start with our main result regarding the limit of the solutions of DIN.

3.3.1.1 Main convergence results

For convenience, we denote by $\mathsf{S}_{<0} \subset \mathsf{S}$ the set of stationary points $(\theta^\star, \psi^\star)$ such that $\theta^\star$ is a strict saddle point of $\mathcal{J}$, namely,

$$\mathsf{S}_{<0} \overset{\text{def}}{=} \left\{ (\theta^\star, \psi^\star) \in \mathsf{S} \,\middle|\, \nabla^2 \mathcal{J}(\theta^\star) \text{ has at least one negative eigenvalue} \right\}. \tag{3.2}$$

Theorem 3.2. *Suppose that Assumption 3.1 holds for $\mathcal{J}$, then for almost any initialization, the corresponding solution of (3.1) does not converge to a point in $\mathsf{S}_{<0}$.*

Before proving the theorem, the following corollary is an immediate consequence suited for practical applications.

Corollary 3.3. *Assume that $\mathcal{J}$ is a twice continuously differentiable Morse function. Assume also that $\mathcal{J}$ is coercive (i.e., that $\lim_{\|\theta\| \to \infty} \mathcal{J}(\theta) = +\infty$). Then for any initialization the associated solution of (3.1) converges. Moreover, let (θ_0, ψ_0) be a non-degenerate random variable on $\mathbb{R}^P \times \mathbb{R}^P$, and let (θ, ψ) be the solution of (3.1) initialized at (θ_0, ψ_0) and converging to $(\theta^\star, \psi^\star) \in \mathbb{R}^P \times \mathbb{R}^P$. Then with probability one with respect to the draw of (θ_0, ψ_0), $\theta^\star$ is a local minimizer of $\mathcal{J}$.*

This corollary states in particular that for a coercive Morse function, we can pick an initialization sampled from a non-degenerate distribution on $\mathbb{R}^P \times \mathbb{R}^P$, for example a Gaussian or uniform distribution, and with probability one, the first coordinate of the limit of the solution (with respect to the initialization) is a local minimizer of $\mathcal{J}$.

Proof of Corollary 3.3. Using the remarks from Section 2.4.2, the coercivity of $\mathcal{J}$ guarantees that any solution of (3.1) remains bounded, and from Alvarez et al. (2002), any bounded solution is converging. Then the limit of any solution belongs to S. Let (θ_0, ψ_0) be a random variable sampled from a non-degenerate distribution on $\mathbb{R}^P \times \mathbb{R}^P$, by definition the support of the distribution has non-zero measure. In addition, according to Theorem 3.2, the set of initializations such that the solutions of (3.1) converge to $\mathsf{S}_{<0}$ has zero measure. So, almost surely with respect to the random variable (θ_0, ψ_0), the solution of (3.1) initialized at (θ_0, ψ_0) converges toward $\mathsf{S} \setminus \mathsf{S}_{<0}$. Finally, since $\mathcal{J}$ is a Morse function, $\mathsf{S} \setminus \mathsf{S}_{<0}$ is exactly the set of local minimizers. □

Remark 3.4. *We could state a more general (but more abstruse) result than Corollary 3.3 which would not require the coercivity assumption but only that the set of initializations such that the associated solution of (3.1) converges has positive Lebesgue measure. We will do so for INNA (see Corollary 3.10).*

We now introduce the main tool to prove Theorem 3.2: the stable manifold theorem.

3.3.1.2 The stable manifold theorem

To simplify the notations we introduce the following mapping,

$$G : (\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P \mapsto \begin{pmatrix} -\left(\alpha - \frac{1}{\beta}\right)\theta - \frac{1}{\beta}\psi - \beta\nabla\mathcal{J}(\theta) \\ -\left(\alpha - \frac{1}{\beta}\right)\theta - \frac{1}{\beta}\psi \end{pmatrix},$$

so that (3.1) can be re-written,

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix}\theta(t)\\ \psi(t)\end{pmatrix} = G(\theta(t), \psi(t)), \quad \text{for all } t > 0. \tag{3.3}$$

For any $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$, we also denote by $\mathrm{Jac}_G(\theta, \psi) \in \mathbb{R}^{2P\times 2P}$ the Jacobian matrix of G at (θ, ψ). Remark that for any $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$, it holds $(\theta, \psi) \in \mathsf{S} \iff G(\theta, \psi) = 0$, so the stationary points of (3.1) are exactly the zeros of G. We now state the stable manifold theorem which is the keystone to prove Theorem 3.2.

Theorem 3.5 (Stable manifold theorem (Haragus and Iooss, 2010; Perko, 2013))**.** *Let $F\colon \mathbb{R}^{2P} \to \mathbb{R}^{2P}$ be a C^1 mapping and denote by Jac_F the Jacobian of F, consider the autonomous ODE,*

$$\frac{\mathrm{d}\Theta}{\mathrm{d}t}(t) = F(\Theta(t)), \quad \textit{for all } t > 0. \tag{3.4}$$

Let $\Theta^\star \in \mathbb{R}^{2P}$ such that $F(\Theta^\star) = 0$. Let $E^{sc}(\Theta^\star)$ be the linear subspace of $\mathbb{R}^{2P}$ spanned by the eigenvalues of $\mathrm{Jac}_F(\Theta^\star)$ with non-positive real part. There exists a neighborhood Ω of $\Theta^\star$ and a C^1 manifold $\mathsf{W}^{sc}(\Theta^\star)$ tangent to $E^{sc}(\Theta^\star)$ at $\Theta^\star$—whose dimension is the number of eigenvalues of $\mathrm{Jac}_F(\Theta^\star)$ with non-positive real part—such that, for any solution Θ of (3.4),

(i) If $\Theta(0) \in \mathsf{W}^{sc}(\Theta^\star) \cap \Omega$ and for $T \geq 0$, $\Theta([0, T]) \subset \Omega$, then $\Theta([0, T]) \subset \mathsf{W}^{sc}(\Theta^\star)$. (Invariance)

(ii) If $\forall t \geq 0$, $\Theta(t) \in \Omega$, then $\Theta(0) \in \mathsf{W}^{sc}(\Theta^\star)$.

We see why this theorem plays an important role in the proof of Theorem 3.2. It states in

particular that all the solutions of (3.4) converging to $\Theta^\star$ must enter inside $\mathsf{W}^{sc}(\Theta^\star)$ after some time, and that $\mathsf{W}^{sc}(\Theta^\star)$ has zero measure as soon as $\mathrm{Jac}_F(\Theta^\star)$ has at least one positive eigenvalue. We will show that this holds true for G and for any point in $\mathsf{S}_{<0}$. Now that we introduced our main tool, we can prove Theorem 3.2.

3.3.1.3 Proof of Theorem 3.2

We state an elementary useful lemma.

Lemma 3.6. *Let $\alpha \geq 0$, $\beta > 0$ and $\lambda \in \mathbb{R}$. The quantity $(\alpha+\beta\lambda)^2 - 4\lambda$ is non-positive if and only if $\alpha\beta \leq 1$ and $\lambda \in \left[\frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, \frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}\right]$.*

Proof of Lemma 3.6. Let $\alpha \geq 0$, and $\beta > 0$, the function $\lambda \in \mathbb{R} \mapsto (\alpha+\beta\lambda)^2 - 4\lambda = \beta^2\lambda^2 + 2(\alpha\beta-2)\lambda + \alpha^2$ is a second-order polynomial in λ whose discriminant is $16(1-\alpha\beta)$. If $\alpha\beta > 1$ this discriminant is negative thus the polynomial has no real roots and hence is always positive. If $\alpha\beta \leq 1$, then the discriminant is non-negative and the roots of the polynomial are $\frac{(2-\alpha\beta)}{\beta^2} \pm \frac{2\sqrt{1-\alpha\beta}}{\beta^2}$. □

We now prove Theorem 3.2.

Proof of Theorem 3.2. Let $(\theta^\star, \psi^\star) \in \mathbb{R}^P \times \mathbb{R}^P$ such that $G(\theta^\star, \psi^\star) = 0$. We first compute the eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$, in order to apply Theorem 3.5 to (3.3) around $(\theta^\star, \psi^\star)$. By differentiating G we obtain the following Jacobian matrix, displayed by block,

$$\mathrm{Jac}_G(\theta^\star, \psi^\star) = \begin{pmatrix} -\beta\nabla^2\mathcal{J}(\theta^\star) - \left(\alpha - \frac{1}{\beta}\right) I_P & -\frac{1}{\beta} I_P \\ -\left(\alpha - \frac{1}{\beta}\right) I_P & -\frac{1}{\beta} I_P \end{pmatrix}, \tag{3.5}$$

where I_P denotes the identity matrix of $\mathbb{R}^{P\times P}$. We need to compute the eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ and study the sign of their real parts. In particular, we want to show that if $\nabla^2\mathcal{J}(\theta)$ has a (strictly) negative eigenvalue (i.e., $(\theta^\star, \psi^\star) \in \mathsf{S}_{<0}$), then $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ has at least one (strictly) positive eigenvalue, and thus according to Theorem 3.5, the stable manifold associated to $(\theta^\star, \psi^\star)$ has zero measure.

First, $\nabla^2\mathcal{J}(\theta^\star)$ is real and symmetric, so the spectral theorem states that there exists an orthogonal matrix V such that $V^T\nabla^2\mathcal{J}(\theta^\star)V$ is a diagonal matrix. Thus the matrix,

$$\begin{pmatrix} V^T & 0 \\ 0 & V^T \end{pmatrix} \mathrm{Jac}_G(\theta^\star, \psi^\star) \begin{pmatrix} V & 0 \\ 0 & V \end{pmatrix} = \begin{pmatrix} -\beta V^T\nabla^2\mathcal{J}(\theta^\star)V - \left(\alpha - \frac{1}{\beta}\right) I_P & -\frac{1}{\beta} I_P \\ -\left(\alpha - \frac{1}{\beta}\right) I_P & -\frac{1}{\beta} I_P \end{pmatrix} \tag{3.6}$$

is a sparse matrix with only 3 non-zero diagonals and whose eigenvalues are the same as those of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$. Exploiting the tridiagonal structure, there exists a symmetric permutation $U \in \mathbb{R}^{2P\times 2P}$, specified in (3.36) in Section 3.6 of the appendix of this chapter, such that we can transform (3.6) into a block diagonal matrix,

$$U^T \begin{pmatrix} V^T & 0 \\ 0 & V^T \end{pmatrix} \mathrm{Jac}_G(\theta^\star, \psi^\star) \begin{pmatrix} V & 0 \\ 0 & V \end{pmatrix} U = \begin{pmatrix} M_1 & & \\ & \ddots & \\ & & M_P \end{pmatrix}, \tag{3.7}$$

where for each $p \in \{1, \dots, P\}$, M_p is a 2×2 matrix defined as follows. Denote by $(\lambda_p)_{p\in\{1,\dots,P\}}$ the eigenvalues of $\nabla^2 \mathcal{J}(\theta^\star)$, then—up to a symmetric permutation—for all $p \in \{1, \dots, P\}$, $M_p = \begin{pmatrix} -\left(\alpha - \frac{1}{\beta}\right) - \beta\lambda_p & -\frac{1}{\beta} \\ -\left(\alpha - \frac{1}{\beta}\right) & -\frac{1}{\beta} \end{pmatrix}$.

The eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ are obtained by computing those of the matrices M_p. Let $p \in \{1, \dots, P\}$, the eigenvalues of M_p are the roots of its characteristic polynomial: $\chi_{M_p} : X \in \mathbb{R} \mapsto X^2 - \mathrm{trace}(M_p)X + \det(M_p)$, which gives for any $X \in \mathbb{R}$,

$$\chi_{M_p}(X) = X^2 + (\alpha + \beta\lambda_p)X + \lambda_p. \tag{3.8}$$

This is a second-order polynomial, whose discriminant is,

$$\Delta_{M_p} \overset{\text{def}}{=} (\alpha + \beta\lambda_p)^2 - 4\lambda_p. \tag{3.9}$$

The eigenvalues of M_p depend on the sign of Δ_{M_p} which is given by Lemma 3.6. We now show that if $\lambda_p < 0$, then M_p has a positive eigenvalue.

First assume that $\Delta_{M_p} \leq 0$. Lemma 3.6 states that in this case, we have $\alpha\beta \leq 1$ and $\lambda_p \in \left[\frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, \frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}\right]$. An elementary study of the function $x \in [0,1] \mapsto 2 - x - 2\sqrt{1-x}$ shows however that for $0 \leq \alpha\beta \leq 1$, $\frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2} \geq 0$, so $\Delta_{M_p} \leq 0$ implies $\lambda_p \geq 0$. Thus, we do not need to further investigate the case $\Delta_{M_p} \leq 0$ since this case never occurs when $\lambda_p < 0$.

Suppose now that $\Delta_{M_p} > 0$, then M_p has two real eigenvalues,

$$\begin{cases} \sigma_{p,+} = \frac{-(\alpha+\beta\lambda_p)}{2} + \frac{\sqrt{\Delta_{M_p}}}{2} = \frac{-(\alpha+\beta\lambda_p)}{2} + \frac{\sqrt{(\alpha+\beta\lambda_p)^2-4\lambda_p}}{2} \\ \sigma_{p,-} = \frac{-(\alpha+\beta\lambda_p)}{2} - \frac{\sqrt{\Delta_{M_p}}}{2} = \frac{-(\alpha+\beta\lambda_p)}{2} - \frac{\sqrt{(\alpha+\beta\lambda_p)^2-4\lambda_p}}{2} \end{cases}. \tag{3.10}$$

In this case, assume that $\lambda_p < 0$. If $\alpha + \beta\lambda_p \leq 0$, then $\sigma_{p,+}$ is a sum of a non-negative and a positive term, so $\sigma_{p,+} > 0$. If $\alpha + \beta\lambda_p \geq 0$, then

$$2\sigma_{p,+} = -(\alpha + \beta\lambda_p) + \sqrt{(\alpha + \beta\lambda_p)^2 + 4(-\lambda_p)} > 0,$$

since $4(-\lambda_p) > 0$. Overall, we showed that in every case, $\lambda_p < 0 \implies \sigma_{p,+} > 0$. So whenever there exists $p \in \{1, \dots, P\}$ such that $\lambda_p < 0$, $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ has at least one positive eigenvalue.

We can now apply the stable manifold theorem. Let $(\theta^\star, \psi^\star) \in \mathsf{S}_{<0}$. Let an initialization (θ_0, ψ_0) such that the corresponding solution (θ, ψ) of (3.3) converges to $(\theta^\star, \psi^\star)$. Denote $\Phi : \mathbb{R}^P \times \mathbb{R}^P \times \mathbb{R} \to \mathbb{R}^P \times \mathbb{R}^P \times \mathbb{R}$ the flow of the solutions of (3.3), so that we have in particular for all $t \geq 0$, $(\theta(t), \psi(t)) = \Phi((\theta_0, \psi_0), t)$ and $(\theta_0, \psi_0) = \Phi((\theta(t), \psi(t), -t)$. Consider the manifold $\mathsf{W}^{sc}(\theta^\star, \psi^\star)$ and the neighborhood Ω as defined in Theorem 3.5. The convergence of (θ, ψ) implies that there exists $t_0 \geq 0$ such that for all $t \geq t_0$, $(\theta(t), \psi(t)) \in \Omega$, so according to Theorem 3.5, $\forall t \geq t_0$, $(\theta(t), \psi(t)) \in \Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)$. Expressing this in terms of flows, $\forall t \geq t_0$, $\Phi((\theta_0, \psi_0), t) \in \Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)$ and hence $\forall t \geq t_0$,

$$\Phi((\theta_0, \psi_0), t) \in \bigcup_{k \in \mathbb{N}} \Phi\left(\Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star), -k\right), \tag{3.11}$$

where the right-hand side in (3.11) corresponds to the union over $k \in \mathbb{N}$ of initial conditions such that the associated solution has reached $\Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)$ at time k. Let

$$\mathsf{W}(\theta^\star, \psi^\star) = \left\{ (\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P \middle| \Phi((\theta_0, \psi_0), t) \xrightarrow[t \to +\infty]{} (\theta^\star, \psi^\star) \right\}, \tag{3.12}$$

the set of all initial conditions such that the associated solution converges to $(\theta^\star, \psi^\star)$. According to (3.11), we have proved that,

$$\mathsf{W}(\theta^\star, \psi^\star) \subset \bigcup_{k \in \mathbb{N}} \Phi\left(\Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star), -k\right). \tag{3.13}$$

Now, we previously showed that since $(\theta^\star, \psi^\star) \in \mathsf{S}_{<0}$, $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ has one or more positive eigenvalues, so according to the stable manifold theorem, the dimension of $\mathsf{W}^{sc}(\theta^\star, \psi^\star)$ is strictly less than $2P$, hence this manifold has zero measure. Due to the uniqueness of the solutions of (3.3), for any $k \in \mathbb{N}$, $\Phi(\cdot, -k)$ is a local diffeomorphism, hence it maps zero-measure sets to zero-measure sets. Consequently, the right-hand side in (3.13) is a countable union of zero-measure sets, so it has zero measure as well, and the same goes for $\mathsf{W}(\theta^\star, \psi^\star)$.

To conclude the proof of the theorem, by Assumption 3.1, the critical points are isolated so their number is countable. So $\bigcup_{(\theta^\star,\psi^\star)\in\mathsf{S}_{<0}} \mathsf{W}(\theta^\star,\psi^\star)$ is a countable union of zero-measure sets so it has zero measure.

□

Remark 3.7. *Keeping the notations of the proof of Theorem 3.2, we proved that for $p \in \{1,\dots,P\}$, if $\lambda_p < 0$ then $\sigma_{p,+} > 0$. Looking at the proof, note that we could also show quite easily that $\lambda_p > 0 \implies \sigma_{p,+} < 0$, so for any local minimizer with non-singular Hessian, the associated stable manifold does not have zero measure. In particular, the stable manifold associated to any local minima of a twice differentiable Morse functions does not have zero measure.*

3.3.1.4 On the complex eigenvalues of Jac_G

In the proof of Theorem 3.2 we showed that for any $(\theta^\star,\psi^\star) \in \mathsf{S}$, if an eigenvalue λ_p of the Hessian $\nabla\mathcal{J}(\theta^\star)$ is negative, then the associated discriminant Δ_{M_p} is positive and thus the eigenvalues of $\mathrm{Jac}_G(\theta^\star,\psi^\star)$ are real. Note however that when there exists $p \in \{1,\dots,P\}$ such that $\lambda_p \geq 0$ (and hence in particular around local minima), we may have $\Delta_{M_p} \leq 0$. This is the case whenever $\alpha\beta \leq 1$, and $\lambda_p \in \left[\frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, \frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}\right]$. When the aforementioned conditions hold, $\mathrm{Jac}_G(\theta^\star,\psi^\star)$ has complex eigenvalues,

$$\begin{cases} \sigma_{p,+} = \frac{-(\alpha+\beta\lambda_p)}{2} + i\frac{\sqrt{-\Delta_{M_p}}}{2} = \frac{-(\alpha+\beta\lambda_p)}{2} + i\frac{\sqrt{4\lambda_p-(\alpha+\beta\lambda_p)^2}}{2} \\ \sigma_{p,-} = \frac{-(\alpha+\beta\lambda_p)}{2} - i\frac{\sqrt{-\Delta_{M_p}}}{2} = \frac{-(\alpha+\beta\lambda_p)}{2} - i\frac{\sqrt{4\lambda_p-(\alpha+\beta\lambda_p)^2}}{2} \end{cases} . \tag{3.14}$$

Overall, we proved that DIN is likely to avoid strict saddle points, and we additionally observed that around local minima, the Jacobian Jac_G may or may not have eigenvalues with non-zero imaginary part. The existence of complex eigenvalues may change the behavior of the solutions around local minima. Next section is devoted to studying this matter.

3.3.2 Behavior of the solutions of DIN around stationary points

Accordingly to what we just discussed in Section 3.3.1.4, we wish to characterize the qualitative asymptotic behavior of the convergent trajectories of DIN.

3.3.2.1 The Hartman-Grobman theorem

To this aim, we introduce the Hartman-Grobman theorem.

Theorem 3.8 (Hartman–Grobman (Perko, 2013))**.** *Consider the following dynamical system,*

$$\frac{\mathrm{d}\Theta}{\mathrm{d}t}(t) = F(\Theta(t)), \quad t \in \mathbb{R} \tag{3.15}$$

where $\Theta : \mathbb{R} \to \mathbb{R}^{2P}$, $F : \mathbb{R}^{2P} \to \mathbb{R}^{2P}$ *is* C^1 *and denote by* Jac_F *the Jacobian matrix of* F*. Assume that there exists* $\Theta^\star \in \mathbb{R}^{2P}$ *such that* $F(\Theta^\star) = 0$ *and* $\mathrm{Jac}_F(\Theta^\star)$ *has only non-zero eigenvalues. Then, there exists a neighborhood* Ω *of* $\Theta^\star$ *and a homeomorphism* H *(a bijective continuous function whose inverse is continuous) such that, for any* $\Theta_0 \in \Omega$*, if* Θ *is a solution of* (3.15) *with* $\Theta(0) = \Theta_0$*, there exists an open interval of time* $\mathsf{T} \subset \mathbb{R}$ *containing* 0 *such that the function* $\Phi = H \circ \Theta$ *is the solution of*

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t}(t) = \mathrm{Jac}_F(\Theta^\star)\Phi(t), \quad t \in \mathsf{T}, \tag{3.16}$$

with initial condition $\Phi(0) = H(\Theta_0)$*. The homeomorphism* H *preserves the parameterization by time (it does not reverse time).*

This theorem essentially states that, in a neighborhood of a stationary point $\Theta^\star$ where the Jacobian matrix $\mathrm{Jac}_F(\Theta^\star)$ is non-singular the solutions of (3.15) have a qualitative behavior similar to those of the linearized system (3.16).

3.3.2.2 Application to DIN

Application of the theorem. Let $(\theta^\star, \psi^\star) \in \mathsf{S}$ be a stationary point of (3.1) such that $\theta^\star$ is a local minimizer of $\mathcal{J}$ and such that $\nabla^2\mathcal{J}(\theta^\star)$ has only non-zero eigenvalues (this is guaranteed for all local minima if $\mathcal{J}$ is a Morse function). Consider the differential equation,

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix}\widetilde{\theta}(t)\\ \widetilde{\psi}(t)\end{pmatrix} = \mathrm{Jac}_G(\theta^\star, \psi^\star)\begin{pmatrix}\widetilde{\theta}(t)\\ \widetilde{\psi}(t)\end{pmatrix}, \quad t \in \mathbb{R}. \tag{3.17}$$

According to Remark 3.7 and the proof of Theorem 3.2, all the eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ have strictly negative real parts. So there exists a homeomorphism H and a neighborhood Ω of $(\theta^\star, \psi^\star)$ on which Theorem 3.8 holds. In particular, for any initial condition $(\theta_0, \psi_0) \in \Omega$, the associated solution of (3.3) converges to $(\theta^\star, \psi^\star)$ (because the corresponding solution of (3.17) converges to $(\theta^\star, \psi^\star)$ and H preserves the parameterization by time).

So for any initialization in $(\theta_0, \psi_0) \in \Omega$, the corresponding solution (θ, ψ) of (3.3) remains in Ω, i.e., $\forall t \geq 0, (\theta(t), \psi(t)) \in \Omega$. Thus we can use the Hartman-Grobman around $(\theta(t), \psi(t))$ for any $t \geq 0$. As a result, for any initialization in $(\theta_0, \psi_0) \in \Omega$, and for all $t \geq 0$, the associated solution (θ, ψ) of (3.3) reads, $(\theta(t), \psi(t)) = H^{-1}(\widetilde{\theta}(t), \widetilde{\psi}(t))$ where $(\widetilde{\theta}, \widetilde{\psi})$ is the solution of (3.17) with initial condition $(\widetilde{\theta}(t_0), \widetilde{\psi}(t_0)) = H(\theta(0), \psi(0))$.

We give a more precise expression for this solution. As done in the proof of Theorem 3.2 we can diagonalize $\mathrm{Jac}_G(\theta^\star, \psi^\star)$: there exists a matrix $Q \in \mathbb{R}^{2P \times 2P}$ such that $\mathrm{Jac}_G(\theta^\star, \psi^\star) = QDQ^{-1}$, where $D = \mathrm{diag}(\sigma_1, \dots, \sigma_{2P})$, and $(\sigma_p)_{\{1,\dots,2P\}}$ are the eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$. Using the diagonalization, the solution of (3.17) is given for all $t \in \mathbb{R}$ by $\begin{pmatrix} \widetilde{\theta}(t) \\ \widetilde{\psi}(t) \end{pmatrix} = Qe^{tD}Q^{-1} \begin{pmatrix} \widetilde{\theta}(0) \\ \widetilde{\psi}(0) \end{pmatrix}$. So going back to (θ, ψ), we have,

$$\Big(\theta(t), \psi(t)\Big) = H^{-1}\left(Qe^{tD}Q^{-1}H(\theta(0), \psi(0))\right), \quad \text{for all } t \geq 0. \tag{3.18}$$

Finally, let any initialization $(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P$ (not necessarily belonging to Ω), such that the corresponding solution (θ, ψ) of (3.3), converges to $(\theta^\star, \psi^\star)$. Then there exists $t_0 \geq 0$ such that for all $t \geq t_0$, $(\theta(t), \psi(t)) \in \Omega$, and the arguments above apply after t_0.

Form of the solutions. We proved that after some time, a solution (θ, ψ) of (3.3) which converges to $(\theta^\star, \psi^\star)$ can be expressed with formula (3.18) (up to a time shift). If $\alpha\beta \leq 1$ and all the eigenvalues of $\nabla^2 \mathcal{J}(\theta^\star)$ are not in $\left(\frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, \frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}\right)$, or if $\alpha\beta > 1$, then all the eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ are real so the coordinates of $Q^{-1}e^{tD}Q$ in (3.18) are sums of exponential functions decreasing in time.

However, if $\alpha\beta \leq 1$ and there exists eigenvalues of $\nabla^2 \mathcal{J}(\theta^\star)$ belonging to the interval mentioned above, then there exists eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ with non-zero imaginary part. Let $p \in \{1, \dots, P\}$ such that λ_p is an eigenvalue of $\nabla^2 \mathcal{J}(\theta^\star)$ belonging to the aforementioned interval. From (3.14), there exists two complex eigenvalues: $\frac{-(\alpha+\beta\lambda_p)}{2} \pm i\frac{\sqrt{4\lambda_p - (\alpha+\beta\lambda_p)^2}}{2}$, and thus the coordinates of the matrix e^{tD} in (3.18) contain terms of the form,

$$e^{\frac{-(\alpha+\beta\lambda_p)}{2}t}\left(\cos(\omega_p t) \pm i \sin(\omega_p t)\right),$$

where $\omega_p = \frac{\sqrt{4\lambda_p - (\alpha+\beta\lambda_p)^2}}{2}$. So in this setting, and in this setting only, the imaginary parts of the eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ generate oscillating terms and the solution of the linearized model $t \mapsto Qe^{tD}Q^{-1}$ (plus initial condition) spirals around $(\theta^\star, \psi^\star)$ as it converges toward it. This is illustrated on Figure 3.2 where such a behavior is indeed observed for a

(numerically approximated) solution of (3.1). Note finally that ω_p is a decreasing function of β, hence increasing the parameter β reduces the oscillations, which corroborates the intuition discussed in Section 2.6.1 of Chapter 2.

3.3.2.3 Numerical illustration of the spiraling phenomenon

Setting. To illustrate the spiraling phenomenon, we consider a simple quadratic function $\mathcal{J}\colon (\theta_1, \theta_2) \in \mathbb{R}^2 \mapsto \theta_1^2 + 2\theta_2^2$. This loss function is $\mathcal{C}^2(\mathbb{R}^2)$, and for all $(\theta_1, \theta_2) \in \mathbb{R}^2$, it has a constant diagonal Hessian $\nabla^2 \mathcal{J}(\theta_1, \theta_2) = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}$. This is a convex function whose unique global minimizer is $(\theta^\star, \psi^\star) = (0, 0)$. Instead of solving exactly (3.1), we find an approximate solution via the a full-batch version of INNA derived from (3.1) and presented in next section. To do so, we ran the algorithm with very small step-sizes. The algorithm is initialized at $(1, 1)$. We consider two choices of parameters: $(\alpha, \beta) = (2, 0.1)$ and $(\alpha, \beta) = (2, 1)$. The former illustrates the case $\alpha\beta < 1$ while the second corresponds to the case where $\alpha\beta > 1$. According to Section 3.3.1.4, with the configuration $(\alpha, \beta) = (2, 0.1)$, the range of eigenvalues for which we should observe spirals is approximately $[1, 359]$ so both eigenvalues of the Hessian of $\mathcal{J}$ lie in this interval.

Results. The expected behavior (discussed on Section 3.3.2.2) can be observed on the left of Figure 3.2. When $\alpha\beta < 1$ (red curve), the trajectory spirals around the critical point $(0, 0)$. On the contrary, the phenomenon does not occur when $\alpha\beta > 1$ (orange curve). Remark also that when zooming infinitesimally close to $(0, 0)$, the oscillating behavior is still present. Note however that this qualitative result says nothing about the speed of convergence, as evidenced on the right of Figure 3.2. Despite the presence of spirals, the setting where $\alpha\beta < 1$ yields a faster algorithm both in terms of loss function values and distance to the objective. From a theoretical point of view, the Hartman-Grobman theorem connects the solutions of (3.1) and those of its linearized approximation through a mapping which is homeomorphic (hence continuous) but not necessarily differentiable. As a consequence, the theorem does not guarantee that the speed of convergence is preserved. Regarding the study of the speed of convergence, we refer to Attouch et al. (2020) and Attouch et al. (2021) in a convex setting and to Theorem 2.10 for the non-convex case.

Vanishing viscous damping. To finish this section, we empirically investigate the oscillating phenomenon when using an asymptotically vanishing damping. More precisely, we consider a viscous damping $\alpha(t)$ that may vary over time, and in particular that pro-

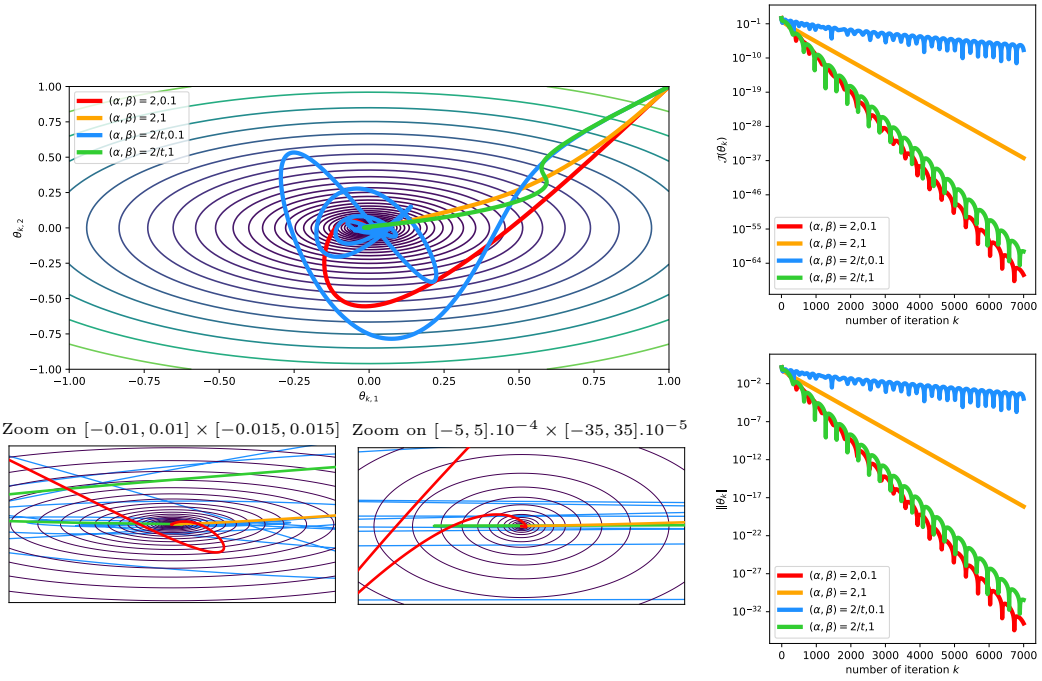


Figure 3.2: Illustration of the spiral phenomenon discussed in Section 3.3.2 on the function $\mathcal{J}\colon (\theta_1, \theta_2) \in \mathbb{R}^2 \mapsto \theta_1^2 + 2\theta_2^2$. Top-left figure displays the evolution of the iterates on the landscape of the loss function with two zooms on bottom-left figures. Right figures show the value of the loss function and the distance to the global minimizer $(0, 0)$ as a function of the iterations.

gressively decreases to zero as $t \to \infty$. Such damping has been given a lot of attention after the work of Su et al. (2014) who made a connection between Nesterov's accelerated gradient (Nesterov, 1983) and a differential equation with a damping proportional to $1/t$. As for DIN, if such a damping is used while keeping β fixed, we eventually have $\alpha(t)\beta \leq 1$ after some time. Our approach is however purely empirical since the Hartman-Grobman theorem holds only for autonomous ODEs[1] (hence with α remaining constant). In this setting we do observe spirals (see Figure 3.2), actually, we see on the blue curve that for $(\alpha(t), \beta) = (2/t, 0.1)$, the spirals are so large that the algorithm is much slower than it was for fixed values of (α, β). However when taking a larger β (green curve), these oscillations are damped (although still noticeable), yielding better performances in terms of speed.

[1]The theorem can be extended to some non-autonomous ODEs (Palmer, 1973), which we do not consider for the sake of simplicity.

3.4 Discrete case: INNA almost surely avoids saddle points

We now turn our attention to the asymptotic behavior of INNA introduced in Chapter 2. Let $\alpha \geq 0$ and $\beta > 0$ be two hyper-parameters. We designed INNA for non-smooth and stochastic applications, however, in order to study its asymptotic behavior, we consider a simpler framework. We analyze the algorithm for a loss function $\mathcal{J}$ that is twice continuously differentiable and consider a deterministic version of the algorithm. In this framework, we may use fixed step-sizes: let $\gamma > 0$ be a step-size, in this setting, we consider the deterministic version of INNA as,

$$\begin{cases} \theta_{k+1} &= \theta_k + \gamma \left[-(\alpha - \frac{1}{\beta})\theta_k - \frac{1}{\beta}\psi_k - \beta \nabla \mathcal{J}(\theta_k) \right] \\ \psi_{k+1} &= \psi_k + \gamma \left[-(\alpha - \frac{1}{\beta})\theta_k - \frac{1}{\beta}\psi_k \right] \end{cases}. \tag{3.19}$$

For the rest of this chapter, when we mention INNA, we refer to the deterministic smooth version (3.19). Since INNA is obtained by discretizing (3.1), we expect a similar behavior for both dynamics. Actually, we showed in the previous chapter that the set of stationary points of INNA is the same as that of DIN:

$$\mathsf{S} = \left\{ (\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P \middle| \nabla \mathcal{J}(\theta) = 0,\ \psi = (1 - \alpha\beta)\theta \right\}.$$

In this section, we prove that INNA is unlikely to converge to strict saddle points, that is, for almost any initialization. To this aim we recall the definition of the set $\mathsf{S}_{<0}$ previously introduced,

$$\mathsf{S}_{<0} = \left\{ (\theta, \psi) \in \mathsf{S} \middle| \nabla^2 \mathcal{J}(\theta) \text{ has at least one negative eigenvalue} \right\}.$$

3.4.1 INNA generically avoids strict saddles

In order to derive results for INNA similar to those for DIN, we will have to carefully choose the step-size $\gamma > 0$. To this aim, we need to assume the following.

Assumption 3.2. *There exists $L_{\nabla \mathcal{J}} > 0$ such that the gradient $\nabla \mathcal{J}$ of the loss function $\mathcal{J}$ is $L_{\nabla \mathcal{J}}$-Lipschitz continuous on $\mathbb{R}^P$ (with respect to a given norm $\|\cdot\|$ on $\mathbb{R}^P$).*

Recall from the introduction chapter that this assumption guarantees in particular that at any point of $\mathbb{R}^P$, the eigenvalues of $\nabla^2 \mathcal{J}$ are bounded by $L_{\nabla \mathcal{J}}$. Under this assumption our main result regarding INNA follows.

Theorem 3.9. *Under Assumption 3.1 and 3.2, if $\alpha > 0$ and the step-size γ is such that,*

$$0 < \gamma < \min\left(\frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}, \beta\right), \tag{3.20}$$

—where the right-hand side in (3.20) is always positive— then for almost any initialization, INNA does not converge to a point in $\mathsf{S}_{<0}$.

We can again formulate a corollary suited for practical applications.

Corollary 3.10. *Assume that $\mathcal{J}$ is a twice continuously differentiable Morse function and that Assumption 3.2 holds. Let $\alpha > 0$, $\beta > 0$ and a step-size γ such that $0 < \gamma < \min\left(\frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}, \beta\right)$. Consider the algorithm INNA for such a choice of α, β and γ. Denote by $\mathsf{C}_{\mathcal{J}}$ the set of initializations such that the associated sequences of iterates of INNA remain bounded and converge. Assume that $\mathsf{C}_{\mathcal{J}}$ has positive Lebesgue measure. Let (θ_0, ψ_0) be a non-degenerate random variable on $\mathsf{C}_{\mathcal{J}}$, and let $(\theta_k, \psi_k)_{k\in\mathbb{N}}$ be a sequence generated by (3.19), initialized at (θ_0, ψ_0) and converging to $(\theta^\star, \psi^\star) \in \mathbb{R}^P \times \mathbb{R}^P$. Then with probability one with respect to the draw of (θ_0, ψ_0), $\theta^\star$ is a local minimizer of $\mathcal{J}$.*

The proof follows similar lines as that of Corollary 3.3 and the practical consequences of Corollary 3.10 are the same as those discussed for DIN. In order to prove Theorem 3.9, we will use a version of the stable manifold theorem suited to the analysis of discrete processes.

3.4.2 Stable manifold theorem for discrete processes

We introduce a different version of the stable manifold theorem. This version was used by Lee et al. (2016) and O'Neill and S. J. Wright (2019) to analyze gradient descent and the HBF methods respectively. For a function $F : \mathbb{R}^P \to \mathbb{R}^P$ and for all $k \in \mathbb{N}_{>0}$, we introduce the following notation: $F^k = \underbrace{F \circ \ldots \circ F}_{k \text{ compisitions}}$. The theorem is the following.

Theorem 3.11 (III.7 (Shub, 2013))**.** *Let $\Theta^\star \in \mathbb{R}^{2P}$ be a fixed point for the C^1 local diffeomorphism $F : \mathsf{U} \to \mathbb{R}^{2P}$ where $\mathsf{U} \subset \mathbb{R}^{2P}$ is a neighborhood of $\Theta^\star$. Let $\mathsf{E}^{sc}(\Theta^\star)$ be the linear subspace spanned by the (complex) eigenvalues of $\mathrm{Jac}_F(\Theta^\star)$ with magnitude less than one. There exists a neighborhood Ω of $\Theta^\star$ and a C^1 manifold $\mathsf{W}^{sc}(\Theta^\star)$ tangent to $\mathsf{E}^{sc}(\Theta^\star)$ at $\Theta^\star$—whose dimension is the number of eigenvalues of $\mathrm{Jac}_F(\Theta^\star)$ with magnitude less than one—such that, for $\Theta_0 \in \mathbb{R}^{2P}$,*

(i) If $\Theta_0 \in \mathsf{W}^{sc}(\Theta^\star)$ and $F(\Theta_0) \in \Omega$ then $F(\Theta_0) \in \mathsf{W}^{sc}(\Theta^\star)$ (Invariance).

(ii) If $\forall k \in \mathbb{N}_{>0},\ F^k(\Theta_0) \in \Omega$, *then* $\Theta_0 \in \mathsf{W}^{sc}(\Theta^\star)$.

Although we study an iterative algorithm and not the solutions of an ODE, the results stated in Theorem 3.11 are very similar to those of Theorem 3.5, thus, the proof of Theorem 3.9 follows steps similar to those of the proof of Theorem 3.2.

Formulating INNA to use Theorem 3.11. Proceeding similarly to Section 3.3.1, for any $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$, we redefine the mapping G as,

$$G\begin{pmatrix}\theta\\ \psi\end{pmatrix} = \begin{pmatrix}\theta + \gamma\left[-(\alpha - \frac{1}{\beta})\theta - \frac{1}{\beta}\psi - \beta\nabla\mathcal{J}(\theta)\right]\\ \psi + \gamma\left[-(\alpha - \frac{1}{\beta})\theta - \frac{1}{\beta}\psi\right]\end{pmatrix}, \tag{3.21}$$

so that an iteration $k \in \mathbb{N}$ of INNA reads $(\theta_{k+1}, \psi_{k+1}) = G(\theta_k, \psi_k)$. Remark that unlike for (3.1), we now study the fixed points of G and not its zeros. Indeed, the iterative process INNA consists in successive compositions of the operator G and the set of fixed points of G is exactly S. Indeed, let $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$,

$$\begin{aligned} G(\theta,\psi) = (\theta,\psi) \iff &\begin{cases} -(\alpha - \frac{1}{\beta})\theta - \frac{1}{\beta}\psi - \beta\nabla\mathcal{J}(\theta) = 0 \\ -(\alpha - \frac{1}{\beta})\theta - \frac{1}{\beta}\psi = 0 \end{cases} \\ &\iff \begin{cases} \nabla\mathcal{J}(\theta) = 0 \\ \psi = (1-\alpha\beta)\theta \end{cases} . \end{aligned} \tag{3.22}$$

So (θ, ψ) is a fixed point of G if and only if $\nabla\mathcal{J}(\theta) = 0$ and $\psi = (1 - \alpha\beta)\theta$.

3.4.3 Proof of Theorem 3.9

Block-diagonal transformation. Throughout the proof we will use a block-diagonal transformation. Let $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$, since $\mathcal{J}$ is $C^2(\mathbb{R}^P)$ then G is $C^1(\mathbb{R}^P \times \mathbb{R}^P)$ and the Jacobian matrix of G at (θ, ψ) (displayed by block) reads,

$$\mathrm{Jac}_G(\theta,\psi) = \begin{pmatrix}(1 - \gamma(\alpha - \frac{1}{\beta}))I_P - \gamma\beta\nabla^2\mathcal{J}(\theta) & -\frac{\gamma}{\beta}I_P \\ -\gamma(\alpha - \frac{1}{\beta})I_P & (1 - \frac{\gamma}{\beta})I_P\end{pmatrix}. \tag{3.23}$$

Proceeding like in Section 3.3.1, their exists an orthogonal matrix $V \in \mathbb{R}^{P\times P}$ and a permutation $U \in \mathbb{R}^{2P\times 2P}$ (defined in Section 3.6) such that,

$$U^T \begin{pmatrix} V^T & 0 \\ 0 & V^T \end{pmatrix} \mathrm{Jac}_G(\theta,\psi) \begin{pmatrix} V & 0 \\ 0 & V \end{pmatrix} U = \begin{pmatrix} M_1 & & \\ & \ddots & \\ & & M_P \end{pmatrix}, \tag{3.24}$$

where for each $p \in \{1,\dots,P\}$, $M_p = \begin{pmatrix} 1-\gamma(\alpha-\frac{1}{\beta})-\gamma\beta\lambda_p & -\frac{\gamma}{\beta} \\ -\gamma(\alpha-\frac{1}{\beta}) & 1-\frac{\gamma}{\beta} \end{pmatrix}$—up to a symmetric permutation—and $(\lambda_p)_{p\in\{1,\dots,P\}}$ are the eigenvalues of $\nabla^2\mathcal{J}(\theta)$. To apply the stable manifold theorem and prove Theorem 3.9 we need G to be a local diffeomorphism. This result is non-straightforward to obtain, so we state it as a theorem before proving Theorem 3.9.

Theorem 3.12. *Under Assumption 3.2, for any $\alpha > 0$, $\beta > 0$ and*

$$0 < \gamma < \min\left(\frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}, \beta\right),$$

the mapping G defined in (3.21) is a local diffeomorphism from $\mathbb{R}^P \times \mathbb{R}^P$ to $\mathbb{R}^P \times \mathbb{R}^P$.

We show this result later in Section 3.7 of the appendix of this chapter.

Application of Theorem 3.11 to prove Theorem 3.9. We can now prove Theorem 3.9.

Proof of Theorem 3.9. Consider the mapping G defined in (3.21) with $\alpha > 0$, $\beta > 0$ and $0 < \gamma < \min\left(\frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2-4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}, \beta\right)$. By direct application of Theorem 3.12, G is a local diffeomorphism. Let $(\theta^\star,\psi^\star) \in \mathbb{R}^P \times \mathbb{R}^P$ be a fixed point of G. Our goal is to apply the stable manifold theorem in a neighborhood of this point.

To this aim, we study under which conditions on the eigenvalues of $\nabla^2\mathcal{J}(\theta^\star)$ the eigenvalues of $\mathrm{Jac}_G(\theta^\star,\psi^\star)$ have magnitude less than one. Throughout the proof we consider the same block-diagonal transformation of $\mathrm{Jac}_G(\theta^\star,\psi^\star)$ as in (3.24), and we keep the same notations. Let $p \in \{1,\dots,P\}$, the eigenvalues of M_p are the roots of the following polynomial,

$$\begin{aligned}\chi_{M_p}(X) &= X^2 - \mathrm{trace}(M_p)X + \det(M_p) \\ &= X^2 - (2-\gamma(\alpha+\beta\lambda_p))X + 1 - \gamma(\alpha+\beta\lambda_p) + \gamma^2\lambda_p. \end{aligned}\tag{3.25}$$

The discriminant of χ_{M_p} is,

$$\begin{aligned}\Delta_{M_p} &= (2-\gamma(\alpha+\beta\lambda_p))^2 - 4(1-\gamma(\alpha+\beta\lambda_p)+\gamma^2\lambda_p)\\ &= 4+\gamma^2(\alpha+\beta\lambda_p)^2 - 4\gamma(\alpha+\beta\lambda_p) - 4 + 4\gamma(\alpha+\beta\lambda_p) - 4\gamma^2\lambda_p\\ &= \gamma^2\left((\alpha+\beta\lambda_p)^2 - 4\lambda_p\right).\end{aligned}$$

Remark that up to a factor $\gamma^2 > 0$, this is the same discriminant as in (3.9) from Section 3.3.1. Therefore, we can once again use Lemma 3.6 to deduce that Δ_{M_p} is non-positive if and only if $\alpha\beta \leq 1$ and $\lambda_p \in \left[\frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, \frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}\right]$. We split the study with respect to the sign of Δ_{M_p}.

- If $\Delta_{M_p} > 0$, then M_p has two real eigenvalues,

$$\begin{cases}\sigma_{p,+} &= 1 - \frac{1}{2}\gamma(\alpha+\beta\lambda_p) + \frac{1}{2}\gamma\sqrt{(\alpha+\beta\lambda_p)^2 - 4\lambda_p}\\ \sigma_{p,-} &= 1 - \frac{1}{2}\gamma(\alpha+\beta\lambda_p) - \frac{1}{2}\gamma\sqrt{(\alpha+\beta\lambda_p)^2 - 4\lambda_p}\end{cases}. \tag{3.26}$$

  We then study whether the magnitudes of the eigenvalues are smaller or larger than 1, the computations are very similar to those of Section 3.3.1. If $\lambda_p < 0$, then $|(\alpha+\beta\lambda_p)| < \sqrt{(\alpha+\beta\lambda_p)^2 - 4\lambda_p}$, so $\sigma_{p,+} > 1$ and $\sigma_{p,-} < 1$, so we have at least one eigenvalue with magnitude larger than one. If $\lambda_p = 0$, then $\sigma_{p,+} = 1$ and[2] $|\sigma_{p,-}| = |1-\gamma\alpha| \leq 1$.

  In order to be exhaustive, remark that if $\lambda_p > 0$, then $(\alpha+\beta\lambda_p) > \sqrt{(\alpha+\beta\lambda_p)^2 - 4\lambda_p}$, so $0 \leq \sigma_{p,+} < 1$ and $-\infty < \sigma_{p,-} < 1$, so the bounds enforced on γ ensure that both eigenvalues have magnitude less than 1. This is indeed the case whenever $-\sigma_{p,-} < 1$, which is equivalent to $\gamma\left[\alpha+\beta\lambda_p + \sqrt{(\alpha+\beta\lambda_p)^2 - 4\lambda_p}\right] < 4$ and the latter always holds true. Indeed, when $\alpha\beta > 1$, one can show that the function $x > 0 \mapsto \alpha + \beta x + \sqrt{(\alpha+\beta x)^2 - 4x}$ is increasing (by differentiating it). Then, using Assumption 3.2 and the upper bound enforced on γ it holds,

$$\begin{aligned}&\gamma\left[\alpha+\beta\lambda_p + \sqrt{(\alpha+\beta\lambda_p)^2 - 4\lambda_p}\right]\\ &< \frac{(\alpha+\beta L_{\nabla\mathcal{J}}) - \sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}\left[(\alpha+\beta L_{\nabla\mathcal{J}}) + \sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}\right]\\ &< \frac{4L_{\nabla\mathcal{J}}}{2L_{\nabla\mathcal{J}}} = 2 < 4.\end{aligned} \tag{3.27}$$

[2] This is ensured by the boundaries enforced on γ as shown in the proof of Theorem 3.12.

On the other hand, when $\alpha\beta \leq 1$, by studying again the function $x > 0 \mapsto \alpha + \beta x + \sqrt{(\alpha+\beta x)^2 - 4x}$, one can show[3] that,

$$\alpha + \beta\lambda_p + \sqrt{(\alpha+\beta\lambda_p)^2 - 4\lambda_p} \\ \leq \max\left(2\alpha, \alpha + \beta L_{\nabla\mathcal{J}} + \sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}\right). \quad (3.28)$$

Then if the maximum in the right-hand side of (3.28) is 2α, it holds,

$$\gamma\left[\alpha + \beta\lambda_p + \sqrt{(\alpha+\beta\lambda_p)^2 - 4\lambda_p}\right] \leq 2\alpha\gamma \leq 2\alpha\beta \leq 2 < 4,$$

and if the maximum is the other value, we use (3.27) again. To summarize, when $\Delta_{M_p} > 0$, $\lambda_p \geq 0 \iff |\sigma_{p,+}| \leq 1$ and $|\sigma_{p,-}| \leq 1$.

- If $\Delta_{M_p} \leq 0$ then this implies that $\lambda_p \geq 0$ so $(\theta^\star, \psi^\star) \notin \mathsf{S}_{<0}$ and we do not need additional arguments. However, for the sake of completeness, we check whether the manifold in Theorem 3.11 may have positive measure around local minimizers with non-singular Hessian (in particular around any minimizer of a Morse function). The eigenvalues of M_p are,

$$\begin{cases} \sigma_{p,+} &= 1 - \frac{1}{2}\gamma(\alpha+\beta\lambda_p) + \frac{i}{2}\gamma\sqrt{4\lambda_p - (\alpha+\beta\lambda_p)^2} \\ \sigma_{p,-} &= 1 - \frac{1}{2}\gamma(\alpha+\beta\lambda_p) - \frac{i}{2}\gamma\sqrt{4\lambda_p - (\alpha+\beta\lambda_p)^2} \end{cases}. \quad (3.29)$$

Both eigenvalues have the same magnitude,

$$|\sigma_{p,+}|^2 = |\sigma_{p,-}|^2 = \left(1 - \frac{1}{2}\gamma(\alpha+\beta\lambda_p)\right)^2 + \frac{1}{4}\gamma^2(4\lambda_p - (\alpha+\beta\lambda_p)^2) \\ = 1 - \gamma(\alpha+\beta\lambda_p) + \gamma^2\lambda_p, \quad (3.30)$$

so,

$$|\sigma_{p,+}|^2 < 1 \iff -\gamma(\alpha+\beta\lambda_p) + \gamma^2\lambda_p < 0 \\ \iff (\gamma-\beta)\lambda_p < \alpha \iff \gamma < \beta + \frac{\alpha}{\lambda_p}. \quad (3.31)$$

[3]The proof is similar to the one of Lemma 3.15 given in the appendix of this chapter.

This is always true since,

$$\gamma < \frac{1}{2}(\beta + \frac{\alpha}{L_{\nabla\mathcal{J}}}) - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}} \leq \frac{1}{2}(\beta + \frac{\alpha}{L_{\nabla\mathcal{J}}}) \leq \beta + \frac{\alpha}{\lambda_p}. \tag{3.32}$$

We just proved that the eigenvalues of $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ have magnitude less than one if and only if $(\theta^\star, \psi^\star) \in \mathsf{S} \setminus \mathsf{S}_{<0}$.

We can now use the stable manifold theorem. Let $(\theta^\star, \psi^\star) \in \mathsf{S}_{<0}$. Let an initialization (θ_0, ψ_0) such that the associated realization $(\theta_k, \psi_k)_{k\in\mathbb{N}}$ of INNA converges to $(\theta^\star, \psi^\star)$. Consider the manifold $\mathsf{W}^{sc}(\theta^\star, \psi^\star)$ and the neighborhood Ω as defined in Theorem 3.11. Since $(\theta_k, \psi_k)_{k\in\mathbb{N}}$ converges, there exists $k_0 \in \mathbb{N}$ such that for all $k \geq k_0$, $(\theta_k, \psi_k) \in \Omega$, so according to Theorem 3.11, $\forall k \geq k_0$, $(\theta_k, \psi_k) \in \Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)$. Rewriting this with the operator G, $\forall k \geq k_0$, $G^k(\theta_0, \psi_0) \in \Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)$, and hence $\forall k \geq k_0$,

$$G^k(\theta_0, \psi_0) \in \bigcup_{j\in\mathbb{N}} G^{-j}\left(\Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)\right), \tag{3.33}$$

where $G^{-j}\left(\Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)\right)$ corresponds to all the initial conditions such that INNA has reached $\Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)$ after j iterations. Let

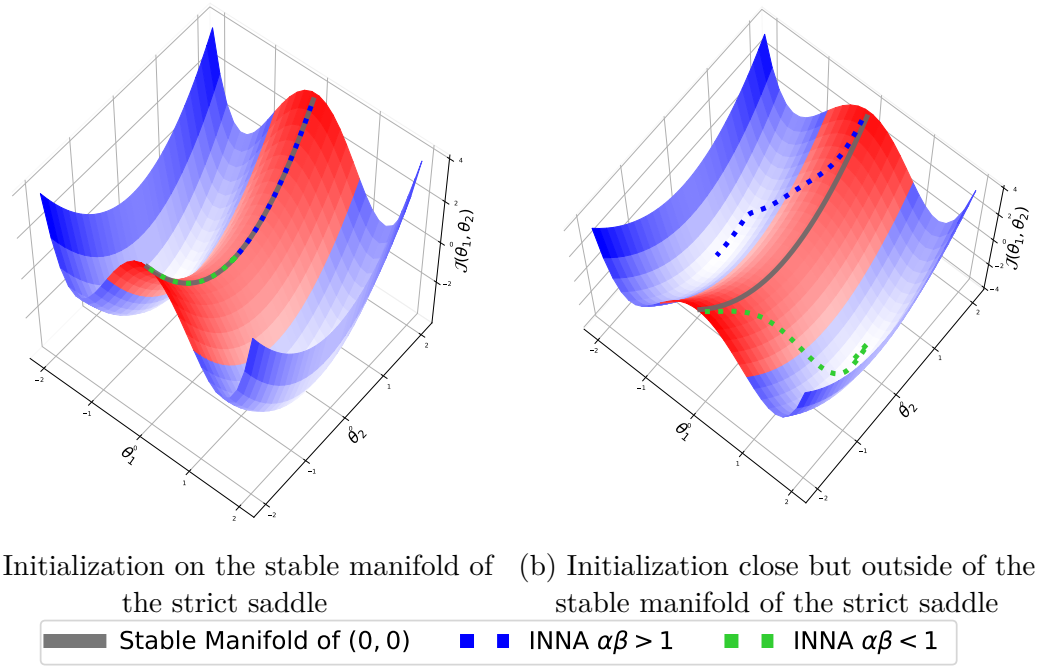
$$\mathsf{W}(\theta^\star, \psi^\star) = \left\{(\theta_0, \psi_0) \in \mathbb{R}^P \times \mathbb{R}^P \middle| G^k(\theta_0, \psi_0) \xrightarrow[k\to+\infty]{} (\theta^\star, \psi^\star)\right\}, \tag{3.34}$$

the set of all initial conditions such that INNA converges to $(\theta^\star, \psi^\star)$. From (3.33), it holds that,

$$\mathsf{W}(\theta^\star, \psi^\star) \subset \bigcup_{j\in\mathbb{N}} G^{-j}\left(\Omega \cap \mathsf{W}^{sc}(\theta^\star, \psi^\star)\right). \tag{3.35}$$

Then, we showed that since $(\theta^\star, \psi^\star) \in \mathsf{S}_{<0}$, then $\mathrm{Jac}_G(\theta^\star, \psi^\star)$ has at least one eigenvalue with magnitude strictly larger than one, so according to the stable manifold theorem, the dimension of $\mathsf{W}^{sc}(\theta^\star, \psi^\star)$ is strictly less than $2P$, hence this manifold has zero measure. By assumption the step-size γ is chosen such that G is a local diffeomorphism (from Theorem 3.12), so $\forall k \in \mathbb{N}$, G^{-k} is also a local diffeomorphism, hence it maps zero-measure sets to zero-measure sets. As a result, the right-hand side in (3.35) is a countable union of zero-measure sets, so it has zero measure, as well as $\mathsf{W}(\theta^\star, \psi^\star)$.

This proves the theorem since Assumption 3.1 guarantees that there is at most a countable number of critical points. So $\bigcup_{(\theta^\star,\psi^\star)\in\mathsf{S}_{<0}} \mathsf{W}(\theta^\star, \psi^\star)$ is a countable union of zero-measure sets so it has zero measure.



(a) Initialization on the stable manifold of the strict saddle

(b) Initialization close but outside of the stable manifold of the strict saddle

Figure 3.3: Evolution of the iterates of INNA on the landscape of the toy function $\mathcal{J}\colon (\theta_1, \theta_2) \in \mathbb{R}^2 \mapsto \theta_1^4 - 4\theta_1^2 + \theta_2^2$. This function has two minimizers $(-\sqrt{2}, 0)$ and $(\sqrt{2}, 0)$ and one strict saddle point $(0, 0)$. The red and blue surfaces represent the parts where $\mathcal{J}$ is locally concave and convex respectively. The stable manifold of $\mathcal{J}$ around $(0, 0)$ is represented by the grey curve. Left figure shows the behavior of INNA for two choices of hyper-parameters and for two initializations belonging to the stable manifold of $(0, 0)$. In this setting the algorithm does converge to the strict saddle $(0, 0)$. When initialized near but outside the manifold (right figure), the algorithm avoids the strict saddle and converges to a local minimizer for both choices of hyper-parameters.

□

3.4.4 Numerical Illustration

We finish the study of INNA with a short empirical illustration of Theorem 3.9 on a toy example. To this aim we consider the function $\mathcal{J} : (\theta_1, \theta_2) \in \mathbb{R}^2 \mapsto \theta_1^4 - 4\theta_1^2 + \theta_2^2$. This function is twice continuously differentiable on $\mathbb{R}^2$, non-convex, has a diagonal Hessian and three critical points: two local minimizers $(-\sqrt{2}, 0)$ and $(\sqrt{2}, 0)$ and one strict saddle point $(0, 0)$. The landscape of $\mathcal{J}$ is displayed on Figure 3.3. The set of initializations such that INNA converges to the strict saddle point $(0, 0)$ is the manifold $\theta_2 \in \mathbb{R} \mapsto (0, \theta_2)$ which has indeed zero measure on $\mathbb{R}^2$. Figure 3.3 shows that when initialized on this manifold,

Table 3.1: Empirical validation of the results of Theorem 3.9

		Percentage of convergence to each critical point			Average number of iterations to escape a saddle point
		$(-\sqrt{2}, 0)$	$(\sqrt{2}, 0)$	$(0, 0)$	
Initialization outside the stable manifold, very close to $(0, 0)$	INNA $\alpha\beta < 1$	48,8%	51,2%	0%	36
	INNA $\alpha\beta > 1$	50,7%	49,3%	0%	35
	GD	50,2%	49,8%	0%	37
Initialization on the stable manifold	INNA $\alpha\beta < 1$	0%	0%	100%	-
	INNA $\alpha\beta > 1$	0%	0%	100%	-
	GD	0%	0%	100%	-

the algorithm does converge to $(0, 0)$ but when initialized anywhere else, it avoids the strict saddle.

In addition to this illustration, we ran INNA and gradient descent—which is also known for almost surely escaping strict saddle points (Lee et al., 2016)—for 1000 random Gaussian initializations sampled from $\mathcal{N}_2(0, 10^{-24})$, hence extremely close to the saddle point $(0, 0)$. We also perform the same experiment but with a random initialization on the stable manifold. The results reported on Table 3.1 demonstrate that the algorithm always escapes the saddle point and converges to one of the two local minimizers. This empirically illustrates Theorem 3.9 and Corollary 3.10.

3.5 Conclusion

In this chapter, we provided a better understanding of the role played by the hyper-parameters α and β. This could help users of INNA to choose these parameters in practical applications. More importantly, we proved that the asymptotic behaviors of INNA and DIN make them relevant to tackle non-convex minimization problems. In particular, we provided conditions so that the deterministic version of INNA is likely to avoid strict saddle points of smooth functions. We may expect a similar behavior for INNA in its stochastic form. Indeed, one may intuitively think that the convergence to a strict saddle point is even more unlikely to occur in this case since the random noise will make the iterates escape the zero-measure stable manifold associated to this point (if they enter inside this manifold, which is already unlikely). Yet, proving such results in the stochastic setting is not straightforward and is left for future work. This concludes the part of this thesis on INNA, we will now

move on to another algorithm.

Appendices to Chapter 3

3.6 Permutation matrices

In this section, we specify the permutations matrices necessary to obtain the block diagonalization in (3.7) and (3.24). Denote by mod the modulo operator and let $P \in \mathbb{N}_{>0}$. We can choose the permutation matrix $U \in \mathbb{R}^{2P\times 2P}$ as the matrix whose coefficients are all zero except the following, for all $p \in \{1, \dots, P\}$,

$$\begin{aligned} \text{if } P \text{ is odd: } & \begin{cases} U_{P-p+1,p} & = 1 - \mathrm{mod}(p,2) \\ U_{P+p,2P-p+1} & = \mathrm{mod}(p,2) \\ U_{p,2P-p} & = \mathrm{mod}(p,2) \end{cases}, \\ \text{if } P \text{ is even: } & \begin{cases} U_{p,p} & = \mathrm{mod}(p,2) \\ U_{P+p,P+p} & = 1 - \mathrm{mod}(p,2) \\ U_{P+p,p} & = \mathrm{mod}(p,2) \\ U_{p,P+p} & = \mathrm{mod}(p,2) \end{cases}. \end{aligned} \tag{3.36}$$

For example, for $P = 3$ and $P = 4$ this yields the following matrices (where the zero coefficients are omitted for the sake of readability),

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix}.$$

3.7 Proof of Theorem 3.12

To prove Theorem 3.12, we introduce three technical lemmas.

Lemma 3.13. *For any $\alpha > 0$ and $\beta > 0$ such that $\alpha\beta > 1$, the function*

$$x \in \mathbb{R}_{>0} \mapsto \frac{\beta}{2} + \frac{\alpha}{2x} - \frac{\sqrt{(\alpha+\beta x)^2 - 4x}}{2x}$$

is continuous and decreasing both on $\mathbb{R}_{>0}$ and $\mathbb{R}_{<0}$.

Proof of Lemma 3.13. Let $\alpha > 0$ and $\beta > 0$ such that $\alpha\beta > 1$. The function $x \in \mathbb{R}_{>0} \mapsto \frac{\beta}{2} + \frac{\alpha}{2x} - \frac{\sqrt{(\alpha+\beta x)^2-4x}}{2x}$ is clearly $C^\infty(\mathbb{R}_{>0})$, and its first-order derivative is the function $x \in \mathbb{R}_{>0} \mapsto -\frac{\alpha\sqrt{(\beta x+\alpha)^2-4x}+(2-\alpha\beta)x-\alpha^2}{2x^2\sqrt{(\beta x+\alpha)^2-4x}}$. Since the denominator is always positive, we study the numerator of this derivative: define $h : x \in \mathbb{R}_{>0} \mapsto -\alpha\sqrt{(\beta x+\alpha)^2-4x}-(2-\alpha\beta)x+\alpha^2$. We will prove that h is negative by differentiating it: for all $x \in \mathbb{R}_{>0}$,

$$\frac{\partial h}{\partial x}(x) = -\frac{\alpha(2\beta(\beta x+\alpha)-4)}{2\sqrt{(\beta x+\alpha)^2-4x}} + \alpha\beta - 2. \tag{3.37}$$

$$\frac{\partial^2 h}{\partial x^2}(x) = -\frac{4\alpha(\alpha\beta-1)}{((\beta x+\alpha)^2-4x)^{\frac{3}{2}}}. \tag{3.38}$$

Since $\alpha\beta > 1$, for all $x \in \mathbb{R}_{>0}$, $\frac{\partial^2 h}{\partial x^2}(x) < 0$ and hence for all $x \in \mathbb{R}_{>0}$, $\frac{\partial h}{\partial x}(x) < \lim_{t\to 0} \frac{\partial h}{\partial x}(t) = 0$. So h is also decreasing on $\mathbb{R}_{>0}$, and $\lim_{x\to 0} h(x) = 0$, so for all $x \in \mathbb{R}_{>0}$, $h(x) \leq 0$ and the claim is thus proved on $\mathbb{R}_{>0}$. The proof is very similar on $\mathbb{R}_{<0}$ except that h is increasing on $\mathbb{R}_{<0}$ but $\lim_{x\to 0} h(x) = 0$, hence the result. □

Lemma 3.14. *For $\alpha > 0$, $\beta > 0$ such that $\alpha\beta \leq 1$, the function*

$$x \in \mathbb{R}_{<0} \mapsto \frac{\beta}{2} + \frac{\alpha}{2x} - \frac{\sqrt{(\alpha+\beta x)^2 - 4x}}{2x}$$

is continuous and increasing on $\mathbb{R}_{<0}$.

Proof of Lemma 3.14. The proof follows the exact same steps as those of the proof of Lemma 3.13, except that in (3.38), $\frac{\partial^2 h}{\partial x^2}$ is always positive on $\mathbb{R}_{<0}$, and we use that to deduce that $\frac{\partial h}{\partial x}$ defined in (3.37) is negative on $\mathbb{R}_{<0}$. So h is decreasing on $\mathbb{R}_{<0}$ and since $h(0) = 0$ we eventually obtain the result. □

Lemma 3.15. *Let $\alpha > 0$, $\beta > 0$ such that $\alpha\beta \leq 1$, the function*

$$x \in \mathbb{R}_{>0} \setminus [\frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, \frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}] \mapsto \frac{\beta}{2} + \frac{\alpha}{2x} - \frac{\sqrt{(\alpha+\beta x)^2 - 4x}}{2x}$$

is continuous and increasing for $x \in \left(0, \frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}\right)$ and continuous and decreasing for $x \in \left(\frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, +\infty\right)$.

Proof of Lemma 3.15. Let $\alpha > 0$, $\beta > 0$ such that $\alpha\beta \leq 1$. Denote by $x^- = \frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}$ and $x^+ = \frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}$ The function $x \in \mathbb{R}_{>0} \setminus (x^-, x^+) \mapsto \frac{\beta}{2} + \frac{\alpha}{2x} - \frac{\sqrt{(\alpha+\beta x)^2-4x}}{2x}$ is C^∞ on $(0, x^-)$ and on $(x^+, +\infty)$. Its first-order derivative is $x \in \mathbb{R}_{>0} \setminus (x^-, x^+) \mapsto -\frac{\alpha\sqrt{(\beta x+\alpha)^2-4x}+(2-\alpha\beta)x-\alpha^2}{2x^2\sqrt{(\beta x+\alpha)^2-4x}}$. The denominator is positive, so we focus on the numerator, define $h : x \in \mathbb{R}_{>0} \setminus (x^-, x^+) \mapsto -\alpha\sqrt{(\beta x+\alpha)^2 - 4x} - (2-\alpha\beta)x + \alpha^2$. For all $x \in \mathbb{R}_{>0} \setminus (x^-, x^+)$, the first and second-order derivatives of h are given by (3.37) and (3.38) respectively.

Since $\alpha\beta < 1$, $\frac{\partial^2 h}{\partial x^2}$ is always positive, so $\frac{\partial h}{\partial x}$ is increasing on both intervals. First, when $x \to 0$ with $0 < x < x^-$, $\frac{\partial h}{\partial x}(x) \to 0$, so $\frac{\partial h}{\partial x}$ is positive on $(0, x^-)$ and h is increasing on $(0, x^-)$. Since $h(0) = 0$ and h is increasing, we proved the first part of the lemma. Then, when $x \to +\infty$, $\frac{\partial h}{\partial x} \to -2$, so $\frac{\partial h}{\partial x}$ is negative on $(x^+, +\infty)$ and h is decreasing on $(x^+, +\infty)$. Finally, $h(x^+) = -\frac{4(1-\alpha\beta)}{\beta^2} - \frac{2(2-\alpha\beta)\sqrt{1-\alpha\beta}}{\beta^2} \leq 0$. □

We finally use these lemmas to prove the theorem.

Proof of Theorem 3.12. Let $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$. Since $\mathcal{J}$ is $C^2(\mathbb{R}^P)$ then G is $C^1(\mathbb{R}^{2P})$ and the Jacobian matrix $\mathrm{Jac}_G(\theta, \psi)$ can be transformed into a block diagonal matrix as in (3.24) where for any $p \in \{1, \dots, P\}$, M_p is a 2×2 block of the diagonal and λ_p is the associated eigenvalue of $\mathcal{J}(\theta)$. To prove that G is a local diffeomorphism we prove that $\mathrm{Jac}_G(\theta, \psi)$ is invertible (i.e., that it has non-zero determinant) and then use the local inversion theorem. It holds that $\det(\mathrm{Jac}_G(\theta, \psi)) = \prod_{p=1}^P \det(M_p)$, and for each $p \in \{1, \dots, P\}$,

$$\det(M_p) = (1 - \gamma(\alpha - \frac{1}{\beta}) - \gamma\beta\lambda_p)(1 - \frac{\gamma}{\beta}) - \frac{\gamma}{\beta}\gamma(\alpha - \frac{1}{\beta}) = 1 - \gamma(\alpha + \beta\lambda_p) + \gamma^2\lambda_p. \quad (3.39)$$

Let $p \in \{1, \dots, P\}$, we want to choose γ such that $\det(M_p) \neq 0$ for any $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$, hence for any $\lambda_p \in [-L_{\nabla\mathcal{J}}, L_{\nabla\mathcal{J}}]$ (since the eigenvalues are bounded by $L_{\nabla\mathcal{J}}$ from Assumption 3.2).

First, if $\lambda_p = 0$, from (3.39), we must take $\gamma \neq 1/\alpha$. From now on, we assume $\lambda_p \neq 0$, so (3.39) is a second-order polynomial in γ and its discriminant is $\Delta_\gamma = (\alpha + \beta\lambda_p)^2 - 4\lambda_p$. Notice that Δ_γ is a polynomial in λ_p and is exactly the discriminant that we studied in Section 3.3.1; its sign is given by Lemma 3.6. When this discriminant is non-negative, there exists two real roots to (3.39),

$$\begin{cases} \gamma^+ = \frac{(\alpha+\beta\lambda_p)}{2\lambda_p} + \frac{\sqrt{(\alpha+\beta\lambda_p)^2-4\lambda_p}}{2\lambda_p} = \frac{\beta}{2} + \frac{\alpha}{2\lambda_p} + \frac{\sqrt{(\alpha+\beta\lambda_p)^2-4\lambda_p}}{2\lambda_p} \\ \gamma^- = \frac{(\alpha+\beta\lambda_p)}{2\lambda_p} - \frac{\sqrt{(\alpha+\beta\lambda_p)^2-4\lambda_p}}{2\lambda_p} = \frac{\beta}{2} + \frac{\alpha}{2\lambda_p} - \frac{\sqrt{(\alpha+\beta\lambda_p)^2-4\lambda_p}}{2\lambda_p} \end{cases} . \tag{3.40}$$

As before, we split the study with respect to the value of $\alpha\beta$.

- If $\alpha\beta > 1$, then $\Delta_\gamma \geq 0$ and the roots of $\det(M_p)$ are given by (3.40).
  - First, when $\lambda_p < 0$, $\sqrt{(\alpha + \beta\lambda_p)^2 - 4\lambda_p} > |\alpha + \beta\lambda_p|$, so $\gamma^+ < 0$ and any positive choice of γ will never be equal to γ^+ in this case. Then by Lemma 3.13, γ^- is a decreasing function of λ_p for $\lambda_p < 0$ and when $\lambda_p \to 0$, $\gamma^- \to 1/\alpha$ (using L'Hôpital's rule), this yields a first condition $\gamma < 1/\alpha$.
  - When $\lambda_p > 0$, observe that $\gamma^+ \geq \gamma^- > 0$ so we focus the study on γ^-. Lemma 3.13 exactly states that γ^- is a decreasing function of $\lambda_p > 0$. Since $\mathcal{J}$ has $L_{\nabla\mathcal{J}}$-Lipschitz gradient, $\lambda_p \leq L_{\nabla\mathcal{J}}$, so, for all $\lambda_p \in (0, L_{\nabla\mathcal{J}}]$, $\gamma^- \geq \frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2-4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}$. Note in addition that when $\lambda_p \to 0$, $\gamma^- \to 1/\alpha$, this is a simple way to prove that $1/\alpha > \frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2-4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}$ when $\alpha\beta > 1$.

  To summarize, we had three conditions, $\gamma \neq 1/\alpha$, $\gamma < 1/\alpha$ and $\gamma < \frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2-4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}$ and we proved that the latter implies the first-two conditions. Remark that the condition $\gamma < \beta$ holds but is not necessary in the case $\alpha\beta > 1$, it is present in the statement of the theorem to keep it as simple as possible.

- We now assume that $\alpha\beta \leq 1$.
  - If $\lambda_p < 0$, then $\Delta_\gamma > 0$ and the roots are given by (3.40). As above, it holds that $\sqrt{(\alpha + \beta\lambda_p)^2 - 4\lambda_p} > |\alpha + \beta\lambda_p|$, so $\gamma^+ < 0$. Then Lemma 3.14 states that γ^- is an increasing function of $\lambda_p < 0$, and when $\lambda_p \to -\infty$, $\gamma^- \to \beta$. So we need $\gamma < \beta$.
  - If $\lambda_p > 0$, then whenever $\lambda_p \in [\frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, \frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}]$, there are no real roots so $\det(M_p) \neq 0$ regardless the choice of $\gamma > 0$. If λ_p does not belong to interval previously mentioned, the roots are given by (3.40). Remark

that $\gamma^+ > \gamma^- > 0$ so we focus on γ^-. Using Lemma 3.15, γ^- is increasing on $(0, \frac{2-\alpha\beta}{\beta^2} - \frac{2\sqrt{1-\alpha\beta}}{\beta^2})$ and tends to $1/\alpha$ when $\lambda_p \to 0$ (using L'Hôpital's rule). The same lemma also state that γ^- is decreasing on $(\frac{2-\alpha\beta}{\beta^2} + \frac{2\sqrt{1-\alpha\beta}}{\beta^2}, +\infty)$, so using the $L_{\nabla\mathcal{J}}$-Lipschitz gradient of $\mathcal{J}$, $\gamma^- \leq \frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}$ on this interval. Note however that we do not necessarily have $1/\alpha > \frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}$.

Overall, for $\alpha\beta \leq 1$ we must have $0 < \gamma < \min\left(\frac{\beta}{2} + \frac{\alpha}{2L_{\nabla\mathcal{J}}} - \frac{\sqrt{(\alpha+\beta L_{\nabla\mathcal{J}})^2 - 4L_{\nabla\mathcal{J}}}}{2L_{\nabla\mathcal{J}}}, \frac{1}{\alpha}, \beta\right)$ and $\alpha\beta \leq 1 \implies \beta \leq 1/\alpha$ hence the result.

In every case we proved that the conditions mentioned in the theorem are sufficient to ensure that for all $(\theta, \psi) \in \mathbb{R}^P \times \mathbb{R}^P$, $\det(\mathrm{Jac}_G(\theta, \psi)) \neq 0$. So by the local inversion theorem, G is a local diffeomorphism from $\mathbb{R}^P \times \mathbb{R}^P$ to $\mathbb{R}^P \times \mathbb{R}^P$. □

Chapter 4

Second-order Step-size Tuning of SGD for Non-convex Optimization

This chapter is adapted from Castera et al. (2021b).

Contents

4.1 Introduction

In this last chapter, we focus on using second-order information for tuning the step-sizes of SGD according to the discussion at the end of Section 1.4.1. Given $P \in \mathbb{N}_{>0}$, we consider

twice-differentiable loss functions $\mathcal{J} : \mathbb{R}^P \to \mathbb{R}$ such that each term that constitutes its sum structure in (1.5) is also twice differentiable. Recall from the introduction that for all $k \in \mathbb{N}$ and $\theta_k \in \mathbb{R}^P$, an iteration of SGD reads,

$$\theta_{k+1} = \theta_k - \gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k), \tag{4.1}$$

where $\mathsf{B}_k \subset \{1, \dots, N\}$ is a mini-batch and $\gamma_k > 0$ is a step-size. While adaptive methods (ADAGRAD, RMSprop, etc.) act as preconditioners and prescribe vector step-sizes (see Section 1.4.3.2), we focus exclusively on fine-tuning the scalar step-sizes of vanilla SGD. Our goal here is to adapt the step-sizes to the local shape (the local *curvature*) of the landscape of the loss function, and account in particular for local convexity, local concavity, etc. Such information can be estimated using second-order derivatives as we explain hereafter. However, we will have to overcome once more the computational cost of second-order information and the noise produced by mini-batch sub-sampling strategies.

Our starting point is an infinitesimal second-order variational model along the negative gradient direction, close to what we did for GD in (1.25). Such infinitesimal model is particularly relevant in DL since one must use vanishing step-sizes to control the effects of mini-batch sub-sampling (recall Figure 1.7). Second-order information is approximated with first-order quantities using finite differences. We will first derive a deterministic (or full-batch) method. This deterministic method is not really new: it corresponds to a non-convex version of the Barzilai-Borwein (BB) method (Barzilai and Borwein, 1988) and is somehow a discrete non-convex adaption of the continuous gradient system considered in Alvarez and Cabot (2004). It is also close to earlier work (Raydan, 1997), with the major difference that our algorithm is supported by a variational model. These variational ideas will then be essential to adapt our method to mini-batch sub-sampling.

The main contribution of this chapter is to efficiently adapt the deterministic method mentioned above to the mini-batch sub-sampling setting. The resulting algorithm is called Step-Tuned SGD, it features fine-tuned step-sizes which accelerate SGD in most of our numerical experiments. We also provide convergence guarantees and rates for this new algorithm. The proofs of this chapter are not based on the ODE approach that we used for INNA. Instead, we use the smoothness assumption stated at the beginning of this chapter and follow a more "classical" approach based on a descent lemma (see Section 4.4).

Regarding the organization of the chapter, the preliminary deterministic algorithm is derived in Section 4.3.1 and is then adapted to mini-batch sub-sampling in Section 4.3.2. Theoretical results are stated in Section 4.4 and DL experiments are presented in Section 4.5.

We first review some literature that is specifically related to this chapter.

4.2 Literature related to this chapter

Our method belongs to the class of BB methods. Many variations of this method have been proposed for deterministic optimization (Dai et al., 2002; Xiao et al., 2010; Biglari and Solimanpur, 2013; Babaie-Kafaki and Fatemi, 2013; Curtis and Guo, 2016). Our methods aims to detect the sign of the local curvature of the loss function using a convexity test similar to those provided in Babaie-Kafaki and Fatemi (2013) and Curtis and Guo (2016). BB methods may be quite unstable and are often stabilized with line-search techniques (Armijo, 1966). As we explained in Chapter 1, classical line-search approaches are not well suited for stochastic optimization since we never evaluate $\mathcal{J}$ exactly. It is thus hard to combine BB methods with noisy gradient estimates. Most stochastic BB algorithms (Tan et al., 2016; Liang et al., 2019; Robles-Kelly and Nazari, 2019) overcome this issue with averaging techniques in the style of SVRG (Johnson and T. Zhang, 2013). This reduces the instability caused by mini-batch sub-sampling but only allows prescribing a new step-size at every epoch. As such, they fail to capture local variations of curvature at each iteration. In addition, the step-sizes of the vanilla BB algorithm are positive for strongly-convex functions only. Since the methods stated above cannot capture local information, they cannot exploit non-convexity and are limited to using absolute values to prevent negative step-sizes.

On the contrary, our stochastic method can adapt to local curvature every two iterations and can thus discriminate flat, locally concave and locally convex regions. Regarding the utilization of flatness and concavity of DL loss functions, the AdaBelief algorithm of Zhuang et al. (2020) is worth mentioning. The latter uses the difference between the current stochastic gradient estimate and the average over past gradients, this difference is then used to prescribe a vector step-size in the style of ADAM. In comparison, our method uses scalar step-sizes and aims to capture subtle variations as it computes a stabilized difference between consecutive gradient estimates *before* averaging.

Finally, the proofs of convergence of this chapter partially follow arguments provided by Li and Orabona (2019) for the scalar version of ADAGRAD with the difference that we do not assume the global Lipschitz continuity of the gradient.

4.3 Design of the algorithm

We first build a preliminary full-batch algorithm based upon a simple second-order variational model. We then adapt this algorithm to address mini-batch stochastic approximations.

4.3.1 Deterministic full-batch algorithm

Second-order infinitesimal step-size tuning. Let $\mathcal{J}\colon \mathbb{R}^P \to \mathbb{R}$ a loss function, that we assume to be twice differentiable. Let $\theta \in \mathbb{R}^P$, given an update direction $d \in \mathbb{R}^P$, a natural strategy is to choose $\gamma \in \mathbb{R}$ that minimizes $\mathcal{J}(\theta+\gamma d)$. Let us approximate $\gamma \in \mathbb{R} \mapsto \mathcal{J}(\theta+\gamma d)$ around 0 with a Taylor expansion,

$$q_d(\gamma) \stackrel{\text{def}}{=} \mathcal{J}(\theta) + \gamma\langle \nabla\mathcal{J}(\theta), d\rangle + \frac{\gamma^2}{2}\langle \nabla^2\mathcal{J}(\theta)d, d\rangle. \tag{4.2}$$

If the curvature term $\langle \nabla^2\mathcal{J}(\theta)d, d\rangle$ is positive, then q_d has a unique minimizer at,

$$\gamma^\star = -\frac{\langle \nabla\mathcal{J}(\theta), d\rangle}{\langle \nabla^2\mathcal{J}(\theta)d, d\rangle}. \tag{4.3}$$

On the contrary when $\langle \nabla^2\mathcal{J}(\theta)d, d\rangle \leq 0$, the infinitesimal model q_d is concave (or equivalently $\mathcal{J}$ is locally concave in the direction d) which suggests taking a large step-size $\gamma > 0$. Indeed, since $\mathcal{J}$ is locally convex around local minima, one will want to escape regions where local concavity is detected. These considerations are illustrated on Figure 4.1.

Tuning gradient descent. In order to tune GD we choose the direction $d = -\nabla\mathcal{J}(\theta)$ which gives,

$$\gamma(\theta) \stackrel{\text{def}}{=} \frac{\|\nabla\mathcal{J}(\theta)\|^2}{\langle \nabla^2\mathcal{J}(\theta)\nabla\mathcal{J}(\theta), \nabla\mathcal{J}(\theta)\rangle}. \tag{4.4}$$

According to the previous discussion, we would ideally use $\gamma_k = \gamma(\theta_k)$ for GD when $\gamma(\theta_k) > 0$. Yet, we will later use a discretization procedure to overcome the computational cost of (4.4), and it will then appear that seeking a step-size γ_k such that, $\gamma_k \simeq \gamma(\theta_{k-1})$ (again when $\gamma(\theta_{k-1}) > 0$) is not ideal but is less expensive. Let $k \geq 1$ be an iteration index of GD and assume that $\theta_{k-1} \in \mathbb{R}^P$ and $\gamma_{k-1} > 0$ are known. Let us approximate the quantity,

$$\gamma(\theta_{k-1}) = \frac{\|\nabla\mathcal{J}(\theta_{k-1})\|^2}{\langle \nabla^2\mathcal{J}(\theta_{k-1})\nabla\mathcal{J}(\theta_{k-1}), \nabla\mathcal{J}(\theta_{k-1})\rangle}, \tag{4.5}$$

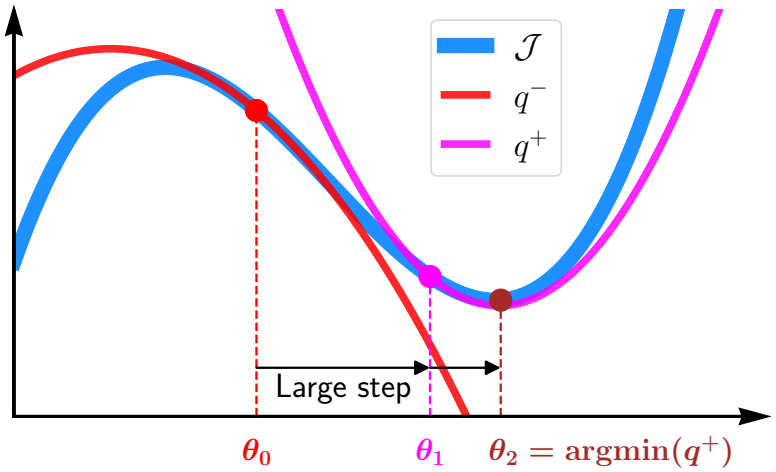


Figure 4.1: Illustration of negative and positive curvature steps. The function q^- represents the variational model at θ_0, with negative curvature. Concavity suggests taking a large step to reach θ_1. Then, at θ_1, the variational model q^+ has positive curvature and can be minimized to obtain θ_2.

using only first-order objects. We rely on two identities,

$$\Delta\theta_k \overset{\text{def}}{=} \theta_k - \theta_{k-1} = -\gamma_{k-1}\nabla\mathcal{J}(\theta_{k-1}), \tag{4.6}$$

$$\Delta g_k \overset{\text{def}}{=} \nabla\mathcal{J}(\theta_k) - \nabla\mathcal{J}(\theta_{k-1}) \simeq -\gamma_{k-1}\mathcal{C}_{\mathcal{J}}(\theta_{k-1}), \tag{4.7}$$

where for all $\theta \in \mathbb{R}^P$, $\mathcal{C}_{\mathcal{J}}(\theta) \overset{\text{def}}{=} \nabla^2\mathcal{J}(\theta)\nabla\mathcal{J}(\theta)$ and (4.7) is obtained by Taylor's formula. Combining the above leads us to consider the following step-size,

$$\gamma_k = \begin{cases} \frac{\|\Delta\theta_k\|^2}{\langle\Delta\theta_k, \Delta g_k\rangle} & \text{if } \langle\Delta\theta_k, \Delta g_k\rangle > 0 \\ \nu & \text{otherwise} \end{cases}, \tag{4.8}$$

where $\nu > 0$ is a hyper-parameter of the algorithm representing the large step-sizes to use in locally-concave regions.

The resulting full-batch non-convex optimization method is detailed in Algorithm 2, in which $\alpha > 0$ is the learning rate as in ADAGRAD or ADAM, not to be confused with the friction parameter of INNA. We insist on the fact that Algorithm 2 is present in the literature under subtle variants (Raydan, 1997; Dai et al., 2002; Xiao et al., 2010; Biglari and Solimanpur, 2013). It belongs to the class of non-convex BB methods. In particular, (4.8)

Algorithm 2 Full-batch preliminary algorithm

Input: $\alpha > 0$, $\nu > 0$
Initialize $\theta_0 \in \mathbb{R}^P$
$\theta_1 = \theta_0 - \alpha \nabla \mathcal{J}(\theta_0)$
for $k = 1, \dots$ **do**
  $\Delta g_k = \nabla \mathcal{J}(\theta_k) - \nabla \mathcal{J}(\theta_{k-1})$
  $\Delta \theta_k = \theta_k - \theta_{k-1}$
  if $\langle \Delta g_k, \Delta \theta_k \rangle > 0$ **then**
    $\gamma_k = \frac{\|\Delta \theta_k\|^2}{\langle \Delta g_k, \Delta \theta_k \rangle}$
  else
    $\gamma_k = \nu$
  end if
  $\theta_{k+1} = \theta_k - \alpha \gamma_k \nabla \mathcal{J}(\theta_k)$
end for

coincides with the vanilla BB step-size when $\langle \Delta \theta_k, \Delta g_k \rangle$ is positive. The main difference is that we introduce a scaling factor α to overcome the need of stabilizing the method with line-search procedures (see Section 4.2). We do so in anticipation of the mini-batch sub-sampling adaptation carried out in the next section, where line-search is unavailable.

Such a scaling factor α, is present in most DL optimizers (ADAGRAD, ADAM, RMSprop, etc.) and generally requires tuning. Recall that our goal here is not to get rid of hyper-parameter pre-tuning (which is an important problem as well) but rather to combine this α with an automatic fine-tuned sequence $(\gamma_k)_{k \in \mathbb{N}}$ in order to accelerate the training. Although Algorithm 2 is close to existing methods, the interest of our variational viewpoint is the characterization of the underlying geometrical mechanism supporting the algorithm. This will be the key to designing an efficient stochastic version of Algorithm 2 in Section 4.3.2.

Illustrative experiment. Before presenting the mini-batch version, we illustrate the interest of exploiting negative curvature through the *large-step* parameter ν with a synthetic experiment inspired from Carmon et al. (2017). We apply Algorithm 2 to a non-convex regression problem of the form $\min_{\theta \in \mathbb{R}^P} \sum_{n=1}^{N} \phi(A_n \theta - b_n)$, specified in Section 4.7 of the appendix of this chapter, and where ϕ is non-convex. We compare Algorithm 2 with a classical BB method where absolute values are used when the step-size is negative[1] (see, e.g., Tan et al. (2016) and Liang et al. (2019) in stochastic settings), we also compare the results with GD used with Armijo's line-search method. As shown on Figure 4.2, Algorithm 2 efficiently exploits local curvature and converges much faster than other methods.

[1]For a fair comparison we implement this method with the scaling-factor α of Algorithm 2.

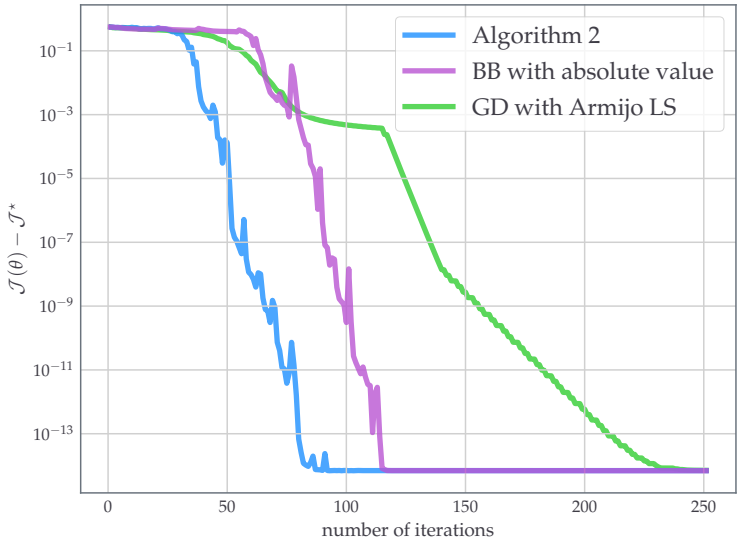


Figure 4.2: Values of the loss function $\mathcal{J}(\theta)$ against iterations (each corresponding to a gradient step) for the synthetic non-convex regression problem detailed in Section 4.7 of the appendix of this chapter. The optimal value $\mathcal{J}^\star$ is unknown and is estimated by taking the best value obtained among all algorithms after 10^5 iterations.

4.3.2 Stochastic mini-batch algorithm

We wish to adapt Algorithm 2 for DL applications, and in particular, make it compatible with mini-batch sub-sampling. As in the previous chapters, let $N \in \mathbb{N}_{>0}$ and consider a loss function $\mathcal{J} : \mathbb{R}^P \to \mathbb{R}$ which takes the form of a sum $\mathcal{J} = \sum_{n=1}^N \mathcal{J}_n$, where each $\mathcal{J}_n$ is assumed to be twice continuously differentiable.

Reminder on mini-batch sub-sampling. We consider the mini-batch sub-sampling procedure described in Section 1.3.4. Since $\mathcal{J}_n$ is differentiable for all $n \in \{1, \dots, N\}$, for any $\mathsf{B} \subset \{1, \dots, N\}$, we recall the definition of the following quantities, for any $\theta \in \mathbb{R}^P$,

$$\mathcal{J}_\mathsf{B}(\theta) = \frac{1}{|\mathsf{B}|} \sum_{n \in \mathsf{B}} \mathcal{J}_n(\theta), \quad \text{and} \quad \nabla \mathcal{J}_\mathsf{B}(\theta) = \frac{1}{|\mathsf{B}|} \sum_{n \in \mathsf{B}} \nabla \mathcal{J}_n(\theta). \tag{4.9}$$

where again $|\mathsf{B}|$ denotes the number of elements of the set B. Like in Section 1.3.4, the mini-batches are independent copies of a random subset $\mathsf{A} \subset \{1, \dots, N\}$ such that, $\mathcal{J} = \mathbb{E}[\mathcal{J}_\mathsf{A}]$ and $\nabla \mathcal{J} = \mathbb{E}[\nabla \mathcal{J}_\mathsf{A}]$ (where the expectation is taken over the random draw of A). This is valid for example if A is taken uniformly at random over all possible subsets of fixed size,

Algorithm 3 Step-Tuned SGD

Input: $\alpha > 0$, $\nu > 0$
Input: $\beta \in [0,1]$, $\tilde{m} > 0$, $\tilde{M} > 0$, $\delta \in (0, 1/2)$
Initialize $\theta_0 \in \mathbb{R}^P$, $G_{-1} = \mathbf{0}_P$, $\gamma_0 = 1$
Draw independent random mini-batches $(\mathsf{B}_k)_{k\in\mathbb{N}}$.
for $k = 0, 1, \ldots$ **do**
  $\theta_{k+\frac{1}{2}} = \theta_k - \frac{\alpha}{(k+1)^{1/2+\delta}}\gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)$
  $\theta_{k+1} = \theta_{k+\frac{1}{2}} - \frac{\alpha}{(k+1)^{1/2+\delta}}\gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})$
  $\Delta\theta_{\mathsf{B}_k} = \theta_{k+\frac{1}{2}} - \theta_k$
  $\Delta g_{\mathsf{B}_k} = \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}}) - \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)$
  $G_k = \beta G_{k-1} + (1-\beta)\Delta g_{\mathsf{B}_k}$
  $\hat{G}_k = G_k/(1-\beta^{k+1})$
  if $\langle \hat{G}_k, \Delta\theta_{\mathsf{B}_k}\rangle > 0$ **then**
    $\gamma_{k+1} = \frac{\|\Delta\theta_{\mathsf{B}_k}\|^2}{\langle \hat{G}_k, \Delta\theta_{\mathsf{B}_k}\rangle}$
  else
    $\gamma_{k+1} = \nu$
  end if
  $\gamma_{k+1} = \min(\max(\gamma_{k+1}, \tilde{m}), \tilde{M})$
end for

as we shown in (1.19).

Second-order tuning of mini-batch SGD: Step-Tuned SGD. Our goal is to devise a step-size strategy, based on the variational ideas developed earlier and on the quantity $\mathcal{C}_{\mathcal{J}}$ (defined right after Equation 4.6), in the context of mini-batch sub-sampling. First observe that for $\theta \in \mathbb{R}^P$,

$$\mathcal{C}_{\mathcal{J}}(\theta) = \nabla^2 \mathcal{J}(\theta)\nabla \mathcal{J}(\theta) = \nabla\left(\frac{1}{2}\|\nabla \mathcal{J}(\theta)\|^2\right). \tag{4.10}$$

So rewriting $\mathcal{J}$ as an expectation over the random subset A, for all $\theta \in \mathbb{R}^P$ it holds

$$\mathcal{C}_{\mathcal{J}}(\theta) = \nabla\left(\frac{1}{2}\|\mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{A}}(\theta)\right]\|^2\right). \tag{4.11}$$

In order to build an infinitesimal model as in (4.4), this suggests the following mini-batch estimator of $\mathcal{C}_{\mathcal{J}}$,

$$\mathcal{C}_{\mathcal{J}_{\mathsf{B}}}(\theta) \overset{\text{def}}{=} \nabla\left(\frac{1}{2}\|\nabla \mathcal{J}_{\mathsf{B}}(\theta)\|^2\right) = \nabla^2 \mathcal{J}_{\mathsf{B}}(\theta)\nabla \mathcal{J}_{\mathsf{B}}(\theta),$$

where $\mathsf{B} \subset \{1, \ldots, N\}$ and $\theta \in \mathbb{R}^P$.

Like in the deterministic case, we reduce the computational cost by approximating our new target quantity (4.12) with a Taylor expansion of $\mathcal{J}_{\mathsf{B}}$ between two iterations of SGD. We obtain for any $\mathsf{B} \subset \{1, \dots, N\}$, $\theta \in \mathbb{R}^P$, and small $\gamma > 0$,

$$-\gamma \mathcal{C}_{\mathcal{J}_{\mathsf{B}}}(\theta) \simeq \nabla \mathcal{J}_{\mathsf{B}}(\underbrace{\theta - \gamma \nabla \mathcal{J}_{\mathsf{B}}(\theta)}_{\text{next iterate}}) - \nabla \mathcal{J}_{\mathsf{B}}(\theta). \tag{4.12}$$

In order to accurately approximate our target quantity, (4.12) indicates that we must compute two gradient estimates on the same mini-batch. This suggests that we should use each mini-batch twice in SGD and compute a difference of gradients only every two iterations. Note that we could also compute an additional gradient estimate at each iteration of SGD similarly to what Schraudolph et al. (2007) previously did for a stochastic BFGS algorithm, but this would double the computational cost.

We now build the algorithm based on this principle. Let an initialization $\theta_0 \in \mathbb{R}^P$, and a sequence of i.i.d. random mini-batches $(\mathsf{B}_k)_{k \in \mathbb{N}}$, whose common distribution is the same as A. We change the way of enumerating the iterations compared to the previous chapters. We adopt the following convention: at iteration $k \in \mathbb{N}$ and for $\theta_k \in \mathbb{R}^P$, the random mini-batch B_k is used to compute a stochastic gradient estimate $\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)$ and one iteration of SGD is then performed (with a step-size that we discuss here-after). We denote $k + \frac{1}{2}$ (rather than $k + 1$) the index of the next iteration and $\theta_{k+\frac{1}{2}}$ the corresponding iterate. Then, *the same mini-batch* is used again to compute another stochastic gradient estimate $\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})$ and to perform the next iteration of SGD. Doing so, we obtain the next iterate θ_{k+1}, at this iteration we use the next mini-batch B_{k+1}, etc. This new way of using mini-batches is convenient in view of (4.12). Let us define,

$$\Delta g_{\mathsf{B}_k} \stackrel{\text{def}}{=} \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}}) - \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k), \tag{4.13}$$

thereby Δg_{B_k} forms an approximation of $-\gamma_k \mathcal{C}_{\mathcal{J}_{\mathsf{B}_k}}(\theta_k)$ that we can use to compute the next step-size γ_{k+1}. We define the difference between two iterates accordingly,

$$\Delta \theta_{\mathsf{B}_k} \stackrel{\text{def}}{=} \theta_{k+\frac{1}{2}} - \theta_k. \tag{4.14}$$

We could now build a mini-batch adaptation of Algorithm 2. Before that, we additionally stabilize the approximation of the target quantity in (4.12) by using an exponential moving average of the previously computed $(\Delta g_{\mathsf{B}_j})_{j \leq k}$. More precisely, we recursively compute G_k defined by,

$$G_k = \beta G_{k-1} + (1 - \beta) \Delta g_{\mathsf{B}_k}. \tag{4.15}$$

We finally introduce $\hat{G}_k = G_k/(1-\beta^{k+1})$ to debias the estimator G_k such that the sum of the weights in the average equals 1. This mostly impacts the first iterations as β^{k+1} vanishes quickly; a similar process is often used for ADAM, see Kingma and Ba (2015). The motivation for averaging is discussed in Section 4.3.3.

Altogether we obtain our main method: Algorithm 3, which we name *Step-Tuned SGD*, as it aims to tune the step-size every two iterations and not only at each epoch like most stochastic BB methods. Note that the main idea behind Step-Tuned SGD remains the same as in the deterministic setting: we exploit the curvature properties of the $(\mathcal{J}_{\mathsf{B}_k})_{k\in\mathbb{N}}$ through the quantities $\langle \hat{G}_k, \Delta\theta_{\mathsf{B}_k}\rangle$ to devise our method. Note that compared to Algorithm 2, the iteration index is shifted by 1 so that the estimated step-size γ_{k+1} only depends on mini-batches B_0 up to B_k and is therefore conditionally independent of B_{k+1}. This conditional dependency structure is crucial to obtain the convergence guarantees given in Section 4.4.

Remark 4.1. *Like Algorithm 2 and like most methods, Algorithm 3 does not alleviate the need of tuning the scaling factor α, tuning this parameter remains indeed important in most practical applications. Our main goal here is to accelerate SGD with fine-tuned step-sizes, this is analogous to our previous discussion on BB methods for deterministic applications which often accelerate algorithms but must be stabilized with line-search strategies (replaced here by the introduction of α). To the best of our knowledge, in comparison to the epoch-wise BB methods (Tan et al., 2016; Liang et al., 2019), Algorithm 3 is the first method that manages to mimic the iteration-wise behavior of deterministic BB methods for mini-batch applications.*

4.3.3 Heuristic construction of Step-Tuned SGD

In the previous section, we described the main elements from which Algorithm 3 is made. We now present the main ideas that led us to build Algorithm 3 this way and discuss the hyper-parameters. Throughout this paragraph, we use the term *gradient variation* (GV) which refers to the local variations of the gradient: it is simply the difference of the gradients at two consecutive iterates. Our heuristic discussion blends discretization arguments and experimental considerations. We use the non-convex regression experiment of Section 4.3.1 as a test for our intuition and algorithms. A complete description of the methods introduced below is given in Section 4.9, we only sketch the main ideas.

First heuristic experiment with exact GVs. Assume that along any ordered collection of points $\theta_0, \dots, \theta_k \in \mathbb{R}^P$, one is able to evaluate the GVs of $\mathcal{J}$, that is, terms of the form

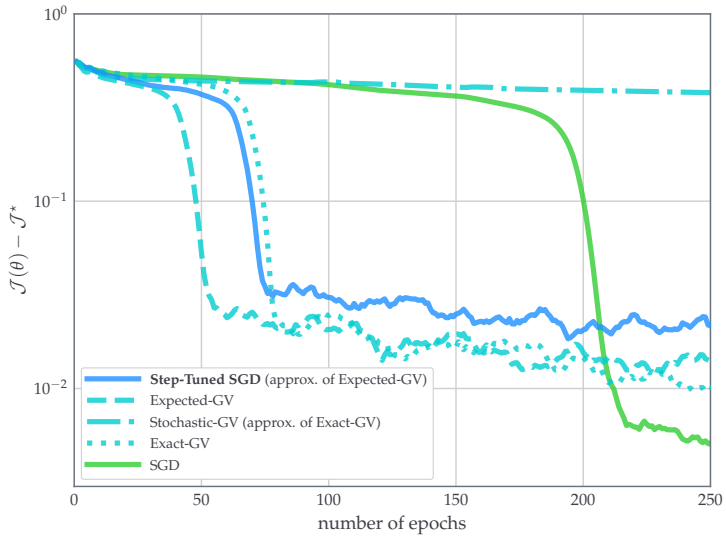


Figure 4.3: Values of the loss function against epochs for non-convex regression: heuristic methods (dashed lines) of Section 4.3.3 are compared with Step-Tuned SGD (plain blue). SGD serves as a reference to evidence the fast drop down effect of other methods. The additional computational cost of Expected-GV and Exact-GV is ignored as these methods are here only for illustration purposes (see Section 4.3.3).

$\nabla\mathcal{J}(\theta_i) - \nabla\mathcal{J}(\theta_{i-1})$, for $1 \leq i \leq k$. Recall that for all $i \geq 1$, we denote $\Delta\theta_i = \theta_i - \theta_{i-1}$ the difference between two consecutive iterates. In the deterministic setting, Algorithm 2 is based on these GVs, indeed,

$$\theta_{k+1} = \theta_k - \frac{\alpha\|\Delta\theta_k\|^2}{\langle\nabla\mathcal{J}(\theta_k) - \nabla\mathcal{J}(\theta_{k-1}), \Delta\theta_k\rangle}\nabla\mathcal{J}(\theta_k), \tag{4.16}$$

whenever the denominator is positive. Given a sequence of independent random mini-batches $(\mathsf{B}_k)_{k\in\mathbb{N}}$ chosen according to the previous discussions, a heuristic mini-batch version of this recursion could be as follows,

$$\theta_{k+1} = \theta_k - \frac{\alpha_k\|\Delta\theta_k\|^2}{\langle\nabla\mathcal{J}(\theta_k) - \nabla\mathcal{J}(\theta_{k-1}), \Delta\theta_k\rangle}\nabla\mathcal{J}_{\mathsf{B}_k}(\theta_k), \tag{Exact-GV}$$

where the difference between (4.16) and Exact-GV lies in the mini-batch estimation of the update direction and the dependency of the scaling factor α_k which aims to moderate the effect of noise (generally $\alpha_k \to 0$ according to the discussion about vanishing step-sizes in Section 1.3.4). As shown in Figure 4.3 the recursion Exact-GV is much faster than SGD

especially for the first ~ 150 epochs which is often the main concern for DL applications. Indeed, although SGD achieves a smaller value of $\mathcal{J}$ after a larger number of iterations (due to other methods using larger step-sizes), this happens when the value of $\mathcal{J}$ is already low. Overall the quantity Exact-GV seems very promising.

Yet, for large sums like in DL, the gradient-variation in Exact-GV is too computationally expensive. One should therefore adapt (Exact-GV). A direct adaption would simply consist in the algorithm,

$$\theta_{k+1} = \theta_k - \frac{\alpha_k \|\Delta\theta_k\|^2}{\left\langle \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) - \nabla \mathcal{J}_{\mathsf{B}_{k-1}}(\theta_{k-1}), \Delta\theta_k \right\rangle} \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k), \qquad \text{(Stochastic-GV)}$$

where mini-batches are used both to obtain a update direction and to approximate the GV. Stochastic-GV yields almost no additional computational cost compared to vanilla SGD since $\nabla \mathcal{J}_{\mathsf{B}_{k-1}}(\theta_{k-1})$ is the previous update direction and is thus already computed at iteration k. For this "naive" approach, we observe a dramatic loss of performance, as illustrated in Figure 4.3. This reveals the need for using accurate stochastic approximations of GVs.

Second heuristic experiment using expected gradient variations. Towards a more stable approximation of the GVs, we consider the following recursion,

$$\theta_{k+1} = \theta_k - \frac{\alpha_k \|\Delta\theta_k\| \|\nabla \mathcal{J}_{\mathsf{B}_{k-1}}(\theta_{k-1})\|}{\langle -\mathbb{E}[\mathcal{C}_{\mathcal{J}_{\mathsf{A}}}(\theta_{k-1})], \Delta\theta_k \rangle} \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k), \qquad \text{(Expected-GV)}$$

where $\mathcal{C}_{\mathcal{J}_{\mathsf{A}}}$ is defined in (4.12) and the expectation is taken over the independent draw of the random subset A, conditioned on the other random variables. The main difference with Exact-GV is the use of expected GVs instead of exact GVs, the minus sign ensures a coherent interpretation in terms of GVs. The numerator in Expected-GV is also modified to ensure homogeneity of the steps with the other variations of the algorithm. Indeed, $\mathcal{C}_{\mathcal{J}_{\mathsf{A}}}(\theta_k)$ approximates a difference of gradients modulo a step-size, see (4.12). As illustrated in Figure 4.3, the recursion Expected-GV provides performances comparable (and actually superior) to Exact-GV, and in particular for both algorithms, we also recover the loss drop which was observed in the deterministic setting (Figure 4.2 and Figure 4.3).

Then, remark that Algorithm 3 is nothing less than an approximate version of Expected-GV which combines a double use of mini-batches with a moving average. Indeed, from (4.12), considering the expectation over the random draw of A, for any $\theta \in \mathbb{R}^P$ and small

$\gamma > 0$, we have,

$$-\gamma \mathbb{E}[\mathcal{C}_{\mathcal{J}_{\mathsf{A}}}(\theta)] \simeq \mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{A}}\left(\theta - \gamma \nabla \mathcal{J}_{\mathsf{A}}(\theta)\right) - \nabla \mathcal{J}_{\mathsf{A}}(\theta)\right]. \tag{4.17}$$

The purpose of the term $\hat{G}_k$ in Algorithm 3 is precisely to mimic this last quantity, i.e., to approximate the expectation in (4.17) with a moving average. The experimental results of Algorithm 3 are very similar to those of Expected-GV, see Figure 4.3.

The above considerations on gradient variations (GVs) led us to propose Algorithm 3 as a possible mini-batch version of Algorithm 2. The underlying geometric aspects discussed in Section 4.3.2 were of course a major motivation as well.

Parameters of the algorithm. Algorithm 3 contains more hyper-parameters than in the deterministic case, yet, we recommend keeping the default values for most of them.[2] Like in most optimizers (SGD, ADAM, RMSprop, etc.), only the parameter $\alpha > 0$ has to be carefully tuned to get the most of Algorithm 3. Note that we enforce $\gamma_k \in [\tilde{m}, \tilde{M}]$ for all $k \in \mathbb{N}$. The bounds stabilize the algorithm and also play an important role for the convergence as we will show in Section 4.4. Note that we also enforce the step-size to decrease using a decay of the form $1/k^{1/2+\delta}$ where $0 < \delta < 1/2$ is usually very close to 0, this way the Robbins-Monro condition (1.22) holds. This is again necessary to obtain the convergence results presented next.

4.4 Theoretical results

We study the convergence of Step-Tuned SGD for general smooth non-convex stochastic optimization which encompasses in particular smooth DL problems.

Main result. We recall that $\mathcal{J}$ is a finite sum of twice continuously differentiable functions $(\mathcal{J}_n)_{n=1,\dots,N}$. Hence, the gradient of $\mathcal{J}$ and the gradients of each $\mathcal{J}_n$, $n \in \{1, \dots, N\}$ are locally Lipschitz continuous (see Definition 1.4). We denote by $\frac{1}{2}\mathbb{N} = \{0, \frac{1}{2}, 1, \frac{3}{2}, 2, \dots\}$ the set of half integers so that the iterations of Step-Tuned SGD are indexed by $k \in \frac{1}{2}\mathbb{N}$. We assume that Assumption 1.1 holds for $\mathcal{J}$, the main theoretical result for Step-Tuned SGD follows.

Theorem 4.2. *Let $\theta_0 \in \mathbb{R}^P$, and let $(\theta_k)_{k \in \frac{1}{2}\mathbb{N}}$ be a sequence generated by Step-Tuned SGD*

[2] We suggest the following default values: $(\nu, \beta, \tilde{m}, \tilde{M}, \delta) = (2, 0.9, 0.5, 2, 0.001)$.

initialized at θ_0. *Assume that there exists* $C_1 > 0$ *such that almost surely* $\sup_{k\in\frac{1}{2}\mathbb{N}} \|\theta_k\| < C_1$. *Then the sequence of values* $(\mathcal{J}(\theta_k))_{k\in\mathbb{N}}$ *converges almost surely and* $(\|\nabla\mathcal{J}(\theta_k)\|^2)_{k\in\mathbb{N}}$ *converges to* 0 *almost surely. In addition,*

$$\min_{j\in\{0,\dots,k-1\}} \mathbb{E}\left[\|\nabla\mathcal{J}(\theta_j)\|^2\right] = O\left(\frac{1}{k^{1/2-\delta}}\right).$$

The results above state in particular that a realization of the algorithm reaches a point where the gradient is arbitrarily small with probability one. Note that the rate depends on the parameter $\delta \in (0, 1/2)$ which can be chosen by the user and corresponds to the decay schedule $1/(k+1)^{1/2+\delta}$. In most cases, one will want to slowly decay the step-size so $\delta \simeq 0$ and the rate is close to $1/\sqrt{k+1}$.

An alternative to the boundedness assumption. In Theorem 4.2 we make the assumption that almost surely the iterates $(\theta_k)_{k\in\frac{1}{2}\mathbb{N}}$ are uniformly bounded, like we did for INNA on Chapter 2. One can alternatively leverage additional regularity assumptions on the loss function as Li and Orabona (2019) did for example for the scalar variant of ADAGRAD. This is more restrictive than the locally-Lipschitz-continuous property of the gradient that we used but for completeness we provide below an alternative version of Theorem 4.2 under such assumptions.

Corollary 4.3. *Let* $\theta_0 \in \mathbb{R}^P$, *and let* $(\theta_k)_{k\in\frac{1}{2}\mathbb{N}}$ *be a sequence generated by Step-Tuned SGD initialized at* θ_0. *Assume that each* $\mathcal{J}_n$ *and* $\nabla\mathcal{J}_n$ *are Lipschitz continuous on* $\mathbb{R}^P$ *for all* $n \in \{1, \dots, N\}$, *and assume that* $\mathcal{J}$ *is bounded below. Then the same conclusions as in Theorem 4.2 apply.*

Proof sketch of Theorem 4.2. The proof of our main theorem relies on more classical arguments and computations than for INNA. Thus, we first present the key elements here and postpone the fully-detailed proof to Section 4.8 of the appendix of this chapter. Here are the main arguments.

– The proof relies on a descent lemma similar to (1.25): for any compact subset $\mathsf{K} \subset \mathbb{R}^P$ there exists $L_\mathsf{K} > 0$ such that for any $\theta \in \mathsf{K}$ and $d \in \mathbb{R}^P$ such that $\theta + d \in \mathsf{K}$,

$$\mathcal{J}(\theta + d) \leq \mathcal{J}(\theta) + \langle\nabla\mathcal{J}(\theta), d\rangle + \frac{L_\mathsf{K}}{2}\|d\|^2. \tag{4.18}$$

– Let $(\theta_k)_{k\in\frac{1}{2}\mathbb{N}}$ be a realization of the algorithm. Using the boundedness assumption, almost surely the iterates belong to a compact subset K on which $\nabla\mathcal{J}$ and the gradient

estimates $(\nabla \mathcal{J}_{\mathsf{B}_k})_{k\in\mathbb{N}}$ are uniformly bounded. So at any iteration $k \in \mathbb{N}$, we may use the descent lemma (4.18) on the update direction $d = -\gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)$ to bound the difference $\mathcal{J}(\theta_{k+1}) - \mathcal{J}(\theta_k)$.

– As stated in Section 4.3.2, conditioning on $\mathsf{B}_0, \ldots, \mathsf{B}_{k-1}$ the step-size γ_k is constructed to be independent of the current mini-batch B_k. Using this and the descent lemma, we show that there exist $M_1, M_2 > 0$ such that, for all $k \in \mathbb{N}_{>0}$,

$$\mathbb{E}\left[\mathcal{J}(\theta_{k+1}) \mid \mathsf{B}_0, \ldots \mathsf{B}_{k-1}\right] \leq \mathcal{J}(\theta_k) - \frac{M_1}{(k+1)^{1/2+\delta}} \|\nabla \mathcal{J}(\theta_k)\|^2 + \frac{M_2}{(k+1)^{1+2\delta}}. \quad (4.19)$$

– Applying Robbins-Siegmund convergence theorem (Robbins and Siegmund, 1971) for martingales to (4.19), using the fact that $\sum_{k=0}^{+\infty} \frac{1}{(k+1)^{1+2\delta}} < \infty$, we obtain almost surely that $(\mathcal{J}(\theta_k))_{k\in\mathbb{N}}$ converges and

$$\sum_{k=0}^{+\infty} \frac{1}{(k+1)^{1/2+\delta}} \|\nabla \mathcal{J}(\theta_k)\|^2 < +\infty, \quad (4.20)$$

Since $\sum_{k=0}^{+\infty} \frac{1}{(k+1)^{1/2+\delta}} = +\infty$, we deduce that $\nabla \mathcal{J}(\theta_k)$ converges to zero almost surely, using the local Lipschitz continuity of the gradient (from twice differentiability) and an argument of Alber et al. (1998), see Lemma 4.6. The rate follows from considering expectations on both sides of (4.19).

4.5 Application to deep learning

We finally evaluate the performance of Step-Tuned SGD for training DNNs. We consider six different problems presented next and fully-specified in Table 4.1. The results for Problems (a) to (d) are first presented here while the results for Problems (e) and (f) are discussed at the end of this section. We compare Step-Tuned SGD with SGD, RMSprop, ADAM and our previously-introduced method INNA. The methodology is detailed below.

4.5.1 Settings of the experiments

We specify the two types of DL experiments that we will perform.

Details on the comparative experiments.

- We consider image classification problems on CIFAR-10 and CIFAR-100 (Krizhevsky, 2009) and the training of an auto-encoder on MNIST (LeCun et al., 2010).
- The networks are slightly modified versions of Lenet (LeCun et al., 1998), ResNet-20 (He et al., 2016), Network-in-Network (M. Lin et al., 2014) and the auto-encoder of G. E. Hinton and Salakhutdinov (2006).
- As specified in Table 4.1 of the Supplementary, we used either smooth (ELU, SiLU) or nonsmooth (ReLU) activation functions (although our theoretical analysis only applies to smooth activation functions).
- For image classification tasks, the dissimilarity measure is the cross-entropy, and for the auto-encoder, it is the mean-squared error. In each problem we also add a ℓ^2-regularization parameter (a.k.a. weight decay) of the form $\frac{\lambda}{2}\|\theta\|_2^2$.
- For each algorithm, we selected the learning rate parameter α (or γ_0 for INNA and SGD) from the set $\{10^{-4}, \dots, 10^{0}\}$. The value is selected as the one yielding the minimum training loss after 10% of the total number of epochs. For example, if we intend to train the network during 100 epochs, the grid-search is carried on the first 10 epochs. For Step-Tuned SGD, the parameter ν was selected with the same criterion from the set $\{1, 2, 5\}$ and for ADAM the momentum parameter was chosen in $\{0.1, 0.5, 0.9, 0.99\}$. All other parameters of the algorithms are left to their default values even INNA for which we keep the default value $(\alpha, \beta) = (0.5, 0.1)$.
- Decay-schedule: To meet the conditions of Theorem 4.2 the step-size decay schedule of SGD and Step-Tuned SGD takes the form $1/q^{1/2+\delta}$ where q is the current epoch index and $\delta = 0.001$. It slightly differs from what is given in Algorithm 3 as we apply the decay at each epoch instead of each iteration. This slower schedule still satisfies the convergence conditions of Theorem 4.2.[3] To ease the comparison we used the same step-size decay schedule for INNA (although slower decays are possible as explained in Section 2.4.2). RMSprop and ADAM rely on their own adaptive procedure and are usually used without a step-size decay schedule.
- The experiments were run on a Nvidia GTX 1080 TI GPU, with an Intel Xeon 2640 V4 CPU. The code was written in `python` 3.6.9 and `pytorch` 1.4 (Paszke et al., 2019).

Remark 4.4. *The settings for these experiments slightly differs from those of Chapter 2. We test the algorithms on more problems, use additional types of DNNs, etc. This is mainly*

[3] An alternative common practice consists in manually decaying the step-size at pre-defined epochs. This technique, although efficient in practice to achieve state-of-the-art results, makes the comparison of algorithms harder, hence we stick to a usual Robbins-Monro type of decay.

because two years passed between the experiments of each chapter. During this period, we gained access to additional computational resources allowing us to consider more DL problems, with in particular the classification of CIFAR-10 and CIFAR-100 with ResNets which became the most popular benchmark problem. We added a weight-decay regularization as it is now a standard practice. We also switched DL libraries, from `keras` *and* `tensorflow` *to* `pytorch`*.*

Second experiment: mini-batch sub-sampling. Step-Tuned SGD departs from the usual process of using a new mini-batch after each gradient update. Indeed, we use each mini-batch twice in order to properly approximate curvature information, but also to maintain a computing time similar to standard algorithms. We performed additional experiments to understand the consequences of using the same mini-batch twice, and in particular make sure that this is not the source of the observed advantage of Step-Tuned SGD. In these experiments all the methods are used with the mini-batch sub-sampling procedure of Step-Tuned SGD detailed in Algorithm 3 (each mini-batch being used to perform two consecutive gradient steps). The other settings remain the same as for the comparative experiments.

4.5.2 Results

We describe the results for the two types of experiments, the comparative one to assess the quality of Step-Tuned SGD against concurrent optimization algorithms, and the other one to study the effect of changing the way mini-batches are used.

Comparison with standard methods. The results for Problems (a) to (d) are displayed on Figure 4.4. For each problem we display the evolution of the values of the loss function and of the test accuracy during the training phase. We observe a recurrent behavior: during early training Step-Tuned SGD has performances similar to other methods, then there is a sudden drop of the loss (combined with an improvement in terms of test accuracy which we discuss below). As a result, Step-Tuned SGD achieves the best training performance among all algorithms on Problems (a) and (b) and at least outperforms SGD in five of the six problems considered (result for Problems (e) and (f) are displayed on Figure 4.6). The sudden drop observed is in accordance with our preliminary observations in Figure 4.3. We note that a similar drop and improved results are reported for SGD and ADAM when used with a manually enforced reduction of the learning rate, see e.g., He et al. (2016). Our experiments show however that Step-Tuned SGD behaves similarly but automatically: the

Table 4.1: Settings of the six different deep learning experiments.

	Problem (a)	**Problem (b)**	**Problem (c)**
Type	Classification	Classification	Classification
Dataset	CIFAR-10	CIFAR-100	CIFAR-10
Network	ResNet-20 (Residual)	ResNet-20 (Residual)	Network-in-Network (Nested)
BatchNorm	Yes	Yes	Yes
Batch-size	128	128	128
Activation functions	ReLU	ReLU	ELU
Dissimilarity measure	Cross-entropy	Cross-entropy	Cross-entropy
Regularization	$\lambda = 10^{-4}$	$\lambda = 10^{-4}$	$\lambda = 10^{-4}$
Grid-search	50 epochs	50 epochs	30 epochs
Stop-criterion	500 epochs	500 epochs	300 epochs
	Problem (d)	**Problem (e)**	**Problem (f)**
Type	Auto-encoder	Classification	Classification
Dataset	MNIST	CIFAR-10	CIFAR-10
Network	Auto-Encoder (Dense)	LeNet (Convolutional)	LeNet (Convolutional)
BatchNorm	No	No	Yes
Batch-size	128	128	128
Activation functions	SiLU	ELU	ELU
Dissimilarity measure	Mean square	Cross-entropy	Cross-entropy
Regularization	$\lambda = 10^{-4}$	$\lambda = 10^{-4}$	$\lambda = 10^{-4}$
Grid-search	50 epochs	30 epochs	30 epochs
Stop-criterion	500 epochs	300 epochs	300 epochs

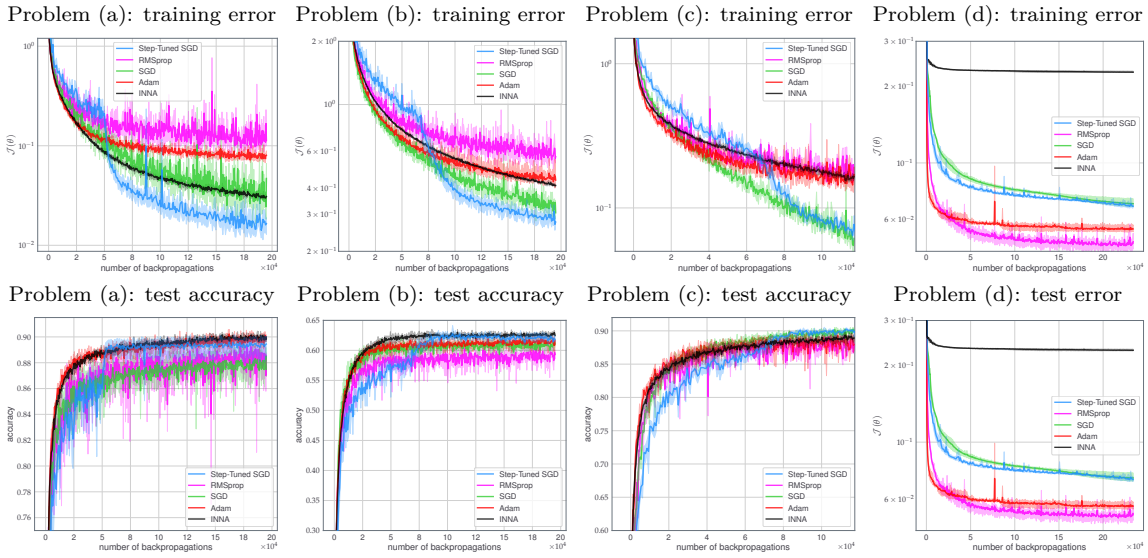


Figure 4.4: Classification of CIFAR-10 and CIFAR-100 with ResNet-20 (left and middle-left respectively), CIFAR-10 with NiN (middle-right) and training of an auto-encoder on MNIST (right). This corresponds to Problems (a) to (d) specified in Table 4.1. Continuous lines: average values from 3 random initializations. Limits of shadow area: best and worst runs (in training loss). For fair comparison values are plotted against the number of gradient estimates computed (using back-propagation).

drop down is caused by the automatic fine-tuning that the algorithm is designed to achieve and not by a user-defined reduction of the step-size.

We remark that in Problem (d), ADAM and RMSprop are notably better than SGD and Step-Tuned SGD. This may be explained by their vector step-sizes since auto-encoders are often ill-conditioned, making methods with scalar step-sizes less efficient. To conclude on these comparative experiments, in most cases Step-Tuned SGD represents a significant improvement compared to SGD. It also seems to be a good alternative to adaptive methods like RMSprop or ADAM especially on residual networks. Note also that while some stochastic second-order methods perform well mostly when combined with large mini-batches, hence with less-noisy gradients (see for example the experiments of Martens and Grosse (2015)), we obtain satisfactory performances with mini-batches of standard sizes.

In addition to efficient training performances (in terms of loss function values), Step-Tuned SGD generalizes well (as measured by test accuracy). For example Figure 4.4 shows a correlation between test accuracy and training loss. Conditions or explanations for when this happens are not fully understood to this day. Yet, SGD is often said to behave well

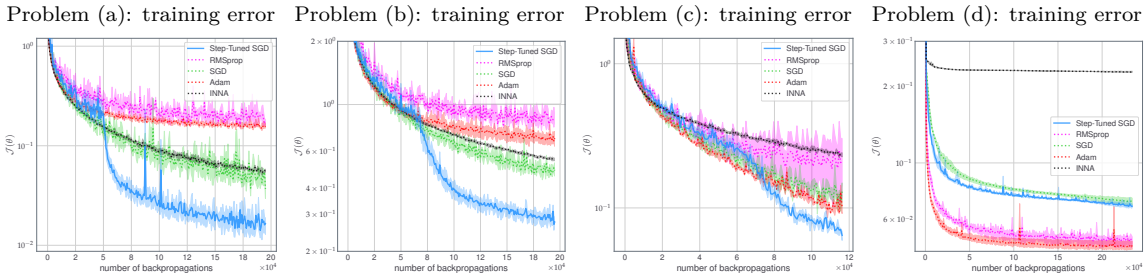


Figure 4.5: Experiment where each algorithms receives the same mini-batch for two consecutive iterations as in Algorithm 3. This allows comparing algorithms with respect to the number of data processed. The problems and the framework are the same as in Figure 4.4.

with respect to this matter (Wilson et al., 2017) and hence it is satisfactory to observe that Step-Tuned SGD seems to inherit this property.

Effect of the mini-batch sub-sampling of Step-Tuned SGD. The results are presented on Figure 4.5. We observe that using each mini-batch twice usually reduces the performance of SGD, INNA, ADAM and RMSprop except on Problem (c) where it benefits the latter two in terms of training error. Thus, on these problems, changing the way of using mini-batches is not the reason for the success of our method. On the contrary, it seems that our goal which was to obtain a fine-tuned step-size specifically for each iteration is clearly achieved, but processing data more slowly, like Step-Tuned SGD does, can sometime impact the performances of the algorithm.

Arguably these results show that the need for using each mini-batch twice appears to be the main downside of Step-Tuned SGD. Thus, in problems where mini-batches may be very different we should expect other methods to be more efficient as they process data twice faster. We actually remark that our method achieves its best results on networks where batch normalization (BatchNorm)—a technique that aims to normalize the inputs of neural networks (Ioffe and Szegedy, 2015)—is used. Figure 4.6 corroborates these observations: BatchNorm has a positive effect on Step-Tuned SGD.

Remark 4.5 (On additional experimental results for INNA)**.** *Let us take the opportunity to use these experiments to make additional comments on the numerical performances of INNA. Here we used* $(\alpha, \beta) = (0.5, 0.1)$ *that we previously proposed as default values. We can see on Figure 4.4 and Figure 4.6 that INNA performs quite well: it is always among the three best algorithms in terms of training performances except for Problem (e). On this*

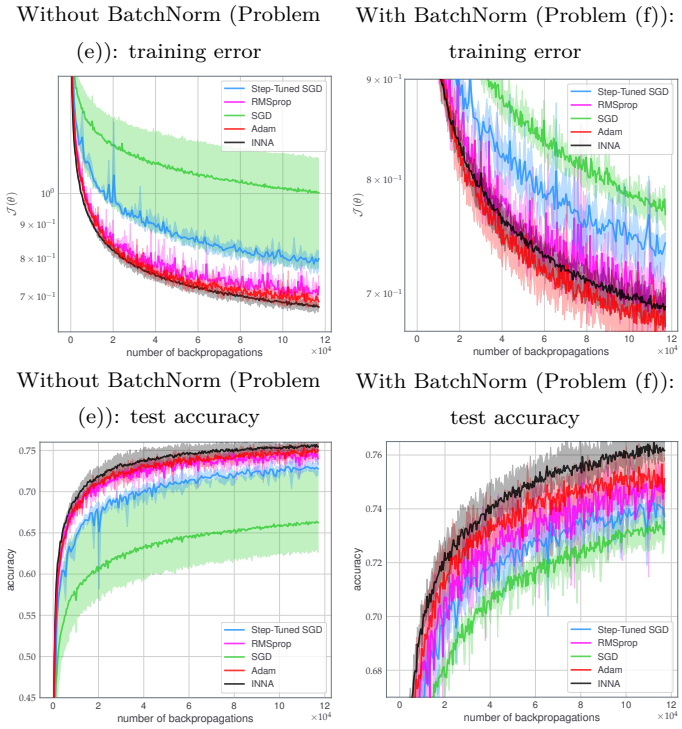


Figure 4.6: Classification of CIFAR-10 with LeNet with and without batch normalization, corresponding to Problems (e) and (f) specified in Table 4.1. These experiments illustrate how batch normalization has a positive effect on Step-Tuned SGD.

problems, which is quite different from the others, it seems that tuning (α, β) *is necessary to obtain more satisfactory results. It is worth noticing that INNA seems to achieve very good generalization scores (on all problems except Problem (d)). In particular, it generalizes better than any other method even when it is not the best algorithm in terms of training (see for example Problems (b) and (f)). These are empirical observations which may have many causes. Yet, this very promising behavior may come from the damping properties illustrated on Figure 2.1 or the ability to escape strict saddle points proved in the previous chapter.*

4.6 Conclusion

We presented a new method to tune SGD's step-sizes for stochastic non-convex optimization within a first-order computational framework. In addition to the new algorithm, we also presented a generic strategy (Section 4.3.1 and 4.3.3) on how to use empirical and geometrical

considerations to address the major difficulty of preserving favorable behaviors of deterministic algorithms while dealing with mini-batches. In particular, we tackled the problem of adapting the step-sizes to the local landscape of non-convex loss functions with noisy gradient estimations. For a computational cost similar to SGD, Step-tuned SGD uses a step-size changing every two iterations unlike other stochastic methods à la Barzilai-Borwein. We proved asymptotic convergence results and convergence rates for our algorithm.

While our method does not alleviate hyper-parameter pre-tuning, it shows how an efficient *automatic* fine-tuning of a *simple* scalar step-size can improve the training of DNNs. Step-Tuned SGD processes data more slowly than other methods but by doing so manages to fine-tune step-sizes, leading to faster training in some DL problems with a typical sudden drop of the error rate at medium stages, especially on ResNets.

Appendices to Chapter 4

4.7 Details on the synthetic experiments

We detail the non-convex regression problem that we presented in Figure 4.2 and 4.3. Given a matrix $A \in \mathbb{R}^{N \times P}$ and a vector $b \in \mathbb{R}^N$, denote A_n the n-th row of A. The problem consists in minimizing a loss function of the form,

$$\theta \in \mathbb{R}^P \mapsto \mathcal{J}(\theta) = \frac{1}{N} \sum_n^N \phi(A_n^T \theta - b_n), \tag{4.21}$$

where the non-convexity comes from the function $t \in \mathbb{R} \mapsto \phi(t) = t^2/(1+t^2)$. For more details on the initialization of A and b we refer to Carmon et al. (2017) where this problem is initially proposed. In the experiments of Figure 4.3, the mini-batch approximation was made by selecting a subset of the lines of A, which amounts to computing only a few terms of the full sum in (4.21). We used $N = 500$, $P = 30$ and mini-batches of size 50.

In the deterministic setting we ran each algorithm during 250 iterations and selected the hyper-parameters of each algorithm such that they achieve $|\mathcal{J}(\theta) - \mathcal{J}^\star| < 10^{-1}$ as fast as possible. In the mini-batch experiments we ran each algorithm during 250 epochs and selected the hyper-parameters that yielded the smallest value of $\mathcal{J}(\theta)$ after 50 epochs.

4.8 Proof of the theoretical results

We state a lemma that we will use to prove Theorem 4.2.

4.8.1 Preliminary lemma

The result is the following.

Lemma 4.6 (Alber et al. (1998, Proposition 2))**.** *Let $(u_k)_{k\in\mathbb{N}}$ and $(v_k)_{k\in\mathbb{N}}$ two non-negative real sequences. Assume that $\sum_{k=0}^{+\infty} u_k v_k < +\infty$, and $\sum_{k=0}^{+\infty} v_k = +\infty$. If there exists a*

constant $C > 0$ *such that* $\forall k \in \mathbb{N}, |u_{k+1} - u_k| \leq Cv_k$, *then* $u_k \xrightarrow[k \to +\infty]{} 0$.

4.8.2 Proof of the main theorem

We can now prove Theorem 4.2.

Proof of Theorem 4.2. We first clarify the random process induced by the mini-batch subsampling. Algorithm 3 takes a sequence of mini-batches as input. This sequence is represented by the random variables $(\mathsf{B}_k)_{k \in \mathbb{N}}$ as described in Section 4.3.2. Each of these random variables is independent of the others. In particular, for $k \in \mathbb{N}_{>0}$, B_k is independent of the previous mini-batches $\mathsf{B}_0, \ldots, \mathsf{B}_{k-1}$. For convenience, we will denote $\underline{\mathsf{B}}_k = \{\mathsf{B}_0, \ldots, \mathsf{B}_k\}$, the mini-batches up to iteration k. Due to the randomness of the mini-batches, the algorithm is a random process as well. As such, the current iterate θ_k is a random variable with a deterministic dependence on $\underline{\mathsf{B}}_{k-1}$ and is independent of B_k. However, $\theta_{k+\frac{1}{2}}$ and B_k are not independent. Similarly, we constructed γ_k such that it is a random variable with a deterministic dependence on $\underline{\mathsf{B}}_{k-1}$, which is independent of B_k. This dependency structure will be crucial to derive and bound conditional expectations. Finally, we highlight the following important identity, for any $k \geq 1$,

$$\mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) | \underline{\mathsf{B}}_{k-1}\right] = \nabla \mathcal{J}(\theta_k). \tag{4.22}$$

Indeed, the iterate θ_k is a deterministic function of $\underline{\mathsf{B}}_{k-1}$, so taking the expectation over B_k, which is independent of $\underline{\mathsf{B}}_{k-1}$, we recover the full gradient of $\mathcal{J}$ since the distribution of B_k is the same as that of A in Section 4.3.2. Notice in addition that a similar identity does not hold for $\theta_{k+\frac{1}{2}}$ (as it depends on B_k).

We now provide estimates that will be used extensively in the rest of the proof. Recall that the gradient of the loss function $\nabla \mathcal{J}$ is locally Lipschitz continuous since $\mathcal{J}$ is twice continuously differentiable. By assumption, there exists a compact convex set $\mathsf{K} \subset \mathbb{R}^P$, such that with probability 1, the sequence of iterates $(\theta_k)_{k \in \frac{1}{2}\mathbb{N}}$ belongs to K. Therefore, by local Lipschitz continuity, the restriction of $\nabla \mathcal{J}$ to K is Lipschitz continuous on K. Similarly, each $\nabla \mathcal{J}_n$ is also Lipschitz continuous on K. We denote by $L_{\mathsf{K}} > 0$ a Lipschitz constant common to each $\nabla \mathcal{J}_n$, for $n = 1, \ldots, N$. Notice that the Lipschitz continuity is preserved by averaging, in other words,

$$\forall\, \mathsf{B} \subseteq \{1, \ldots, N\}, \forall \psi_1, \psi_2 \in \mathsf{K}, \quad \|\nabla \mathcal{J}_{\mathsf{B}}(\psi_1) - \nabla \mathcal{J}_{\mathsf{B}}(\psi_2)\| \leq L_{\mathsf{K}} \|\psi_1 - \psi_2\|. \tag{4.23}$$

In addition, using the continuity of the $\nabla \mathcal{J}_n$'s, there exists a constant $C_2 > 0$, such that,

$$\forall \mathsf{B} \subseteq \{1, \dots, N\}, \forall \psi \in \mathsf{K}, \quad \|\nabla \mathcal{J}_{\mathsf{B}}(\psi)\| \leq C_2. \tag{4.24}$$

Finally, for a differentiable function $g : \mathbb{R}^P \to \mathbb{R}$ with $L_{\nabla g}$-Lipschitz continuous gradient, we recall the descent lemma (see for example Bertsekas (1999, Proposition A.24) or Equation 1.25). For any $\theta \in \mathbb{R}^P$ and any $d \in \mathbb{R}^P$,

$$g(\theta + d) \leq g(\theta) + \langle \nabla g(\theta), d \rangle + \frac{L_{\nabla g}}{2} \|d\|^2. \tag{4.25}$$

In our case since we only have the L_{K}-Lipschitz continuity of $\nabla \mathcal{J}$ on K which is convex, we have a similar bound for $\nabla \mathcal{J}$ on K: for any $\theta \in \mathsf{K}$ and any $d \in \mathbb{R}^P$ such that $\theta + d \in \mathsf{K}$,

$$\mathcal{J}(\theta + d) \leq \mathcal{J}(\theta) + \langle \nabla \mathcal{J}(\theta), d \rangle + \frac{L_{\mathsf{K}}}{2} \|d\|^2. \tag{4.26}$$

Let $\theta_0 \in \mathbb{R}^P$ and let $(\theta_k)_{k \in \frac{1}{2}\mathbb{N}}$ a sequence generated by Algorithm 3 initialized at θ_0. By assumption this sequence belongs to K almost surely. To simplify, for all $k \in \mathbb{N}$ we denote $\eta_k = \alpha \gamma_k (k+1)^{-(1/2+\delta)}$ the step-size. Fix an iteration $k \in \mathbb{N}$, we can use (4.26) with $\theta = \theta_k$ and $d = -\eta_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)$, almost surely (with respect to the boundedness assumption),

$$\mathcal{J}(\theta_{k+\frac{1}{2}}) \leq \mathcal{J}(\theta_k) - \eta_k \langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \rangle + \frac{\eta_k^2}{2} L_{\mathsf{K}} \|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\|^2. \tag{4.27}$$

Similarly with $\theta = \theta_{k+\frac{1}{2}}$ and $d = -\eta_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})$, almost surely,

$$\mathcal{J}(\theta_{k+1}) \leq \mathcal{J}(\theta_{k+\frac{1}{2}}) - \eta_k \langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}}) \rangle + \frac{\eta_k^2}{2} L_{\mathsf{K}} \|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\|^2. \tag{4.28}$$

We combine (4.27) and (4.28), almost surely,

$$\begin{aligned} \mathcal{J}(\theta_{k+1}) \leq \mathcal{J}(\theta_k) &- \eta_k \left(\langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) \rangle + \langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}}) \rangle \right) \\ &+ \frac{\eta_k^2}{2} L_{\mathsf{K}} \left(\|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\|^2 + \|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\|^2 \right). \end{aligned} \tag{4.29}$$

Using the boundedness assumption and (4.24), almost surely,

$$\|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\|^2 \leq C_2 \quad \text{and} \quad \|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\|^2 \leq C_2. \tag{4.30}$$

So almost surely,

$$\begin{aligned}\mathcal{J}(\theta_{k+1}) \leq \mathcal{J}(\theta_k) - \eta_k \left(\langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle + \langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\rangle\right) \\ + \eta_k^2 L_{\mathsf{K}} C_2.\end{aligned} \tag{4.31}$$

Then, we take the conditional expectation of (4.31) over B_k conditionally on $\underline{\mathsf{B}}_{k-1}$ (the mini-batches used up to iteration $k-1$), we have,

$$\begin{aligned}\mathbb{E}\left[\mathcal{J}(\theta_{k+1})|\underline{\mathsf{B}}_{k-1}\right] \leq \mathbb{E}\left[\mathcal{J}(\theta_k)|\underline{\mathsf{B}}_{k-1}\right] + \mathbb{E}\left[\eta_k^2 L_{\mathsf{K}} C_2 \middle| \underline{\mathsf{B}}_{k-1}\right] \\ - \mathbb{E}\left[\eta_k \left(\langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle + \langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\rangle\right) \middle| \underline{\mathsf{B}}_{k-1}\right].\end{aligned} \tag{4.32}$$

As explained at the beginning of the proof, θ_k is a deterministic function of $\underline{\mathsf{B}}_{k-1}$, thus using (4.22), $\mathbb{E}\left[\mathcal{J}(\theta_k)|\underline{\mathsf{B}}_{k-1}\right] = \mathcal{J}(\theta_k)$. Similarly, by construction η_k is independent of the current mini-batch B_k, it is a deterministic function of $\underline{\mathsf{B}}_{k-1}$. Hence, (4.32) reads,

$$\begin{aligned}\mathbb{E}\left[\mathcal{J}(\theta_{k+1})|\underline{\mathsf{B}}_{k-1}\right] \leq & \mathcal{J}(\theta_k) + \eta_k^2 L_{\mathsf{K}} C_2 - \eta_k \langle \nabla \mathcal{J}(\theta_k), \mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)|\underline{\mathsf{B}}_{k-1}\right]\rangle \\ & - \eta_k \mathbb{E}\left[\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\rangle \middle| \underline{\mathsf{B}}_{k-1}\right].\end{aligned} \tag{4.33}$$

Then, we use the fact that $\mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)|\underline{\mathsf{B}}_{k-1}\right] = \nabla \mathcal{J}(\theta_k)$. Overall, we obtain,

$$\begin{aligned}\mathbb{E}\left[\mathcal{J}(\theta_{k+1})|\underline{\mathsf{B}}_{k-1}\right] \leq & \mathcal{J}(\theta_k) + \eta_k^2 L_{\mathsf{K}} C_2 - \eta_k \|\nabla \mathcal{J}(\theta_k)\|^2 \\ & - \eta_k \mathbb{E}\left[\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\rangle \middle| \underline{\mathsf{B}}_{k-1}\right].\end{aligned} \tag{4.34}$$

We will now bound the last term of (4.34). First we write,

$$\begin{aligned}& - \langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\rangle \\ & = -\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}}) - \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle - \langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle.\end{aligned} \tag{4.35}$$

Using the Cauchy-Schwarz inequality, as well as (4.23) and (4.24), almost surely,

$$\begin{aligned}|\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}}) - \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle| &\leq \|\nabla \mathcal{J}(\theta_{k+\frac{1}{2}})\| \|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}}) - \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\| \\ &\leq \|\nabla \mathcal{J}(\theta_{k+\frac{1}{2}})\| L_{\mathsf{K}} \|\theta_{k+\frac{1}{2}} - \theta_k\| \\ &\leq \|\nabla \mathcal{J}(\theta_{k+\frac{1}{2}})\| L_{\mathsf{K}} \| - \eta_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\| \\ &\leq L_{\mathsf{K}} C_2^2 \eta_k.\end{aligned} \tag{4.36}$$

Hence,

$$-\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\rangle \leq L_{\mathsf{K}} C_2^2 \eta_k - \langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle. \tag{4.37}$$

We perform similar computations on the last term of (4.37), almost surely

$$\begin{aligned}
&-\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle \\
&= -\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}) - \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle - \langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle \\
&\leq \|\nabla \mathcal{J}(\theta_{k+\frac{1}{2}}) - \nabla \mathcal{J}(\theta_k)\| \|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\| - \langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle \\
&\leq L_{\mathsf{K}} C_2 \|\theta_{k+\frac{1}{2}} - \theta_k\| - \langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle \\
&\leq L_{\mathsf{K}} C_2^2 \eta_k - \langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle.
\end{aligned} \tag{4.38}$$

Finally by combining (4.35), (4.37) and (4.38), we obtain almost surely,

$$-\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\rangle \leq 2 L_{\mathsf{K}} C_2^2 \eta_k - \langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle. \tag{4.39}$$

Going back to the last term of (4.34), taking the conditional expectation of (4.39), we have almost surely,

$$\begin{aligned}
-\eta_k \mathbb{E}\left[\langle \nabla \mathcal{J}(\theta_{k+\frac{1}{2}}), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\rangle \middle| \underline{\mathsf{B}}_{k-1}\right] &\leq 2 L_{\mathsf{K}} C_2^2 \eta_k^2 - \eta_k \mathbb{E}\left[\langle \nabla \mathcal{J}(\theta_k), \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)\rangle | \underline{\mathsf{B}}_{k-1}\right] \\
\leq 2 L_{\mathsf{K}} C_2^2 \eta_k^2 - \eta_k \langle \nabla \mathcal{J}(\theta_k), \mathbb{E}\left[\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) | \underline{\mathsf{B}}_{k-1}\right]\rangle &= 2 L_{\mathsf{K}} C_2^2 \eta_k^2 - \eta_k \|\nabla \mathcal{J}(\theta_k)\|^2.
\end{aligned} \tag{4.40}$$

In the end we obtain, for an arbitrary iteration $k \in \mathbb{N}$, almost surely,

$$\mathbb{E}\left[\mathcal{J}(\theta_{k+1}) | \underline{\mathsf{B}}_{k-1}\right] \leq \mathcal{J}(\theta_k) - 2\eta_k \|\nabla \mathcal{J}(\theta_k)\|^2 + \eta_k^2 L_{\mathsf{K}} (C_2 + 2C_2^2). \tag{4.41}$$

To simplify we assume that $\tilde{M} > \nu$ (otherwise set $\tilde{M} = \max(\tilde{M}, \nu)$). We use the fact that, $\eta_k \in [\frac{\alpha \tilde{m}}{(k+1)^{1/2+\delta}}, \frac{\alpha \tilde{M}}{(k+1)^{1/2+\delta}}]$, to obtain almost surely,

$$\mathbb{E}\left[\mathcal{J}(\theta_{k+1}) | \underline{\mathsf{B}}_{k-1}\right] \leq \mathcal{J}(\theta_k) - 2\frac{\alpha \tilde{m}}{(k+1)^{1/2+\delta}} \|\nabla \mathcal{J}(\theta_k)\|^2 + \frac{\alpha^2 \tilde{M}^2}{(k+1)^{1+2\delta}} L_{\mathsf{K}} (C_2 + 2C_2^2). \tag{4.42}$$

Since by assumption, the last term is summable, we can now use the Robbins-Siegmund convergence theorem (Robbins and Siegmund, 1971) to obtain that, almost surely, $(\mathcal{J}(\theta_k))_{k \in \mathbb{N}}$

converges and,

$$\sum_{k=0}^{+\infty} \frac{1}{(k+1)^{1/2+\delta}} \|\nabla \mathcal{J}(\theta_k)\|^2 < +\infty. \tag{4.43}$$

Since $\sum_{k=0}^{+\infty} \frac{1}{(k+1)^{1/2+\delta}} = +\infty$, this implies at least that almost surely,

$$\liminf_{k\to\infty} \|\nabla \mathcal{J}(\theta_k)\|^2 = 0. \tag{4.44}$$

To prove that in addition $\lim_{k\to\infty} \|\nabla \mathcal{J}(\theta_k)\|^2 = 0$, we will use Lemma 4.6 with $u_k = \|\nabla \mathcal{J}(\theta_k)\|^2$ and $v_k = \frac{1}{(k+1)^{1/2+\delta}}$. So we need to prove that there exists $C_3 > 0$ such that $|u_{k+1} - u_k| \leq C_3 v_k$. To do so, we use the L_{K}-Lipschitz continuity of the gradients on K, triangle inequalities and (4.24). It holds, almost surely, for all $k \in \mathbb{N}$,

$$\begin{aligned}
&\left| \|\nabla \mathcal{J}(\theta_{k+1})\|^2 - \|\nabla \mathcal{J}(\theta_k)\|^2 \right| \\
=& \left(\|\nabla \mathcal{J}(\theta_{k+1})\| + \|\nabla \mathcal{J}(\theta_k)\| \right) \times \left| \|\nabla \mathcal{J}(\theta_{k+1})\| - \|\nabla \mathcal{J}(\theta_k)\| \right| \\
\leq& 2C_2 \left| \|\nabla \mathcal{J}(\theta_{k+1})\| - \|\nabla \mathcal{J}(\theta_k)\| \right| \\
\leq& 2C_2 \|\nabla \mathcal{J}(\theta_{k+1}) - \nabla \mathcal{J}(\theta_k)\| \\
\leq& 2C_2 L_{\mathsf{K}} \|\theta_{k+1} - \theta_k\| \\
\leq& 2C_2 L_{\mathsf{K}} \left\| -\eta_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) - \eta_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}}) \right\| \\
\leq& 2C_2 L_{\mathsf{K}} \frac{\alpha \tilde{M}}{(k+1)^{1/2+\delta}} \|\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) + \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_{k+\frac{1}{2}})\| \\
\leq& 4C_2^2 L_{\mathsf{K}} \frac{\alpha \tilde{M}}{(k+1)^{1/2+\delta}}.
\end{aligned} \tag{4.45}$$

So taking $C_3 = 4C_2^2 L_{\mathsf{K}} \alpha \tilde{M}$, by Lemma 4.6, almost surely, $\lim_{k\to+\infty} \|\nabla \mathcal{J}(\theta_k)\|^2 = 0$. This concludes the part of the proof on almost sure convergence.

Regarding the rate, consider the expectation of (4.42) (with respect to the random variables $(\mathsf{B}_k)_{k\in\mathbb{N}}$). The tower property of the conditional expectation gives

$$\mathbb{E}_{\underline{\mathsf{B}}_{k-1}}[\mathbb{E}[\mathcal{J}(\theta_{k+1})|\underline{\mathsf{B}}_{k-1}]] = \mathbb{E}\left[\mathcal{J}(\theta_{k+1})\right],$$

so we obtain, for all $k \in \mathbb{N}$,

$$2\frac{\alpha\tilde{m}}{(k+1)^{1/2+\delta}} \mathbb{E}\left[\|\nabla \mathcal{J}(\theta_k)\|^2\right] \leq \mathbb{E}\left[\mathcal{J}(\theta_k)\right] - \mathbb{E}\left[\mathcal{J}(\theta_{k+1})\right] + \frac{\alpha^2 \tilde{M}^2}{(k+1)^{1+2\delta}} L_{\mathsf{K}}(C_2 + 2C_2^2). \tag{4.46}$$

Then for $K \geq 1$, we sum from 0 to $K-1$,

$$\begin{aligned}
&\sum_{k=0}^{K-1} 2\frac{\alpha\tilde{m}}{(k+1)^{1/2+\delta}}\mathbb{E}\left[\|\nabla\mathcal{J}(\theta_k)\|^2\right] \\
&\leq \sum_{k=0}^{K-1}\mathbb{E}\left[\mathcal{J}(\theta_k)\right] - \sum_{k=0}^{K-1}\mathbb{E}\left[\mathcal{J}(\theta_{k+1})\right] + \sum_{k=0}^{K-1}\frac{\alpha^2\tilde{M}^2}{(k+1)^{1+2\delta}}L_{\mathsf{K}}(C_2+2C_2^2) \\
&= \mathcal{J}(\theta_0) - \mathbb{E}\left[\mathcal{J}(\theta_K)\right] + \sum_{k=0}^{K-1}\frac{\alpha^2\tilde{M}^2}{(k+1)^{1+2\delta}}L_{\mathsf{K}}(C_2+2C_2^2) \\
&\leq \mathcal{J}(\theta_0) - \inf_{\psi\in\mathbb{R}^P}\mathcal{J}(\psi) + \sum_{k=0}^{K-1}\frac{\alpha^2\tilde{M}^2}{(k+1)^{1+2\delta}}L_{\mathsf{K}}(C_2+2C_2^2),
\end{aligned} \tag{4.47}$$

The right-hand side is finite, so there exists a constant $C_4 > 0$ such that for any $K \in \mathbb{N}$, it holds,

$$\begin{aligned}
C_4 \geq \sum_{k=0}^{K}\frac{1}{(k+1)^{1/2+\delta}}\mathbb{E}\left[\|\nabla\mathcal{J}(\theta_k)\|^2\right] &\geq \min_{k\in\{1,\dots,K\}}\mathbb{E}\left[\|\nabla\mathcal{J}(\theta_k)\|^2\right]\sum_{k=0}^{K}\frac{1}{(k+1)^{1/2+\delta}} \\
&\geq (K+1)^{1/2-\delta}\min_{k\in\{1,\dots,K\}}\mathbb{E}\left[\|\nabla\mathcal{J}(\theta_k)\|^2\right],
\end{aligned} \tag{4.48}$$

and we obtain the rate. □

4.8.3 Proof of the corollary

Before proving the corollary we recall the following result.

Lemma 4.7. *Let $g : \mathbb{R}^P \to \mathbb{R}$ a Lipschitz continuous and differentiable function. Then ∇g is uniformly bounded on $\mathbb{R}^P$.*

We can now prove the corollary.

Proof of Corollary 4.3. The proof is very similar to the one of Theorem 4.2. Using the Lipschitz continuity of $\nabla\mathcal{J}$, the descent lemma (4.27) holds surely on $\mathbb{R}^P$. Furthermore, since for all $n \in \{1,\dots,N\}$, each $\mathcal{J}_n$ is Lipschitz continuous, so is $\mathcal{J}$. Furthermore, from Lemma 4.7, globally Lipschitz continuous functions have uniformly bounded gradients. This is enough to obtain (4.42). It also holds that for all $k \in \mathbb{N}$, $\mathbb{E}\left[\|\nabla\mathcal{J}_{\mathsf{B}_k}(\theta_k)\|\right]$ is uniformly bounded. Overall these arguments allow to follow the lines of the proof of Theorem 4.2 and the same conclusions follow by repeating the same arguments. □

4.9 Description of auxiliary algorithms

We precise the heuristic algorithms used in Figure 4.3 and discussed in Section 4.3.3. Note that the step-size in Algorithm 6 corresponds to Expected-GV but is written differently to avoid storing an additional gradient estimate.

Algorithm 4 Stochastic-GV SGD

Input: $\alpha > 0$, $\nu > 0$
Input: $\tilde{m} > 0$, $\tilde{M} > 0$, $\delta \in (0, 1/2)$
Initialize $\theta_0 \in \mathbb{R}^P$, $\gamma_0 = 1$
Draw independent random mini-batches $(\mathsf{B}_k)_{k\in\mathbb{N}}$
$\theta_1 = \theta_0 - \alpha\gamma_0 \nabla \mathcal{J}_{\mathsf{B}_0}(\theta_0)$
for $k = 1, \dots$ **do**
  $\Delta\theta_k = \theta_k - \theta_{k-1}$
  $\Delta g_k^{\text{naive}} = \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) - \nabla \mathcal{J}_{\mathsf{B}_{k-1}}(\theta_{k-1})$
  if $\langle \Delta g_k^{\text{naive}}, \Delta\theta_{\mathsf{B}_k} \rangle > 0$ **then**
    $\gamma_k = \frac{\|\Delta\theta_k\|^2}{\langle \Delta g_k^{\text{naive}}, \Delta\theta_k \rangle}$
  else
    $\gamma_k = \nu$
  end if
  $\gamma_k = \min(\max(\gamma_k, \tilde{m}), \tilde{M})$
  $\theta_{k+1} = \theta_k - \frac{\alpha}{(k+1)^{1/2+\delta}} \gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)$
end for

Algorithm 5 Exact-GV SGD

Input: $\alpha > 0$, $\nu > 0$
Input: $\tilde{m} > 0$, $\tilde{M} > 0$, $\delta \in (0, 1/2)$
Initialize $\theta_0 \in \mathbb{R}^P$, $\gamma_0 = 1$
Draw independent random mini-batches $(\mathsf{B}_k)_{k\in\mathbb{N}}$
$\theta_1 = \theta_0 - \alpha\gamma_0 \nabla \mathcal{J}_{\mathsf{B}_0}(\theta_0)$
for $k = 1, \ldots$ **do**
  $\Delta\theta_k = \theta_k - \theta_{k-1}$
  $G_k = \nabla \mathcal{J}(\theta_k) - \nabla \mathcal{J}(\theta_{k-1})$
  if $\langle G_k, \Delta\theta_{\mathsf{B}_k} \rangle > 0$ **then**
    $\gamma_k = \frac{\|\Delta\theta_k\|^2}{\langle G_k, \Delta\theta_k \rangle}$
  else
    $\gamma_k = \nu$
  end if
  $\gamma_k = \min(\max(\gamma_k, \tilde{m}), \tilde{M})$
  $\theta_{k+1} = \theta_k - \frac{\alpha}{(k+1)^{1/2+\delta}} \gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)$
end for

Algorithm 6 Expected-GV SGD

Input: $\alpha > 0$, $\nu > 0$
Input: $\tilde{m} > 0$, $\tilde{M} > 0$, $\delta \in (0, 1/2)$
Initialize $\theta_0 \in \mathbb{R}^P$, $\gamma_0 = 1$
Draw independent random mini-batches $(\mathsf{B}_k)_{k\in\mathbb{N}}$
Let A the random subset defined in Section 4.3.2
$\theta_1 = \theta_0 - \alpha\gamma_0 \nabla \mathcal{J}_{\mathsf{B}_0}(\theta_0)$
for $k = 1, \ldots$ **do**
  $\Delta\theta_k = \theta_k - \theta_{k-1}$
  $G_k = -\frac{\alpha}{(k-1)^{1/2+\delta}} \gamma_{k-1} \mathbb{E}\left[\mathcal{C}_{\mathcal{J}_{\mathsf{A}}}(\theta_{k-1})\right]$
  if $\langle G_k, \Delta\theta_{\mathsf{B}_k} \rangle > 0$ **then**
    $\gamma_k = \frac{\|\Delta\theta_k\|^2}{\langle G_k, \Delta\theta_k \rangle}$
  else
    $\gamma_k = \nu$
  end if
  $\gamma_k = \min(\max(\gamma_k, \tilde{m}), \tilde{M})$
  $\theta_{k+1} = \theta_k - \frac{\alpha}{(k+1)^{1/2+\delta}} \gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k)$
end for

Conclusion

Closing discussion

The primary goal of this thesis was to leverage second-order information in order to design efficient algorithms for training DNNs. We had two major practical challenges to address: overcoming the computational cost of second-order information and adapting deterministic methods to mini-batch sub-sampling while preserving their performances. We followed two different approaches in Chapter 2 and Chapter 4, and designed two algorithms.

- We built INNA from a second-order ODE rewritten into an equivalent first-order system where the Hessian is implicit. This solved both problems at the same time: it allowed building a second-order algorithm for no additional computational cost, but it also allowed avoiding the discretization of second-order derivatives of $\mathcal{J}$ which makes the algorithm more robust to mini-batch sub-sampling.
- Step-Tuned SGD takes inspiration from a variational model rather than an ODE. Unlike INNA, a discretization of second-order derivatives was necessary to maintain an affordable computational cost. A straightforward discretization does not mix well with mini-batch sub-sampling. Thus, we rather used empirical and theoretical considerations to find a surrogate variational quantity which takes the form of an expectation. As such, the discretization of this surrogate quantity is more robust to mini-batch sub-sampling which helps to preserve the behavior of deterministic BB methods in a mini-batch setting.

As a result, we can answer positively to the first two questions raised in Section 1.4.2: we provided two new practical methods for training DNNs, both benefiting from second-order information for a computational cost similar to the algorithms classically used to train DNNs. In addition, our algorithms are motivated from a geometrical point of view, while having simple update rules (this is true in particular for INNA). Second-order information does benefit our algorithms in practice despite non-smoothness and mini-batch sub-sampling. This is evidenced by higher training speeds and better generalization perfor-

mances compared to standard methods on several DL benchmark problems.

As for our third question on theoretical guarantees, we analyzed our algorithms in a framework that we tried to keep as close as possible to the reality of DNN training. To achieve this, we combined existing techniques and overcame new challenges.

- We proved the convergence of INNA in a very general non-smooth stochastic framework. To do so, we followed the ODE technique, but we showed that the mini-batch sub-sampling of non-smooth functions brings new difficulties. We thus introduced a new operator D and provided calculus results for D in order to prove convergence. In a smooth deterministic framework we also proved that INNA is likely to avoid spurious critical points.
- Regarding Step-Tuned SGD, we restrained the analysis to smooth loss functions but used no global Lipschitz continuity assumptions. The proofs are based on a classical descent lemma, yet, Step-Tuned SGD is harder to analyze than SGD with this technique. We had to carefully design our algorithm so that convergence may hold and use several tricks to prove the convergence.

Overall, we proved the almost-sure convergence of the algorithms in DL and derived rates of convergence which is difficult in general for non-convex functions. These theoretical guarantees support the fact that our algorithms can be used to train DNNs and help us to get a better understanding of the behavior of these methods.

Perspectives and future work

While we carried out extensive theoretical analyses of our algorithms, we present some possible improvements and ideas to investigate in the future.

- The convergence results of both algorithms rely on the same assumption: the almost-sure boundedness of the iterates. Since this assumption is hard to ensure in practice, we could consider a variant of each algorithm where we project the iterates on an arbitrarily large compact subset of $\mathbb{R}^P$. However, proving the convergence in this case would be non-straightforward, in particular for INNA since the results of Benaïm et al. (2005) do not apply directly. We believe that the ODE technique can be adapted to this setting but this remains to be done. Note that these concerns are mostly theoretical. Indeed, since INNA and Step-Tuned SGD seem to provide satisfactory results in numerical experiments, practitioners are unlikely to add a projection but will rather use them in their present form.

– Similarly, we proved the almost-sure convergence of INNA to D-critical points for non-smooth tame functions. Actually, we believe that INNA is likely to converge to Clarke critical points in practice. The work of Bolte and Pauwels (2020a) shows that SGD almost surely avoids D-critical points, we could prove similar results for INNA. Again, this is mostly a theoretical matter, since in practice we observe that INNA provides "valuable" results for the parameter θ.

Finally, although INNA and Step-Tuned SGD benefit from promising numerical and theoretical results, they are far from being as used as ADAM and SGD to this day. This is not surprising since our algorithms are recent, yet, one may wonder whether this will change in the near future. Since they are famous and provided by default in DL libraries, ADAM and SGD will probably remain the first methods that practitioners try when training a DNN. However, Step-Tuned SGD may be helpful in situations where choosing manually an efficient step-size decay is difficult. Indeed, it seems to automatically find an efficient decay as evidenced by the "drop down" effect observed in numerical experiments. INNA is a versatile algorithm with good generalization performances. Its flexibility comes from its hyper-parameters α and β which require being tuned for some problems. This task will probably be eased by the improvement of computational resources and in particular, the increasing possibilities of performing parallel computations. This will favor INNA which might become a suitable choice for DNNs that are hard to train with standard methods, in particular due to its ability to reduce parasitic oscillations.

Appendix A

Résumé Détaillé de la Thèse en Français

Mise en contexte

Deep Learning

Un principe fondateur du machine learning (ou apprentissage machine) consiste à supposer qu'une réalité (biologique, physique, etc.) peut s'exprimer à travers une relation de la forme,

$$y = f_{\text{truth}}(x),$$

où $x \in \mathbb{R}^M$ et $y \in \mathbb{R}^D$ sont respectivement appelées variables d'entrée et de sortie, et où M et D sont des entiers naturels non nuls. La fonction f_{truth}, parfois appelée «fonction vérité terrain» est inconnue et potentiellement très complexe et donc extrêmement difficile à déterminer. En machine learning, on cherche alors plutôt à trouver un modèle qui soit une bonne approximation de f_{truth}. Les réseaux de neurones sont une classe de ces modèles, ils sont représentés par une fonction f paramétrée par un vecteur $\theta \in \mathbb{R}^P$ (où $P \in \mathbb{N}^\star$). Étant donné x et un choix de paramètre θ, un réseau de neurones produit une variable de sortie $\hat{y}$ via la relation,

$$\hat{y} = f(x, \theta).$$

L'objectif est de choisir le paramètre θ de manière à ce que $\hat{y}$ soit «le plus proche possible» de la vérité y.

Afin de déterminer ce paramètre, on a recours à une procédure appelée «l'entraînement» du réseau de neurones. Celle-ci se fait en utilisant une collection de $N \in \mathbb{N}^\star$ données: $(x_n, y_n)_{\{1,\dots,N\}}$ liées pour tout $n \in \{1, \dots, N\}$ par la relation $y_n = f_{\text{truth}}(x_n)$ (cette relation peut éventuellement contenir un terme supplémentaire de perturbation). On cherche alors à minimiser les erreurs de prédiction du réseau f sur ces données. Cela revient à trouver $\theta \in \mathbb{R}^P$ qui minimise une fonction de la forme,

$$\mathcal{J}(\theta) = \sum_{n=1}^{N} \ell(y_n, f(x_n, \theta)),$$

où ℓ est une mesure de dissimilarité (typiquement, $\ell\colon (y,\hat{y}) \in \mathbb{R}^D \times \mathbb{R}^D \mapsto \|\hat{y}_- y\|^2$) et $\mathcal{J}$ est appelée «fonction de perte». D'autres problématiques sont liées à la minimisation de cette fonction, par exemple la question de la généralisation (la prédiction sur des données non utilisées pour l'entraînement). Le deep learning (ou apprentissage profond) désigne le fait d'entraîner et d'utiliser des réseaux de neurones et tout ce qui y est lié.

Optimisation en grande dimension

L'entraînement d'un réseaux de neurones se formule donc comme un problème d'optimisation non contraint:

$$\min_{\theta \in \mathbb{R}^P} \mathcal{J}(\theta).$$

En raison de la structure du réseau f, la fonction $\mathcal{J}$ est en général non-convexe et parfois non-differentiable. Le problème ci-dessus est donc posé dans un cadre très général, le rendant difficile à résoudre. En effet, dans un cadre non-convexe et non-lisse, de nombreux problèmes se posent tel que l'existence de minimiseurs locaux qui ne soient pas globaux, de points selles, de singularités, etc. De plus, le jeu de donné d'entraînement et le vecteur de paramètres du réseau sont généralement très grands, il en résulte un cout de calcul de $\mathcal{J}$ très élevé, rendant coûteuse la recherche numérique de solution.

Supposons pour le moment que $\mathcal{J}$ soit différentiable, la méthode numérique la plus classique pour résoudre ce genre de problème est la descente de gradient. Pour une initialisation $\theta_0 \in \mathbb{R}^P$, cet algorithme consiste en le processus itératif suivant, pour tout $k \in \mathbb{N}$,

$$\theta_{k+1} = \theta_k - \gamma \nabla \mathcal{J}(\theta_k),$$

où $\nabla \mathcal{J}$ désigne le gradient de $\mathcal{J}$ et $\gamma > 0$ est appelé «longueur de pas». Cet algorithme se formule de manière assez simple et requiert simplement l'évaluation du gradient de la fonction de perte. Pour cela on utilise la méthode de backpropagtion (ou rétropropagation) qui est une manière optimisée de calculer des gradients en deep learning. Bien que cela rende l'évaluation du gradient raisonnable, le cout de calcul (et donc le temps d'exécution) de la descente de gradient reste important car de nombreuses itérations sont en général nécessaires pour obtenir des résultats satisfaisants.

Ainsi, une approche alternative s'avère plus efficace en pratique: le sous-échantillonnage par mini-lots. Cette technique consiste à remplacer $\nabla \mathcal{J}(\theta_k)$ dans la descente de gradient, par une approximation calculée sur un sous-ensemble du jeu de données d'entraînement. Plus précisément, on choisit $\mathsf{B}_k \subset \{1,\dots,N\}$ et on calcule $\nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k) = \sum_{n \in \mathsf{B}_k} \ell(y_n, f(x_n,\theta))$. L'algorithme obtenu est du type descente gradient stochastique (SGD), dont les itérations $k \in \mathbb{N}$ sont,

$$\theta_{k+1} = \theta_k - \gamma_k \nabla \mathcal{J}_{\mathsf{B}_k}(\theta_k),$$

où $(\gamma_k)_{k\in\mathbb{N}}$ est une suite de longueurs de pas. Cet algorithme, bien qu'ancien (Robbins and Monro, 1951) reste encore aujourd'hui l'outil de base pour entraîner des réseaux de neurones.

Motivations et problématiques

L'objectif principal de cette thèse est justement de proposer de nouveaux algorithmes pour l'entraînement de réseaux de neurones. En particulier nous voulons aller plus loin que les méthodes du premier ordre (telles que SGD) et construire des algorithmes exploitant de l'information de second ordre (des dérivées secondes de $\mathcal{J}$). Une telle approche présente de nombreux avantages. Les méthodes d'ordre deux peuvent s'avérer plus rapides sur certains problèmes et plus robustes à un mauvais conditionnement (en particulier la méthode de Newton). L'information d'ordre deux permet aussi de déterminer le signe de la courbure locale de $\mathcal{J}$ et ainsi éviter plus efficacement les maximums et les points selles. Elle peut enfin aider à choisir la suite de longueurs de pas $(\gamma_k)_{k\in\mathbb{N}}$.

Malgré les avantages indéniables d'une telle approche, celle-ci implique de relever plusieurs défis de taille. Premièrement, et comme dit précédemment, la fonction de perte $\mathcal{J}$ est parfois non-differentiable (elle l'est en revanche presque partout), ce qui limite l'utilité des dérivées d'ordre deux, quand celles-ci sont bien définies. Ensuite, l'information du second ordre peut s'avérer très coûteuse à calculer, et perdre tout bénéfice lorsque combinée au sous-échantillonnage par mini-lots. En résumé, voici les trois principales problématiques auxquelles nous nous intéressons.

- *Peut-on construire des algorithmes exploitant efficacement l'information de second-ordre en dépit de toutes les limites techniques et théoriques du deep learning ?*
- *Pour ces algorithmes, est-ce que l'information de second-ordre profite réellement à l'entraînement malgré le sous-échantillonnage par mini-lots ?*
- *Quelles garanties et quelles vitesses de convergence pouvons-nous obtenir pour ces algorithmes dans un cadre théorique où la convergence peut sembler incertaine ?*

Nous résumons maintenant le contenu des trois chapitres qui tentent de répondre à ces questions.

Résumé du Chapitre 2

Le point de se départ de ce chapitre est l'équation différentielle suivante, introduite par Alvarez et al. (2002),

$$\underbrace{\ddot{\theta}(t)}_{\text{Inertie}} + \underbrace{\alpha\,\dot{\theta}(t)}_{\text{Frottement visqueux}} + \underbrace{\beta\,\nabla^2\mathcal{J}(\theta(t))\dot{\theta}(t)}_{\text{Effet Newtonien}} + \underbrace{\nabla\mathcal{J}(\theta(t))}_{\text{Gravité}} = 0, \quad \text{pour tout } t\in[0,+\infty),$$

où, θ est une fonction différentiable du temps (appelée solution ou trajectoire) et $\dot{\theta}$ et $\ddot{\theta}$ désignent ses dérivées premières et secondes, enfin $\alpha \geq 0$ et $\beta > 0$ sont des hyper-paramètres. Cette équation différentielle, appelée DIN pour «Dynamical Inertial Newton», modélise un mélange entre les méthodes de type gradient accéléré et de celles de type Newton. Elle peut également s'interpréter du point de vue des lois de la mécanique, comme détaillé dans l'équation ci-dessus.

En approximant les solutions de DIN par un processus de discrétisation, on peut construire des algorithmes d'optimisation afin de minimiser $\mathcal{J}$. Cependant, DIN dans la forme ci-dessus n'est pas adapté aux fonctions non-différentiables rencontrées en deep learning. Une partie du chapitre est dédiée à trouver une formulation plus adaptée. Nous obtenons celle-ci en reformulant DIN sous forme d'un système du premier ordre. Nous l'adaptons ensuite au sous-échantillonnage par mini-lots et mettons en évidence que ce sous-échantillonnage pose problème pour des fonctions non-lisses. Nous introduisons alors un nouvel opérateur D (à valeur ensembliste) étendant les notions de gradients et sous gradients usuelles et non adaptées au deep learning.

Nous obtenons un nouvel algorithme que nous appelons INNA et dont une itération $k \in \mathbb{N}$ prend la forme suivante,

$$\begin{cases} v_k & \in D\mathcal{J}_{\mathsf{B}_k}(\theta_k) \\ \theta_{k+1} & = \theta_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k - \beta v_k \right) , \\ \psi_{k+1} & = \psi_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k \right) \end{cases}$$

où $\alpha \geq 0$, $\beta > 0$ et $(\gamma_k)_{k\in\mathbb{N}}$ sont des hyper-paramètres de l'algorithme et $\forall k \in \mathbb{N}$, $\mathsf{B}_k \subset \{1, \dots, N\}$. À l'aide des résultats de Benaïm et al. (2005), nous établissons un lien entre les comportements asymptotiques d'INNA et des solutions de DIN et prouvons ainsi la convergence presque sûre d'INNA vers les points critiques de $\mathcal{J}$. Enfin nous entraînons des réseaux de neurones avec INNA et obtenons des résultats numériques très prometteurs.

Résumé du Chapitre 3

Le chapitre 3 poursuit l'analyse asymptotique de INNA et de DIN commencée dans le chapitre 2. Alors que l'on a précédemment montré la convergence de INNA vers des points critiques de $\mathcal{J}$, on étudie maintenant la nature (minimums, points selles, maximums) des points critiques trouvés en pratique, cette question est importante afin de minimiser $\mathcal{J}$. INNA est un mélange de la méthode du gradient accéléré—connue pour éviter les points selles stricts et les maximums (Lee et al., 2016; O'Neill and S. J. Wright, 2019)—et la méthode de Newton—qui elle converge vers tout type de points critiques. Pour des fonctions différentiables et pour l'algorithme utilisé sans sous-échantillonnage par mini-lots, nous montrons qu'INNA a tendance à éviter les points selles strictes et les maximums quel que soit le choix des hyper-paramètres dès lors que $\alpha > 0$. Ces résultats sont obtenus en utilisant le théorème de la variété stable. Nous apportons également un éclairage nouveau sur DIN et INNA grâce au théorème d'Hartman-Grobman et nous illustrons les résultats numériquement.

Résumé du Chapitre 4

Ce dernier chapitre est indépendant des deux précédents. Dans celui-ci nous considérons exclusivement des réseaux de neurones pour lesquels la fonction de perte $\mathcal{J}$ est deux fois différentiable. Notre objectif est de construire une procédure automatique afin d'adapter la suite $(\gamma_k)_{k\in\mathbb{N}}$ de longueurs de pas de SGD à la courbure de la fonction $\mathcal{J}$. Notre approche se base sur le modèle variationnel suivant. Si l'on souhaite mettre à jour le paramètre $\theta \in \mathbb{R}^P$ en se déplaçant dans la direction $d \in \mathbb{R}^P$ alors la longueur de pas $\gamma > 0$ optimale (à l'ordre deux) est celle qui minimise le modèle suivant:

$$q_d(\gamma) = \mathcal{J}(\theta) + \gamma\langle\nabla\mathcal{J}(\theta), d\rangle + \frac{\gamma^2}{2}\langle\nabla^2\mathcal{J}(\theta)d, d\rangle.$$

Ainsi, si l'on considère $d = -\nabla\mathcal{J}(\theta)$ comme dans la descente de gradient, et si $\mathcal{J}$ est localement convexe dans cette direction, ce modèle suggère le choix de longueur pas suivant,

$$\gamma = \frac{\|\nabla\mathcal{J}(\theta)\|^2}{\langle\nabla^2\mathcal{J}(\theta)\nabla\mathcal{J}(\theta), \nabla\mathcal{J}(\theta)\rangle}.$$

À l'inverse quand $\mathcal{J}$ est localement concave, on choisit d'ignorer ce modèle et de prendre une longueur de pas plutôt grande. Ce choix de longueur de pas proposé par Alvarez and Cabot (2004) étant trop coûteux à calculer, on a recours à une procédure de discrétisation. À chaque itération $k \in \mathbb{N}^\star$, le pas proposé est finalement le suivant,

$$\gamma_k = \begin{cases} \frac{\|\theta_k - \theta_{k-1}\|^2}{\langle\theta_k - \theta_{k-1}, \nabla\mathcal{J}(\theta_k) - \nabla\mathcal{J}(\theta_{k-1})\rangle} & \text{if } \langle\Delta\theta_k, \Delta g_k\rangle > 0 \\ \nu & \text{sinon} \end{cases},$$

où $\nu > 0$ est un hyper-paramètre à choisir. Cette longueur de pas est de type Barzilai-Borwein (Barzilai and Borwein, 1988).

Elle nécessite de faire des calculs exacts de gradients, elle est donc adaptée à la descente de gradient mais pas à SGD. Dans la suite du chapitre nous utilisons des considérations empiriques et théoriques afin de construire une adaptation de ce choix de longueur de pas qui soit appropriée à SGD et dont les performances soient bonnes malgré le sous-échantillonnage par mini-lots. On obtient finalement un nouvel algorithme, appelé Step-Tuned SGD. Celui-ci est une modification simple et peu coûteuse de SGD qui s'avère très efficace dans certaines de nos expériences d'entraînement de réseaux de neurones, en particulier sur les réseaux résiduels et ceux utilisant la fonctionnalité de «BatchNorm».

En nous basant sur un «lemme de descente», nous montrons la convergence presque sûre de Step-Tuned SGD vers les points critiques de $\mathcal{J}$ et explicitons une vitesse de convergence.

Bibliography

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. (2016). "Tensorflow: A system for large-scale machine learning." In: *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation OSDI*, pp. 265–283 (cit. on pp. 2, 21, 64).

Adil, S. (2018). *Opérateurs monotones aléatoires et application à l'optimisation stochastique*. PhD Thesis, Paris Saclay (cit. on p. 37).

Agarwal, N., B. Bullins, X. Chen, E. Hazan, K. Singh, C. Zhang, and Y. Zhang (2019). "Efficient full-matrix adaptive regularization." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 102–110 (cit. on p. 30).

Alber, Y. I., A. N. Iusem, and M. V. Solodov (1998). "On the projected subgradient method for nonsmooth convex optimization in a Hilbert space." In: *Mathematical Programming* 81.1, pp. 23–35 (cit. on pp. 115, 123).

Alvarez, F. and A. Cabot (2004). "Steepest descent with curvature dynamical system." In: *Journal of Optimization Theory and Applications* 120.2, pp. 247–273 (cit. on pp. 32, 102, 141).

Alvarez, F., H. Attouch, J. Bolte, and P. Redont (2002). "A second-order gradient-like dissipative dynamical system with Hessian-driven damping: Application to optimization and mechanics." In: *Journal de Mathématiques Pures et Appliquées* 81.8, pp. 747–779 (cit. on pp. 31, 37, 39, 42, 60, 61, 70, 73, 74, 139).

Alvarez, F. and J. M. Pérez (1998). "A dynamical system associated with Newton's method for parametric approximations of convex minimization problems." In: *Applied Mathematics and Optimization* 38, pp. 193–217 (cit. on p. 61).

Amari, S.-I. (1998). "Natural gradient works efficiently in learning." In: *Neural Computation* 10.2, pp. 251–276 (cit. on p. 30).

Armijo, L. (1966). "Minimization of functions having Lipschitz continuous first partial derivatives." In: *Pacific Journal of Mathematics* 16.1, pp. 1–3 (cit. on pp. 24, 103).

Asi, H. and J. C. Duchi (2019). "The importance of better models in stochastic optimization." In: *Proceedings of the National Academy of Sciences* 116.46, pp. 22924–22930 (cit. on p. 30).

Attouch, H., J. Bolte, P. Redont, and A. Soubeyran (2010). "Proximal alternating minimization and projection methods for nonconvex problems: An approach based on the Kurdyka-Łojasiewicz inequality." In: *Mathematics of Operations Research* 35.2, pp. 438–457 (cit. on p. 40).

Attouch, H., Z. Chbani, J. Fadili, and H. Riahi (2020). "First-order optimization algorithms via inertial systems with Hessian driven damping." In: *Mathematical Programming* (cit. on pp. 62, 70, 82).

Attouch, H., Z. Chbani, J. Fadili, and H. Riahi (2021). "Convergence of iterates for first-order optimization algorithms with inertia and Hessian driven damping." In: *arXiv preprint:2107.05943* (cit. on pp. 70, 82).

Attouch, H., J. Peypouquet, and P. Redont (2016). "Fast convex optimization via inertial dynamics with Hessian driven damping." In: *Journal of Differential Equations* 261.10, pp. 5734–5783 (cit. on p. 70).

Attouch, H., J. Peypouquet, and P. Redont (2014). "A dynamical approach to an inertial forward-backward algorithm for convex minimization." In: *SIAM Journal on Optimization* 24.1, pp. 232–256 (cit. on p. 70).

Attouch, H. and P. Redont (2001). "The second-order in time continuous Newton method." In: *Approximation, Optimization and Mathematical Economics*. Springer, pp. 25–36 (cit. on p. 37).

Aubin, J.-P. and A. Cellina (2012). *Differential inclusions: Set-valued maps and viability theory*. Springer (cit. on p. 43).

Babaie-Kafaki, S. and M. Fatemi (2013). "A modified two-point stepsize gradient algorithm for unconstrained minimization." In: *Optimization Methods and Software* 28.5, pp. 1040–1050 (cit. on p. 103).

Barakat, A. and P. Bianchi (2021). "Convergence and dynamical behavior of the ADAM algorithm for nonconvex stochastic optimization." In: *SIAM Journal on Optimization* 31.1, pp. 244–274 (cit. on p. 37).

Barzilai, J. and J. M. Borwein (1988). "Two-point step size gradient methods." In: *IMA Journal of Numerical Analysis* 8.1, pp. 141–148 (cit. on pp. 32, 102, 141).

Benaïm, M. (1999). "Dynamics of stochastic approximation algorithms." In: *Séminaire de Probabilités XXXIII*. Springer, pp. 1–68 (cit. on p. 36).

Benaïm, M., J. Hofbauer, and S. Sorin (2005). "Stochastic approximations and differential inclusions." In: *SIAM Journal on Control and Optimization* 44.1, pp. 328–348 (cit. on pp. 32, 36, 47–50, 53, 134, 140).

Bengio, Y. (2012). "Practical recommendations for gradient-based training of deep architectures." In: *Neural networks: Tricks of the Trade*. Springer, pp. 437–478 (cit. on p. 11).

Bengio, Y., Y. LeCun, et al. (2007). "Scaling learning algorithms towards AI." In: *Large-scale Kernel Machines* 34.5, pp. 1–41 (cit. on p. 2).

Berahas, A. S., R. Bollapragada, and J. Nocedal (2020). "An investigation of Newton-sketch and subsampled Newton methods." In: *Optimization Methods and Software* 35.4, pp. 661–680 (cit. on p. 24).

Bertsekas, D. P. (1999). *Nonlinear programming, second edition*. Athena Scientific (cit. on pp. 18, 26, 125).

Bianchi, P., W. Hachem, and S. Schechtman (2020). "Convergence of constant step stochastic gradient descent for non-smooth non-convex functions." In: *arXiv preprint:2005.08513* (cit. on p. 48).

Biglari, F. and M. Solimanpur (2013). "Scaling on the spectral gradient method." In: *Journal of Optimization Theory and Applications* 158, pp. 626–635 (cit. on pp. 103, 105).

Bolte, J., A. Daniilidis, and A. Lewis (2007a). "The Łojasiewicz inequality for nonsmooth subanalytic functions with applications to subgradient dynamical systems." In: *SIAM Journal on Optimization* 17.4, pp. 1205–1223 (cit. on p. 60).

Bolte, J., A. Daniilidis, A. Lewis, and M. Shiota (2007b). "Clarke subgradients of stratifiable functions." In: *SIAM Journal on Optimization* 18.2, pp. 556–572 (cit. on pp. 49, 50, 54).

Bolte, J., A. Daniilidis, O. Ley, and L. Mazet (2010). "Characterizations of Łojasiewicz inequalities: Subgradient flows, talweg, convexity." In: *Transactions of the American Mathematical Society* 362.6, pp. 3319–3363 (cit. on p. 54).

Bolte, J. and E. Pauwels (2020a). "A mathematical model for automatic differentiation in machine learning." In: *Advances in Neural Information Processing Systems (NIPS)* (cit. on pp. 48, 135).

Bolte, J. and E. Pauwels (2020b). "Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning." In: *Mathematical Programming*, pp. 1–33 (cit. on p. 45).

Bolte, J., S. Sabach, and M. Teboulle (2014). "Proximal alternating linearized minimization for nonconvex and nonsmooth problems." In: *Mathematical Programming* 146.1-2, pp. 459–494 (cit. on p. 54).

Borkar, V. S. (2009). *Stochastic approximation: A dynamical systems viewpoint*. Springer (cit. on p. 36).

Bottou, L. and O. Bousquet (2008). "The tradeoffs of large scale learning." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 161–168 (cit. on pp. 2, 23).

Bottou, L., F. E. Curtis, and J. Nocedal (2018). "Optimization methods for large-scale machine learning." In: *SIAM Review* 60.2, pp. 223–311 (cit. on pp. 2, 26).

Boyer, C. and A. Godichon-Baggioni (2020). "On the asymptotic rate of convergence of stochastic Newton algorithms and their weighted averaged versions." In: *arXiv preprint:2011.09706* (cit. on p. 27).

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). "Language models are few-shot learners." In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 33 (cit. on pp. 2, 19).

Broyden, C. G. (1967). "Quasi-Newton methods and their application to function minimisation." In: *Mathematics of Computation* 21.99, pp. 368–381 (cit. on p. 28).

Byrd, R. H., G. M. Chin, W. Neveitt, and J. Nocedal (2011). "On the use of stochastic Hessian information in optimization methods for machine learning." In: *SIAM Journal on Optimization* 21.3, pp. 977–995 (cit. on p. 27).

Byrd, R. H., S. L. Hansen, J. Nocedal, and Y. Singer (2016). "A stochastic quasi-Newton method for large-scale optimization." In: *SIAM Journal on Optimization* 26.2, pp. 1008–1031 (cit. on p. 28).

Carmon, Y., J. C. Duchi, O. Hinder, and A. Sidford (2017). "Convex until proven guilty: Dimension-free acceleration of gradient descent on non-convex functions." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 654–663 (cit. on pp. 31, 106, 123).

Castera, C. (2021). "Inertial Newton algorithms avoiding strict saddle points." In: *preprint arXiv:2111.04596* (cit. on pp. 33, 69).

Castera, C., J. Bolte, C. Févotte, and E. Pauwels (2019a). "An inertial Newton algorithm for deep learning." In: *NeurIPS Workshop: Beyond First-order methods in Machine Learning* (cit. on p. 33).

Castera, C., J. Bolte, C. Févotte, and E. Pauwels (2021a). "An inertial Newton algorithm for deep learning." In: *Journal of Machine Learning Research* 22.134, pp. 1–31 (cit. on pp. 33, 35).

Castera, C., J. Bolte, C. Févotte, and E. Pauwels (2021b). "Second-order step-size tuning of SGD for non-convex optimization." In: *Neural Processing Letters*, To appear (cit. on pp. 33, 101).

Castera, C., J. Bolte, C. Févotte, and E. Pauwels (2019b). "An Inertial Newton Algorithm for Deep Learning." In: *arXiv preprint:1905.12278v1* (cit. on p. 45).

Castera, C. (2019). *INNA for Deep Learning*. https://github.com/camcastera/Inna-for-DeepLearning (cit. on p. 64).

Chen, L. and H. Luo (2019). "First order optimization methods based on Hessian-driven Nesterov accelerated gradient flow." In: *arXiv preprint:1912.09276* (cit. on p. 70).

Chollet, F. (2015). *Keras*. https://github.com/fchollet/keras (cit. on p. 64).

Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). "The loss surfaces of multilayer networks." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 192–204 (cit. on p. 15).

Clarke, F. H. (1990). *Optimization and nonsmooth analysis*. SIAM (cit. on pp. 17, 36, 42, 45).

Coste, M. (2000). *An introduction to o-minimal geometry*. Istituti Editoriali e Poligrafici Internazionali Pisa (cit. on pp. 40, 41).

Curtis, F. E. and W. Guo (2016). "Handling nonpositive curvature in a limited memory steepest descent method." In: *IMA Journal of Numerical Analysis* 36.2, pp. 717–742 (cit. on p. 103).

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314 (cit. on p. 9).

Dai, Y., J. Yuan, and Y.-X. Yuan (2002). "Modified two-point stepsize gradient methods for unconstrained optimization." In: *Computational Optimization and Applications* 22.1 (cit. on pp. 103, 105).

Daniilidis, A. and D. Drusvyatskiy (2020). "Pathological subgradient dynamics." In: *SIAM Journal on Optimization* 30.2, pp. 1327–1338 (cit. on p. 16).

Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014). "Identifying and attacking the saddle point problem in high-dimensional non-convex op-

timization." In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 27 (cit. on pp. 15, 70).

Davis, D., D. Drusvyatskiy, S. Kakade, and J. D. Lee (2020). "Stochastic subgradient method converges on tame functions." In: *Foundations of Computational Mathematics* 20.1, pp. 119–154 (cit. on pp. 3, 16, 37, 40, 48, 49).

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255 (cit. on pp. 2, 17, 19).

van den Dries, L. (1998). *Tame topology and o-minimal structures*. Cambridge University Press (cit. on pp. 40, 50).

Duchi, J. C. and F. Ruan (2018). "Stochastic methods for composite and weakly convex optimization problems." In: *SIAM Journal on Optimization* 28.4, pp. 3229–3259 (cit. on p. 48).

Duchi, J., E. Hazan, and Y. Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of Machine Learning Research* 12.7, pp. 2121–2159 (cit. on pp. 2, 29, 64).

Dudar, V., G. Chierchia, E. Chouzenoux, J.-C. Pesquet, and V. Semenov (2017). "A two-stage subspace trust region approach for deep neural network training." In: *European Signal Processing Conference (EUSIPCO)*, pp. 291–295 (cit. on p. 31).

Fukushima, K. and S. Miyake (1982). "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition." In: *Competition and Cooperation in Neural Nets*. Springer, pp. 267–285 (cit. on p. 6).

Ghadimi, S. and G. Lan (2013). "Stochastic first-and zeroth-order methods for nonconvex stochastic programming." In: *SIAM Journal on Optimization* 23.4, pp. 2341–2368 (cit. on p. 3).

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT Press (cit. on pp. 9, 20).

Goudou, X. and J. Munier (2009). "The gradient and heavy ball with friction dynamical systems: The quasiconvex case." In: *Mathematical Programming* 116.1, pp. 173–191 (cit. on pp. 70, 71).

Grobman, D. M. (1959). "Homeomorphism of systems of differential equations." In: *Doklady Akademii Nauk SSSR* 128.5, pp. 880–881 (cit. on p. 71).

Haragus, M. and G. Iooss (2010). *Local bifurcations, center manifolds, and normal forms in infinite-dimensional dynamical systems*. Springer Science & Business Media (cit. on p. 75).

Hartman, P. (1960). "A lemma in the theory of structural stability of differential equations." In: *Proceedings of the American Mathematical Society* 11.4, pp. 610–620 (cit. on p. 71).

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The elements of statistical learning: Data mining, inference, and prediction*. Springer Science & Business Media (cit. on pp. 3, 10).

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778 (cit. on pp. 6, 7, 116, 117).

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Psychology Press (cit. on pp. 2, 6).

Heinonen, J. (2005). *Lectures on Lipschitz analysis*. University of Jyväskylä (cit. on p. 16).

Hinton, G. E., S. Osindero, and Y.-W. Teh (2006). "A fast learning algorithm for deep belief nets." In: *Neural Computation* 18.7, pp. 1527–1554 (cit. on p. 2).

Hinton, G. E. and R. R. Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks." In: *Science* 313.5786, pp. 504–507 (cit. on pp. 10, 116).

Hornik, K., M. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are universal approximators." In: *Neural Networks* 2.5, pp. 359–366 (cit. on p. 9).

Ioffe, S. and C. Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 448–456 (cit. on p. 120).

Johnson, R. and T. Zhang (2013). "Accelerating stochastic gradient descent using predictive variance reduction." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 315–323 (cit. on p. 103).

Kelley, A. (1966). "The stable, center-stable, center, center-unstable, unstable manifolds." In: *Journal of Differential Equations* (cit. on p. 71).

Kingma, D. P. and J. Ba (2015). "Adam: A method for stochastic optimization." In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 30, 64, 110).

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images*. Tech. rep. Canadian Institute for Advanced Research (cit. on pp. 63, 116).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems (NIPS)* 25, pp. 1097–1105 (cit. on pp. 2, 17).

Krogh, A. and J. A. Hertz (1992). "A simple weight decay can improve generalization." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 950–957 (cit. on p. 11).

Kurdyka, K. (1998). "On gradients of functions definable in o-minimal structures." In: *Annales de l'Institut Fourier*. Vol. 48, pp. 769–783 (cit. on p. 50).

Kushner, H. and G. G. Yin (2003). *Stochastic approximation and recursive algorithms and applications*. Springer (cit. on p. 36).

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). "Backpropagation applied to handwritten zip Code recognition." In: *Neural Computation* 1.4, pp. 541–551 (cit. on pp. 2, 6, 17).

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on pp. 63, 116).

LeCun, Y., C. Cortes, and C. Burges (2010). "MNIST handwritten digit database." In: *ATT Labs* (cit. on p. 116).

Lee, J. D., M. Simchowitz, M. I. Jordan, and B. Recht (2016). "Gradient descent only converges to minimizers." In: *Conference on Learning Theory (COLT)*, pp. 1246–1257 (cit. on pp. 32, 70, 71, 85, 92, 140).

Lemaréchal, C. (2012). "Cauchy and the gradient method." In: *Doc Math Extra* 251.254, p. 10 (cit. on p. 18).

Leshno, M., V. Y. Lin, A. Pinkus, and S. Schocken (1993). "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function." In: *Neural Networks* 6.6, pp. 861–867 (cit. on p. 9).

Li, X. and F. Orabona (2019). "On the convergence of stochastic gradient descent with adaptive stepsizes." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 983–992 (cit. on pp. 3, 103, 114).

Liang, J., Y. Xu, C. Bao, Y. Quan, and H. Ji (2019). "Barzilai–Borwein-based adaptive learning rate for deep learning." In: *Pattern Recognition Letters* 128, pp. 197–203 (cit. on pp. 31, 103, 106, 110).

Lin, M., Q. Chen, and S. Yan (2014). "Network In Network." In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 64, 116).

Ljung, L. (1977). "Analysis of recursive stochastic algorithms." In: *IEEE Transactions on Automatic Control* 22.4, pp. 551–575 (cit. on p. 36).

Loshchilov, I. and F. Hutter (2019). "Decoupled weight decay regularization." In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 31).

Marcus, M., B. Santorini, and M. A. Marcinkiewicz (1993). "Building a large annotated corpus of English: The Penn Treebank." In: *Computational Linguistics* 19.2, pp. 313–330 (cit. on p. 19).

Mark, C. (2017). *Neural network*. https://github.com/battlesnake/neural (cit. on p. 5).

Martens, J. (2010). "Deep learning via Hessian-free optimization." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 735–742 (cit. on p. 27).

Martens, J. and R. Grosse (2015). "Optimizing neural networks with Kronecker-factored approximate curvature." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2408–2417 (cit. on pp. 31, 119).

Martens, J., I. Sutskever, and K. Swersky (2012). "Estimating the Hessian by backpropagating curvature." In: *Proceedings of the International Conference on Machine Learning (ICML)* (cit. on p. 28).

McCulloch, W. S. and W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity." In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133 (cit. on p. 17).

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). "Efficient estimation of word representations in vector space." In: *arXiv preprint:1301.3781* (cit. on p. 2).

Mizutani, E. and S. E. Dreyfus (2008). "Second-order stagewise backpropagation for Hessian-matrix analyses and investigation of negative curvature." In: *Neural Networks* 21, pp. 193–203 (cit. on p. 31).

Moulines, E. and F. R. Bach (2011). "Non-asymptotic analysis of stochastic approximation algorithms for machine learning." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 451–459 (cit. on p. 3).

Nesterov, Y. (2003). *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media (cit. on pp. 14, 17).

Nesterov, Y. (1983). "A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$." In: *Doklady an USSR*. Vol. 269, pp. 543–547 (cit. on pp. 29, 83).

Nocedal, J. and S. Wright (2006). *Numerical optimization*. Springer Science & Business Media (cit. on pp. 25, 71).

O'Neill, M. and S. J. Wright (2019). "Behavior of accelerated gradient methods near critical points of nonconvex functions." In: *Mathematical Programming* 176.1, pp. 403–427 (cit. on pp. 32, 70, 71, 85, 140).

Owens, J. D., M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips (2008). "GPU computing." In: *Proceedings of the IEEE* 96.5, pp. 879–899 (cit. on p. 19).

Palmer, K. J. (1973). "A generalization of Hartman's linearization theorem." In: *Journal of Mathematical Analysis and Applications* 41.3, pp. 753–758 (cit. on p. 83).

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. (2019). "Pytorch: An imperative style, high-performance deep learning library." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 8026–8037 (cit. on pp. 2, 21, 116).

Pearlmutter, B. A. (1994). "Fast exact multiplication by the Hessian." In: *Neural Computation* 6.1, pp. 147–160 (cit. on p. 28).

Perko, L. (2013). *Differential equations and dynamical systems*. Springer Science & Business Media (cit. on pp. 75, 80).

Pliss, V. A. (1964). "A reduction principle in the theory of stability of motion." In: *Izvestiya Akademii Nauk SSSR. Seriya Matematicheskaya* 28 (cit. on p. 71).

Polyak, B. T. (1964). "Some methods of speeding up the convergence of iteration methods." In: *USSR Computational Mathematics and Mathematical Physics* 4.5, pp. 1–17 (cit. on p. 28).

Raydan, M. (1997). "The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem." In: *SIAM Journal on Optimization* 7.1, pp. 26–33 (cit. on pp. 102, 105).

Reddi, S. J., S. Kale, and S. Kumar (2018). "On the convergence of Adam and beyond." In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 31).

Rios-Zertuche, R. (2020). "Examples of pathological dynamics of the subgradient method for Lipschitz path-differentiable functions." In: *arXiv preprint:2007.11699* (cit. on p. 16).

Robbins, H. and S. Monro (1951). "A stochastic approximation method." In: *The Annals of Mathematical Statistics* 22.1, pp. 400–407 (cit. on pp. 2, 23, 24, 138).

Robbins, H. and D. Siegmund (1971). "A convergence theorem for non negative almost supermartingales and some applications." In: *Optimizing Methods in Statistics*. Elsevier, pp. 233–257 (cit. on pp. 115, 127).

Robles-Kelly, A. and A. Nazari (2019). "Incorporating the Barzilai-Borwein adaptive step size into subgradient methods for deep network training." In: *Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1–6 (cit. on p. 103).

Rockafellar, R. T. (1996). *Convex analysis*. Princeton University Press (cit. on p. 17).

Rosenblatt, F. (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6, p. 386 (cit. on pp. 2, 4, 17).

Rossum, G. (1995). *Python reference manual*. CWI (cit. on p. 2).

Rumelhart, D. E. and G. E. Hinton (1986). "Learning representations by back-propagating errors." In: *Nature* 323.9, pp. 533–536 (cit. on pp. 2, 17, 20).

Schraudolph, N. N., J. Yu, and S. Günter (2007). "A stochastic quasi-Newton method for online convex optimization." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (cit. on pp. 28, 109).

Shi, B., S. S. Du, M. I. Jordan, and W. J. Su (2021). "Understanding the acceleration phenomenon via high-resolution differential equations." In: *Mathematical Programming* (cit. on p. 70).

Shiota, M. (2012). *Geometry of subanalytic and semialgebraic sets*. Springer Science & Business Media (cit. on p. 40).

Shub, M. (2013). *Global stability of dynamical systems*. Springer Science & Business Media (cit. on p. 85).

Su, W., S. Boyd, and E. Candes (2014). "A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 2510–2518 (cit. on pp. 62, 83).

Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). "On the importance of initialization and momentum in deep learning." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1139–1147 (cit. on p. 29).

Tan, C., S. Ma, Y.-H. Dai, and Y. Qian (2016). "Barzilai-Borwein step size for stochastic gradient descent." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 685–693 (cit. on pp. 31, 103, 106, 110).

Tieleman, T. and G. Hinton (2012). "RMSprop: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning* 4.2, pp. 26–31 (cit. on p. 29).

Wilson, A. C., R. Roelofs, M. Stern, N. Srebro, and B. Recht (2017). "The marginal value of adaptive gradient methods in machine learning." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 4148–4158 (cit. on p. 120).

Xiao, Y., Q. Wang, and D. Wang (2010). "Notes on the Dai–Yuan–Yuan modified spectral gradient method." In: *Journal of Computational and Applied Mathematics* 234.10, pp. 2986–2992 (cit. on pp. 103, 105).

Xu, P., F. Roosta, and M. W. Mahoney (2020). "Second-order optimization for non-convex machine learning: An empirical study." In: *Proceedings of the SIAM International Conference on Data Mining (SDM20)*. SIAM, pp. 199–207 (cit. on p. 24).

Zeiler, M. D. (2012). "Adadelta: An adaptive learning rate method." In: *arXiv preprint:1212.5701* (cit. on p. 31).

Zhang, M., J. Lucas, J. Ba, and G. E. Hinton (2019). "Lookahead optimizer: k steps forward, 1 step back." In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 32 (cit. on p. 31).

Zhuang, J., T. Tang, Y. Ding, S. C. Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan (2020). "AdaBelief optimizer: Adapting stepsizes by the belief in observed gradients." In: *Advances in Neural Information Processing Systems (NIPS)* 33 (cit. on pp. 31, 103).