Université de Montréal

Steepest Descent as Linear Quadratic Regulation

par

Simon Dufort-Labbé

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

31 août 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Steepest Descent as Linear Quadratic Regulation

présenté par

Simon Dufort-Labbé

a été évalué par un jury composé des personnes suivantes :

Simon Lacoste-Julien
(président-rapporteur)

Pierre-Luc Bacon
(directeur de recherche)

Ioannis Mitliagkas
(membre du jury)

Résumé

Concorder un modèle à certaines observations, voilà qui résume assez bien ce que l'apprentissage machine cherche à accomplir. Ce concept est maintenant omniprésent dans nos vies, entre autre grâce aux percées récentes en apprentissage profond. La stratégie d'optimisation prédominante pour ces deux domaines est la minimisation d'un objectif donné. Et pour cela, la méthode du gradient, méthode de premier-ordre qui modifie les paramètres du modèle à chaque itération, est l'approche dominante. À l'opposé, les méthodes dites de second-ordre n'ont jamais réussi à s'imposer en apprentissage profond. Pourtant, elles offrent des avantages reconnus qui soulèvent encore un grand intérêt. D'où l'importance de la méthode du col, qui unifie les méthodes de premier et second-ordre sous un même paradigme.

Dans ce mémoire, nous établissons un parralèle direct entre la méthode du col et le domaine du contrôle optimal ; domaine qui cherche à optimiser mathématiquement une séquence de décisions. Et certains des problèmes les mieux compris et étudiés en contrôle optimal sont les commandes linéaires quadratiques. Problèmes pour lesquels on connaît très bien la solution optimale. Plus spécifiquement, nous démontrerons l'équivalence entre une itération de la méthode du col et la résolution d'une Commande Linéaire Quadratique (CLQ).

Cet éclairage nouveau implique une approche unifiée quand vient le temps de déployer nombre d'algorithmes issus de la méthode du col, tel que la méthode du gradient et celle des gradients naturels, sans être limitée à ceux-ci. Approche que nous étendons ensuite aux problèmes à horizon infini, tel que les modèles à équilibre profond. Ce faisant, nous démontrons pour ces problèmes que calculer les gradients via la différentiation implicite revient à employer l'équation de Riccati pour solutionner la CLQ associée à la méthode du gradient. Finalement, notons que l'incorporation d'information sur la courbure du problème revient généralement à rencontrer une inversion matricielle dans la méthode du col. Nous montrons que l'équivalence avec les CLQ permet de contourner cette inversion en utilisant une approximation issue des séries de Neumann. Surprenamment, certaines observations empiriques suggèrent que cette approximation aide aussi à stabiliser le processus d'optimisation quand des méthodes de second-ordre sont impliquées ; en agissant comme un

régularisateur adaptif implicite.

Mots-clés : optimisation, apprentissage profond, apprentissage machine, méthode du col, réseaux de neurones, commande linéaire quadratique, algorithme du gradient, méthode des gradients naturels, équation de Riccati, contrôle optimal, modèle à équilibre profond.

Abstract

Machine learning entails training a model to fit some given observations, and recent advances in the field, particularly in deep learning, have made it omnipresent in our lives. Fitting a model usually requires the minimization of a given objective. When it comes to deep learning, first-order methods like gradient descent have become a default tool for optimization in deep learning. On the other hand, second-order methods did not see widespread use in deep learning. Yet, they hold many promises and are still a very active field of research. An important perspective into both methods is steepest descent, which allows you to encompass first and second-order approaches into the same framework.

In this thesis, we establish an explicit connection between steepest descent and optimal control, a field that tries to optimize sequential decision-making processes. Core to it is the family of problems known as Linear Quadratic Regulation; problems that have been well-studied and for which we know optimal solutions. More specifically, we show that performing one iteration of steepest descent is equivalent to solving a Linear Quadratic Regulator (LQR). This perspective gives us a convenient and unified framework for deploying a wide range of steepest descent algorithms, such as gradient descent and natural gradient descent, but certainly not limited to. This framework can also be extended to problems with an infinite horizon, such as deep equilibrium models. Doing so reveals that retrieving the gradient via implicit differentiation is equivalent to recovering it via Riccati's solution to the LQR associated with gradient descent. Finally, incorporating curvature information into steepest descent usually takes the form of a matrix inversion. However, casting a steepest descent step as a LQR also hints toward a trick that allows to sidestep this inversion, by leveraging Neumann's series approximation. Empirical observations provide evidence that this approximation actually helps to stabilize the training process, by acting as an adaptive damping parameter.

Keywords: optimization, deep learning, machine learning, steepest descent, neural networks, linear quadratic regulator, gradient descent, natural gradient descent, Riccati's equation, optimal control, deep equilibrium models.

Contents

List of Tables

List of Figures

List of acronyms and abbreviations

CARE	Continuous Time Algebraic Riccati Equation
CLQ	Commande Linéaire Quadratique
DARE	Discrete Algebraic Riccati Equation
DL	Deep Learning
DNN	Deep Neural Nets
GD	Gradient Descent
KL	Kullback–Leibler (used for Kullback–Leibler divergence)
LQR	Linear Quadratic Regulator
MVP	Matrix-Vector Product
ML	Machine Learning

MNIST	Modified National Institute of Standards and Technology
NGD	Natural Gradient Descent
NN	Neural Network
OC	Optimal Control
OCP	Optimal Control Problem
ODEs	Ordinary Differential Equations
PMP	Pontryagin's Maximum Principle
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent
SVD	Singular Values Decomposition

Acknowledgments

It would be a sin not to first thank my supervisor, Dr. Pierre-Luc Bacon. It was a chance and a privilege to be given the opportunity to work with him. Without his vision and knowledge, bringing to a satisfactory completion this project in 8 months would not have been possible.
Special thanks also need to be addressed to my colleague, Evgenii Nikishin. It is hard to describe how helpful it is to be granted access to a living encyclopedia.
To Ryan D'Orazio and David Yu-Tung Hui, thank you for providing specific comments when it mattered, it was very appreciated.
It goes without saying, discussions and feedback from the members in the research group I am lucky enough to be part of were always insightful. To all of you, thank you.

Je ne peux non plus manquer de remercier ma famille et mon entourage, dont le support constant constitue le socle duquel je peux m'élancer vers les nombreux projets qui m'aspirent. À Ariane, ma conjointe, qui pallie mes défauts et équilibre ma vie. Et finalement à Léonard, mon fils, qui du haut de son an se révéle être une source inépuisable de motivation.

Chapter 1

Introduction

It took a while for Deep Learning (DL) to shine and become the workhorse of artificial intelligence, with many pieces falling into place to make it happens. One of the key components that enabled neural networks to be trained at scale is the development of efficient optimization methods tailored to the high dimensional and noisy regime under which deep neural networks typically operate. First-order optimization methods have predominantly been used for training deep neural networks due to their low computational overhead (see Goodfellow and Vinyals 2015). While computationally advantageous, first-order optimization methods fall short of leveraging the local geometry of the loss landscape; for example, first-order methods will ignore curvature while second-order methods can converge faster by leveraging this information. In order to increase the computational efficiency of second-order methods, Pearlmutter 1994; Martens and Sutskever 2011 proposed matrix-free variants using the "forward over reverse" approach of automatic differentiation (Griewank and Naumann 2003) combined with a matrix-free linear solver such as Conjugate Gradient method (Hestenes and Stiefel 1952).

Despite these improvements, second-order optimization methods have not been adopted by practitioners. We believe that the problem is twofold: 1) the inadequacy of the local geometry implied by the chosen second-order solver, 2) the need for a layer-aware approach in improving computational efficiency.

This thesis addresses those two challenges jointly by proposing a framework in which a large class of steepest descent methods can be implemented efficiently with the same blueprints. The key insight in this work is to recognize that under certain assumptions, the problem of finding the direction of steepest descent is amenable to solving a local Linear Quadratic Regulation problem.

This provided us a direct entry point to try to address these issues more generally in the Optimal Control (OC) framework which, as in Reinforcement Learning (RL), is concerned with how to optimize sequential decision-making processes (Pontriagin *et al.* 1962; Bellman 1952). Our results offer a new perspective on the nature of the computation performed during an optimization step: the parameters update step of a whole family of steepest descent algorithms can itself be expressed as LQR problems.

The history of deep learning is intertwined with that of optimal control. Indeed, backpropagation was first described via the Optimal Control Problem (OCP) framework and studied under the Hamiltonian formalism (see Bryson and Ho 1969, Lecun 1988 and Linnainmaa 1976); this formalism being the backbone of OC. The specific case of the neural net optimization was also connected to optimal control theory explicitly by Dreyfus 1990, who applied the Kelley-Bryson procedure, a.k.a backpropagation, to optimize neural nets. More recently, Zhang *et al.* 2019 and Seidman *et al.* 2020 studied min-max optimization as an optimal control problem to improve training stability. Optimal control has also recently gained attention within the DL community for modeling continuous dynamical systems governed by Ordinary Differential Equations (ODEs). This line of work came from Chen *et al.* 2018 who developed Neural ODEs from the question: what if the discretization step in ResNets (He *et al.*, 2016) was made increasingly smaller? To efficiently optimize such models, the authors used the adjoint method from Bryson and Ho 1969 in optimal control to evaluate the derivatives. Using insights from optimal control, Haber and Ruthotto 2017 also investigated the stability of forward propagation in neural networks by studying a continuous-time counterpart that led them to propose new neural architectures. The work developed in this thesis extends those ideas by showing for the first time that each gradient step during training can be computed as the solution to a specific kind of optimal control problem.

1.1 Deep Learning Formulation in Optimal Control

We follow the path outlined above, and formulate some of the concepts encountered in ML in the language of OC. However, we currently restrict ourselves to the discrete setting. The training task of a NN can thus be posed as a Mayer problem (see Bliss 1968); that is, an optimal control problem (OCP) of the form:

$$\begin{aligned} &\min_{\boldsymbol{u}} \quad J(\boldsymbol{x}, \boldsymbol{u}) \\ &\text{subject to} \quad \boldsymbol{x}_{i+1} = f_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \\ &\text{given} \quad \boldsymbol{x}_0 \end{aligned} \tag{1.1.1}$$

Indeed, when training a NN, one would receive an input ($\boldsymbol{x}_0$) that will then be processed sequentially through all the layers f_i before evaluating the loss ($J(\boldsymbol{x}, \boldsymbol{u})$) over the final representation only.

Since we will be leveraging tools from the OC setting to perform optimization in ML, we decided to align our notation with as is conventional in OC. When dealing with the specific task of optimizing a NN in ML, the correspondence between the concepts are as follows:

ML terminology		Control terminology	
Name	Notation	Name	Notation
Input	$\boldsymbol{x}_0$	Initial state	$\boldsymbol{x}_0$
Feature/Activation	$\boldsymbol{a}_i$	Transition state	$\boldsymbol{x}_i$
Output	$\boldsymbol{y}$	Final state	$\boldsymbol{x}_N$
Parameters	$\boldsymbol{\theta}$	Controls	$\boldsymbol{u}$
Loss function	$l(\cdot)$	Cost function	$J(\cdot)$
Layer		Transition function	f_i

Table 1.1. Notations used in ML vs OC

1.2 Contributions

The work in this thesis was sparked by our attempt to leverage a connection shown in Bertsekas (2016) (section 2.6.2) between a particular LQR problem and Newton's descent step for an OCP. This particular implementation of Newton's method was first formalized simultaneously by Dunn and Bertsekas (1989) and independently by De O. Pantoja and Mayne (1989). We realized that this was only a specific case and that an equivalence could be built more broadly for the steepest descent minimization objective. A similar connection has also been found by Jin *et al.* (2020), but only in the context of an end-to-end differentiable architecture for reinforcement learning. Our results apply more broadly to unconstrained optimization in other areas of ML.

Thus the contributions of this thesis, to the best of our knowledge, include:

- We describe how a gradient descent step, a steepest descent step under a Euclidean norm, over a discrete dynamical system with terminal cost (which encompasses neural networks) is always equivalent to the solution of a LQR (section 3.1).
- We generalize this equivalence between LQR and an individual steepest descent step for a variety of divergence measures that encompasses second-order methods such as natural gradient descent and Newton's method (section 3.2).

- As a specific case we derive the LQR equivalent to performing a natural gradient descent update (section 3.2.3).
- We extend the above results to the infinite horizon setting, thus making the technique appropriate to train models such as deep equilibrium models. As a consequence, this gives us an algorithm to train such models with second-order methods, something that does not seem to have been attempted yet (chapter 4).
- We show that, as a special case, solving the LQR equivalent to gradient descent in the infinite horizon setting is equivalent to retrieving it via implicit differentiation (section 4.2).
- The matrix inversion involved with the general steepest procedure descent is usually computationally expensive. We show how we can sidestep this difficulty for NN that possesses ReLu activation functions by deriving a trick based on Neumann's series (section 5.1).

Chapter 2

Background

Steepest descent is a paradigm that encompasses a whole family of optimization algorithms, but it does not necessarily provide a unified method to solve the optimization problems formulated through it. We will attempt to cover some of this ground, by bringing a few OC results into the realm of DL. We assume most readers will be acquainted with one of the two fields, but not necessarily both. As such, we will give a very brief overview of the optimization trends in DL before going more technical and providing the details that will be leveraged from steepest descent. We will continue in this vein and explains the exploited concepts about LQR and the formulation associated with it in OC. Finally, we will succinctly introduce the deep implicit models' framework as we will be extending some of our results to this area.

2.1 Optimization Methods for Deep Learning

2.1.1 First-Order Methods

The optimization landscape in Deep Learning (DL) is largely dominated by first-order stochastic gradient descent methods. This simple approach is extremely efficient in DL, providing an unbiased estimate of the gradient and being able to escape local minima (see Bottou 1998 for more information about SGD). It also scales very well with the number of examples, it is thus suited to big data (see Robbins 2007). Yet, this method still has few disadvantages: it often requires manual tuning of hyperparameters such as the learning rate and may make very slow progress in a noisy setting. These issues are well discussed in Ruder (2016). Multiple variants were later used for training Deep Neural Nets (DNN) in order to mitigate the challenges faced by SGD.

Momentum (Polyak 1964) is a key idea to accelerate training; it accumulates past gradients and adds them (scaled by an exponentially decaying rate) to the current update. This technique is particularly efficient in noisy high curvature settings. It is also a core idea

when it comes to accelerating learning and has many derivatives. As an example, Nesterov acceleration (Nesterov 1983) extends this idea but evaluates the gradient after applying it. This strategy brings a better rate of convergence in deterministic settings and a practical speed up in stochastic scenarios, such as the training of DNN (see Sutskever *et al.* 2013).

Even with those improvements, setting the learning rate correctly remains a difficult hyperparameter optimization problem. Momentum somewhat improved the situation but did so by introducing a new hyperparameter, namely the mass parameter. Methods were thus designed to specifically address the problem, by introducing an adaptive learning rate tailored separately to each weight. AdaGrad (Duchi *et al.* 2011) did so by proportionally scaling a learning rate decay to the size of the partial derivatives. Weights that adapt more slowly will therefore decay less and the training procedure should make better progress in the directions for which the gradient slope is smoother. RMSProp (Hinton 2012) modified AdaGrad by discarding the earlier part of the updates history when calculating the decay rate, allowing for better performance in a non-convex setting.

Finally, when it comes to first-order methods, Adam (Kingma and Ba 2015) is probably the default choice for DL practitioners nowadays, as it is considered to be more robust to the choice of hyperparameters. It incorporates momentum as an estimate of the first-order moments of the gradients and includes a bias correction term to account for the initialization at the origin. The bias correction leverages an estimate of the second-order moments along with the estimate of the first-order moments.

2.1.2 Second-Order Methods

In contrast, second-order methods never saw wide applications in DL, suffering drastically from the curse of dimensionality as the algorithms implementing them traditionally require computation time and memory storage scaling with $\mathcal{O}(n^3)$. However, there is experimental evidence (Martens 2016) that they allow training convergence in fewer iterations, even if the time to do so may not be smaller than via first-order methods (for which an individual update is computed much more quickly). Martens 2016 provides a good overview of the difficulties as well as the advantages that come with those methods when applied to DL.

Because of those challenges, most early applications of second-order methods to Machine Learning (ML) were through the methods known as Quasi-Newton that replace the Hessian inverse by an approximation. The most popular among those is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method for which a limited memory variant (Liu and Nocedal 1989) can offer better performance than plain SGD in some instances (see Le

et al. 2011). However, their applicability to DL remains limited as they still do not scale well with the large number of parameters that a DNN usually possesses.

Recently, another second-order method has drawn some attention: the Natural Gradient Descent (NGD). This method leverages the Fischer Information matrix to condition the gradient update and was first introduced by Amari 1998. In the context of DL, empirical evidence was gathered about the robustness of the approach when facing limited training data by Pascanu and Bengio 2014.

The main difficulty regarding the widespread use of NGD remains its scalability since in theory it also requires to invert a matrix scaling with the number of parameters (the Fischer information matrix). A recent approach, K-FAC (Martens and Grosse 2015), alleviated part of the problem by first performing a block-diagonal approximation of the Fischer information matrix and then using Kronecker factorization to ease the inversion task. It was shown that this strategy could be scaled and adapted to distributed systems by Osawa *et al.* 2019, but the result requires more overheads than typical first-order methods.

2.2 Steepest Descent

Steepest descent can be seen as a generalization of gradient descent. It should be noted also that the *mirror descent* approach can be cast as the steepest direction in the Riemannian space induced by the chosen divergence (see Raskutti and Mukherjee 2015; Beck and Teboulle 2003). The core idea behind those methods is that two quantities need to be determined in order to optimize a function locally. Thus, one has to decide on a direction in which to search to improve the objective as well as the magnitude of the step to take in the chosen direction.

If we first fix the step size, what is left to find is the optimal direction in which to take that step. For a given direction $\boldsymbol{v}$, the first-order Taylor's approximation (see Nocedal and Wright, 2006) yields:

$$f(\boldsymbol{x}+\boldsymbol{v}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \boldsymbol{v}$$

The *steepest direction* of norm 1 will therefore be the vector $\boldsymbol{v}$ that minimizes:

$$\begin{aligned} \boldsymbol{v}^* &= \arg\min_{\boldsymbol{v}} f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \boldsymbol{v} \quad \text{s.t. } ||\boldsymbol{v}|| = 1 \\ &= \arg\min_{\boldsymbol{v}} \nabla f(\boldsymbol{x})^T \boldsymbol{v} \quad \text{s.t. } ||\boldsymbol{v}|| = 1 \end{aligned} \tag{2.2.1}$$

Already, we can see that the steepest descent algorithm can come in many flavors. Indeed depending on the norm choice, we can obtain very different updates. Although this problem might seem difficult to solve because of the constraint requiring $||\boldsymbol{v}|| = 1$, we can relax the problem so that we recover the same direction, but rescaled. That is, there exists a constant α such that (2.2.1) is equivalent to:

$$\boldsymbol{v}^* = \alpha\Big[\arg\min_{\boldsymbol{v}}\{\nabla f(\boldsymbol{x})^T\boldsymbol{v} + \frac{1}{2}\|\boldsymbol{v}\|^2\}\Big] \tag{2.2.2}$$

which is a much more convenient form. The details for this equivalence are provided in Appendix A and a more formal introduction to steepest descent can be found in section 9.4 of Boyd and Vandenberghe 2004.

2.2.1 Gradient Descent

Gradient descent is an instance of steepest descent, and we recover this specific algorithm by choosing the Euclidean norm in the steepest descent formulation. Indeed, in this case, we want to minimize:

$$\alpha\Big[\arg\min_{\boldsymbol{v}}\{\nabla f(\boldsymbol{x})^T\boldsymbol{v} + \frac{1}{2}\boldsymbol{v}^T\boldsymbol{v}\Big]$$

This is a simple quadratic so that we can recover the minimum by setting the derivative to zero:

$$0 = \alpha\nabla f(\boldsymbol{x}) + \boldsymbol{v}^* \implies \boldsymbol{v}^* = -\alpha\nabla f(\boldsymbol{x}) \ ,$$

where such a $\boldsymbol{v}^*$ is exactly the update we want to apply when we perform gradient descent.

2.2.2 Steepest Descent under a Quadratic Norm

Let us now consider the quadratic norm $\|\cdot\|_P$ defined as $\|\boldsymbol{z}\|_P = (\boldsymbol{z}^T\boldsymbol{P}\boldsymbol{z})^{1/2}$ for a matrix $\boldsymbol{P}$ that is positive semidefinite.

Then, just as above, we can seek the minimum by setting the derivative to zero, which yields:

$$\boldsymbol{v}^* = -\alpha\boldsymbol{P}^{-1}\nabla f(\boldsymbol{x})$$

With this formulation, we can therefore also cast NGD and Newton's descent as special instances of steepest descent, where $\boldsymbol{P}$ is chosen to be the Fisher matrix, $\boldsymbol{F}$, for the former or the Hessian $\boldsymbol{H}$ for the latter.

2.3 Linear Quadratic Regulator

Linear quadratic regulators are the canonical problems of control theory. Although far from being a simple problem, LQRs are well-studied problems that have the nice property of being analytically solvable. Moreover, LQRs have many practical applications and thus many numerical solvers have been developed for this family of problems over the years. Readers interested in a more thorough introduction can refer themselves to Anderson and Moore 1990.

LQR are control problems for which the dynamics are described by a linear system. In control theory, $\boldsymbol{x}_t \in \mathbb{R}^n$ is often used to denote the *state* of the system at time t while $\boldsymbol{u}_t \in \mathbb{R}^m$ denotes the *control* taken at time t. The control is usually the quantity that one can vary to obtain the desired trajectory. Under this notation, the system dynamics can be described by:

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t \tag{2.3.1}$$

The goal is then to minimize a cost associated with the trajectory obtained by following the system dynamics. For LQRs problems, this cost must be at most quadratic with respect to its states and controls and of the following form:

$$J(\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{a}_T^T \boldsymbol{x}_T + \boldsymbol{x}_T^T \boldsymbol{Q}_T \boldsymbol{x}_T + \sum_{i=0}^{T-1} \left[\boldsymbol{a}_i^T \boldsymbol{x}_i + \boldsymbol{x}_i^T \boldsymbol{Q}_i \boldsymbol{x}_i + \boldsymbol{b}_i^T \boldsymbol{u}_i + \boldsymbol{u}_i^T \boldsymbol{R}_i \boldsymbol{u}_i + \boldsymbol{u}_i^T \boldsymbol{M}_i \boldsymbol{x}_i \right] \tag{2.3.2}$$

Note that objective function is the sum of per-state costs, and the immediate cost at a given time step depends solely on the current state and control. That is, there are no cross-terms in the cost between parameters of different time steps.

For such problems, the optimal values for the controls can be retrieved via the Riccati equation. To do so, we have to solve backward for the gain matrix that will define the optimal controls. The Riccati equation is provided below for the two cases that we encounter when building the equivalence between a steepest descent step and a LQR.

Though LQR problems are well defined both for continuous and discrete settings, we will focus on the latter case. This is because DL models currently consist mostly of discrete optimization problems; as seen in section 1.1 the passage from one layer to another can be seen as moving forward by one time step. The connection made later between an optimization step and an LQR will be grounded in the discrete setting.

2.3.1 Finite Horizon

In OC, a finite horizon means that the dynamic system reaches a final state after N steps. For us here, the final state would be the final output given by the last layer of a NN.

LQR problems that are well formulated will fit the form:

$$\begin{aligned} &\min_{\boldsymbol{u}} \quad J(\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{a}_T^T\boldsymbol{x}_T + \boldsymbol{x}_T^T\boldsymbol{Q}_T\boldsymbol{x}_T + \sum_{i=0}^{T-1}\left[\boldsymbol{a}_i^T\boldsymbol{x}_i + \boldsymbol{x}_i^T\boldsymbol{Q}_i\boldsymbol{x}_i + \boldsymbol{b}_i^T\boldsymbol{u}_i + \boldsymbol{u}_i^T\boldsymbol{R}_i\boldsymbol{u}_i + \boldsymbol{u}_i^T\boldsymbol{M}_i\boldsymbol{x}_i\right] \\ &\text{subject to} \quad \boldsymbol{x}_{i+1} = \boldsymbol{A}_i\boldsymbol{x}_i + \boldsymbol{B}_i\boldsymbol{u}_i \\ &\text{given} \quad \boldsymbol{x}_0 \end{aligned} \tag{2.3.3}$$

To solve such problems, we can leverage the Riccati equation to first retrieve the gain matrices $\boldsymbol{K}_i$ and the vector $\boldsymbol{\lambda}_i$ (see Reid, 1972).

$$\boldsymbol{K}_i = \boldsymbol{A}_i^T\boldsymbol{K}_{i+1}\boldsymbol{A}_i + \boldsymbol{Q}_i - (\boldsymbol{A}_i^T\boldsymbol{K}_{i+1}\boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T\boldsymbol{K}_{i+1}\boldsymbol{B}_i)^{-1}(\boldsymbol{M}_i + \boldsymbol{B}_i^T\boldsymbol{K}_{i+1}\boldsymbol{A}_i) \tag{2.3.4}$$

$$\boldsymbol{\lambda}_i = \boldsymbol{a}_i + \boldsymbol{A}_i^T\boldsymbol{\lambda}_{i+1} - (\boldsymbol{A}_i^T\boldsymbol{K}_{i+1}\boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T\boldsymbol{K}_{i+1}\boldsymbol{B}_i)^{-1}(\boldsymbol{b}_i + \boldsymbol{B}_i^T\boldsymbol{\lambda}_{i+1}) \tag{2.3.5}$$

With $\boldsymbol{K}_N = \boldsymbol{Q}_N$ and $\boldsymbol{\lambda}_N = \boldsymbol{a}_N$.
The optimal controls are then calculated with:

$$\begin{aligned} \boldsymbol{u}_0^* &= -(\boldsymbol{R}_0 + \boldsymbol{B}_0^T\boldsymbol{K}_1\boldsymbol{B}_0)^{-1}(\boldsymbol{b}_0 + \boldsymbol{B}_0^T\boldsymbol{\lambda}_1) \\ \boldsymbol{x}_0^* &= \boldsymbol{x}_0 \\ \boldsymbol{x}_{i+1}^* &= \boldsymbol{A}_i\boldsymbol{x}_i^* + \boldsymbol{B}_i\boldsymbol{u}_i^* \\ \boldsymbol{u}_i^* &= -(\boldsymbol{R}_i + \boldsymbol{B}_i^T\boldsymbol{K}_{i+1}\boldsymbol{B}_i)^{-1}\left((\boldsymbol{M}_i + \boldsymbol{B}_i^T\boldsymbol{K}_{i+1}\boldsymbol{A}_i)\boldsymbol{x}_i^* + \boldsymbol{b}_i + \boldsymbol{B}_i^T\boldsymbol{\lambda}_{i+1}\right) \end{aligned} \tag{2.3.6}$$

Since a full development of those equations that encompasses the case when $\boldsymbol{a}_i \neq 0$ and $\boldsymbol{b}_i \neq 0$ is lengthy, we provide a proof in Appendix B.

2.3.2 Infinite Horizon

The infinite horizon setting deals with the case when $N \to \infty$, with both the cost terms $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{Q}, \boldsymbol{M}, \boldsymbol{R})$ and the transition matrices $(\boldsymbol{A}, \boldsymbol{B})$ remaining constant through time. In

such cases, it makes no sense to have a cost associated with the final state, and the problem formulation becomes:

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & J(\boldsymbol{x},\boldsymbol{u}) = \sum_{i=0}^{\infty} \left[\boldsymbol{a}^T \boldsymbol{x}_i + \boldsymbol{x}_i^T \boldsymbol{Q} \boldsymbol{x}_i + \boldsymbol{b}^T \boldsymbol{u}_i + \boldsymbol{u}_i^T \boldsymbol{R} \boldsymbol{u}_i + \boldsymbol{u}_i^T \boldsymbol{M} \boldsymbol{x}_i \right] \\ \text{subject to} \quad & \boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i \\ \text{given} \quad & \boldsymbol{x}_0 \end{aligned} \tag{2.3.7}$$

Again, we can use the associated Riccati equation to obtain the gain matrix $\boldsymbol{K}$ and $\boldsymbol{\lambda}$.

$$\boldsymbol{K} = \boldsymbol{A}^T \boldsymbol{K} \boldsymbol{A} + \boldsymbol{Q} - (\boldsymbol{A}^T \boldsymbol{K} \boldsymbol{B} + \boldsymbol{M}^T)(\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B})^{-1} (\boldsymbol{M} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{A}) \tag{2.3.8}$$

$$\boldsymbol{\lambda} = \boldsymbol{a} + \boldsymbol{A}^T \boldsymbol{\lambda} - (\boldsymbol{A}^T \boldsymbol{K} \boldsymbol{B} + \boldsymbol{M}^T)(\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B})^{-1} (\boldsymbol{b} + \boldsymbol{B}^T \boldsymbol{\lambda}) \tag{2.3.9}$$

The equation (2.3.8) is known as the Discrete Algebraic Riccati Equation (DARE), and its solution has many practical uses. Many efficient numerical DARE solvers already exist, and it is still an active area of research (see Bini *et al.* 2011).

A solution exists to (2.3.8) if the *closed loop state transfer matrix*, $\boldsymbol{A} - \boldsymbol{B}(\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B})^{-1}(\boldsymbol{M} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{A})$, has a spectral radius smaller than one (it is stable). The optimal trajectory can then be retrieved in the same fashion:

$$\begin{aligned} \boldsymbol{u}_0^* &= -(\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B})^{-1} (\boldsymbol{b} + \boldsymbol{B}^T \boldsymbol{\lambda}) \\ \boldsymbol{x}_0^* &= \boldsymbol{x}_0 \\ \boldsymbol{x}_{i+1}^* &= \boldsymbol{A}\boldsymbol{x}_i^* + \boldsymbol{B}\boldsymbol{u}_i^* \\ \boldsymbol{u}_i^* &= -(\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B})^{-1} \left((\boldsymbol{M} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{A}) \boldsymbol{x}_i^* + \boldsymbol{b} + \boldsymbol{B}^T \boldsymbol{\lambda} \right) \end{aligned} \tag{2.3.10}$$

Note that the optimal controls are guaranteed to stabilize when the closed-loop state transfer matrix is stable.

2.3.3 Cyclic Case

Finally, as it will be relevant later, we mention that a *cyclic* case exists for LQR in the infinite horizon setting. If a given LQR cycles through N different linear system, it takes the form:

$$
\begin{aligned}
&\min_{\boldsymbol{u}} \quad J(\boldsymbol{x},\boldsymbol{u}) = \sum_{i=0}^{\infty}\sum_{n=1}^{N}\left[\boldsymbol{a}_n^T\boldsymbol{x}_{in} + \boldsymbol{x}_{in}^T\boldsymbol{Q}_n\boldsymbol{x}_{in} + \boldsymbol{b}_n^T\boldsymbol{u}_{in} + \boldsymbol{u}_{in}^T\boldsymbol{R}_n\boldsymbol{u}_{in} + \boldsymbol{u}_{in}^T\boldsymbol{M}_n\boldsymbol{x}_{in}\right] \\
&\text{subject to} \;\; \boldsymbol{x}_{i(n+1)} = \boldsymbol{A}_n\boldsymbol{x}_{in} + \boldsymbol{B}_n\boldsymbol{u}_{in} \\
&\text{with} \;\; \boldsymbol{x}_{iN} = \boldsymbol{x}_{(i+1)0} \\
&\text{given} \;\; \boldsymbol{x}_{00}
\end{aligned}
\tag{2.3.11}
$$

The Riccati solution is similar, the difference being that we now need to solve a system of gain matrices.

$$
\begin{aligned}
\boldsymbol{K}_{N-1} &= \boldsymbol{A}_{N-1}^T\boldsymbol{K}_N\boldsymbol{A}_{N-1} + \boldsymbol{Q}_{N-1} - \\
&\qquad (\boldsymbol{A}_{N-1}^T\boldsymbol{K}_N\boldsymbol{B}_{N-1} + \boldsymbol{M}_{N-1}^T)(\boldsymbol{R}_{N-1} + \boldsymbol{B}_{N-1}^T\boldsymbol{K}_N\boldsymbol{B}_{N-1})^{-1}(\boldsymbol{M}_{N-1} + \boldsymbol{B}_{N-1}^T\boldsymbol{K}_N\boldsymbol{A}_{N-1}) \\
&\;\;\vdots \\
\boldsymbol{K}_1 &= \boldsymbol{A}_1^T\boldsymbol{K}_2\boldsymbol{A}_1 + \boldsymbol{Q}_1 - (\boldsymbol{A}_1^T\boldsymbol{K}_2\boldsymbol{B}_1 + \boldsymbol{M}_1^T)(\boldsymbol{R}_1 + \boldsymbol{B}_1^T\boldsymbol{K}_2\boldsymbol{B}_1)^{-1}(\boldsymbol{M}_1 + \boldsymbol{B}_1^T\boldsymbol{K}_2\boldsymbol{A}_1) \\
\boldsymbol{K}_N &= \boldsymbol{A}_N^T\boldsymbol{K}_1\boldsymbol{A}_N + \boldsymbol{Q}_N - (\boldsymbol{A}_N^T\boldsymbol{K}_1\boldsymbol{B}_N + \boldsymbol{M}_N^T)(\boldsymbol{R}_N + \boldsymbol{B}_N^T\boldsymbol{K}_1\boldsymbol{B}_N)^{-1}(\boldsymbol{M}_N + \boldsymbol{B}_N^T\boldsymbol{K}_1\boldsymbol{A}_N)
\end{aligned}
\tag{2.3.12}
$$

Likewise, we need to solve the cyclic system for $\boldsymbol{\lambda}_i$.

The good news for us is that, again, there are multiple cyclic DARE solvers already available to solve such problems.

2.4 Deep Implicit Models

Implicit models were recently brought into light within the deep learning community first by its use in modeling Ordinary Differential Equations (ODEs) system Chen *et al.* 2018 and then the introduction of the Deep Equilibrium Models (DEQ) Bai *et al.* 2019 that leveraged the same ideas over modern architectures. Let us remark that those papers made popular the uses of implicit layers within the ML community, but the core ideas behind date back to the '80s (e.g. Almeida 1990 and Pineda 1987).

The key point to the implicit layers is that instead of representing a typical transformation of the form :

$$
\boldsymbol{z} = f(\boldsymbol{x}) \tag{2.4.1}
$$

it instead focus on its *implicit* representation of the form :

$$g(\boldsymbol{z},\boldsymbol{x}) = f(\boldsymbol{x}) - \boldsymbol{z} = 0 \tag{2.4.2}$$

and the goal is now to find the root of the above equation.

This formulation becomes particularly useful when the transformation f has a dependency on $\boldsymbol{z}$ ($f : \mathcal{X} \times \mathcal{Z} \to \mathcal{Z}$) and is applied repeatedly until convergence to a fixed point, that we can denote $\boldsymbol{z}^*$.

The root-finding problem is then:

$$g(\boldsymbol{z},\boldsymbol{x}) = f(\boldsymbol{x},\boldsymbol{z}) - \boldsymbol{z} = 0 \tag{2.4.3}$$

With a solution at the fixed point $\boldsymbol{z}^* = f(\boldsymbol{x},\boldsymbol{z}^*)$. The fixed point has an implicit dependency over $\boldsymbol{x}$, and we can design the function $\varphi : \mathcal{X} \to \mathcal{Z}$ to be the function returning the fixed point of $f(\boldsymbol{x},\boldsymbol{z})$, i.e. $\varphi(\boldsymbol{x}) = \boldsymbol{z}^*$.

For such problems, when a fixed point exists, we can then leverage the Implicit Function Theorem to find the derivative to (2.4.3) around the fixed point $\boldsymbol{z}^*$, given by:

$$\nabla_{\boldsymbol{x}}^T \varphi(\boldsymbol{x}) = \left(\nabla_{\boldsymbol{z}}^T g(\boldsymbol{x}, \boldsymbol{z}) \Big|_{z=z^*} \right)^{-1} \nabla_{\boldsymbol{x}}^T g(\boldsymbol{x},\boldsymbol{z}^*) \tag{2.4.4}$$

Under the condition that the matrix $\left(\nabla_{\boldsymbol{z}}^T g(\boldsymbol{x}, \boldsymbol{z}) \Big|_{z=z^*} \right)^{-1}$ is invertible.

This relationship has practical consequences: the Jacobian of the implicit layer f can be recovered *without* having to backpropagate through the fixed-point solver/iterator.

This means that we are free to use any fixed point solver to find $\boldsymbol{z}^*$ and then recover analytically the Jacobian of interest via (2.4.4): without representing explicitely the computational graph (as build by typical automatic differentiation tools) of the fixed-point solver.

2.4.1 Deep Equilibrium Models

DEQ models deploy the above ideas in the realm of deep learning. Indeed, conceptually, one could think of a DEQ as a recurrent neural network of infinite depth. If the representation induced by the infinitely repeated layer eventually reaches a fixed point, such a NN could be written as:

$$
\begin{aligned}
\boldsymbol{z}_0 &= 0 \\
\boldsymbol{z}_{i+1} &= f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z}_i) \quad \text{until } \boldsymbol{z}_{i=1} = \boldsymbol{z}_i \\
J(\boldsymbol{x}) &= l(\boldsymbol{z}^*(\boldsymbol{x}, \boldsymbol{\theta}))
\end{aligned} \tag{2.4.5}
$$

Where $\boldsymbol{z}^*$ is the fixed point of the layer $f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z})$, $\boldsymbol{x}$ is the input, $\boldsymbol{\theta}$ are the layer parameters and $l(\cdot)$ is the loss calculated with respect to the NN output.

In practice, we often want to optimize such a network with a variant of the SGD algorithm. In all cases, the optimization step will most likely require the computation of the quantity:

$$\nabla_{\boldsymbol{\theta}} \varphi(\boldsymbol{x}, \boldsymbol{\theta}) \nabla_{\boldsymbol{z}^*} l(\boldsymbol{z}^*) = \nabla_{\boldsymbol{\theta}} l(\boldsymbol{z}^*)$$

That is, the vector-Jacobian product between the Jacobian of the fixed point with respect to the parameters and the gradient of the loss with respect to the final representation. To recover such a quantity, we can leverage the Implicit Function Theorem showcased above that tells us:

$$\nabla_{\boldsymbol{\theta}}^T \varphi(\boldsymbol{x}, \boldsymbol{\theta}) = \left(\nabla_{\boldsymbol{z}}^T f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z}) \Big|_{\boldsymbol{z}=\boldsymbol{z}^*} - \boldsymbol{I} \right)^{-1} \nabla_{\boldsymbol{\theta}}^T f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z}^*) \tag{2.4.6}$$

For any vector $\boldsymbol{b}$ (such as $\nabla_{\boldsymbol{z}^*} l(\boldsymbol{z}^*)$), we have that:

$$\nabla_{\boldsymbol{\theta}} \varphi(\boldsymbol{x}, \boldsymbol{\theta}) \boldsymbol{b} = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z}^*) \left(\nabla_{\boldsymbol{z}}^T f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z}) \Big|_{\boldsymbol{z}=\boldsymbol{z}^*} - \boldsymbol{I} \right)^{-T} \boldsymbol{b} \tag{2.4.7}$$

Finally, to be compatible with the automatic differentiation framework, one will want to retrieve the quantity $\left(\nabla_{\boldsymbol{z}}^T f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z}) \Big|_{\boldsymbol{z}=\boldsymbol{z}^*} - \boldsymbol{I} \right)^{-T} \boldsymbol{b}$ by solving instead the system:

$$\bar{\boldsymbol{x}} = \left(\nabla_{\boldsymbol{z}}^T f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z}) \Big|_{\boldsymbol{z}=\boldsymbol{z}^*} \right) \bar{\boldsymbol{x}} - \boldsymbol{b} \tag{2.4.8}$$

The system above also describes a fixed-point equation, and one key concept of the DEQ networks is that we have access to a fixed-point solver. We thus have all the tools needed to derive the gradient update with automatic differentiation. Moreover, if a fixed point exists for the implicit layer of the DEQ (2.4.5), then we are guaranteed that a fixed point also exists for (2.4.8). Indeed, the local convergence condition for both system is the same : that $\left(\nabla_{\boldsymbol{z}}^T f(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{z}) \Big|_{\boldsymbol{z}=\boldsymbol{z}^*} \right)$ has a spectral radius smaller than 1 (and is thus a contraction,

see Nayfeh and Balachandran 1995 section 2.2).

We now have in hand everything needed to derive the equivalences proposed in the introduction.

Chapter 3

Steepest Descent as Linear Quadratic Regulation

We begin by showing that performing a single GD step with steepest descent is equivalent to solving a corresponding unique LQR. From there, we will be able to move toward our main result, which extends this equivalence to steepest descent under various divergence measures. Please note that the results presented in this chapter are developed for finite horizon, discrete OCP – a setting that holds when trying to optimize a traditional NN made from a finite number of layers.

3.1 Problem Statement for a Single Gradient Step

For consistency, we will now denote the various quantities via their typical OC denomination, following the equivalences put forward in table 1.1. Let $F : \mathbb{R}^n \to \mathbb{R}^d$ be a NN mapping the input $\boldsymbol{x}_0^k$ to the output $\boldsymbol{x}_N^k$ through the applications of $N-1$ layers labelled $f_0, f_1, ..., f_{N-1}$. Each layers has its own parameters $\boldsymbol{u}_i$ so that $\boldsymbol{x}_{i+1}^k = f(\boldsymbol{x}_i^k, \boldsymbol{u}_i)$. The whole set of parameters can be written as $\boldsymbol{u} = \{\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{N-1}\}$. A loss function of the form $J(\boldsymbol{x}_N^k)$ is defined, with an implicit dependency over the parameters $\boldsymbol{u}$, so that we can in fact label the loss as $J(\boldsymbol{u})$.

Then, posed in the framework of steepest descent, performing a single GD step for this NN over the parameters $\boldsymbol{u}$ can be seen as a Bolza problem (Bliss 1968) that minimizes the following objective for the current input $\boldsymbol{x}_0^k$ and current parameters $\boldsymbol{u}^k$:

$$\min_{\Delta \boldsymbol{u}} \nabla J(\boldsymbol{u}^k)^T(\Delta \boldsymbol{u}) + \frac{1}{2}(\Delta \boldsymbol{u})^T(\Delta \boldsymbol{u}) = \min_{\boldsymbol{u}} \nabla J(\boldsymbol{u}^k)^T(\boldsymbol{u}^k - \boldsymbol{u}) + \frac{1}{2}\sum_{i=0}^{N-1}(\boldsymbol{u}_i^k - \boldsymbol{u}_i)^T(\boldsymbol{u}_i^k - \boldsymbol{u}_i) \tag{3.1.1}$$

The above problem is quadratic and thus convex, a unique optimal solution exists given by $\Delta \boldsymbol{u}^*$ (or $\boldsymbol{u}^*$). The GD update to the parameters for the current input is then given by:

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \Delta\boldsymbol{u}^* = \boldsymbol{u}^*$$

We will now show that (3.1.1) is in fact a proper LQR problem. Already, we can notice that a quadratic cost is present; what is lacking is a linear dynamic system describing the system evolution. But, we will see that the linearization arise naturally from the differentiation, that is from the term $\nabla J(\boldsymbol{u}^k)^T(\Delta\boldsymbol{u})$ in (3.1.1).

Intuitively, this property can be understood easily from the multivariate chain rule. Indeed, if we think of a given system with $\boldsymbol{x}_1 = \phi(\boldsymbol{u})$ and $\boldsymbol{x}_2 = g(\boldsymbol{x}_1,\boldsymbol{u})$, we have :

$$\begin{aligned}
\mathcal{D}_{\boldsymbol{u}}(f(\boldsymbol{x}_2,\boldsymbol{u}) \circ g)(\boldsymbol{x},\boldsymbol{u}) &= \mathcal{D}_{\boldsymbol{x}_2} f \mathcal{D}_{\boldsymbol{u}} g + \mathcal{D} f_{\boldsymbol{x}_2} \mathcal{D}_{\boldsymbol{x}_1} g \mathcal{D}_{\boldsymbol{u}} \phi + \mathcal{D}_{\boldsymbol{u}} f \\
&= \mathcal{D}_{\boldsymbol{x}_2} f \Big(\mathcal{D}_{\boldsymbol{x}_1} g \mathcal{D}_{\boldsymbol{u}} \phi + \mathcal{D}_{\boldsymbol{u}} g \Big) + \mathcal{D}_{\boldsymbol{u}} f \\
&= \boldsymbol{A}_2(\boldsymbol{A}_1 \cdot \boldsymbol{C} + \boldsymbol{B}_1) + \boldsymbol{B}_2
\end{aligned}$$

That is, differentiation and linearization are closely related. Moreover, the above can be seen as a discrete-time linear dynamical system.

More formally, by applying the chain rule to the first term in (3.1.1) and denoting $\nabla^T J(\boldsymbol{u}^k)$ by $\nabla^T J^k$ (and $f_i(\boldsymbol{x}_i^k, \boldsymbol{u}_i^k)$ by f_i^k) we have that:

$$\begin{aligned}
\nabla_{\boldsymbol{u}}^T J^k(\boldsymbol{u} - \boldsymbol{u}^k) &= \nabla_{\boldsymbol{x}_N}^T J^k(\nabla_{\boldsymbol{x}_{N-1}}^T f_{N-1}^k \nabla_{\boldsymbol{u}_{0:N-2}}^T f_{N-1}^k + \nabla_{\boldsymbol{u}_{N-1}}^T f_{N-1}^k)(\boldsymbol{u} - \boldsymbol{u}^k) \\
&= \nabla_{\boldsymbol{x}_N}^T J^k(\nabla_{\boldsymbol{x}_{N-1}}^T f_{N-1}^k(\nabla_{\boldsymbol{x}_{N-2}}^T f_{N-2}^k \nabla_{\boldsymbol{u}_{0:N-3}}^T f_{N-3}^k + \nabla_{\boldsymbol{u}_{N-2}}^T f_{N-2}^k) + \nabla_{\boldsymbol{u}_{N-1}}^T f_{N-1}^k)(\boldsymbol{u} - \boldsymbol{u}^k) \\
&= \vdots \\
&= \nabla_{\boldsymbol{x}_N}^T J^k\Big(...\nabla_{\boldsymbol{x}_1}^T f_1^k(\nabla_{\boldsymbol{x}_0}^T f_0^k \nabla_{\boldsymbol{u}_0}^T f_0^k + \nabla_{\boldsymbol{u}_0}^T f_0^k) + \nabla_{\boldsymbol{u}_1}^T f_1^k ...\Big)(\boldsymbol{u} - \boldsymbol{u}^k)
\end{aligned} \tag{3.1.2}$$

We chose to denote the Jacobians via the matrix of gradients, $\nabla_{\boldsymbol{v}_i} f_i$, for which each columns are the gradients of the elements of the vector function f_i. As such, we have that the Jacobian of f_i is the transpose of this matrix, i.e. $\frac{\partial f_i(\boldsymbol{v}_i)}{\partial \boldsymbol{v}_i} = \nabla_{\boldsymbol{v}_i}^T f_i$. Moreover, the notation $\nabla_{\boldsymbol{u}_{0:i}}^T$ is used to indicate that the derivative is taken with respect to the parameters $\boldsymbol{u}_0$ up to $\boldsymbol{u}_i$.

Now we note that $\boldsymbol{x}_0$ is constant, so $\nabla^T_{\boldsymbol{x}_0} f_1^k \nabla^T_{\boldsymbol{u}} f_0 = 0$ and also that we can distribute $(\boldsymbol{u} - \boldsymbol{u}^k)$ over the sum, taking into account that f_i has only partial derivatives over $\boldsymbol{u}_i$ and $\boldsymbol{x}_i$:

$$\nabla^T_{\boldsymbol{u}} J^k(\boldsymbol{u} - \boldsymbol{u}^k) = \nabla^T_{\boldsymbol{x}_N} J^k\Big(...\nabla^T_{\boldsymbol{x}_2} f_2^k\Big(\nabla^T_{\boldsymbol{x}_1} f_1^k(\nabla^T_{\boldsymbol{u}_0} f_0^k)(\boldsymbol{u}_0 - \boldsymbol{u}_0^k) + \nabla^T_{\boldsymbol{u}_1} f_1^k(\boldsymbol{u}_1 - \boldsymbol{u}_1^k)\Big) + \nabla^T_{\boldsymbol{u}_2} f_2^k(\boldsymbol{u}_2 - \boldsymbol{u}_2^k)...\Big) \tag{3.1.3}$$

Finally, by adopting the notation $\boldsymbol{A}_i = \nabla^T_{\boldsymbol{x}_i} f_i^k$, $\boldsymbol{B}_i = \nabla^T_{\boldsymbol{u}_i} f_i^k$; $\tilde{\boldsymbol{u}}_i = \boldsymbol{u}_i - \boldsymbol{u}_i^k$ and $\tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i \tilde{\boldsymbol{x}}_i + \boldsymbol{B}_i \tilde{\boldsymbol{u}}_i$ we get:

$$\begin{aligned} \nabla^T_{\boldsymbol{u}} J^k(\boldsymbol{u} - \boldsymbol{u}^k) &= \nabla^T_{\boldsymbol{x}_N} J^k\Big(...\boldsymbol{A}_2\Big(\boldsymbol{A}_1(\boldsymbol{B}_0 \tilde{\boldsymbol{u}_0}) + \boldsymbol{B}_1 \tilde{\boldsymbol{u}_1}\Big) + \boldsymbol{B}_2 \tilde{\boldsymbol{u}_2}...\Big) \\ &= \nabla^T_{\boldsymbol{x}_N} J^k \cdot \tilde{\boldsymbol{x}}_N \end{aligned} \tag{3.1.4}$$

As it will be useful later, note that if we denote explicitly the implicit dependence of $\boldsymbol{x}$ over $\boldsymbol{u}$ via $\boldsymbol{x} = \phi(\boldsymbol{u})$ (so that $J(\boldsymbol{x},\boldsymbol{u}) = J(\phi(\boldsymbol{u}),\boldsymbol{u})$), then (3.1.4) shows that:

$$\nabla^T_{\boldsymbol{u}} \phi(\boldsymbol{u}) \cdot (\boldsymbol{u} - \boldsymbol{u}^k) = \tilde{\boldsymbol{x}}_N \tag{3.1.5}$$

Since the cost J has only a direct dependency over the variable $\boldsymbol{x}_N$ and $\nabla^T_{\boldsymbol{u}} J^k(\boldsymbol{u} - \boldsymbol{u}^k) = \nabla^T_{\boldsymbol{x}_N} J^k\Big(\nabla^T_{\boldsymbol{u}} \phi(\boldsymbol{u}) \cdot (\boldsymbol{u} - \boldsymbol{u}^k)\Big)$.

3.1.1 Equivalence with Gradient Descent

What we have in (3.1.4) is a discrete linear dynamical system. Indeed the transition layers of this system are obtained via the linearization of the original layers in the model. Thus, (3.1.1) can be rewritten as a standard OCP, allowing us to put forward a preliminary result of this thesis, obtained by construction from the above problem statement:

Theorem 3.1.1. *Retrieving the update for a single gradient descent step over the parameters $\boldsymbol{u}^k$ for a given example $\boldsymbol{x}^k$ in a neural network is equivalent to solving the following linear quadratic regulator problem:*

$$\begin{aligned} &\min_{\tilde{\boldsymbol{u}}} \;\; \nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_N + \frac{1}{2} \sum_{i=1}^{N-1} \tilde{\boldsymbol{u}}_i^T \tilde{\boldsymbol{u}}_i \\ &\text{subject to } \; \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i \tilde{\boldsymbol{x}}_i + \boldsymbol{B}_i \tilde{\boldsymbol{u}}_i \\ &\text{where } \; \boldsymbol{A}_i = \nabla^T_{\boldsymbol{x}_i} f_i^k, \, \boldsymbol{B}_i = \nabla^T_{\boldsymbol{u}_i} f_i^k \\ &\text{given } \; \tilde{\boldsymbol{x}}_0 = 0 \end{aligned} \tag{3.1.6}$$

Based on the material presented in section 2.3, we can see that (3.1.6) *is a proper LQR problem: the cost to minimize is quadratic and the transition functions are linear. Therefore, the optimal solution,* $\tilde{\boldsymbol{u}}^*$ *to* (3.1.6) *can be obtained by solving the Riccati equation* (2.3.4)*, starting with* $\boldsymbol{K}_N = 0$ *and* $\boldsymbol{\lambda}_N = \nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k)$

$$\begin{aligned} \boldsymbol{K}_i &= \boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)(\boldsymbol{I} + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} (\boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \\ \boldsymbol{\lambda}_i &= \boldsymbol{A}_i^T \boldsymbol{\lambda}_{i+1} - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)(\boldsymbol{I} + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1} \end{aligned} \tag{3.1.7}$$

The equivalent GD update is then given by:

$$\begin{aligned} \tilde{\boldsymbol{u}}_0^* &= -(\boldsymbol{I} + \boldsymbol{B}_0^T \boldsymbol{K}_1 \boldsymbol{B}_0)^{-1} \boldsymbol{B}_0^T \boldsymbol{\lambda}_1 \\ \tilde{\boldsymbol{x}}_{i+1}^* &= \boldsymbol{A}_i \tilde{\boldsymbol{x}}_i^* + \boldsymbol{B}_i \tilde{\boldsymbol{u}}_i^* \\ \tilde{\boldsymbol{u}}_i^* &= -(\boldsymbol{I} + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} \left((\boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \tilde{\boldsymbol{x}}_i^* + \boldsymbol{B}_i^T \lambda_{i+1} \right) \end{aligned} \tag{3.1.8}$$

With a learning rate α*, the GD update can therefore be applied via:*

$$\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha \tilde{\boldsymbol{u}}^* \tag{3.1.9}$$

Note that the initial condition $\tilde{\boldsymbol{x}}_0 = 0$ arises from $\nabla^T_{\boldsymbol{x}_0} f_1^k \nabla^T_{\boldsymbol{u}} f_0 = 0$; defining the subsequent $\tilde{\boldsymbol{x}}_i$ by $\tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i \tilde{\boldsymbol{x}}_i + \boldsymbol{B}_i \tilde{\boldsymbol{u}}_i$ is then a convenient choice to describe the state transitions happening in the linear system of (3.1.4).

Also, it is worth noting that the learning rate for the GD step can equivalently be incorporated in the optimization problem (3.1.6) by minimizing instead

$$\min_{\tilde{\boldsymbol{u}}} \ \nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_N + \frac{1}{2} \alpha \sum_{i=1}^{N-1} \tilde{\boldsymbol{u}}_i^T \tilde{\boldsymbol{u}}_i$$

and the solution follows as above, but taking $\alpha \boldsymbol{I}$ instead of $\boldsymbol{I}$ in (3.1.7) and (3.1.8).

3.2 Steepest Descent Objective

The result shown in Theorem 3.1.1 was obtained by minimizing the steepest descent objective under the Euclidean norm (see section 2.2.1). But, the steepest descent procedure extends well beyond gradient descent and will lead to other optimization algorithms (such as NGD and Newton's method) when the divergence measure is chosen differently. We will now show

that an equivalent LQR can be obtained to the steepest descent problem for a wide choice of divergence measures.

3.2.1 General LQR Equivalence

To perform a steepest descent step under a given P-norm, $\|\cdot\|_P$, we need to solve:

$$\min_{\boldsymbol{v}} \nabla f(x)^T \boldsymbol{v} + \frac{1}{2}\|\boldsymbol{v}\|_P$$

The P-norm is often induced by a divergence measure, and for such cases, we can recover an equivalent LQR. We provide this result below, by simply extending the demonstration of the equivalence between a LQR problem and one Newton's method step stated in Bertsekas 2016. The use of the notation H in (3.2.3) refers to the *Hamiltonian* defined in the development of the Pontryagin's maximum principle (PMP) Pontriagin *et al.* 1962.

Theorem 3.2.1. *If the matrix $P(\boldsymbol{x},\boldsymbol{u})$ inducing the quadratic norm $\|\cdot\|_P$ can be written as $P(\boldsymbol{x},\boldsymbol{u}) = \nabla^2_{\boldsymbol{uu}} D(\boldsymbol{x},\boldsymbol{u})$ and D is a function that can be expressed layer-wise (i.e. $D(\boldsymbol{x},\boldsymbol{u}) = D_N(\boldsymbol{x}_N) + \sum_i^{N-1} D_i(\boldsymbol{x}_i,\boldsymbol{u}_i)$), then the minimization problem*

$$\begin{aligned} &\min_{\boldsymbol{v}} \nabla^T J(F(\boldsymbol{x}))\boldsymbol{v} + \frac{1}{2}\|\boldsymbol{v}\|_P \\ &= \min_{\tilde{\boldsymbol{u}}} \nabla^T J(\boldsymbol{u})\tilde{\boldsymbol{u}} + \frac{1}{2}\tilde{\boldsymbol{u}}^T\Big(\nabla^2_{\boldsymbol{uu}} D(\boldsymbol{x},\boldsymbol{u})\Big)\tilde{\boldsymbol{u}} \end{aligned} \tag{3.2.1}$$

is equivalent to the following LQR when F is the dynamical system of an OCP (for example, when F is a NN) and J the associated cost :

$$\begin{aligned} \min_{\tilde{\boldsymbol{u}}} \quad & \nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_N + \frac{1}{2}\{\tilde{\boldsymbol{x}}^T \nabla^2_{\boldsymbol{xx}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{x}}^T \nabla^2_{\boldsymbol{xu}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))\tilde{\boldsymbol{u}} + \\ & \qquad \tilde{\boldsymbol{u}}^T \nabla^2_{\boldsymbol{ux}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{u}}^T \nabla^2_{\boldsymbol{uu}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))\tilde{\boldsymbol{u}}\} \\ \text{subject to} \quad & \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i\tilde{\boldsymbol{x}}_t + \boldsymbol{B}_i\tilde{\boldsymbol{u}}_i \\ \text{where} \quad & \boldsymbol{A}_i = \nabla^T_{\boldsymbol{x}_i} f_i^k,\ \boldsymbol{B}_i = \nabla^T_{\boldsymbol{u}_i} f_i^k \\ \text{given} \quad & \tilde{\boldsymbol{x}}_0 = 0 \end{aligned} \tag{3.2.2}$$

Where:

$$\phi(\boldsymbol{u}) = [\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_N]^T$$

$$H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) = D(\phi(\boldsymbol{u}),\boldsymbol{u}) + h(\phi(\boldsymbol{u}),\boldsymbol{u})^T p(\boldsymbol{u}) \tag{3.2.3}$$

$$h(\phi(\boldsymbol{u}),\boldsymbol{u}) = \begin{bmatrix} f_0(\boldsymbol{u}_0,\boldsymbol{x}_0) - \boldsymbol{x}_1 \\ \vdots \\ f_{N-1}(\boldsymbol{u}_{N-1},\boldsymbol{x}_{N-1}) - \boldsymbol{x}_N \end{bmatrix} = \overrightarrow{0}$$

And

$$p(\boldsymbol{u}) = \begin{bmatrix} \nabla_{\boldsymbol{x}_1} f_1 \cdot \boldsymbol{p}_2 + \nabla_{\boldsymbol{x}_1} D \\ \vdots \\ \nabla_{\boldsymbol{x}_{N-1}} f_{N-1} \cdot \boldsymbol{p}_N + \nabla_{\boldsymbol{x}_{N-1}} D \\ \nabla_{\boldsymbol{x}_N} D \end{bmatrix}$$

Proof. When going over the proof, it is important to realize that the objective to minimize in (3.2.1) is *not* a valid cost function for LQR problems, because it is not decoupled with respect to the controls and the states. This is why introducing the Hamiltonian is so helpful: the Hamiltonian is such that the controls are minimizers at each transition step (i.e. at every layer) (see Lenhart and Workman, 2007). It is this property that allows us to cast the optimization objective as *a sum of decoupled terms*, thus making it a proper objective for a LQR problem.

We already know that the term $\nabla^T J(\boldsymbol{u})\tilde{\boldsymbol{u}}$ gives rise to the "skeleton" of the LQR, so to complete the proof we focus on showing that

$$\begin{aligned}\tilde{\boldsymbol{u}}^T\Big(\nabla^2_{\boldsymbol{uu}} D(\boldsymbol{x},\boldsymbol{u})\Big)\tilde{\boldsymbol{u}} =& \tilde{\boldsymbol{x}}^T\nabla^2_{\boldsymbol{xx}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{x}}^T\nabla^2_{\boldsymbol{xu}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))\tilde{\boldsymbol{u}} \\ &+ \tilde{\boldsymbol{u}}^T\nabla^2_{\boldsymbol{ux}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{u}}^T\nabla^2_{\boldsymbol{uu}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))\tilde{\boldsymbol{u}}\end{aligned}$$

To see this, we need to view D as an "alternate loss term" for which we can develop the appropriate adjoint terms; an approach inspired by the PMP.
Since h is a zero valued vector, we have that $\forall \boldsymbol{p}$, $D(\phi(\boldsymbol{u}),\boldsymbol{u}) = H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))$ when H is defined as:

$$H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) = D(\phi(\boldsymbol{u}),\boldsymbol{u}) + h(\phi(\boldsymbol{u}),\boldsymbol{u})^T p(\boldsymbol{u})$$

By taking the first derivative with respect to $\boldsymbol{u}$ and applying the chain rule, we get

$$\nabla_{\boldsymbol{u}} H = \nabla_{\boldsymbol{u}} \phi(\boldsymbol{u}) \nabla_{\boldsymbol{x}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) + \nabla_{\boldsymbol{u}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u}))$$

This is true for any $\boldsymbol{p}$, and in particular for the given $\boldsymbol{p}$ s.t. $\nabla_{\boldsymbol{x}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) = 0$. This means that we can rewrite $\boldsymbol{p}_i$ as being

$$\begin{aligned} \boldsymbol{p}_i &= \nabla_{\boldsymbol{x}_i} f_i \cdot \boldsymbol{p}_{i+1} + \nabla_{\boldsymbol{x}_i} D \\ \boldsymbol{p}_N &= \nabla_{\boldsymbol{x}_N} D \end{aligned}$$

Taking the derivative of $\nabla_{\boldsymbol{u}} H$ once more, we now get:

$$\begin{aligned} \nabla^2_{\boldsymbol{u}} L = \nabla^2_{\boldsymbol{u}} \phi(\boldsymbol{u}) \nabla_{\boldsymbol{x}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) + \nabla_{\boldsymbol{u}} \phi(\boldsymbol{u}) \nabla_{\boldsymbol{xx}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) \nabla^T_{\boldsymbol{u}} \phi(\boldsymbol{u}) \\ + \nabla_{\boldsymbol{u}} \phi(\boldsymbol{u}) \nabla_{\boldsymbol{xu}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) + \nabla_{\boldsymbol{ux}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) \nabla^T_{\boldsymbol{u}} \phi(\boldsymbol{u}) \\ + \nabla_{\boldsymbol{uu}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) \end{aligned}$$

Remembering that $\tilde{\boldsymbol{u}}^T \nabla_{\boldsymbol{u}} \phi(\boldsymbol{u}) = \tilde{\boldsymbol{x}}^T$ (3.1.5) and $\nabla^2_{\boldsymbol{u}} \phi(\boldsymbol{u}) \nabla_{\boldsymbol{x}} H(\phi(\boldsymbol{u}),\boldsymbol{u},p(\boldsymbol{u})) = 0$ because of our choice of $p(\boldsymbol{u})$, we get the desired result.

□

The conditions imposed on D in Theorem 3.2.1 may seem quite restrictive, but we argue that it should not be the case in practice. Indeed, in most scenario, the divergence is only measured over the final state, i.e.: $D(\boldsymbol{x},\boldsymbol{u}) = D_N(\boldsymbol{x}_N)$. Moreover, the divergence measure can often be approximated (sometimes exactly) via their second-order Taylor's expansion (again, Nocedal and Wright, 2006):

Theorem 3.2.2. *For a divergence measure $D \in \mathbf{C}^2$ having the following properties:*

$$D(\boldsymbol{x},\boldsymbol{x}) = 0 \; ; \; D(\boldsymbol{x},\boldsymbol{y}) \geq 0$$

We have that:

$$D(\boldsymbol{u}) \approx \boldsymbol{u}^T \left(\nabla^2_{\boldsymbol{uu}} D(\boldsymbol{u}') \right) \boldsymbol{u} \tag{3.2.4}$$

Proof. Taking the second order Taylor's expansion of a divergence measure $D(\boldsymbol{u}) = D(\boldsymbol{u}',\boldsymbol{u})$ around $\boldsymbol{u} = \boldsymbol{u}'$:

$$D(\boldsymbol{u}) \approx D(\boldsymbol{u}') + \nabla_{\boldsymbol{u}} D(\boldsymbol{u}')\boldsymbol{u} + \boldsymbol{u}^T \Big(\nabla^2_{\boldsymbol{uu}} D(\boldsymbol{u}')\Big)\boldsymbol{u}$$

By definition $D(\boldsymbol{u}') = D(\boldsymbol{u}', \boldsymbol{u}') = 0$, and $\nabla_{\boldsymbol{u}} D(\boldsymbol{u}') = 0$ as well since $D(\boldsymbol{u})$ is minimal at $\boldsymbol{u}'$. □

3.2.2 Riccati's Solution to the Steepest Descent Objective

Since (3.2.2) is a LQR problem, just as for the GD case, we can recover the steepest descent update by solving the LQR with Riccati's equation. By defining H_i as:

$$H_i(\boldsymbol{x}_i, \boldsymbol{u}, \boldsymbol{p}_i) = D_i(\boldsymbol{x}_i, \boldsymbol{u}_i) + h_i(\boldsymbol{x}_i, \boldsymbol{u}_i)^T \boldsymbol{p}_i$$

And adopting the notation $\boldsymbol{Q}_N = \nabla^2 D_N(\boldsymbol{x}_N)$, $\boldsymbol{a}_N = \nabla_{\boldsymbol{x}_N} J(\boldsymbol{u}^k)$, $\boldsymbol{Q}_i = \nabla^2_{\boldsymbol{x}_i \boldsymbol{x}_i} H_i$, $\boldsymbol{R}_i = \nabla^2_{\boldsymbol{u}_i \boldsymbol{u}_i} H_i$, $\boldsymbol{M}_i = \nabla^2_{\boldsymbol{u}_i \boldsymbol{x}_i} H_i$, we can rewrite (3.2.2) more succinctly as:

$$\begin{aligned} &\min_{\tilde{\boldsymbol{u}}} \;\; \boldsymbol{a}_N^T \tilde{\boldsymbol{x}}_N + \frac{1}{2} \tilde{\boldsymbol{x}}_N^T \boldsymbol{Q}_N \tilde{\boldsymbol{x}}_N + \sum_{i=0}^{N-1} \Big(\frac{1}{2} \tilde{\boldsymbol{x}}_i^T \boldsymbol{Q}_i \tilde{\boldsymbol{x}}_i + \frac{1}{2} \tilde{\boldsymbol{u}}_i^T \boldsymbol{R}_i \tilde{\boldsymbol{u}}_i + \tilde{\boldsymbol{u}}^T \boldsymbol{M} \tilde{\boldsymbol{x}} \Big) \\ &\text{subject to } \; \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i \tilde{\boldsymbol{x}}_t + \boldsymbol{B}_i \tilde{\boldsymbol{u}}_i \\ &\text{where } \; \boldsymbol{A}_i = \nabla^T_{\boldsymbol{x}_i} f_i^k, \, \boldsymbol{B}_i = \nabla^T_{\boldsymbol{u}_i} f_i^k \\ &\text{given } \; \tilde{\boldsymbol{x}}_0 = 0 \end{aligned} \quad , \tag{3.2.5}$$

a problem for which the optimal solution is obtained by solving backward:

$$\boldsymbol{K}_i = \boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i + \boldsymbol{Q}_i - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} (\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \tag{3.2.6}$$

$$\boldsymbol{\lambda}_i = \boldsymbol{A}_i^T \lambda_{i+1} - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1} \;\; , \tag{3.2.7}$$

with $\boldsymbol{K}_N = \boldsymbol{Q}_N$ and $\boldsymbol{\lambda}_N = \boldsymbol{a}_N$.
The updates are then:

$$\tilde{\boldsymbol{u}}_0^* = -(\boldsymbol{R}_0 + \boldsymbol{B}_0^T \boldsymbol{K}_1 \boldsymbol{B}_0)^{-1} \boldsymbol{B}_0^T \boldsymbol{\lambda}_1 \tag{3.2.8}$$

$$\tilde{\boldsymbol{x}}_{i+1}^* = \boldsymbol{A}_i \tilde{\boldsymbol{x}}_i^* + \boldsymbol{B}_i \tilde{\boldsymbol{u}}_i^* \tag{3.2.9}$$

$$\tilde{\boldsymbol{u}}_i^* = -(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} \left((\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \tilde{\boldsymbol{x}}_i^* + \boldsymbol{B}_i^T \lambda_{i+1} \right) \quad . \tag{3.2.10}$$

3.2.3 Equivalence with Natural Gradients

Natural gradient descent Amari 1998 is an optimization algorithm well suited to the modeling of probability measures and has known advantages over GD in the context of deep learning (e.g. Pascanu and Bengio 2014). NGD can be stated in the language of steepest descent as the search for the optimal unitary step in a space scaled by the Fisher's information matrix ($\boldsymbol{F}$):

$$\begin{aligned} \Delta \boldsymbol{v} &= \arg\min_{\boldsymbol{v}} \nabla f(\boldsymbol{u})^T \boldsymbol{v} + \frac{1}{2} \|\boldsymbol{v}\|_F \\ &= \arg\min_{\boldsymbol{v}} \nabla f(\boldsymbol{u})^T \boldsymbol{v} + \frac{1}{2} \boldsymbol{v}^T \boldsymbol{F} \boldsymbol{v} \end{aligned} \quad , \tag{3.2.11}$$

leading to the NGD update rule for some parameters $\boldsymbol{u}$:

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k - \boldsymbol{F}^{-1} \nabla f(\boldsymbol{u})$$

Furthermore, there is a well-known connection between the Hessian of the Kullback–Leibler (KL) divergence and the Fisher's information matrix, allowing to develop the second-order Taylor's approximation of the KL divergence as:

$$\begin{aligned} D_{\mathrm{KL}}(p_{\boldsymbol{u}} || p_{\boldsymbol{u}+\boldsymbol{v}}) &\approx D_{\mathrm{KL}}(p_{\boldsymbol{u}} || p_{\boldsymbol{u}}) + \nabla_{\boldsymbol{u}'}^T D_{\mathrm{KL}}(p_{\boldsymbol{u}} || p_{\boldsymbol{u}'}) \Big|_{\boldsymbol{u}'=\boldsymbol{u}} \cdot \boldsymbol{v} + \frac{1}{2} \boldsymbol{v}^T \left(\nabla_{\boldsymbol{u}'\boldsymbol{u}'}^2 D_{\mathrm{KL}}(p_{\boldsymbol{u}} || p_{\boldsymbol{u}'}) \Big|_{\boldsymbol{u}'=\boldsymbol{u}} \right) \boldsymbol{v} \\ &= D_{\mathrm{KL}}(p_{\boldsymbol{u}} || p_{\boldsymbol{u}}) + \nabla_{\boldsymbol{u}'}^T D_{\mathrm{KL}}(p_{\boldsymbol{u}} || p_{\boldsymbol{u}'}) \Big|_{\boldsymbol{u}'=\boldsymbol{u}} \cdot \boldsymbol{v} + \frac{1}{2} \boldsymbol{v}^T \boldsymbol{F} \boldsymbol{v} \end{aligned}$$

The first two terms are zero (see Theorem 3.2.2), which allows us to equivalently state the steepest descent step above as (see Amari and Nagaoka 2007):

$$\Delta \boldsymbol{v} = \arg\min_{\boldsymbol{v}} \nabla f(\boldsymbol{u})^T \boldsymbol{v} + \frac{1}{2} \boldsymbol{v}^T \left(\nabla_{\boldsymbol{u}'\boldsymbol{u}'}^2 D_{\mathrm{KL}}(p_{\boldsymbol{u}} || p_{\boldsymbol{u}'}) \Big|_{\boldsymbol{u}'=\boldsymbol{u}} \right) \boldsymbol{v} \tag{3.2.12}$$

This form is an exact fit to the formalism developed in Theorem 3.2.1. Thus, one can recover the NGD update step by optimizing the following LQR:

$$\begin{aligned}
&\min_{\tilde{\boldsymbol{u}}} \;\; \boldsymbol{a}_N^T \tilde{\boldsymbol{x}}_N + \frac{1}{2}\tilde{\boldsymbol{x}}_N^T \boldsymbol{Q}_N \tilde{\boldsymbol{x}}_N + \sum_{i=0}^{N-1} \Big(\frac{1}{2}\tilde{\boldsymbol{x}}_i^T \boldsymbol{Q}_i \tilde{\boldsymbol{x}}_i + \frac{1}{2}\tilde{\boldsymbol{u}}_i^T \boldsymbol{R}_i \tilde{\boldsymbol{u}}_i + \tilde{\boldsymbol{u}}_i^T \boldsymbol{M}_i \tilde{\boldsymbol{x}}_i\Big) \\
&\text{subject to} \;\; \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i \tilde{\boldsymbol{x}}_i + \boldsymbol{B}_i \tilde{\boldsymbol{u}}_i \\
&\text{where} \;\; \boldsymbol{A}_i = \nabla_{\boldsymbol{x}_i}^T f_i^k,\, \boldsymbol{B}_i = \nabla_{\boldsymbol{u}_i}^T f_i^k \\
&\text{given} \;\; \tilde{\boldsymbol{x}}_0 = 0
\end{aligned} \tag{3.2.13}$$

where

$$\begin{gathered}
\boldsymbol{Q}_N = \nabla^2_{\boldsymbol{x}'_N} D_{\mathrm{KL}}(p_{\boldsymbol{x}_N} || p_{\boldsymbol{x}'_N})\Big|_{\boldsymbol{x}'_N = \boldsymbol{x}_N} \;;\; \boldsymbol{a}_N = \nabla_{x_N} J(\boldsymbol{u}^k), \\
\boldsymbol{Q}_i = \nabla^2_{x_i x_i} H_i \;;\; \boldsymbol{R}_i = \nabla^2_{\boldsymbol{u}_i \boldsymbol{u}_i} H_i \;;\; \boldsymbol{M}_i = \nabla^2_{\boldsymbol{u}_i \boldsymbol{x}_i} H_i
\end{gathered}$$

and with $H_i = f_i(\boldsymbol{x}^k, \boldsymbol{u}^k)^T \boldsymbol{p}_i$ simply here since the KL-divergence has a direct dependency only over $\boldsymbol{x}_N$, the final state of the dynamic system that is a representation of the distribution we are trying to model.

The solution can be recovered via Riccati equation directly with (3.2.6) and (3.2.8). It is worth noticing that the solution is obtained *without* retrieving the Fisher's information matrix; indeed one only need to recover the Hessian of the KL divergence with respect to the final state (much simpler than recovering the Hessian with respect to the whole system parameters) before applying the Riccati equation to get the gain matrix $\boldsymbol{K}$.

Another advantage of the method is that we do not need to invert the Fisher to recover the NGD update, but instead, we perform $N-1$ matrix inversion of the quantity $(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B})$. When the dynamic system is a NN and with n_l parameters per layer (the total number of parameters is $n = \sum_l n_l$), this means we must invert $N-1$ matrices of size $n_l \times n_l$ instead of inverting the Fisher of size $n \times n$.

3.2.4 Equivalence with Newton's method

As mention earlier, the motivation for this thesis came from the realization that the connection between the use of Newton's method for OCP optimization and LQR formalism generalizes to many steepest descent problems. For completeness, we develop this equivalence (originally shown in Bertsekas 2016; Dunn and Bertsekas 1989; De O. Pantoja and

Mayne 1989) below.

Newton's method can be easily stated as a steepest descent problem by optimizing the second-order Taylor's approximation of our function around the current parameters ($\boldsymbol{u}^k$):

$$\begin{aligned} J(\boldsymbol{u}) &\approx J^k + \nabla_{\boldsymbol{u}}^T J^k (\boldsymbol{u} - \boldsymbol{u}^k) + \frac{1}{2}(\boldsymbol{u} - \boldsymbol{u}^k)^T \nabla_{\boldsymbol{u}}^2 J^k (\boldsymbol{u} - \boldsymbol{u}^k) \\ &= J^k + \nabla_{\boldsymbol{u}}^T J^k (\boldsymbol{u} - \boldsymbol{u}^k) + \frac{1}{2}(\boldsymbol{u} - \boldsymbol{u}^k)^T \boldsymbol{H} (\boldsymbol{u} - \boldsymbol{u}^k) \end{aligned} , \tag{3.2.14}$$

where we used $\boldsymbol{H}$ to denote the Hessian. If $\boldsymbol{H}$ is positive semi-definite, then we are facing a convex problem and (3.2.14) minimum can be obtained by setting the derivative to 0. We get:

$$\begin{aligned} 0 &= \frac{\partial}{\partial \boldsymbol{u}} \Big(J^k + \nabla_{\boldsymbol{u}}^T J^k (\boldsymbol{u} - \boldsymbol{u}^k) + \frac{1}{2}(\boldsymbol{u} - \boldsymbol{u}^k)^T \boldsymbol{H} (\boldsymbol{u} - \boldsymbol{u}^k) \Big) \Big|_{\boldsymbol{u}=\boldsymbol{u}^*} \\ &= \nabla_{\boldsymbol{u}}^T J^k + \boldsymbol{H}(\boldsymbol{u}^* - \boldsymbol{u}^k) \\ &\implies \boldsymbol{u}^* = -\boldsymbol{H}^{-1} \nabla_{\boldsymbol{u}}^T J^k \end{aligned} .$$

Thus, to recover the Newton's update, we can solve the following steepest descent problem:

$$\Delta \boldsymbol{u} = \arg\min_{\boldsymbol{u}} \nabla_{\boldsymbol{u}}^T J^k (\boldsymbol{u} - \boldsymbol{u}^k) + \frac{1}{2}(\boldsymbol{u} - \boldsymbol{u}^k)^T \boldsymbol{H} (\boldsymbol{u} - \boldsymbol{u}^k) \tag{3.2.15}$$

If we are trying to optimize a NN, the equivalence with LQR problems developed in 2.3 allows us to express (3.2.15) as :

$$\begin{aligned} &\min_{\tilde{\boldsymbol{u}}} \ \boldsymbol{a}_N^T \tilde{\boldsymbol{x}}_N + \frac{1}{2}\tilde{\boldsymbol{x}}_N^T \boldsymbol{Q}_N \tilde{\boldsymbol{x}}_N + \sum_{i=0}^{N-1} \Big(\frac{1}{2}\tilde{\boldsymbol{x}}_i^T \boldsymbol{Q}_i \tilde{\boldsymbol{x}}_i + \frac{1}{2}\tilde{\boldsymbol{u}}_i^T \boldsymbol{R}_i \tilde{\boldsymbol{u}}_i + \tilde{\boldsymbol{u}}^T \boldsymbol{M} \tilde{\boldsymbol{x}} \Big) \\ &\text{subject to } \ \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i \tilde{\boldsymbol{x}}_t + \boldsymbol{B}_i \tilde{\boldsymbol{u}}_i \\ &\text{where } \ \boldsymbol{A}_i = \nabla_{\boldsymbol{x}_i}^T f_i^k, \ \boldsymbol{B}_i = \nabla_{\boldsymbol{u}_i}^T f_i^k \\ &\text{given } \ \tilde{\boldsymbol{x}}_0 = 0 \end{aligned} , \tag{3.2.16}$$

where

$$\boldsymbol{Q}_N = \nabla^2_{\boldsymbol{x}_N} J(\boldsymbol{u}^k)\ ;\ \boldsymbol{a}_N = \nabla_{x_N} J(\boldsymbol{u}^k),$$
$$\boldsymbol{Q}_i = \nabla^2_{x_i x_i} H_i\ ;\ \boldsymbol{R}_i = \nabla^2_{\boldsymbol{u}_i \boldsymbol{u}_i} H_i\ ;\ \boldsymbol{M}_i = \nabla^2_{\boldsymbol{u}_i \boldsymbol{x}_i} H_i$$
$$H_i = f_i(\boldsymbol{x}^k, \boldsymbol{u}^k)^T \boldsymbol{p}_i\ \ .$$

The above is a problem we can solve via the Riccati equation (see section 3.2.2).

3.3 Experimental validation

To validate the theory developed in the previous section, we set up a simple experiment: a NN trained with SGD and NGD, via the LQR algorithm, for one epoch. At every time step, before updating, we measured the cosine similarity between the update proposed by the LQR algorithm and the one obtained by plain automatic differentiation.

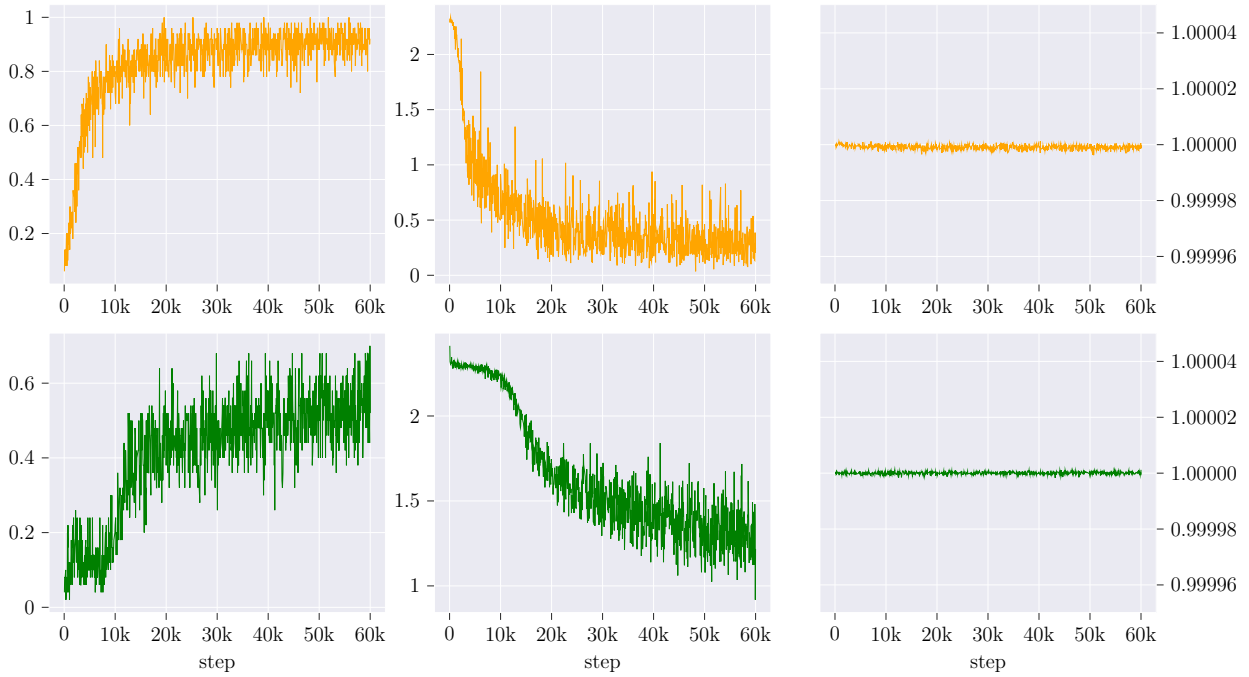

Figure 3.1. top: A NN composed of 3 convolution layers followed by 2 dense layers trained on MNIST database with SGD (5 examples per minibatch) but implemented with the LQR approach. **left:** Accuracy over one epoch of training. **center:** Loss over one epoch. **right:** The cosine similarity, measured at every step, between the gradients calculated via the LQR approach and by traditional automatic differentiation. **bottom:** Same as above, but for NGD.

As we can see in figure 3.1, the cosine similarity between the two schemes always remains close to 1, up to a factor of 10^{-5}, meaning they are pointing in the same direction. The very

small difference measured between the updates can thus safely be attributed to numerical precision.

Yet, the results presented here are mostly theoretical by now, as we are still lacking an efficient implementation that could justify a wide deployment of this approach. Those limitations will be discussed further in the conclusion, and we will touch upon possible improvement directions in chapter 5. But first, we will turn our attention toward a question that arose quickly in the development of our results: could the equivalence be extended to infinite horizon problems?

Chapter 4

Extension to the Infinite Horizon Case

The previous developments were made with the discrete horizon setting in mind. Yet, Linear Quadratic Regulation problems are also well-defined and understood in the infinite horizon setting, and we would like to extend the equivalence developed in the finite setting to this problem family as well.

In the deep learning setting, we can think of the DEQ (2.4.1) as an example of an infinite horizon discrete problem. The implicit block (usually made of multiple individual layers) can be thought of as the sole layer repeated indefinitely until a fixed point is reached. The loss is then usually taken at this fixed point.

Beginning with the equivalent LQR we have for steepest descent in finite horizon setting, we will try to manipulate the obtained solution to make it compatible with the LQR formulation for infinite horizon problems.

One benefit that arises from such a correspondence is that the optimal solution of a discrete LQR problem with infinite horizon is obtained by solving the associated DARE problem. Fortunately for us, the DARE is a well-studied problem for which a variety of powerful solvers are available.

4.1 Problem Statement

The extension of the equivalence developed in (3.2.2) to the infinite-horizon setting leads to an OCP in which the fixed point $\boldsymbol{x}_j$ is reached after j iterations of the dynamic system.

$$
\begin{aligned}
&\min_{\tilde{\boldsymbol{u}}} \ \nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_N + g(\tilde{\boldsymbol{x}}^2, \tilde{\boldsymbol{u}}^2) \\
&\text{where } \ N \to \infty \\
&\text{subject to } \ \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}\tilde{\boldsymbol{x}}_i + \boldsymbol{B}\tilde{\boldsymbol{u}}_i \ \text{ if } i \geq j \\
&\text{subject to } \ \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i\tilde{\boldsymbol{x}}_i + \boldsymbol{B}_i\tilde{\boldsymbol{u}}_i \ \text{ if } i < j \qquad , \\
&\text{where } \ \boldsymbol{A} = \nabla^T_{\boldsymbol{x}} f(\boldsymbol{x}_j, \boldsymbol{u}^k), \ \boldsymbol{B} = \nabla^T_{\boldsymbol{u}} f(\boldsymbol{x}_j, \boldsymbol{u}^k) \\
&\text{where } \ \boldsymbol{A}_i = \nabla^T_{\boldsymbol{x}} f(\boldsymbol{x}_i, \boldsymbol{u}^k), \ \boldsymbol{B}_i = \nabla^T_{u} f(\boldsymbol{x}_i, \boldsymbol{u}^k) \\
&\text{given } \ \tilde{\boldsymbol{x}}_0 = 0
\end{aligned}
\tag{4.1.1}
$$

where we used the notation $g(\tilde{\boldsymbol{x}}^2, \tilde{\boldsymbol{u}}^2)$ to denote the quadratic term in the LQR objective. Note that the above system is equivalent to a linear OCP with transition layers having *frozen weights* $\boldsymbol{u}$ across all layers. Moreover, the linear transition layers become static after reaching the fixed point (i.e. $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ stabilize).

Just as it is the case for a NN with frozen weights, the total gradient over $\boldsymbol{u}$ is given by the sums of the gradients recovered at each layer. This means that if the optimal solution to the above problem is $\tilde{\boldsymbol{u}}^*_i$, then the update over the frozen weights $\boldsymbol{u}^k$ must be:

$$
\Delta \boldsymbol{u}^k = \sum_{i}^{N \to \infty} \tilde{\boldsymbol{u}}^*_i \ . \tag{4.1.2}
$$

We should not expect that $\tilde{\boldsymbol{u}}_i \to 0$ as $N \to \infty$, thus the infinite summation above is obviously divergent.

To overcome that problem, we should not worry about the total gradient value, but instead its *direction*. In that case, the overall direction of the update stabilizes only if the linear system in (4.1.1) obtained after reaching the fixed point also leads to a fixed point. That is, the simplified dynamic system denoted by $\tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}\tilde{\boldsymbol{x}}_i + \boldsymbol{B}\tilde{\boldsymbol{u}}_i$ must reach a fixed point, that we can denote by the pair $(\tilde{\boldsymbol{x}}_g, \tilde{\boldsymbol{u}}_g)$.

We can convince ourselves that this will always be the case by using some notions from dynamical system theory. It is known that to have a fixed point, the dynamic system space must be contractive (i.e. spectral radius smaller than one) in the vicinity of the fixed point (see section 2.2 in Nayfeh and Balachandran 1995). This means that locally around the fixed point, the linearization of the dynamic system must also be contractive. Thus, since $\boldsymbol{A}\tilde{\boldsymbol{x}}_i + \boldsymbol{B}\tilde{\boldsymbol{u}}_i$ is a linearization around a fixed point of our system, it itself must have a fixed

point (see proposition 1, Gilbert 1992).

Now, if the linearized system has reached its fixed point after j' iterations, then we can rewrite (4.1.1) equivalently as :

$$\begin{aligned}
&\min_{\tilde{\boldsymbol{u}}} \ \nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_N + g(\tilde{\boldsymbol{x}}^2, \tilde{\boldsymbol{u}}^2) \\
&\text{where } \ N \to \infty \\
&\text{subject to } \ \tilde{\boldsymbol{x}}_{i+1} = \tilde{\boldsymbol{x}}_g = \boldsymbol{A}\tilde{\boldsymbol{x}}_g + \boldsymbol{B}\tilde{\boldsymbol{u}}_g \ \text{ if } i \geq g = j + j' \\
&\text{subject to } \ \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}\tilde{\boldsymbol{x}}_i + \boldsymbol{B}\tilde{\boldsymbol{u}}_i \ \text{ if } j + j' > i \geq j \\
&\text{subject to } \ \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}_i\tilde{\boldsymbol{x}}_i + \boldsymbol{B}_i\tilde{\boldsymbol{u}}_i \ \text{ if } i < j \\
&\text{where } \ \boldsymbol{A} = \nabla^T_{\boldsymbol{x}} f(\boldsymbol{x}_j, \boldsymbol{u}^k), \ \boldsymbol{B} = \nabla^T_{\boldsymbol{u}} f(\boldsymbol{x}_j, \boldsymbol{u}^k) \\
&\text{where } \ \boldsymbol{A}_i = \nabla^T_{\boldsymbol{x}} f(\boldsymbol{x}_i, \boldsymbol{u}^k), \ \boldsymbol{B}_i = \nabla^T_{u} f(\boldsymbol{x}_i, \boldsymbol{u}^k) \\
&\text{given } \ \tilde{\boldsymbol{x}}_0 = 0
\end{aligned} \tag{4.1.3}$$

Therefore, it means that as $N \to \infty$, we will have a stabilization of the optimal update $\tilde{\boldsymbol{u}}^*_i$ around $\tilde{\boldsymbol{u}}^*_g$ and $\Delta\boldsymbol{u} \to N\tilde{\boldsymbol{u}}^*_g$. That is, the optimal update goes toward infinity, but it does so by being parallel to $\tilde{\boldsymbol{u}}^*_g$.

This means that to recover the *optimal direction* we only need to solve the system after it has reached its fixed point and the resulting problem is then equivalent to solving (4.1.1):

$$\begin{aligned}
&\min_{\tilde{\boldsymbol{u}}} \ \nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_N + g(\tilde{\boldsymbol{x}}^2, \tilde{\boldsymbol{u}}^2) \\
&\text{where } \ N \to \infty \\
&\text{subject to } \ \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}\tilde{\boldsymbol{x}}_i + \boldsymbol{B}\tilde{\boldsymbol{u}}_i \qquad . \\
&\text{where } \ \boldsymbol{A} = \nabla^T_{\boldsymbol{x}} f(\boldsymbol{x}_j, \boldsymbol{u}^k), \ \boldsymbol{B} = \nabla^T_{\boldsymbol{u}} f(\boldsymbol{x}_j, \boldsymbol{u}^k) \\
&\text{given } \ \tilde{\boldsymbol{x}}_0 = 0
\end{aligned} \tag{4.1.4}$$

Keeping the initialization at $\tilde{\boldsymbol{x}}_0 = 0$ makes no difference here since we know that the linear system given by $\tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}\tilde{\boldsymbol{x}}_i + \boldsymbol{B}\tilde{\boldsymbol{u}}_i$ is contractive and will eventually reach the same fixed point, $\tilde{\boldsymbol{x}}_g$ as (4.1.3).

The above OCP is almost a proper LQR in the infinite horizon setting, except that the terms $\nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_N$ makes no sense in such a setting. But since $\tilde{\boldsymbol{x}}_i$ eventually reaches a

fixed point, we can replace it by $\sum_{i=0}^{N\to\infty} \nabla^T_{\boldsymbol{x}_j} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_i$. Indeed, as $N \to \infty$, this sum will tend to $N\Big(\nabla^T_{\boldsymbol{x}_j} J(u^k) \times \tilde{\boldsymbol{x}}_g\Big)$ and is thus a good proxy to the minimization of $\nabla^T_{\boldsymbol{x}_N} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_N$.

By construction, we can then state the following equivalence between steepest descent and LQR problems in a discrete infinite horizon setting:

Theorem 4.1.1. *The steepest descent update over the parameters $\boldsymbol{u}^k$ for a given example (input) $\boldsymbol{x}^k$ in a discrete infinite horizon OCP (for example, a DEQ) is equivalent to solving the following LQR problem:*

$$\begin{aligned} &\min_{\tilde{\boldsymbol{u}}} \ \sum_{i=0}^{\infty} \Big[\nabla^T_{\boldsymbol{x}_j} J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_i + g(\tilde{\boldsymbol{x}}_i^2, \tilde{\boldsymbol{u}}_i^2)\Big] \\ &\text{subjectto } \ \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}\tilde{\boldsymbol{x}}_i + \boldsymbol{B}\tilde{\boldsymbol{u}}_i \\ &\text{where } \ \boldsymbol{A} = \nabla^T_{\boldsymbol{x}} f(\boldsymbol{x}_j, \boldsymbol{u}),\ \boldsymbol{B} = \nabla^T_{\boldsymbol{u}} f(\boldsymbol{x}_j, \boldsymbol{u}) \\ &\text{given } \ \tilde{\boldsymbol{x}}_0 = 0 \end{aligned} \quad , \tag{4.1.5}$$

where $\boldsymbol{x}_j$ is the fixed point reached by the underlying dynamical system and $g(\tilde{\boldsymbol{x}}_i^2, \tilde{\boldsymbol{u}}_i^2)$ denotes the quadratic cost associated with the chosen divergence function used to measure the minimal steepest descent step.

4.1.1 Riccati's Solution in Infinite Horizon Setting

Just as for their finite counterpart, an exact solution exists for the problem of the form (4.1.5) and it can be obtained by solving the discrete algebraic Riccati equation (DARE). For a problem like (4.1.5), we need to solve the following Riccati system:

$$\boldsymbol{K} = \boldsymbol{A}^T\boldsymbol{K}\boldsymbol{A} + \boldsymbol{Q} - (\boldsymbol{M} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{A})^T(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{B})^{-1}(\boldsymbol{M} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{A}) \tag{4.1.6}$$

$$\boldsymbol{\lambda} = \boldsymbol{a} + \boldsymbol{A}^T\boldsymbol{\lambda} - (\boldsymbol{M} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{A})^T(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{B})^{-1}(\boldsymbol{B}^T\boldsymbol{\lambda}) \ , \tag{4.1.7}$$

with

$$\boldsymbol{a} = \nabla_{\boldsymbol{x}_j} J(\boldsymbol{u}^k) \tag{4.1.8}$$

$$\boldsymbol{Q} = \nabla^2_{\boldsymbol{x}\boldsymbol{x}} H \ ; \ \boldsymbol{R} = \nabla^2_{\boldsymbol{u}\boldsymbol{u}} H \ ; \ \boldsymbol{M} = \nabla^2_{\boldsymbol{u}\boldsymbol{x}} H \tag{4.1.9}$$

$$H = D(\boldsymbol{x}_j) + f(\boldsymbol{x}_j, \boldsymbol{u}^k)^T\boldsymbol{p} \ , \tag{4.1.10}$$

where $D(\cdot)$ is the divergence measure we are using to perform the steepest descent. The adjoint vector is recovered by solving the linear system:

$$\begin{aligned} \boldsymbol{p} &= \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_j, \boldsymbol{u}^k) \cdot \boldsymbol{p} + \nabla_{\boldsymbol{x}} D(\boldsymbol{x}_j) \\ &\implies \boldsymbol{p} = (\boldsymbol{I} - \boldsymbol{A}^T)^{-1} \nabla_{\boldsymbol{x}} D(\boldsymbol{x}_j) \end{aligned} \tag{4.1.11}$$

The optimal update is then obtained by solving the following dynamic linear system:

$$(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{u}}^*) = \Big(\boldsymbol{A}\tilde{\boldsymbol{x}}^* + \boldsymbol{B}\tilde{\boldsymbol{u}}^*, -(\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B})^{-1} ((\boldsymbol{M} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{A})\tilde{\boldsymbol{x}}^* + \boldsymbol{B}^T \boldsymbol{\lambda}) \Big) \tag{4.1.12}$$

4.2 Equivalence with Implicit Differentiation

To our surprise, it appears that the LQR approach in infinite horizon leads to the exact same algorithm as the one obtained when calculating the gradient via the Implicit Function Theorem (see section 2.4). We summarize this result in the following theorem:

Theorem 4.2.1. *Taking a gradient step in a NN with infinite depth (such as a DEQ) via the Implicit Function Theorem is equivalent to finding the solution to a discrete-time LQR with an infinite horizon.*

Proof. We know from Theorem 4.1.1 that to retrieve the GD update for a NN with infinite depth that reaches a fixed point, we need to solve the following LQR problem:

$$\begin{aligned} &\min_{\tilde{\boldsymbol{u}}} \ \sum_{i=0}^{\infty} \left[\nabla_{\boldsymbol{x}_j}^T J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_i + \tilde{\boldsymbol{u}}_i^T \tilde{\boldsymbol{u}}_i \right] \\ &\text{subject to } \ \tilde{\boldsymbol{x}}_{i+1} = \boldsymbol{A}\tilde{\boldsymbol{x}}_i + \boldsymbol{B}\tilde{\boldsymbol{u}}_i \\ &\text{where } \ \boldsymbol{A} = \nabla_{\boldsymbol{x}}^T f(\boldsymbol{x}_j, \boldsymbol{u}), \ \boldsymbol{B} = \nabla_{\boldsymbol{u}}^T f(\boldsymbol{x}_j, \boldsymbol{u}) \\ &\text{given } \ \tilde{\boldsymbol{x}}_0 = 0 \end{aligned} \tag{4.2.1}$$

Leveraging Riccati's equation (4.1.6), we need to solve a matrix $\boldsymbol{K}$ that satisfies:

$$\boldsymbol{K} = \boldsymbol{A}^T \boldsymbol{K} \boldsymbol{A} - (\boldsymbol{B}^T \boldsymbol{K} \boldsymbol{A})^T (\boldsymbol{I} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B})^{-1} (\boldsymbol{B}^T \boldsymbol{K} \boldsymbol{A})$$

to find the optimal update. But, the above problem is in fact a trivial case, for which the obvious solution is $\boldsymbol{K} = \boldsymbol{0}$. The optimal update is therefore given by (4.1.12):

$$\tilde{\boldsymbol{u}}^* = -\boldsymbol{B}^T \boldsymbol{\lambda}$$

With $\boldsymbol{\lambda}$ obtained by solving the linear dynamic system:

$$\boldsymbol{\lambda} = \boldsymbol{a} + \boldsymbol{A}^T\boldsymbol{\lambda} = \nabla_{\boldsymbol{x}_j} J(\boldsymbol{u}^k) + \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_j,\boldsymbol{u}^k)\boldsymbol{\lambda}$$

Or equivalently:

$$\begin{aligned} \mathbf{0} &= (\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_j,\boldsymbol{u}^k) - \boldsymbol{I})\boldsymbol{\lambda} + \nabla_{\boldsymbol{x}_j} J(\boldsymbol{u}^k) \\ &\implies \boldsymbol{\lambda} = -(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_j,\boldsymbol{u}^k) - \boldsymbol{I})^{-1}\nabla_{\boldsymbol{x}_j} J(\boldsymbol{u}^k) \end{aligned}$$

And thus, the parameters update can be rewritten as:

$$\tilde{\boldsymbol{u}}^* = \nabla_{\boldsymbol{u}} f(\boldsymbol{x}_j,\boldsymbol{u})(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_j,\boldsymbol{u}^k) - \boldsymbol{I})^{-1}\nabla_{\boldsymbol{x}_j} J(\boldsymbol{u}^k)$$

Remembering that the matrix of gradients is the transpose of the Jacobian, we can recognize that the parameter update recovered by solving the LQR is the same one as when we retrieve the gradient via the Implicit Function Theorem (2.4.7) □

Although this result does not improve or lead to a new approach to recovering the gradient, it provides a novel view on the training process behind the optimization of NN with infinite depth.

4.3 Cyclic Solution

So far, we have treated the implicit block of layers involved in the transformation $f(\boldsymbol{x},\boldsymbol{u})$ as a whole. However, in practice, this transformation is usually made up of individual layers. If the implicit block is composed of N layers, we can rewrite $f(\boldsymbol{x},\boldsymbol{u})$ as:

$$f(\boldsymbol{x},\boldsymbol{u}) = f_N(f_{N-1}(...(f_2(f_1(\boldsymbol{x},\boldsymbol{u})))))$$

Taking into account this decomposition, we can recast the LQR problem (4.1.5) as:

$$\begin{aligned} &\min_{\tilde{\boldsymbol{u}}} \ \sum_{i=0}^{\infty}\sum_{n=1}^{N}\left[\nabla_{\boldsymbol{x}_j}^T J(\boldsymbol{u}^k) \times \tilde{\boldsymbol{x}}_n^i + g((\tilde{\boldsymbol{x}}_n^i)^2,(\tilde{\boldsymbol{u}}_n^i)^2)\right] \\ &\text{subject to } \ \tilde{\boldsymbol{x}}_{n+1}^i = \boldsymbol{A}_n\tilde{\boldsymbol{x}}_n^i + \boldsymbol{B}_n\tilde{\boldsymbol{u}}_x^i \\ &\text{with } \ \tilde{\boldsymbol{x}}_{N+1}^i = \tilde{x}_0^{i+1} \\ &\text{where } \ \boldsymbol{A}_n = \nabla_{\boldsymbol{x}_n}^T f_n(\boldsymbol{x}_n^j,\boldsymbol{u}_n^k),\ \boldsymbol{B}_n = \nabla_{\boldsymbol{u}_n}^T f_n(\boldsymbol{x}_n^j,\boldsymbol{u}_n^k) \\ &\text{given } \ \tilde{\boldsymbol{x}}_0^0 = 0 \end{aligned} \quad , \tag{4.3.1}$$

where $\boldsymbol{x}_n^j$ denotes the fixed point inside the layer f_n. Note that we moved the iteration number i to a superscript to make room for the needed additional index, n, denoting the layer. Overall, (4.3.1) denotes the exact same system as the one in (4.1.5); we simply exposed here the underlying structure of the transformation $f(\boldsymbol{x},\boldsymbol{u})$ that is repeated infinitely.

Nevertheless, a problem with a form such as (4.3.1) is known as a *cyclic LQR* and the solution for such problem can be obtained by solving the cyclic equations system over $\boldsymbol{K}_i$:

$$\begin{aligned}
\boldsymbol{K}_{N-1} &= \boldsymbol{A}_N^T\boldsymbol{K}_N\boldsymbol{A}_N + \boldsymbol{Q}_N - (\boldsymbol{M}_N + \boldsymbol{B}_N^T\boldsymbol{K}_N\boldsymbol{A}_N)^T(\boldsymbol{R}_N + \boldsymbol{B}_N^T\boldsymbol{K}_N\boldsymbol{B}_N)^{-1}(\boldsymbol{M}_N + \boldsymbol{B}_N^T\boldsymbol{K}_N\boldsymbol{A}_N) \\
&\vdots \\
\boldsymbol{K}_1 &= \boldsymbol{A}_2^T\boldsymbol{K}_2\boldsymbol{A}_2 + \boldsymbol{Q}_2 - (\boldsymbol{M}_2 + \boldsymbol{B}_2^T\boldsymbol{K}_2\boldsymbol{A}_2)^T(\boldsymbol{R}_2 + \boldsymbol{B}_2^T\boldsymbol{K}_2\boldsymbol{B}_2)^{-1}(\boldsymbol{M}_2 + \boldsymbol{B}_2^T\boldsymbol{K}_2\boldsymbol{A}_2) \\
\boldsymbol{K}_N &= \boldsymbol{A}_1^T\boldsymbol{K}_1\boldsymbol{A}_1 + \boldsymbol{Q}_1 - (\boldsymbol{M}_1 + \boldsymbol{B}_1^T\boldsymbol{K}_1\boldsymbol{A}_1)^T(\boldsymbol{R}_1 + \boldsymbol{B}_1^T\boldsymbol{K}_1\boldsymbol{B}_1)^{-1}(\boldsymbol{M}_1 + \boldsymbol{B}_1^T\boldsymbol{K}_1\boldsymbol{A}_1)
\end{aligned} \tag{4.3.2}$$

The same cyclic structure arises as well for the $\boldsymbol{\lambda}_i$:

$$\begin{aligned}
\boldsymbol{\lambda}_{N-1} &= \boldsymbol{a} + \boldsymbol{A}_N^T\boldsymbol{\lambda}_N - (\boldsymbol{M}_N + \boldsymbol{B}_N^T\boldsymbol{K}_N\boldsymbol{A}_N)^T(\boldsymbol{R}_N + \boldsymbol{B}_N^T\boldsymbol{K}_N\boldsymbol{B}_N)^{-1}(\boldsymbol{B}_N^T\boldsymbol{\lambda}_N) \\
&\vdots \\
\boldsymbol{\lambda}_1 &= \boldsymbol{a} + \boldsymbol{A}_2^T\boldsymbol{\lambda}_2 - (\boldsymbol{M}_2 + \boldsymbol{B}_2^T\boldsymbol{K}_2\boldsymbol{A}_2)^T(\boldsymbol{R}_2 + \boldsymbol{B}_2^T\boldsymbol{K}_2\boldsymbol{B}_2)^{-1}(\boldsymbol{B}_2^T\boldsymbol{\lambda}_2) \\
\boldsymbol{\lambda}_N &= \boldsymbol{a} + \boldsymbol{A}_1^T\boldsymbol{\lambda}_1 - (\boldsymbol{M}_1 + \boldsymbol{B}_1^T\boldsymbol{K}_1\boldsymbol{A}_1)^T(\boldsymbol{R}_1 + \boldsymbol{B}_1^T\boldsymbol{K}_1\boldsymbol{B}_1)^{-1}(\boldsymbol{B}_1^T\boldsymbol{\lambda}_1)
\end{aligned} \quad , \tag{4.3.3}$$

and the optimal solution is obtained by bringing solving the following system:

$$(\tilde{\boldsymbol{x}}_{n+1}^*, \tilde{\boldsymbol{u}}_{n+1}^*) = \Big(\boldsymbol{A}_n\tilde{\boldsymbol{x}}_n^* + \boldsymbol{B}_N\tilde{\boldsymbol{u}}_n^*, -(\boldsymbol{R}_n + \boldsymbol{B}_n^T\boldsymbol{K}_n\boldsymbol{B}_n)^{-1}((\boldsymbol{M}_n + \boldsymbol{B}_n^T\boldsymbol{K}_n\boldsymbol{A}_N)\tilde{\boldsymbol{x}}_n^* + \boldsymbol{B}_N^T\boldsymbol{\lambda}_n)\Big) \tag{4.3.4}$$

Although it may seem that this formulation only complicates things, we will see later that it allows us to leverage a very useful approximation in the infinite setting. Moreover, we again benefit from the maturity of the LQR field since cyclic counterparts already exist for most of the DARE solvers.

4.4 LQR-based Natural Gradient Method for DEQs

To the best of our knowledge, second-order methods have not been deployed for the optimization of infinite depth NN such as DEQ. A possible explanation is the computational and engineering challenge necessary to recover the second derivative via implicit differentiation.

What is interesting about the LQR equivalence is that it allows us to follow a simple recipe to compute the update for second-order methods optimization. As an example, we detail below the algorithm that arises from the LQR formulation when we want to find the NGD update for a system such as DEQ, given the implicit block f and the final transformation h (usually a dense layer followed by a softmax). We assume we have access to a fixed-point solver, that we denote `fx_point_solve`, taking as input an initial value and a mapping. The solver can simply be the repeated application of the implicit block until convergence or leverage an acceleration scheme such as Anderson iteration (Anderson 1965). We also assume we have access to a DARE solver; for a variety of such see Bini *et al.* 2011.

Algorithm 1 One epoch of training with natural gradient descent over an implicit model

procedure IMPLICIT-NGD(f, $\boldsymbol{x}_0$)
  for $\boldsymbol{x}_0^k \in \mathcal{D}$ **do**
    $\boldsymbol{x}_j \leftarrow$ `fx_point_solve`$(f, \boldsymbol{x}_0^k)$
    $\boldsymbol{A} \leftarrow \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_j)$
    $\boldsymbol{B} \leftarrow \nabla_{\boldsymbol{u}} f(\boldsymbol{u}^k)$
    $\boldsymbol{p} \leftarrow$ `fx_point_solve`$(\boldsymbol{A}^T\boldsymbol{p} + \nabla_{\boldsymbol{x}} D_{\mathrm{KL}}(h(\boldsymbol{x}_j)||h(\boldsymbol{x}'))\big|_{\boldsymbol{x}'=\boldsymbol{x}_j}, \boldsymbol{p} = 0)$
                                ▷ $\boldsymbol{p}$ is the adjoint
    $H(\boldsymbol{x}', \boldsymbol{u}') \leftarrow D_{\mathrm{KL}}(h(\boldsymbol{x}_j)||h(\boldsymbol{x}')) + f(\boldsymbol{x}', \boldsymbol{u}')^T\boldsymbol{p}$
    $\boldsymbol{Q} \leftarrow \nabla_{\boldsymbol{xx}} H(\boldsymbol{x}_j, \boldsymbol{u}^k)$
    $\boldsymbol{R} \leftarrow \nabla_{\boldsymbol{uu}} H(\boldsymbol{x}_j, \boldsymbol{u}^k)$
    $\boldsymbol{Q} \leftarrow \nabla_{\boldsymbol{ux}} H(\boldsymbol{x}_j, \boldsymbol{u}^k)$
    $\boldsymbol{a} \leftarrow \nabla_{\boldsymbol{x}_j} J(\boldsymbol{u}^k)$
    $\boldsymbol{K} \leftarrow$ `DARE`$(\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{M})$
    $\boldsymbol{\lambda} \leftarrow$ `fx_point_solve`$(\boldsymbol{a} + \boldsymbol{A}^T\boldsymbol{\lambda} - (\boldsymbol{M} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{A})^T(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{\lambda}, \boldsymbol{\lambda} = \boldsymbol{a})$
    $\boldsymbol{u}^{k+1} \leftarrow$ `fx_point_solve`$(-(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{B})^{-1}((\boldsymbol{M} + \boldsymbol{B}^T\boldsymbol{K}\boldsymbol{A})\tilde{\boldsymbol{x}} + \boldsymbol{B}^T\boldsymbol{\lambda}), \boldsymbol{u} = \boldsymbol{u}^k)$
                                ▷ $\tilde{\boldsymbol{x}}$ updated simultaneously to $\tilde{\boldsymbol{x}} \leftarrow \boldsymbol{A}\tilde{\boldsymbol{x}} + \boldsymbol{B}u$
  end for
  return $\boldsymbol{u}^N$
end procedure

We tried this algorithm ourselves and noticed that the number of iteration needed to converge was smaller on our toy network trained over MNIST, as showcased in figure 4.1. Sadly, we encountered stability issues and were not able to scale this algorithm to larger DEQs.

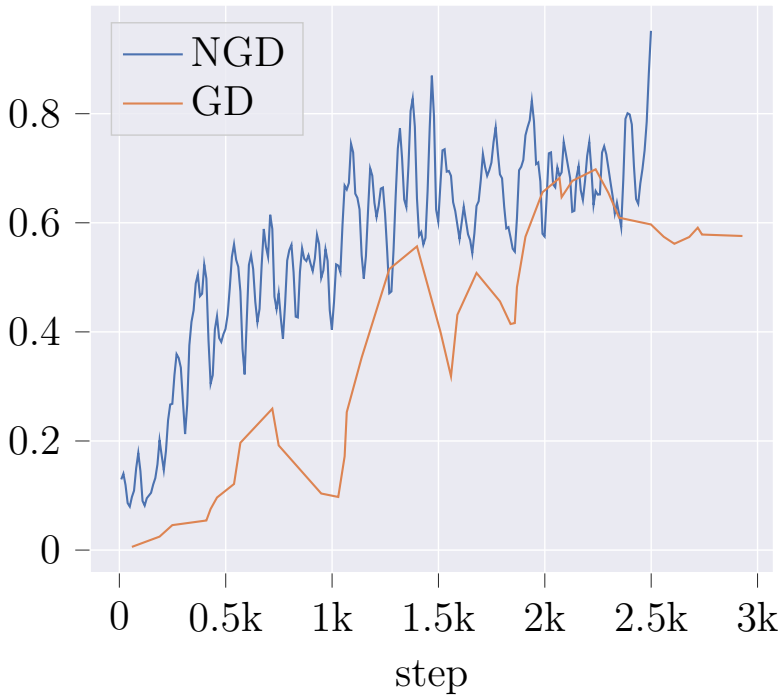

Figure 4.1. Zoom on the accuracy achieved (first 2500 iterations) during the training of a toy DEQ (a Resnet block with 12 inner channels) over MNIST with the NGD algorithm arising from the LQR approach. NGD starts to converge in less iteration, but it is 10x slower than GD.

Chapter 5

Computational Perspective

Computationally speaking, when thinking about second-order method optimizers, the approach described above already has a slight advantage over the direct implementation. Indeed, the difficulties with those methods come from the fact that we need to invert a matrix (Hessian for Newton's method, the Fisher for NGD); a procedure that typically requires $\mathcal{O}(m^{2.3})$ iterations for a matrix of size $m \times m$ when solved with an algorithm such as conjugate gradients (Hestenes and Stiefel 1952). This cost often proves prohibitive with typical architectures in DL, the dimension of the parameters being simply too big.

Fortunately for us, a benefit of building the equivalent LQR is to reduce (slightly) the computation complexity. To see it, let us consider that we want to perform NGD to optimize a NN. Then, as seen with the system of equations shown in (3.2.6) and (4.3.2), the inversion of the Fisher matrix (of size $m \times m$, m being the total number of parameters) is replaced by the inversion of N matrices scaling with the number of parameters per layer (N being the number of layers). Thus, we instead need to invert the N matrices $(\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B})^{-1}$ of size $m_1, m_2, ..., m_N$; an operation that has a computation complexity of the form $\mathcal{O}(m_1^{2.3} + m_2^{2.3} + ... + m_N^{2.3})$. Knowing that $m = m_1 + m_2 + ... + m_N$, the second procedure has thus an advantage.

Yet, this gain is not enough to justify the use of the LQR method since the number of parameters inside an individual layer can still get pretty big with modern NN architectures. To alleviate, this problem we now develop a useful approach when facing a NN composed of layers with activation functions that have *at most* a linear dependency over their respective layer parameters.

5.1 Leveraging Neumann Series

Let consider the term we need to invert when using the LQR approach to make a steepest descent step:

$$(\bar{\boldsymbol{R}}_i + \boldsymbol{B}_i^T \boldsymbol{K}_i \boldsymbol{B})^{-1} \ , \tag{5.1.1}$$

where we have $\bar{\boldsymbol{R}}_i = \boldsymbol{R}_i + \delta_i \boldsymbol{I}$; that is a penalty term δ_i is added over the diagonal of the matrix $\boldsymbol{R}_i$. This decision corresponds to the addition of a damping term and can be justified as a trust-region method. We will discuss this term in more detail later, so let us just accept it as is for now.

Now, let us turn our attention to $\boldsymbol{R}_i$. When the divergence measure is only taken over the final state, this quantity is the Hessian of the transition function (a layer) with respect to its parameters. But, typical layers inside a NN are composed of a linear function followed by an activation (σ), that we can usually write as:

$$\boldsymbol{R}_i = \nabla \boldsymbol{u}_i \boldsymbol{u}_i f_i = \nabla \boldsymbol{u}_i \boldsymbol{u}_i \ \sigma(\boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b}_i) \tag{5.1.2}$$

If we were not to use any activation function, $\boldsymbol{R}_i$ would be 0. What is more interesting though is that the most popular activation function, the Rectified Linear Unit (ReLU) (Jarrett *et al.* 2009; Glorot *et al.* 2011), does not introduce a co-dependency in the activation between the parameters of the current layer. Indeed, the ReLU pushes the negative terms to 0 and leaves unmodified the others. For this particular case, we have:

$$\boldsymbol{R}_i = \nabla \boldsymbol{u}_i \boldsymbol{u}_i \ \text{ReLu}(\boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b}_i) = 0 \ . \tag{5.1.3}$$

This means that for such networks, we are seeking to invert matrices of the form:

$$\begin{aligned} (\bar{\boldsymbol{R}}_i + \boldsymbol{B}_i^T \boldsymbol{K}_i \boldsymbol{B}_i)^{-1} &= (\boldsymbol{I}_{\delta_i} + \bar{\boldsymbol{K}}_i)^{-1} \\ &= (\delta_i (\boldsymbol{I} + \frac{1}{\delta_i} \bar{\boldsymbol{K}}_i)^{-1} \\ &= \frac{1}{\delta_i} + (\boldsymbol{I} + \frac{1}{\delta_i} \bar{\boldsymbol{K}}_i)^{-1} \end{aligned} \ , \tag{5.1.4}$$

where we abbreviated $\boldsymbol{B}_i^T \boldsymbol{K}_i \boldsymbol{B}_i$ by $\bar{\boldsymbol{K}}_i$. If the magnitude of the biggest eigenvalue of $\frac{1}{\delta_i}\bar{\boldsymbol{K}}_i$ is smaller than one, we can use Neumann's series to approximate this inverse, knowing that it would be convergent (e.g., see Neumann 1877; Suzuki 1976):

$$\begin{aligned}(\boldsymbol{I} + \frac{1}{\delta_i}\bar{\boldsymbol{K}}_i)^{-1} &= \sum_{j=0}^{\infty}(-\frac{1}{\delta_i}\bar{\boldsymbol{K}}_i)^j \\ &\approx \boldsymbol{I} - \frac{1}{\delta_i}\bar{\boldsymbol{K}}_i + \frac{1}{\delta_i^2}\bar{\boldsymbol{K}}_i^2 - \frac{1}{\delta_i^3}\bar{\boldsymbol{K}}_i^3 + ...\end{aligned} \quad . \tag{5.1.5}$$

The truncated series is often a pretty good approximation, as long as the spectral radius of $\frac{1}{\delta_i}\bar{\boldsymbol{K}}$ remains smaller than one. What we now propose is to use the penalty δ_i as an adaptive layer parameter that will ensure that it is the case. This idea is in line with the use of a damping factor to rescale a linear system before solving it with Chebyshev iteration, a special case of modified Richardson iteration (e.g. see Golub and Varga 2007)

We will show that this is a valid approach, but first, we need to show that the matrices $\boldsymbol{K}_i$ (and by extension $\bar{\boldsymbol{K}}_i$) are always symmetric:

Theorem 5.1.1. *The gain matrices $\boldsymbol{K}_i$ (or $\boldsymbol{K}$ in the infinite horizon setting) given by* (3.2.6) *(respectively* (4.3.2)*) are symmetric matrices, and therefore the matrices $\boldsymbol{B}_i^T \boldsymbol{K}_i \boldsymbol{B}_i$ (respectively $\boldsymbol{B}^T \boldsymbol{K} \boldsymbol{B}$) are also symmetric.*

Proof. The proof follows from the fact that the matrices $\boldsymbol{K}_i$ are a composition (via addition and subtraction) of symmetric matrices.

By Schwartz's Theorem (see Rudin 1976) we know that the partial second-order derivatives matrices $\boldsymbol{Q}_N, \boldsymbol{Q}_i$, $\boldsymbol{M}_i$ and $\boldsymbol{R}_i$ are symmetric matrices.

Moreover, if $\boldsymbol{K}_{i+1}$ is symmetric, then $\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i$ is also symmetric because:

$$(\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i)^T = \boldsymbol{A}_i^T \boldsymbol{K}_{i+1}^T \boldsymbol{A}_i = \boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i$$

The same argument holds for $\boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i$, and for $(\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i)^T(...)(\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i)$. Finally, remembering that the inverse of a symmetric matrix is also symmetric, we have that $\boldsymbol{K}_i$,

$$\boldsymbol{K}_i = \boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i + \boldsymbol{Q}_i - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1}(\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i)$$

is symmetric if $\boldsymbol{K}_{i+1}$ is also symmetric. This will always be the case as the terminal gain matrix, $\boldsymbol{K}_N = \boldsymbol{Q}_N$ is symmetric. □

We can now use the fact that $\bar{\boldsymbol{K}}_i$ is symmetric to show that we can indeed use the penalty term δ_i to ensure that the new matrix $\frac{1}{\delta_i}\bar{\boldsymbol{K}}_i$ will have a spectral radius smaller than one. Indeed, the symmetry of $\bar{\boldsymbol{K}}_i$ implies that its eigenvalues are real. It also means that its singular values, obtained via Singular Values Decomposition (SVD), are the absolute values of the eigenvalues (see corollary C.5.2 in Lebanon 2012). That is,

$$\bar{\boldsymbol{K}}_i = \bar{\boldsymbol{U}}_i \bar{\boldsymbol{\Sigma}}_i \bar{\boldsymbol{V}}_i^{-1} \ , \tag{5.1.6}$$

where $\bar{\boldsymbol{\Sigma}}_i$ is a diagonal matrix for which the entries are the magnitude of the eigenvalues of $\bar{\boldsymbol{K}}_i$. Therefore, if we denote the largest absolute singular value of $\bar{\boldsymbol{K}}_i$ by λ_m and take $\delta_i > |\lambda_m|$, then we will have that the SVD of $\frac{1}{\delta_i}\bar{\boldsymbol{K}}_i$ can be written as:

$$\frac{1}{\delta_i}\bar{\boldsymbol{K}}_i = \bar{\boldsymbol{U}}_i\Big(\frac{1}{\delta_i}\bar{\boldsymbol{\Sigma}}_i\Big)\bar{\boldsymbol{V}}_i^{-1} \ , \tag{5.1.7}$$

which is an expression that we can use to deduce that the magnitudes of the eigenvalues of $\frac{1}{\delta_i}\bar{\boldsymbol{K}}_i$ would be smaller than one and therefore the Neumann's series approximation will be convergent.

5.1.1 An Algorithm without Matrix Inversion

The proposed approach above means that we now need to compute the spectral radius (equivalent to its maximum absolute eigenvalues) of the matrix $\bar{\boldsymbol{K}}_i$. Although such calculation seems unusual in DL, it is not without precedent. Yoshida and Miyato 2017 for example regularize the weight matrix of the NN they trained by their spectral radius and claim better generalizability from it. Using the same strategy, Miyato *et al.* 2018 showed that it could be used to stabilize the training of the discriminator of a generative adversarial network. It is worth noting that both works use the power iteration method to estimate the spectral radius of their matrix of interest, a method that is relatively quick computation-wise.

For us, this means that we can leverage a second-order method optimization method to train a NN that has ReLu activation units without explicitly inverting a matrix. Moreover, the trick can be applied in the finite horizon setting and **in the infinite horizon one** (using the cyclic formulation for the latter).

5.1.2 Parallel with Damping Parameter

Applying the adaptive penalty δ_i may seem arbitrary at first glance, but if we cast it as a damping parameter over the weights update, it suddenly makes much more sense. The use of a damping parameter is advocated as critical by Martens 2016 and Martens 2020 for second-order optimization of NN because the direction that such algorithms propose in parameter space is not aligned with the distribution space. The resulting path will therefore more often than not veers away from the target pointed to by the second-order method.

To alleviate this problem, we need to dampen the size of the updates. One strategy to do so is via trust-region methods, i.e. by penalizing the update size made in some directions. In fact, that is exactly what the penalty term δ_i does; its introduction is equivalent to adding the following term to the cost of the LQR (3.2.5):

$$\sum_{i=0}^{N-1} \tilde{\boldsymbol{u}}_i^T \delta_i \tilde{\boldsymbol{u}}_i = \sum_{i=0}^{N-1} (\boldsymbol{u}_i - \boldsymbol{u}_i^k)^T \delta_i (\boldsymbol{u}_i - \boldsymbol{u}_i^k) \ .$$

Experimentally, we also used a parameter δ to stabilize the training for second-order methods implemented as a LQR without Neumann's series trick, in the same fashion as above. However, the term was kept constant across all layers and fine-tuned manually.

Setting δ_i via the spectral radius of $\bar{\boldsymbol{K}}_i$ allowed us to dampen adaptively the updates and offer superior performance than without its use. This finding is summarized in figure 5.1. As of now, we do not have a better explanation for this behavior than the intuition discussed above, and it will require further investigation from our side to quantify correctly the impact of δ_i on the training.

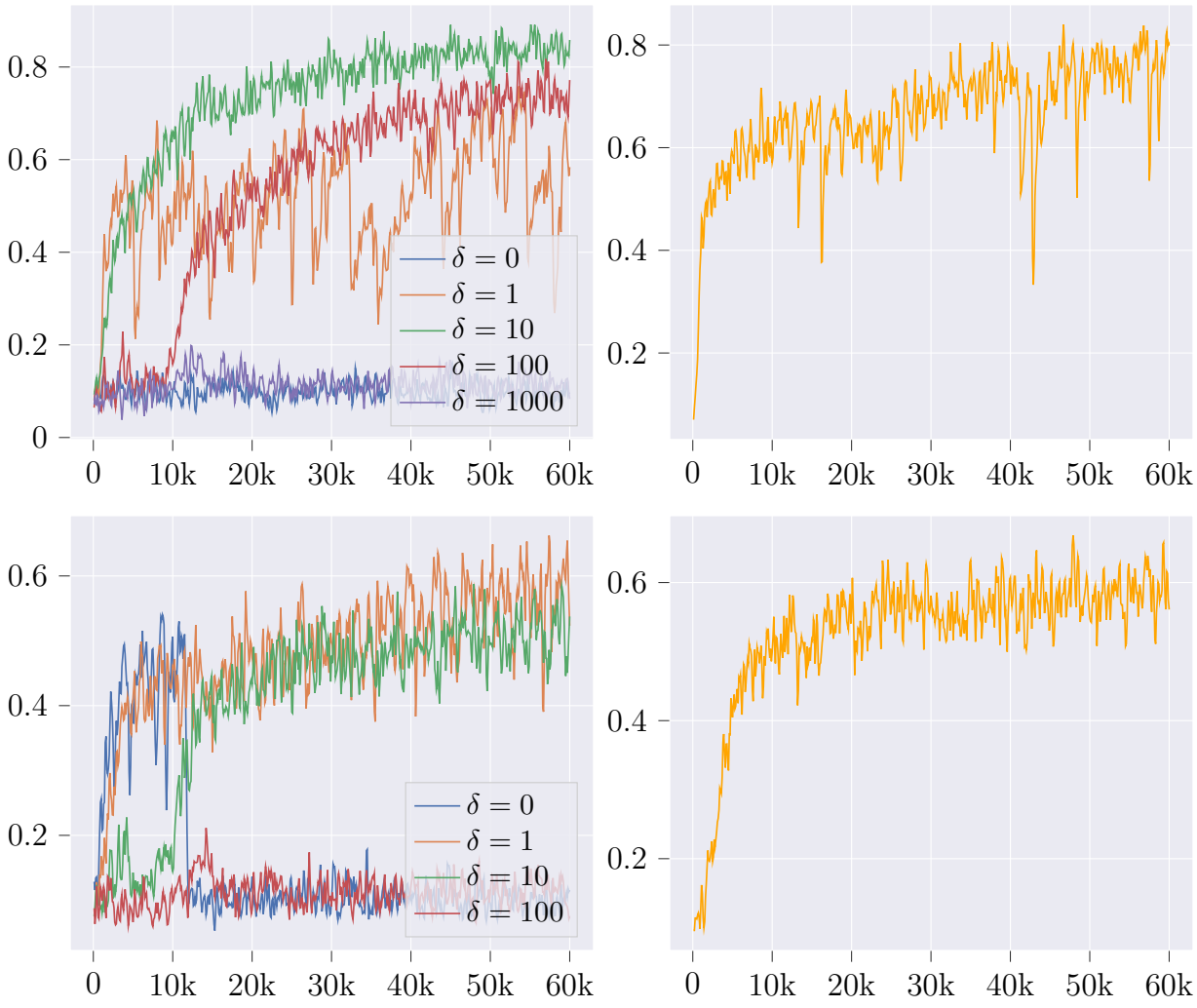

Figure 5.1. top: Accuracy curves of a training experiment, the same Convnet as in figure 3.1 (5 layers deep) trained on MNIST with the LQR approach equivalent to NGD and a learning rate set to $\alpha = 0.1$. **left:** In addition to the learning rate, a damping parameter δ has to be fine-tuned for the training to be stable without Neumann's series trick. **right:** On the opposite, implementing Neumann's series trick also has the added benefit of adding a layer adaptive damping term that stabilizes the training. **bottom:** Same as above, but for a learning rate set to $\alpha = 0.01$.

Chapter 6

Conclusion

Overall, the most important contribution of this thesis is to provide a template to express steepest descent under various divergence, both in the finite and infinite horizon cases, as a special LQR problem. Equipped with this new tool, a multitude of possible paths can now be explored. Most of our experimentation within our framework has been through second-order methods such as NGD and Newton's, but those only represent specific cases. At the moment, two major challenges are preventing us to apply our method at scale: the lack of full matrix-free implementation and stability issues (for which we may have a partial solution).

Matrix-Free Implementation: One of the major advantage of automatic differentiation is the fact that the method is *matrix free* (see Baydin *et al.* 2017). Jacobian matrices are never fully evaluated nor stored, and the computation time is much quicker. Unfortunately, the framework we have described above did not completely preserve this property, in both the finite and the infinite settings. This also means that the memory cost incurred by our algorithm is bigger than for standard backpropagation.

For the finite horizon described in chapter 3, nothing prevents us from formulating the algorithm as matrix-free; all Jacobian and Hessian matrices can be derived as matrix-vector product (MVP) functions. The problem arises from the potential combinatorial explosion in the expression of the matrices $\boldsymbol{K}_i$ in (3.2.6). Hence, if we were to evaluate $\boldsymbol{K}_0 \cdot \boldsymbol{v}$ for a NN with N layers, we would have to evaluate 4^{N-1} times the product $\boldsymbol{K}_N \cdot \boldsymbol{v}$, and then to evaluate 4^{N-2} times the product $\boldsymbol{K}_{N-1} \cdot \boldsymbol{v}$, etc. As such, the total number of MVP evaluations quickly becomes intractable when increasing the number of layers.

To avoid this problem, we currently compute and store the matrices $\boldsymbol{K}_i$. As those matrices are proportional to the size of the representation vector of their respective layer (n_i), they

each require $\mathcal{O}(n_i)$ MVP evaluations and a memory space that scale with $\mathcal{O}(n_i^2)$.

It should be noted that we could be strategic when designing the NN architecture that we want to optimize to mitigate this issue. In practice, only storing the $\boldsymbol{K}_i$ matrices every 2 or 3 layers would not incur too much slow down. Knowing this, one could include a features bottleneck in their NN design and only store/compute the $\boldsymbol{K}_i$ for those layers.

The issue is different in the infinite horizon setting, presented in chapter 4, since typical fast DARE solvers rely on doubling algorithms and therefore perform multiple matrix multiplications (see Bini *et al.* 2011). There is thus no gain in keeping everything as an MVP, and for this setting, we currently compute and store all Jacobians and Hessians when performing steepest descent (the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, and $\boldsymbol{M}$). At least, the implicit block in typical DEQ usually does not consist of many layers, so it is still feasible to do it that way.

Stability: The observed stability that occurs during training with the use of the damping parameter δ_i introduced in chapter 5 came as a surprise to us; we expected a bit of tuning to be necessary. This penalty term directly reduces the spectral radius of the gain matrices $\boldsymbol{K}_i$, but does it implicitly impact the spectral radius of the weight matrices? We plan to investigate further to see if it is the case as it could allow us to understand the stable behavior through the lens of spectral norm regularization (see Yoshida and Miyato 2017 and Miyato *et al.* 2018).

Also, as already mentioned, our formulation allowed us to train DEQ networks under a different divergence measure than the Euclidean norm; a formulation for which popular second-order methods are a specific case. Yet trying it with NGD, we did encounter stability issues during and extensive hyperparameter tuning was required. We hope that bringing the Neumann series trick in the infinite horizon setting will bring the same advantages noticed in the finite setting while allowing faster computation simultaneously.

6.1 Future Work

So far, we have limited our work to the field of discrete-time problems but we do believe it is worth exploring if the same connection can be made in the continuous-time setting. As a starting point, we could cast the optimization objective with mirror descent; a technique that was originally motivated in continuous time (see Nemirovsky and Yudi 1983). The solution of the mirror descent objective in continuous time under various divergence leads to continuous-time descent dynamics to follow to converge to an optimum (e.g. Krichene *et al.* 2015).

Moreover, the PMP (Pontriagin *et al.* 1962) was also first developed for control problems in continuous time; the optimality principle still allows us to design an objective that decouples the controls at every time by handing us a function that must be optimal at every time step. That is, the optimal trajectory will consist of the instantaneous optimal control of the Hamiltonian put end-to-end. We thus believe we will be able to cast the mirror descent objective as a continuous LQR (see Miquel 2020 for more details on the PMP and LQR in a continuous setting).

This would imply that a Continuous-Time Algebraic Riccati Equation (CARE) solver could be used to perform a mirror descent step. The discretization of the optimal trajectory obtained would then provide updates for the parameters of a given NN that preserves the convergence guarantee obtained via mirror descent. It could also be used directly to provide updates to a neural ODEs network (Chen *et al.* 2018). Again, since LQR problems have long been studied and since it exists efficient solver for them, we can hope to leverage those tools to obtain performance gain during the training of a neural net.

Another possible line of work stems from the question: does casting the steepest descent step as a LQR gives us a new tool to analyze the behavior of optimization technique? After all, LQR problems come with great convergence guarantees/conditions and analytical solutions. We do believe our framework opens a new door for the analysis of optimization algorithms, especially when a divergence term other than Euclidean distance is brought in to quantify the size of the updates.

Finally, we hope that establishing an equivalence between a steepest descent and LQR will now enable the design of new efficient optimization algorithms while providing new tools to study their theoretical properties.

References

Almeida, L. B. (1990). *A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment*, page 102–111. IEEE Press.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, **10**, 251–276.

Amari, S. and Nagaoka, H. (2007). *Methods of Information Geometry*. American Mathematical Society.

Anderson, B. D. O. and Moore, J. B. (1990). *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, Inc., USA.

Anderson, D. G. (1965). Iterative procedures for nonlinear integral equations. *J. ACM*, **12**(4), 547–560.

Bai, S., Kolter, J. Z., and Koltun, V. (2019). Deep equilibrium models. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 688–699.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, **18**, 153:1–153:43.

Beck, A. and Teboulle, M. (2003). Mirror descent and nonlinear projected subgradient methods for convex optimization. *Oper. Res. Lett.*, **31**(3), 167–175.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, **38**(8), 716–719.

Bertsekas, D. (2016). *Nonlinear Programming*. Athena scientific optimization and computation series. Athena Scientific.

Bini, D. A., Iannazzo, B., and Meini, B. (2011). *Numerical Solution of Algebraic Riccati Equations*. Society for Industrial and Applied Mathematics.

Bliss, G. A. (1968). *Lectures on the calculus of variations*. Chicago (Ill.) : University of Chicago press.

Bottou, L. (1998). Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK. revised, oct 2012.

Boyd, S. P. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press, Cambridge, UK ; New York.

Bryson, A. E. and Ho, Y.-C. (1969). *Applied optimal control: optimization, estimation, and control*. Blaisdell Publishing Co.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583.

De O. Pantoja, J. and Mayne, D. (1989). A sequential quadratic programming algorithm for discrete optimal control problems with control inequality constraints. In *Proceedings of the 28th IEEE Conference on Decision and Control,*, pages 353–357 vol.1.

Dreyfus, S. E. (1990). Artificial neural networks, back propagation, and the kelley-bryson gradient procedure. *Journal of Guidance, Control, and Dynamics*, **13**(5), 926–928.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, **12**(61), 2121–2159.

Dunn, J. and Bertsekas, D. (1989). Efficient dynamic programming implementations of newton's method for unconstrained optimal control problems. *Journal of Optimization Theory and Applications*, **63**, 23–38.

Gilbert, J. C. (1992). Automatic differentiation and iterative processes. *Optimization methods and software*, **1**(1), 13–21.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

Golub, G. H. and Varga, R. S. (2007). Chebyshev semi-iterative methods, successive over-relaxation iterative methods, and second-order richardson iterative methods, parts I and II. In *Milestones in Matrix Computation - Selected Works of Gene H. Golub, with Commentaries*, pages 45–67. Oxford University Press.

Goodfellow, I. J. and Vinyals, O. (2015). Qualitatively characterizing neural network optimization problems. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Griewank, A. and Naumann, U. (2003). Accumulating jacobians as chained sparse matrix products. *Math. Program.*, **95**(3), 555–571.

Haber, E. and Ruthotto, L. (2017). Stable architectures for deep neural networks. *Inverse Problems*, **34**(1), 014004.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Hestenes, M. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, **49**(6), 409.

Hinton, G. (2012). Neural networks for machine learning. Coursera, video lectures.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153.

Jin, W., Wang, Z., Yang, Z., and Mou, S. (2020). Pontryagin differentiable programming: An end-to-end learning and control framework. In *Advances in Neural Information Processing Systems*, volume 33, pages 7979–7992. Curran Associates, Inc.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Krichene, W., Bayen, A., and Bartlett, P. L. (2015). Accelerated mirror descent in continuous and discrete time. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. Y. (2011). On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 265–272. Omnipress.

Lebanon, G. (2012). *Probability: The Analysis of Data, Volume 1*. CreateSpace Independent Publishing Platform.

Lecun, Y. (1988). A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, pages 21–28. Morgan Kaufmann.

Lenhart, S. and Workman, J. T. (2007). *Optimal Control Applied to Biological Models*. Chapman and Hall/CRC.

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT*, **16**(2), 146–160.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, **45**(1), 503–528.

Martens, J. (2016). *Second-order optimization for neural networks*. University of Toronto (Canada).

Martens, J. (2020). New insights and perspectives on the natural gradient method. *J. Mach. Learn. Res.*, **21**, 146:1–146:76.

Martens, J. and Grosse, R. B. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2408–2417. JMLR.org.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1033–1040. Omnipress.

Miquel, T. (2020). Introduction to Optimal Control. Lecture.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Nayfeh, A. H. and Balachandran, B. (1995). *Equilibrium Solutions*, chapter 2, pages 35–145. John Wiley & Sons, Ltd.

Nemirovsky, A. S. and Yudi, D. B. (1983). *Problem Complexity and Method Efficiency in Optimization*. Wiley-Interscience series in discrete mathematics. Wiley.

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, **269**, 543–547.

Neumann, C. (1877). *Untersuchungen über das logarithmische und Newton'sche Potential*. BG Teubner.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York.

Osawa, K., Tsuji, Y., Ueno, Y., Naruse, A., Yokota, R., and Matsuoka, S. (2019). Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12359–12367. Computer Vision Foundation / IEEE.

Pascanu, R. and Bengio, Y. (2014). Revisiting natural gradient for deep networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Comput.*, **6**(1), 147–160.

Pineda, F. J. (1987). Generalization of backpropagation to recurrent and higher order neural networks. In *Proceedings of the 1987 International Conference on Neural Information Processing Systems*, NIPS'87, page 602–611, Cambridge, MA, USA. MIT Press.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, **4**(5), 1–17.

Pontriagin, L., Boltyanskii, V., Collection, K. M. R., Neustadt, L., Gamkrelidze, R., Trirogoff, K., Mishchenko, E., and Brown, D. (1962). *The Mathematical Theory of Optimal Processes*. International series of monographs in pure and applied mathematics. Interscience Publishers.

Raskutti, G. and Mukherjee, S. (2015). The information geometry of mirror descent. *IEEE Trans. Inf. Theory*, **61**(3), 1451–1457.

Reid, W. (1972). *Riccati Differential Equations*. Mathematics in science and engineering : a series of monographs and textbooks. Academic Press.

Robbins, H. (2007). A stochastic approximation method. *Annals of Mathematical Statistics*, **22**, 400–407.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, **abs/1609.04747**.

Rudin, W. (1976). *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill.

Seidman, J. H., Fazlyab, M., Preciado, V. M., and Pappas, G. J. (2020). Robust deep learning as optimal control: Insights and convergence guarantees.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA. PMLR.

Suzuki, N. (1976). On the convergence of neumann series in banach space. *Mathematische Annalen*, **220**(2), 143–146.

Yoshida, Y. and Miyato, T. (2017). Spectral norm regularization for improving the generalizability of deep learning. *ArXiv*, **abs/1705.10941**.

Zhang, D., Zhang, T., Lu, Y., Zhu, Z., and Dong, B. (2019). You only propagate once: Accelerating adversarial training via maximal principle. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 227–238.

Appendix A

Steepest descent – Minimization Without Constraints

Theorem A.0.1. *The steepest descent direction can be obtained via the following minimization problem.*

$$\boldsymbol{v}^* = \alpha\Big[\operatorname{argmin}_{\boldsymbol{v}}\{\nabla f(\boldsymbol{x})^T\boldsymbol{v} + \frac{1}{2}\|\boldsymbol{v}\|^2\}\Big] \tag{A.0.1}$$

Proof. First, let us suppose that $\boldsymbol{w}^*$ is the solution to normalized steepest descent problem, that is :

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{v}} \nabla f(\boldsymbol{x})^T\boldsymbol{v} \quad \text{s.t. } ||\boldsymbol{v}|| = 1$$

Now, in (A.0.1), we can reexpress $\boldsymbol{v}$ as $\boldsymbol{v} = t\boldsymbol{w}$, $t \geq 0$ being a scalar and with $\boldsymbol{w} = \boldsymbol{v}/\|\boldsymbol{v}\|$ With this substitution, we are facing a minimization problem over two variables, that we can try to optimize successively.
By first taking $\boldsymbol{w}$ as fixed and minimizing for t we have:

$$\hat{t} = \arg\min_{t}\{t\nabla f(\boldsymbol{x})^T\boldsymbol{w} + \frac{t^2}{2}\}$$

This is a quadratic problem for which the minimum can be obtained by putting the derivative to zero:

$$\begin{aligned} 0 &= t + \nabla f(\boldsymbol{x})^T\boldsymbol{w} \\ \implies t &= -\nabla f(\boldsymbol{x})^T\boldsymbol{w} \end{aligned}$$

This in turn implies that $\hat{t} = 0$ if we choose w s.t. $\nabla f(\boldsymbol{x})^T \boldsymbol{w} > 0$ since we have that $t \geq 0$. And therefore, to minimize $t\nabla f(\boldsymbol{x})^T \boldsymbol{w} + \frac{t^2}{2}$ we must pick a $\boldsymbol{w}$ s.t. $\nabla f(\boldsymbol{x})^T \boldsymbol{w} \leq 0$ and $\hat{t} = -\nabla f(\boldsymbol{x})^T \boldsymbol{w}$

Turning our attention toward the minimization over $\boldsymbol{w}$ we have:

$$\begin{aligned}
\hat{\boldsymbol{w}} &= \arg\min_{\boldsymbol{w}, \|\boldsymbol{w}\|=1} \{t\nabla f(\boldsymbol{x})^T \boldsymbol{w} + \frac{t^2}{2}\} \\
&= \arg\min_{\boldsymbol{w}, \|\boldsymbol{w}\|=1} \{t\nabla f(\boldsymbol{x})^T \boldsymbol{w}\} \\
&= \arg\min_{\boldsymbol{w}, \|\boldsymbol{w}\|=1} \{-(\nabla f(\boldsymbol{x})^T \boldsymbol{w})^2\} \\
&= \arg\max_{\boldsymbol{w}, \|\boldsymbol{w}\|=1} \{|\nabla f(\boldsymbol{x})^T \boldsymbol{w}|\}
\end{aligned}$$

A problem for which we already know the solutions, that is $\hat{\boldsymbol{w}} = \pm \boldsymbol{w}^*$ But since we require $\nabla f(\boldsymbol{x})^T \boldsymbol{w} \leq 0$, we get that $\hat{\boldsymbol{w}} = \boldsymbol{w}^*$.

Therefore, we have that:

$$\begin{aligned}
\arg\min_{\boldsymbol{v}} \{\nabla f(\boldsymbol{x})^T \boldsymbol{v} + \frac{1}{2}\|\boldsymbol{v}\|^2\} &= \hat{t}\hat{\boldsymbol{w}} \\
&= -\nabla f(\boldsymbol{x})^T \hat{\boldsymbol{w}}\hat{\boldsymbol{w}} \\
&= -\nabla f(\boldsymbol{x})^T \boldsymbol{w}^* \boldsymbol{w}^* \\
&= \alpha \boldsymbol{w}^* \\
&\quad \text{with } \alpha = -\nabla f(\boldsymbol{x})^T \boldsymbol{w}^* \geq 0
\end{aligned}$$

□

Appendix B

Riccati's Equation – Solution with a Linear Term in the Final Cost

Since an exhaustive development of the discrete Riccati equations for the case of an LQR with a linear term in final cost was hard to come by, we provide one below.

We seek the Riccati equation for the following problem, which is the problem we encounter when transforming the steepest descent step into a LQR:

$$\begin{aligned}
&\min_{\boldsymbol{u}} \;\; \boldsymbol{a}_N \boldsymbol{x}_N + \boldsymbol{x}_N^T \boldsymbol{Q}_N \boldsymbol{x}_N \sum_{i=0}^{N-1} \Big[\frac{1}{2}\boldsymbol{x}_i^T \boldsymbol{Q} \boldsymbol{x}_i + \frac{1}{2} u_i^T \boldsymbol{R} \boldsymbol{u}_i + \boldsymbol{u}_i^T \boldsymbol{M} \boldsymbol{x}_i\Big] \\
&\text{subject to } \;\; \boldsymbol{x}_{i+1} = \boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{B}_i \boldsymbol{u}_i \\
&\text{given } \;\; \boldsymbol{x}_0
\end{aligned}$$

With the Hamiltonian functions given by :

$$\begin{aligned}
H_i(\boldsymbol{x}_i, \boldsymbol{u}_i) &= \boldsymbol{x}_i^T \boldsymbol{Q} \boldsymbol{x}_i + \frac{1}{2} u_i^T \boldsymbol{R} \boldsymbol{u}_i + \boldsymbol{u}_i^T \boldsymbol{M} \boldsymbol{x}_i + \boldsymbol{p}_{i+1}\Big(\boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{B}_i \boldsymbol{u}_i\Big) \\
H_N(\boldsymbol{x}_N) &= \boldsymbol{a}_N \boldsymbol{x}_N + \boldsymbol{x}_N^T \boldsymbol{Q}_N \boldsymbol{x}_N
\end{aligned}$$

The adjoints are defined by $\boldsymbol{p}_i = \nabla_{\boldsymbol{x}_i} H_i$, and in particular when evaluated at the optimal pair $(\boldsymbol{x}_i^*, \boldsymbol{u}_i^*)$:

$$\begin{aligned}
\boldsymbol{p}_i &= \nabla_{\boldsymbol{x}_i} H_i(\boldsymbol{x}_i^*, \boldsymbol{u}_i^*) \\
&= \boldsymbol{Q}_i \boldsymbol{x}_i^* + \boldsymbol{M}_i^T \boldsymbol{u}_i^* + \boldsymbol{A}_i^T \boldsymbol{p}_{i+1}^*
\end{aligned} \tag{B.0.1}$$

They are also chosen such that they satisfy $\nabla_{\boldsymbol{u}_i} H_i = 0$, so that:

$$
\begin{aligned}
0 &= \nabla_{\boldsymbol{u}_i} H_i(\boldsymbol{x}_i^*, \boldsymbol{u}_i^*) \\
&= \boldsymbol{R}_i \boldsymbol{u}_i^* + \boldsymbol{M}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{p}_{i+1}^* \\
&\implies \boldsymbol{u}_i^* = -\boldsymbol{R}_i^{-1}(\boldsymbol{M}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{p}_{i+1}^*)
\end{aligned}
$$

We now want to prove by induction that we can rewrite $\boldsymbol{p}_i^*$ as:

$$
\boldsymbol{p}_i^* = \boldsymbol{K}_i \boldsymbol{x}_i^* + \boldsymbol{\lambda}_i \tag{B.0.2}
$$

$\forall i = 0,...,N-1$ starting with $i = N-1$ with $\boldsymbol{p}_N^* = \boldsymbol{Q}_N \boldsymbol{x}_N^* + \boldsymbol{a}_N$
Supposing it holds for $i+1$, we have that:

$$
\begin{aligned}
\boldsymbol{u}_i^* &= -\boldsymbol{R}_i^{-1}(\boldsymbol{M}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{x}_{i+1}^* + \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1} \\
&= -\boldsymbol{R}_i^{-1}(\boldsymbol{M}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1}(\boldsymbol{A}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i \boldsymbol{u}_i^*) + \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1} \\
\boldsymbol{R}_i \boldsymbol{u}_i^* + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i \boldsymbol{u}_i^* &= -(\boldsymbol{M}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1}) \\
\boldsymbol{u}_i^* &= -(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1}((\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1})
\end{aligned} \tag{B.0.3}
$$

Using this result, we can rewrite $\boldsymbol{x}_{i+1}^*$ as :

$$
\boldsymbol{x}_{i+1}^* = \boldsymbol{A}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i \boldsymbol{u}_i^* = \boldsymbol{A}_i \boldsymbol{x}_i^* - \boldsymbol{B}_i(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1}((\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1})
$$

Multiplying each side by $\boldsymbol{K}_{i+1}$ and then adding $\boldsymbol{\lambda}_{i+1}$ we get from (B.0.2):

$$
\boldsymbol{p}_{i+1}^* = \boldsymbol{K}_{i+1} \boldsymbol{A}_i \boldsymbol{x}_i^* - \boldsymbol{K}_{i+1} \boldsymbol{B}_i(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1}((\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1}) + \boldsymbol{\lambda}_{i+1}
$$

Now multiplying each side by $\boldsymbol{A}_i^T$:

$$
\boldsymbol{A}_i^T \boldsymbol{p}_{i+1}^* = \boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i \boldsymbol{x}_i^* - \boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1}((\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1}) + \boldsymbol{A}_i^T \boldsymbol{\lambda}_{i+1}
$$

And finally adding $(\boldsymbol{Q}_i \boldsymbol{x}_i^* + \boldsymbol{M}_i^T \boldsymbol{u}_i^*)$ on each side, replacing $\boldsymbol{u}_i^*$ its expression in (B.0.3):

$$
\begin{aligned}
\boldsymbol{p}_i^* &= \boldsymbol{Q}_i \boldsymbol{x}_i^* + \boldsymbol{M}_i^T \boldsymbol{u}_i^* + \boldsymbol{A}_i^T \boldsymbol{p}_{i+1}^* \quad \text{from (B.0.1)} \\
&= \Big(\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i + \boldsymbol{Q}_i - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} (\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \Big) \boldsymbol{x}_i^* \\
&\quad + \boldsymbol{A}_i^T \boldsymbol{\lambda}_{i+1} - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1}
\end{aligned}
$$

Completing the induction proof while also giving convenient expressions for $\boldsymbol{K}_i$ and $\boldsymbol{\lambda}_i$:

$$
\boldsymbol{K}_i = \boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i + \boldsymbol{Q}_i - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} (\boldsymbol{M}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i)
\tag{B.0.4}
$$

$$
\boldsymbol{\lambda}_i = \boldsymbol{A}_i^T \boldsymbol{\lambda}_{i+1} - (\boldsymbol{A}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i + \boldsymbol{M}_i^T)(\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1}
\tag{B.0.5}
$$

The optimal trajectory can then be recovered from the system dynamics and from (B.0.3):

$$
\boldsymbol{u}_0^* = -(\boldsymbol{R}_0 + \boldsymbol{B}_0^T \boldsymbol{K}_1 \boldsymbol{B}_0)^{-1} \boldsymbol{B}_0^T \boldsymbol{\lambda}_1
\tag{B.0.6}
$$

$$
\boldsymbol{x}_{i+1}^* = \boldsymbol{A}_i \boldsymbol{x}_i^* + \boldsymbol{B}_i \boldsymbol{u}_i^*
\tag{B.0.7}
$$

$$
\boldsymbol{u}_i^* = (\boldsymbol{R}_i + \boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{B}_i)^{-1} ((\boldsymbol{B}_i^T \boldsymbol{K}_{i+1} \boldsymbol{A}_i) \boldsymbol{x}_i^* + \boldsymbol{B}_i^T \boldsymbol{\lambda}_{i+1})
\tag{B.0.8}
$$