Summer School, Peking University, 2020

Dynamical Systems and Machine Learning

Qianxiao Li

20–24 July 2020

Preface

These lecture notes are compiled for the summer school on *Machine Learning and Dynamical Systems* at Peking University, 20–24 July 2020.

This document is typeset using LaTeXwith a modified theme based on

https://www.overleaf.com/latex/templates/lecture-note-template/dwyrjrnthdcz

If you find any mistakes or typos in the notes, please send me an email at qianxiao@nus.edu.sg.

Contents

1 Introduction **5**
1.1 Overview . . . 5
1.2 Supervised Learning and Neural Networks . . . 6
1.2.1 The Basic Supervised Learning Problem . . . 6
1.2.2 Example: Linear Models . . . 8
1.2.3 The Neural Network Hypothesis Space . . . 11
1.2.4 Optimizing Neural Networks . . . 13
1.2.5 Deep Neural Networks . . . 15
1.3 Ordinary Differential Equations . . . 16
1.3.1 Basic Definitions . . . 17
1.3.2 Flow Map and Dependence on Initial Condition . . . 18
1.3.3 Numerical Solution of ODEs . . . 19

2 Optimal Control Theory **21**
2.1 From Calculus of Variations to Optimal Control . . . 21
2.1.1 A Motivating Example . . . 21
2.1.2 The Problem of Optimal Control . . . 23
2.1.3 Weak vs Strong Minima . . . 23
2.1.4 A Dynamical View on the Calculus of Variations . . . 25
2.1.5 The Optimal Control Formulation . . . 26
2.2 Pontryagin's Maximum Principle . . . 27
2.2.1 The Maximum Principle . . . 28
2.2.2 Other Forms of the Maximum Principle . . . 31
2.2.3 Further Reading . . . 33
2.3 Hamilton-Jacobi-Bellman Equations . . . 34
2.3.1 Motivating Example of Dynamic Programming . . . 34
2.3.2 The Dynamic Programming Principle . . . 35
2.3.3 Hamilton-Jacobi-Bellman Equations . . . 37
2.3.4 Implications for Optimal Control . . . 39
2.3.5 Further Reading . . . 42

3 Dynamical Systems Meets Deep Learning **43**
3.1 A Mean-field Optimal Control Formulation of Deep Learning . . . 43
3.2 Optimality Conditions . . . 45
3.2.1 Mean-field Pontryagin's Maximum Principle . . . 45
3.2.2 Mean-field Hamilton-Jacobi-Bellman Equations . . . 46

3.3 Control Inspired Learning Algorithms . . . 47
3.3.1 Method of Successive Approximations . . . 47
3.3.2 Layer Parallel Training Algorithms . . . 48
3.3.3 Summary and Outlook . . . 50
3.4 Control Inspired Architectures . . . 51
3.4.1 Constraining Weights to Guarantee Stability . . . 51
3.4.2 Architectures from Other Finite Difference Discretization Schemes . . . 53
3.4.3 Architectures from PDE Theory . . . 54
3.4.4 Summary and Outlook . . . 55
3.5 Mathematical Results from the Dynamical Systems Approach . . . 56
3.5.1 Approximation Theory . . . 56
3.5.2 Generalization . . . 57
3.5.3 Connection between Continuous and Discrete Time . . . 58
3.5.4 Summary and Outlook . . . 59

4 Summary **61**

1 Introduction

1.1 Overview

With recent advancements in algorithms and computational hardware, deep learning [LBH15] has risen to the pinnacle of many practical applications, including image analysis, natural language processing and game playing, etc. However, on the theoretical side, we are only at the nascent stages of exploration. In a sense, deep learning is about learning representations, yet the way it does so is quite peculiar: it relies on repeated transformations through the deep layers of a neural network to disentangle features and learn very complex representations. Mathematically, these transformations act together via *compositions*, which connects this to the classical field of study known as dynamical systems [Bir27]. How does this connection help us to unveil certain aspects of deep learning?

These notes presents an pedagogical overview of the connection between dynamical systems and machine learning. Here, the theory of optimal control acts as a bridge between calculus of variations on the one hand, and training deep networks on the other. Hence, we will use the first half of the notes to introduce the basic theory of optimal control, including the central results of Pontryagin and Bellman. This lays the basic theoretical groundwork for its applications to deep learning research. As there are many references to these topics, the focus here is not to present the theory in its utmost generality. Instead, we will sacrifice generality (and sometimes, a bit of rigor) in favor of simplicity and transparency. Nevertheless, where relevant further reading references will be provided for readers interested in the general theory of calculus of variations and optimal control.

In the second part of these notes, we will discuss some applications of the dynamical systems viewpoint on deep learning, including mathematical formulations, optimality conditions, learning algorithms and model architectures. The content in this part is fairly new and our understanding is still far from complete. Hence, the goal of this part is to survey recent research in this direction, as well as point out the limitations and a host of future research directions that one may pursue.

The reader is assumed to have a basic familiarity with linear algebra, calculus/analysis and probability. Knowledge of machine learning, differential equations, numerical analysis and optimization is highly desirable, but the relevant ideas will be introduced along the way, with reference provided for further reading or review.

1.2 Supervised Learning and Neural Networks

We start by giving a quick review of supervised learning and neural networks. The discussion is rather brief and the reader is referred to [BO06, MRT18] (and for deep learning, [GBC16]) for a more thorough introduction.

1.2.1 The Basic Supervised Learning Problem

We begin with a brief introduction of the problem statement of supervised learning. Supervised learning is perhaps the most basic class of machine learning problems. Here, we are given a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$ consisting of *inputs* x_i with their corresponding *labels* y_i and N is the size of the data. The underlying assumption is that each y_i is determined by x_i through some target function F^*, i.e. $y_i = F^*(x_i)$. In classification problems, the function F^* is sometimes called the *oracle*, carrying the meaning that it can determine perfectly the label of any sample presented to it. More generally, one can take into account of noise and uncertainties by assuming that given x_i, y_i is a sample from some "target" conditional distribution $y_i \sim p^*(\cdot|x_i)$. The most common model for this case is when $y_i = F^*(x_i) + \epsilon_i$ with ϵ_i representing some random noise term. For the sake of simplicity, for now we shall discuss supervised learning in the deterministic context. When the labels y_i take values in a continuum, say in $\mathbb{R}$, we say that this is a *regression* problem. Otherwise, if y_i take discrete values, we say that this is a *classification* problem.

The target F^* is unknown to us except from the information contained in the dataset $\mathcal{D} = \{x_i, y_i = F^*(x_i)\}_{i=1}^N$. As such, the over-arching goal of supervised learning is to construct, using $\mathcal{D}$, a good approximation of the target. The word *supervised* means that in our dataset $\mathcal{D}$, the correct label is provided to us as a form of supervision by F^* in our learning process.

So, how can we go about constructing such a predictive model? Notice that without explicit knowledge of F^* and from mere observations of $\mathcal{D}$, it is not clear how we can even represent F^*, say on a computer. Consequently, this motivates the following approach: we take a collection of functions that we *can* represent on a computer or even a piece of paper; From this collection we *pick* one F that "best approximates" F^* in some sense – F is then taken as our learned predictive model. This collection of functions, which is our job to decide, is called the *hypothesis space* and we will denote it by $\mathcal{H}$.

What is missing from the above discussion is how we pick a particular function F to approximate F^*. Clearly, it relies on a precise definition of "best approximation". This is where the concept of *loss functions* comes in. In abstract terms, we want to have a notion of how close any candidate F is to the target F^*. For classification problems, a reasonable measure of the closeness of another classifier F to the target F^* is

$$R(F) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{F(x_i) \neq F^*(x_i)}, \tag{1.1}$$

where $\mathbb{1}_c$ is the *indicator function* which equals 1 if condition c is true and 0 otherwise. Thus,

$R(F)$ is the *accuracy* of f on the training dataset $\mathcal{D}$. In statistical language, $R(F)$ is also called an *empirical risk* associated with the predictor F.

More generally, the closeness of F to F^* can be defined by a suitable *loss function* Φ, so that

$$R(F) = \frac{1}{N}\sum_{i=1}^{N} L(F(x_i), F^*(x_i)) = \frac{1}{N}\sum_{i=1}^{N} \Phi(F(x_i), y_i) \tag{1.2}$$

and we want $\Phi(y', y)$ to decrease as y and y' becomes closer. For the example given in (1.1), we have $\Phi(y', y) = \mathbb{1}_{y \neq y'}$ and this is known as the *zero-one loss*. For various reasons which will become clear, the zero-one loss is not often used, and we will consider a variety of other loss functions in both classification and regression problems. For example, a commonly used loss for regression is the *square loss* $\Phi(y', y) = \frac{1}{2}\|y' - y\|^2$.

Once a suitable loss function (and hence a notion of distance) is defined, the supervised learning problem can now be formalized as an optimization problem

$$\min_{F \in \mathcal{H}} R(F), \tag{1.3}$$

and if we can solve this problem, a minimizer $\widehat{F} \in \mathcal{H}$ of (1.3) is then our obtained predictive model. The process of finding $\widehat{F}$ by minimizing $R(F)$ (which depends on the data $\mathcal{D}$) is called *training*, and $\widehat{F}$ is called a trained model. Hopefully, $\widehat{F}$ performs our task of predicting label from inputs adequately, in which case we have succeeded at this supervised learning task.

Empirical Risk Minimization vs Population Risk Minimization. The preceding discussion, in particular Eq. (1.3) appears to suggest that machine learning is in some sense equivalent to an optimization problem. However, this is not so. In fact, Eq. (1.3) is *not* the actual problem we want to solve. To see this, simply observe that we can easily come up with a $\widehat{F}$ that minimizes the zero-one loss in the digit recognition problem – we simply memorize the label associated with each image x_i, $i = 1, 2, \ldots, N$ found in the training data, i.e.

$$\widehat{F}(x) = \begin{cases} y_i & x = x_i \text{ for some } i = 1, 2, \ldots, N \\ \text{anything} & \text{otherwise} \end{cases}. \tag{1.4}$$

Obviously, $R(\widehat{F}) = 0$ but $\widehat{F}$ is not what we want. What we really want is $\widehat{F}$ to perform well on *new* examples not found in, but distributed identically as, the original training dataset. In mathematical terms, what we really want to solve is the *population risk minimization* problem

$$\min_{F \in \mathcal{H}} R_{\text{pop}}(F) = \mathbb{E}_{x \sim \mu} L(F(x), F^*(x)), \tag{1.5}$$

where μ denotes the probability distribution from which the samples $\{x_i\}_{i=1}^N$ are sampled from. More generally, μ is used to denote a *joint* distribution of the input and label, and so we have

$$\min_{F \in \mathcal{H}} R_{\text{pop}}(F) = \mathbb{E}_{(x,y) \sim \mu} L(F(x), y), \tag{1.6}$$

which includes (1.5) as a special case if $\mu(x, y) \to \mu(x)\delta_{F^*(x)=y}$, but also includes more general cases where the label can depend stochastically on the input.

In contrast, (1.3) is an *empirical risk minimization* problem, which we can rewrite as

$$\min_{F \in \mathcal{H}} R_{\text{emp}}(F) = \frac{1}{N} \sum_{i=1}^{N} L(F(x_i), F^*(x_i)) \qquad x_i \overset{\text{i.i.d.}}{\sim} \mu. \tag{1.7}$$

However, in practice we cannot represent the sample distribution μ, and hence we often resort to solving (1.7) in place of (1.5). Nevertheless, it must be stressed that a good solution of (1.5) is what we are really after. The difference between the solutions of these problems is the study of *generalization*, which sets learning problems apart from pure optimization problems.

Three Paradigms of Supervised Learning. Now that we have formalized the basic problem of supervised learning, it is natural to discuss what sort of questions can we ask in machine learning theory and practice. In a sense, these questions can be grouped into three large categories: *approximation*, *optimization* and *generalization*. Below we list some central questions in each of these aspects.

1. Approximation – *How large is our hypothesis space $\mathcal{H}$?*. In particular, does it include, or at least contain functions that are very close to our target F^*? This is in fact the study of *approximation theory* and some of *harmonic analysis* [DP07, Mal09], although there are also many modern developments, particularly in the area of deep learning.
2. Optimization – *How can we find or get close to an approximation $\widehat{F}$ of F^*?*. This is indeed the empirical risk minimization problem, and questions include the design of large-scale optimization algorithms, their convergence analysis and their efficient implementation. Many methods are extensions of classical methods in convex optimization [Nes04]. See [BCN18] for a modern review. This will be the primary way optimal control theory comes into the picture.
3. Generalization – *Can the $\widehat{F}$ found generalize to unseen examples?* This concerns the fundamental interaction between the size of the data and the complexity of our hypothesis space. In fact, this question is the focus of classical statistical learning theory [FHT01].

Figure 1.1 gives an illustration of these questions. Of course, some of these concepts also apply beyond supervised learning framework.

1.2.2 Example: Linear Models

To illustrate supervised learning, it is useful to introduce the simplest case of *linear models* or *linear basis models*. Consider the dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$ where each input $x_i \in \mathbb{R}^d$ is a vector in d dimensions and scalar label $y_i \in \mathbb{R}$. Consider the hypothesis space

$$\mathcal{H} = \left\{ F : F(x) = \sum_{j=0}^{M-1} w_j \phi_j(x) \right\} \tag{1.8}$$

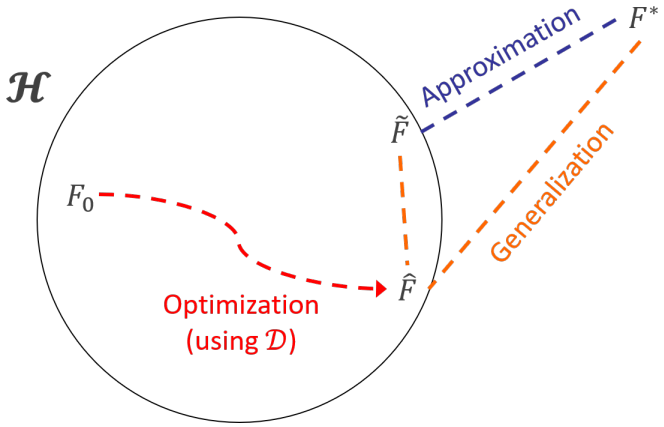


Figure 1.1: A schematic showing the three theoretical paradigms of supervised learning. *Approximation* studies the distance between the best approximator $\widehat{F}$ in our hypothesis space $\mathcal{H}$ and the target F^*. *Optimization* studies the process at which you arrive at or close to $\widehat{F}$, by using the training dataset $\mathcal{D}$ and starting from some initial guess f_0. *Generalization* problems arise because the dataset is finite and so the optimization on training set finds not $\widehat{F}$ but some other $\widetilde{f}$, and we must quantify the distance between them. In fact, what we are really interested in is the distance between $\widehat{F}$ and F^* when it comes to generalization.

where each ϕ_j is a function from $\mathbb{R}^d$ to $\mathbb{R}$. These are called *basis functions* or sometimes *feature maps*, for the reason that their role is to extract some feature from the input data x that is useful for predicting y. We have freedom to define what functions $\{\phi_j\}$ we use.

Observe that this is includes linear regression, if we set $\phi_0(x) = 1$, $\phi_i(x) = x_i$ for $i = 1, \dots, d$, but it also includes other more general spaces. In fact, there are many choices of ϕ_j's, and we give some examples below in the $d = 1$ case:

$$\text{Polynomial basis} \qquad \phi_j(x) = x^j \tag{1.9}$$

$$\text{Gaussian basis} \qquad \phi_j(x) = \exp\left(-\frac{(x - m_j)^2}{2s^2}\right) \tag{1.10}$$

$$\text{Sigmoid basis} \qquad \phi_j(x) = \sigma\left(\frac{x - m_j}{s}\right) \qquad \sigma(b) = \frac{1}{1 + e^{-b}} \tag{1.11}$$

There are in fact many more choices, include splines [DBRDB78], fourier basis, wavelet basis [Mal09] and many more – even neural networks as we will encounter later on.

Ordinary Least Squares. Recall that the empirical risk minimization problem we want to solve here is

$$\min_{w_0,\dots,w_{M-1}} R_{\text{emp}}(w_0,\dots,w_{M-1}) = \frac{1}{2N}\sum_{i=1}^{N}\left(\sum_{j=0}^{M-1} w_j\phi_j(x_i) - y_i\right)^2. \tag{1.12}$$

We can write the above much more compactly as

$$\min_{w\in\mathbb{R}^M} R_{\text{emp}}(w) = \frac{1}{2N}\|\Psi w - y\|^2, \tag{1.13}$$

where we have defined

$$w = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{M-1} \end{pmatrix}, \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \qquad \Psi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_{M-1}(x_N) \end{pmatrix}, \tag{1.14}$$

and $\|\cdot\|$ denotes the usual Euclidean norm. This allows us to derive the ordinary least squares formula by solving $\nabla R_{\text{emp}}(\widehat{w}) = 0$ for $\widehat{w}$.

Proposition 1.1: Ordinary Least Squares Formula

Suppose $\Psi^\top\Psi$ is invertible, then the solution of (1.13) is

$$\widehat{w} = (\Psi^\top\Psi)^{-1}\Psi^\top y. \tag{1.15}$$

Proof: We have $\nabla R_{\text{emp}}(\widehat{w}) = \frac{1}{N}\Psi^\top(\Psi\widehat{w} - y)$. Setting the right hand side to 0 gives $\Psi^\top\Psi\widehat{w} = \Psi^\top y$. Solving for $\widehat{w}$ gives the required solution. □

The OLS formula (1.15) can be seen as a *necessary condition for optimality* for the empirical risk R_{emp}. Here, it turns out to also be sufficient. This is because the function $w \mapsto R_{\text{emp}}(w)$ is smooth and convex, and hence all stationary points are automatically global minima.

Singular Case and Regularization. Proposition 1.1 requires $\Psi^\top\Psi$ to be invertible, but what happens if this is not the case? Note that Ψ is a $N \times M$ matrix whose rank is at most $\max(N, M)$. Suppose $N \geq M$, which is the case when the number of samples is greater than or equal to the number of features, or equivalently, the "complexity" of our hypothesis class. Since $\Psi^\top\Psi$ is a $M \times M$ matrix, it is invertible as long as the columns of Ψ are linearly independent, which we should be able to satisfy by appropriate choices of the ϕ_j's. What happens if $N < M$? In this case the rank of $\Psi^\top\Psi$ is at most N which is smaller M and so it is non-invertible, or *singular*.

So, what is the solution of the least squares problem now? It turns out that there is not one, but an *infinite* number of solutions. They are given by

$$\{\widehat{w}(u) : u \in \mathbb{R}^M\} \quad \text{where} \quad \widehat{w}(u) = \Psi^\dagger y + (I - \Psi^\dagger\Psi)u. \tag{1.16}$$

The matrix $\Psi^\dagger$ denotes the *Moore–Penrose pseudoinverse* [GVL96] of Ψ. Moreover, for any of these solutions, we have $R_{\text{emp}}(\widehat{w}(u)) = 0$, i.e. our training data is perfectly fit. This may not be good in machine learning scenarios.

Exercise 1.2

Show that if $\Psi^\top\Psi$ is invertible then $\widehat{w}(u)$ from (1.16) reduces to the ordinary least squares solution (1.15) for any u. (Hint: recall that for a matrix A, if $A^\top A$ is invertible then $A^\dagger = (A^\top A)^{-1}A^\top$. If you do not remember this, review the basics of pseudoinverse in any linear algebra reference, e.g. [GVL96]).

The fact that we have an infinite number of solutions is often not good as we typically need to pick one to perform our predictions. Which one do we pick? One choice is to pick one that has the smallest norm $\|\widehat{w}(u)\|$, which is $\widehat{w}(0) = \Psi^\dagger y$. More generally, we can consider adding to the empirical risk minimization problem a *regularization* term

$$\min_{w\in\mathbb{R}^M} \frac{1}{2N}\|\Psi w - y\|^2 + \lambda C(w) \tag{1.17}$$

where $C : \mathbb{R}^M \to \mathbb{R}_+$ is the regularization function and $\lambda > 0$ controls its strength. If we pick $C(w) = \|w\|^2$ (ℓ^2 regularization), then we get the unique solution $\widehat{w} = \Psi^\dagger y$ if $\Psi^\top\Psi$ is singular. This is also known as *ridge regression*. However, we can also use other types of regularization, such as $C(w) = \|w\|_1$ (ℓ^1 regularization). In this case, we actually find a *sparse* solution, i.e. many of entries $\widehat{w}$ are 0. This is called *lasso* [FHT01] in the statistics literature and is also related to the field of *compressed sensing* [Don06], which has many interesting applications, including fast Magnetic Resonance Imaging (MRI) [LDP07]. Moreover, regularization can also reduce overfitting, as the following example shows.

1.2.3 The Neural Network Hypothesis Space

Contrary to the linear models, *neural networks* constitutes a class of adaptive basis models, in which the basis functions are fitted according to data. The term "neural networks" originated from the fact that these models were first developed as an attempt to model, in a mathematically precise way, neural interactions in the human brain [MP43, WH60]. However, it became clear quickly that such models are over-simplified and lack complexity required to understand human physiology. Nevertheless, they form a class of powerful machine learning models that admit unique properties that are worth studying. In the following, we start with the basics of shallow neural networks.

Shallow Neural Networks. A shallow neural network corresponds to the following hypothesis space

$$\mathcal{H}_M = \left\{ F : F(x) = \sum_{j=1}^{M} v_j \sigma(w_j^\top x + b_j), w_j \in \mathbb{R}^d, v_j \in \mathbb{R}, b_j \in \mathbb{R} \right\} \tag{1.18}$$

Let us now introduce some nomenclature that is customary in the study of neural networks. The function $\sigma : \mathbb{R} \to \mathbb{R}$ is called an *activation function*. Popular choices include

ReLU (Rectified Linear Unit)	$\sigma(z) = \max(0, z)$	(1.19)
Leaky ReLU	$\sigma(z) = \max(0, z) + \delta \min(0, z)$	(1.20)
Tanh	$\sigma(z) = \tanh(z)$	(1.21)
Sigmoid	$\sigma(z) = \dfrac{1}{1 + e^{-z}}$	(1.22)
Soft-plus	$\sigma(z) = \log(1 + e^z)$	(1.23)

but this list is of course not exhaustive. Next, the parameters w_j are often called *weights* and b_j are called *biases*. Recall that in linear models, we tend to combine them by appending "1" to the input state x. However, here we will write out the bias term explicitly to conform with popular notation. We will refer to v_j as *coefficients*, but in deeper models they can also be regarded as weights. Finally, the number M is the dimensional of the *hidden layer* and this controls the complexity of the model. Often, we refer to $h_j = w_j^\top x + b_j$ as the value on the j^{th} *hidden node*. Thus, M is the number of hidden nodes in the neural network.

Universal Approximation Theorem. We now discuss a foundational result in the approximation theory of neural networks. This is known as the *universal approximation theorem*. In words, it says that given enough hidden nodes, a neural network can approximate *any* function to *arbitrary* accuracy. Let us give the precise statement of this result below.

Theorem 1.3: Universal Approximation Theorem for Neural Networks

Let $K \subset \mathbb{R}^d$ be closed and bounded and $F^* : K \to \mathbb{R}$ be continuous. Assume that the activation function σ is *sigmoidal*, i.e. σ is continuous and $\lim_{z\to\infty} \sigma(z) = 1$, $\lim_{z\to-\infty} \sigma(z) = 0$. Then, for every $\epsilon > 0$ there exists $F \in \cup_{M\geq 1} \mathcal{H}_M$ such that

$$\|F - F^*\|_{C(K)} = \max_{x\in K} |F(x) - F^*(x)| < \epsilon \tag{1.24}$$

A general proof of Theorem 1.3 follows from an application of the Hahn-Banach theorem and the Riesz-Markov representation theorem, together with an argument based on the non-degenerate properties of σ [Cyb89], although there are also many other proofs using different techniques.

1.2.4 Optimizing Neural Networks

The universal approximation theorem (Theorem 1.3) ensures that a neural network can be built to approximate any continuous function F^* on a compact domain. However, it does not tell us *how* to build it. In other words, while the approximation problem is settled, the optimization problem remains. In this section, we shall discuss the *gradient descent* method for optimizing machine learning models, including but not limited to neural networks.

As demonstrated in many previous cases, in practice the empirical risk minimization problem can be written as an optimization problem over certain trainable parameters. In other words, we assume that the Hypothesis space admits a *parameterization*, so that $\mathcal{H} = \{F : F(x) = F_\theta(x) : \theta \in \Theta\}$. The set Θ is the allowed set of trainable parameters. For example, in the case of shallow neural networks (1.18), $\theta = (v, w, b) \in \Theta = \mathbb{R}^{(2+d)M}$. In most applications we can let $\Theta = \mathbb{R}^p$ be a Euclidean space, but there of course exists important exceptions (e.g. quantized networks). In the following, we will assume $\Theta = \mathbb{R}^p$. Consequently, the empirical risk minimization can be written as

$$\min_{F\in\mathcal{H}} R_{\text{emp}}(F) = \min_{\theta\in\in\mathbb{R}^p} R_{\text{emp}}(\theta) = \min_{\theta\in\in\mathbb{R}^p} \frac{1}{N}\sum_{i=1}^{N} \Phi(F_\theta(x_i), y_i). \tag{1.25}$$

Recall that Φ is the loss function.

To introduce optimization methods, it is convenient to abuse notations and simply write

$$\frac{1}{N}\sum_{i=1}^{N} \Phi(F_\theta(x_i), y_i) \qquad \to \qquad \Phi(\theta) \tag{1.26}$$

Then, the empirical risk minimization problem aims to solve $\min_{\theta\in\Theta} \Phi(\theta)$.

Gradient Descent. Except for simple cases (e.g. least squares), minimizing $\Phi(\theta)$ does not admit an explicit solution, and we often resort to iterative approximation methods that are implementable on a computer. We now introduce the simplest of them all, the gradient descent (GD) algorithm. Notice that the gradient vector $\nabla\Phi(\theta)$ always points in the steepest ascent direction on the surface defined by $z = \Phi(\theta)$, and hence to decrease $\Phi(\theta)$ we should go in the *opposite* direction, the steepest descent direction given by $-\nabla\Phi(\theta)$ (Figure 1.2). How long a step should we take? This is controlled by a parameter called the *learning rate* or *step size*, $\eta > 0$. It is typically taken to be small to ensure stability of the algorithm. The algorithm is summarized in 1 and also illustrated on Figure 1.2.

It can be shown that if Φ is sufficiently well-behaved, e.g. if $\nabla\Phi$ is globally Lipschitz, then $\|\nabla\Phi(\theta_k)\| \to 0$ as $k \to \infty$ for η sufficiently small [Nes04]. In other words, there is at least a subsequence of GD iterates that converge to a stationary point. However, we want to to *minimize* Φ and not just find a stationary point. Does GD converge to a minimum? To discuss this question, we need to differentiate between two kinds of minima: local and global.

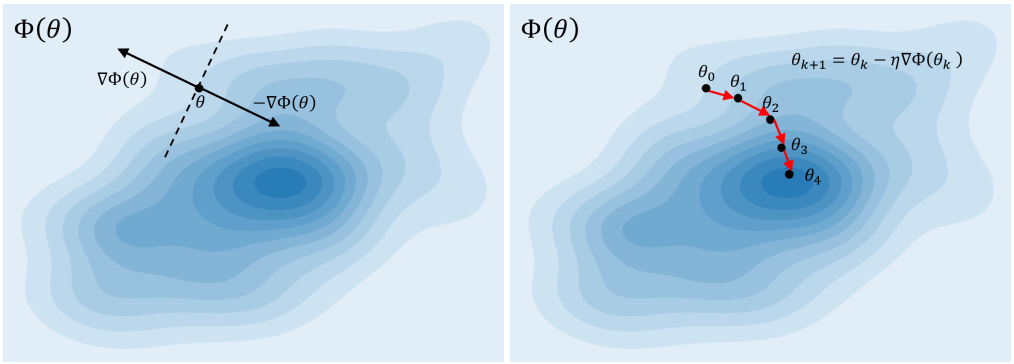


Figure 1.2: Illustration of gradient descent for function minimization. Left: $\nabla\Phi$ always points in the steepest descent direction. Right: the path of the gradient descent algorithm.

Algorithm 1: Gradient Descent

Hyperparameters: K (# iterations), η (learning rate)
Initialize: $\theta_0 \in \mathbb{R}^p$
for $k = 0, 1, \ldots, K-1$ **do**
  $\theta_{k+1} = \theta_k - \eta\nabla\Phi(\theta_k)$
end
return θ_K

Local vs Global Minima. We begin with concrete definitions.

Definition 1.4: Local and Global Minima

Let $\Phi : \mathbb{R}^p \to \mathbb{R}$ be a function. We say that θ^* is *local minimum* of Φ if there exists $\delta > 0$ such that

$$\Phi(\theta^*) \leq \Phi(\theta) \text{ for all } \theta \in \mathbb{R}^p \text{ satisfying } \|\theta - \theta^*\| \leq \delta. \tag{1.27}$$

We say that it is a *global minimum* if

$$\Phi(\theta^*) \leq \Phi(\theta) \text{ for all } \theta \in \mathbb{R}^p. \tag{1.28}$$

It turns out that under general conditions, one can show that gradient descent almost always converges to a local minimum. However, in general it does not converge to a global minimum unless we assume additional conditions on the function Φ. Let us give an example of such a condition that is commonly encountered in machine learning.

Definition 1.5: Convex Functions

We say that a function $\Phi : \mathbb{R}^p \to \mathbb{R}$ is *convex* if

$$\Phi(\lambda\theta + (1-\lambda)\theta') \leq \lambda\Phi(\theta) + (1-\lambda)\Phi(\theta') \tag{1.29}$$

for all $\theta, \theta' \in \mathbb{R}^p$ and $\lambda \in [0,1]$.

It turns out that if Φ is convex, then one can show that all local minima are automatically global minima. We state and prove this result in Proposition 1.6.

Proposition 1.6

Let $\Phi : \mathbb{R}^p \to \mathbb{R}$ be convex. Suppose θ^* is a local minimum of Φ, then it is a global minimum of Φ.

Proof. Without loss of generality we assume $\Phi(\theta^*) = 0$ (otherwise just replace $\Phi(\theta)$ by $\Phi(\theta) - \Phi(\theta^*)$). Suppose for the sake of contradiction that exists $s \in \mathbb{R}^p$ such that $\Phi(s) < 0$. Define $u(\lambda) = \lambda s + (1-\lambda)\theta^*$ for $\lambda \in [0,1]$. By the definition of convexity (1.29), we have

$$\Phi(u(\lambda)) \leq \lambda\Phi(s) + (1-\lambda)\Phi(\theta^*) = \lambda\Phi(s), \tag{1.30}$$

or $\Phi(s) \geq \Phi(u(\lambda))/\lambda$ for all $\lambda \in (0,1]$. But, $\|u(\lambda) - \theta^*\| = \lambda\|s - \theta^*\|$. Picking $\lambda = \min(1, \delta/\|s-\theta^*\|)$ gives $\|u(\lambda) - \theta^*\| \leq \delta$. By definition of local minimum, $\Phi(u(\lambda)) \geq 0$ and so $\Phi(s) \geq \Phi(u(\lambda))/\lambda \geq 0$, contradicting our premise that $\Phi(s) < 0$.

Consequently, as long as Φ is convex, GD can solve the empirical risk minimization problem. Furthermore, for convex functions one can actually give a *rate* at which GD converges: to reach an error of ϵ in the function value, we roughly require at most $O(\epsilon^{-1})$ GD iterations. If stronger conditions are assumed on Φ (e.g. strong convexity), then this rate can be faster.

Remark. *Very often, when the sample size is large we employ the stochastic version of GD, where at each iteration only a subset of all training samples are selected to compute the gradient. This is known as* stochastic gradient descent *(SGD). We will omit this issue in these notes, but see [BCN18].*

1.2.5 Deep Neural Networks

So far, we have introduced shallow neural networks and discussed their optimization and approximation properties. In this section, we discuss the extension of shallow neural networks to *deep* neural networks (DNN), forming the basis of the deep learning revolution we are witnessing today.

The simplest type of DNN are the so-called deep fully-connected networks, where we simply iterate the structure of shallow neural networks K times. K is the *depth* of the DNN. Concretely,

deep neural networks make up the following hypothesis space

$$\begin{aligned}&\mathcal{H} = \left\{F : F(x) = v^\top x(K), v \in \mathbb{R}^{d_K}\right\}\\ &\text{where}\\ &x(k+1) = \sigma(W(k)x(k) + b(k)), \quad W(k) \in \mathbb{R}^{d_{k+1}\times d_k}, \quad b(k) \in \mathbb{R}^{d_{k+1}}, \quad k = 0, \dots, K-1\\ &\text{with} \quad d_0 = d, \quad x(0) = x.\end{aligned} \tag{1.31}$$

The trainable parameters are the weights $\{W(0), \dots, W(K-1)\}$, biases $\{b(0), \dots, b(K-1)\}$ and the final combination (final layer) weights v. The activation function is applied element-wise to each vector, i.e. $\sigma(z)_i = \sigma(z_i)$.

Residual Networks. The deep network architecture proved to be tremendously successful at learning relationships between inputs and outputs. The hypothesis space (1.31) is only the simplest example, and there are many variants. In these notes, we will focus on a particular variant known as *residual networks* [HZRS15], which is among the state of the art architectures for deep learning. ResNet is not a particular architecture but a class of architectures that has a *residual connection*. For example, the residual version of (1.31) is obtained when we replace

$$x(k+1) = \sigma(W(k)x(k) + b(k)) \qquad \longrightarrow \qquad x(k+1) = x(k) + \sigma(W(k)x(k) + b(k)). \tag{1.32}$$

Note that this necessitates the fact that $d_k = d$ for all $k = 0, \dots, K$. More generally, a residual network has the form

$$x(k+1) = x(k) + f(k, x(k), \theta(k)), \tag{1.33}$$

where $\theta(k) \in \Theta$ are the trainable parameters at layer k. Readers familiar with ordinary differential equations will immediately recognize the form of (1.33) resembles a Euler discretization – this observation is in fact the central viewpoint of the work on the dynamical system viewpoint of deep learning, which is the focal point of these notes. Hence, let us now complete the background material by introducing some basics of differential equations.

Back-propagation Algorithm. Just like shallow networks, DNNs can be trained using GD (or SGD). The only complication is, since there are many trainable parameters linked in a deep network, can we have a efficient way to compute the gradient? The well-known *back-propagation algorithm* precisely handles this. Since we are going to introduce an mathematical equivalent way to view back-propagation, we will postpone detailed discussion of this algorithm to later chapters.

1.3 Ordinary Differential Equations

In this section, we introduce some basics of ordinary differential equations that will be useful to us for later chapters. We will not present any proofs since they can be found in any standard

introductory text (e.g. [Arn12, Cod12]) the readers are assumed to have some familiarity with the topic, but we will state without proof a few useful properties and illustrate some relevant phenomenon with examples.

1.3.1 Basic Definitions

We will work in $\mathbb{R}^d$. An ordinary differential equation (ODE) is the equation

$$\dot{x}(t) = f(x(t)), \qquad x(0) = x_0 \in \mathbb{R}^d, \tag{1.34}$$

where $\dot{x}$ denotes the time derivative, $f : \mathbb{R}^d \to \mathbb{R}^d$ is a function or vector field and x_0 is the initial condition. This called a *time homogeneous* ODE since the vector field on the right does not depend explicitly on time t. On the other hand, a *time-inhomogeneous* ODE is given by

$$\dot{x}(t) = f(t, x(t)), \qquad x(0) = x_0 \in \mathbb{R}^d. \tag{1.35}$$

We note that minus technical conditions, these two equations are equivalent. First, obviously (1.35) includes (1.34). For the reverse direction, we define an auxiliary variable $x^0 \in \mathbb{R}$ such that $\dot{x}^0(t) = 1$, $x^0(0) = 0$ so that $x^0(t) = t$. Then, we can rewrite (1.35) by defining $\tilde{x} = (x^0, x)$, $\tilde{f}(\tilde{x}) = (1, f(x^0, x))$ exactly in the form of (1.34). Hence, for convenience we will work with either (1.34) and (1.35), keeping in mind that they are effectively equivalent for most purposes.

By a solution of an ODE on $[0, T]$ we mean a function $\boldsymbol{x} : [0, T] \to \mathbb{R}^d$ with $\boldsymbol{x} := \{x(t) : t \in [0, T]\}$ that satisfies (1.35).

Example 1.7: Linear ODEs

Let $d = 1$ and $f(x) = ax$ with $a \in \mathbb{R}$. Then, check that

$$x(t) = e^{at} x_0, \tag{1.36}$$

is the solution to (1.34). More generally, consider $d \geq 1$ and $f(x) = Ax$ where $A \in \mathbb{R}^{d \times d}$. Then,

$$x(t) = e^{tA} x_0, \tag{1.37}$$

is the solution to (1.34). Here $e^C := \sum_i C^i / i!$ denotes the usual matrix exponential.

The definition of solution requires $\boldsymbol{x}$ to be differentiable on $(0, T)$. But we remark that it is possible to relax this by considering *integral forms*. For example, we can write (1.35) as

$$x(t) = x_0 + \int_0^t f(s, x(s)) ds. \tag{1.38}$$

The advantage here is that we can consider less regular $\boldsymbol{x}$ to be solutions of ODEs, e.g. here it is only required for $\boldsymbol{x}$ to be *absolutely continuous*, meaning that $\boldsymbol{x}$ satisfies (1.35) for almost every t. One of the most basic results concerns when a solution to (1.35) (or (1.38)) exists. The following result gives sufficient conditions, and we will hereafter always assume that a unique solution exists to whichever ODE we deal with.

Theorem 1.8: Picard–Lindelöf Theorem

Let f be continuous in t and uniformly Lipschitz in x, i.e. there exists a constant C such that $\|f(t,x) - f(t,x')\| \leq C\|x - x'\|$ for all $x, x' \in \mathbb{R}^d$ and $t \in [0,T]$. Then, there exists a unique solution to (1.35) on $[0,T]$.

1.3.2 Flow Map and Dependence on Initial Condition

One way to look at ODEs is to look at its solution trajectories given initial condition. Alternatively, we can also look at what the solution does to a set of initial conditions at a fixed terminal time. In other words, we define the *flow* or the *flow map* $\varphi_t : \mathbb{R}^d \to \mathbb{R}^d$

$$\varphi_t(x) := x(t) \qquad \text{where} \qquad \dot{x}(s) = f(s, x(s)), \quad s \in [0,t], \quad x(0) = x \tag{1.39}$$

In fact, the set $\Phi := \{\varphi_t : t \in \mathbb{R}\}$ forms a one-parameter continuous group of transformations on $\mathbb{R}^d$, under the binary operation of function composition. Analyzing the set Φ can be seen as an alternative way to understand ODEs, and is of particular relevance when we connect with the realm of deep learning.

The following properties are well-known and easy to check:

- φ_t is continuous for each t
- φ_0 is the identity mapping, $\varphi_0(x) = x$ for all x
- If f does not depend on t, then $\varphi_t \circ \varphi_s = \varphi_{t+s}$, i.e. $t \to \varphi_t$ is a homomorphism.

One can also ask how sensitive does the terminal state of the ODE is to the initial condition. This can be captured by the jacobian of φ_t, $[\nabla \varphi_t(x)]_{ij} = \partial_j \varphi_{t,i}(x)$. The following result will be useful to us later.

Theorem 1.9: Dependence on Initial Condition

Let f be continuously differentiable in x, and Lipschitz in x uniformly in t. Let $\boldsymbol{x}$ be the solution of the ODE (1.35) with flow map φ_t and $\boldsymbol{v}$ be the solution to the linear time-inhomogeneous ODE

$$\dot{v}(s) = \nabla_x f(s, x(s))v(s), \qquad s \in [0, t], \qquad v(0) = v_0. \tag{1.40}$$

Then, we have

$$\lim_{\varepsilon \to 0^+} \left\| \frac{\varphi_t(x_0 + \varepsilon v_0) - \varphi_t(x_0)}{\varepsilon} - v(t) \right\| \to 0, \tag{1.41}$$

uniformly in $t \in [0, T]$ for $\|v_0\| \leq 1$.

Corollary 1.10

Under the same conditions as in Theorem 1.9, the Jacobian $J(t) := \nabla_x \varphi_t(x_0)$ satisfies the linear equation

$$\dot{J}(t) = \nabla_x f(t, x(t))J(t), \qquad J(0) = I. \tag{1.42}$$

Equation (1.40) is called the variational equation associated with the ODE (1.35). It describes the propagation of variations of the initial condition along the evolution in time, hence its name. We will refer back to these results in our discussion of optimal control theory.

Example 1.11: Flow Map and Variational Equations for Linear Systems

Recall the linear system in Example 1.7. In this case, the flow map is a linear function

$$\varphi_t(x) = e^{tA}x, \tag{1.43}$$

with Jacobian $J(t) = e^{tA}$ (in this case, the Jacobian does not depend on x_0). Check that $J(t)$ satisfies the variational equation

$$\dot{J}(t) = AJ(t), \qquad J(0) = I, \tag{1.44}$$

as shown in the above corollary.

1.3.3 Numerical Solution of ODEs

Often, ODEs do not admit explicit solutions and we have to compute a solution numerically. There are many methods for doing so and it is not the purpose here to give a thorough intro-

duction. Pertaining to the topic discussed in these notes, it is sufficient to first introduce the simplest possible method, the *forward Euler method*.

In this method, we construct a solution to (1.35) by discretizing time and setting

$$\widehat{x}(k+1) = \widehat{x}(k) + \Delta t f(k\Delta t, \widehat{x}(k)), \qquad \widehat{x}(0) = x_0, \tag{1.45}$$

which can be seen as a first-order Taylor expansion of the integral form of the ODE (1.38) for small Δt. The latter is called the *step size*. We expect that this approximation to get better as the step size Δt becomes small. This is made precise in the following result.

Theorem 1.12: Global Truncation Error of Forward Euler Method

Let f be Lipschitz in x uniformly in t and continuous in t. Let $\boldsymbol{x}$ be a solution of the ODE (1.35) with initial condition x_0 and $\widehat{\boldsymbol{x}}$ be the iterates defined in (1.45), then for each $K > 0$ there exists a constant $C > 0$ such that

$$\max_{k \le K} \|\widehat{x}(k) - x(k\Delta t)\| \le C\Delta t. \tag{1.46}$$

2 Optimal Control Theory

The study of optimal control theory originates from the classical theory of the calculus of variations, beginning with the seminal work of Euler and Lagrange in the 1700s. These culminated in the so-called Lagrangian mechanics that reformulate Newtonian mechanics in terms of extremal principles. In a nut shell, the calculus of variations studies optimization over "curves", which one can picture as an infinite dimensional extension of traditional optimization problems.

Optimal control theory is a nontrivial extension of the classical theory of calculus of variations in two main directions: to dynamical and non-smooth settings. This builds on important contributions of Weierstrass and others and led in two inter-related directions: the Pontryagin's maximum principle and the Hamilton-Jacobi-Bellman theory. An interesting historical account of the developments can be found in [Lib12].

In this section, we give a minimal introduction of the problem formulation of optimal control problems, paying particular attention to the so-called *Bolza problems* which are most relevant to deep learning. The reader is referred to comprehensive texts on optimal control theory for a more complete account [AF13, Lib12, BP07].

2.1 From Calculus of Variations to Optimal Control

2.1.1 A Motivating Example

Finite-dimensional optimization problems are of the form

$$\inf_{x \in X} \Phi(x), \qquad \Phi : X \to \mathbb{R}, \tag{2.1}$$

where X is usually a subset of a Euclidean space. On the other hand, a calculus of variations problem minimizes some functional J over some infinite dimensional space $\mathcal{X}$, i.e.

$$\inf_{\boldsymbol{x} \in \mathcal{X}} J[\boldsymbol{x}] \qquad J : \mathcal{X} \to \mathbb{R}. \tag{2.2}$$

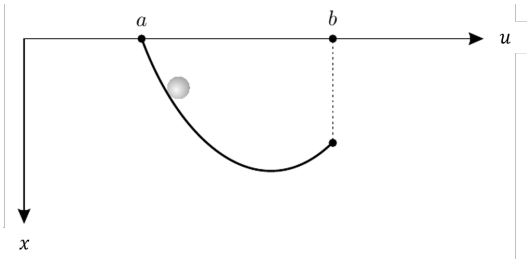
There are many possible forms of the functional J and the space $\mathcal{X}$. For example, one may encounter functionals in the form of an integral, where the argument $\boldsymbol{x} = \{x(u) : u \in [a, b]\}$ is a function of a scalar variable u, i.e.

$$J[\boldsymbol{x}] = \int_a^b L(u, x(u), x'(u)) du \tag{2.3}$$

Let us consider motivating example problem of this nature that is also of substantial historical importance.

Example 2.1: Rolling a Ball Down a Ramp

Let $a < b$ be two points on a horizontal plane, and our goal is to build a ramp such that when we release the ball from point a, it can arrive at a point directly under point b in the *shortest* time possible. See figure below. We will assume that there is no friction.



What shape of the ramp will achieve this task? It turns out that we can phrase this as a calculus of variations problem. Let $s(u)$ be the instantaneous speed of the ball when its horizontal coordinate is at u, and let $\{x(u)\}$ denote the shape of the ramp and that $x(a) = 0$. By conservation of energy we find that

$$\frac{1}{2}ms(u)^2 = mgx(u) \qquad \Rightarrow \qquad s(u) = \sqrt{2gx(u)} \tag{2.4}$$

Hence, the total time taken from a to b is the integral of the arc-length divided speed, i.e.

$$\text{Total time} = J[\boldsymbol{x}] = \int_a^b \frac{\sqrt{1 + x'(u)}}{\sqrt{2gx(u)}} du, \tag{2.5}$$

which is of the form (2.3) where $L(u, x, v) = \sqrt{1 + v}/\sqrt{2gx}$.

The problem in Example 2.1 is known as the *Brachistochrone*[1] problem, and is first posed by Johann Bernoulli in 1696. One can see from the example above that to solve this problem, it is needed to solve optimization problems over curves. A classical result due to Euler and Lagrange gives a necessary condition for optimality that allows us to solve this problem.

[1] In Greek, "Brachistochrone" is literally "shortest time".

Theorem 2.2: Euler-Lagrange Equations

Let $\boldsymbol{x} \in C^1([a,b],\mathbb{R})$ be an extremum of J as defined in (2.3). Then, $\boldsymbol{x}$ satisfies the *Euler-Lagrange Equations*

$$\partial_x L\left(u, x(u), x'(u)\right) = \frac{d}{du}\partial_{x'} L\left(u, x(u), x'(u)\right), \qquad u \in [a,b]. \tag{2.6}$$

We have deliberately left several notions rather undefined, such as the meaning of an extremum. We will revisit this slightly subtle issue in the next part. Here, we will not present a proof of the Euler-Lagrange equations, since it is not required for the rest of our discussions. A proof can be found in any standard texts on the subject of calculus of variations, say [GS00, Lib12].

Exercise 2.3: Brachistochrone Solution

Consider the Brachistochrone problem in Example 2.1. By choosing appropriate units one can set $g = 1/2$. Show that the optimal ramp shapes are *cycloids* whose parametric forms are

$$\begin{aligned} u(\theta) &= a + c(\theta - \sin\theta) \\ x(\theta) &= c(1 - \cos\theta) \end{aligned} \qquad \theta \in [0, 2\pi], \quad c > 0. \tag{2.7}$$

2.1.2 The Problem of Optimal Control

In passing to optimal control, we consider additionally two aspects of the problem, namely the type of extrema studied, as well as the setting in which such calculus of variations problems are phrased.

Throughout these notes, the word "extrema" refers to either a minimum or a maximum in the function/functional under consideration. Since maximization is just equivalent to minimization by replacing the objective function(al) with its negation, we will hereafter only discuss minima, unless otherwise stated.

We start with distinguishing different types of minima.

2.1.3 Weak vs Strong Minima

In finite-dimensional optimization, it is easy to define the notion of local and global minima. Let $\Phi : \mathbb{R}^d \to \mathbb{R}$ be a function.

- We say that x^* is a *local minimum* of Φ if there exists a $\delta > 0$ such that $\Phi(x^*) \leq \Phi(x)$ for all $\|x - x^*\| \leq \delta$.

- We say that x^* is a *global minimum* of Φ if $\Phi(x^*) \leq \Phi(x)$ for all $x \in \mathbb{R}^d$.

Hence, all global minima are automatically local minima. If Φ is differentiable, then a necessary condition for a local minima is that $\nabla\Phi(x^*) = 0$. We have seen this in the least squares formula (1.15).

In extending these ideas to infinite dimensions, one needs to be slightly more careful. Notice that the definition of minima (local or global) depends on the norm $\|\cdot\|$ which gives us a sense of closeness. We did not specify what norm we used in the finite dimensional case above, since all of them are equivalent[2].

In the infinite dimensional case of minimization of functionals, the norm we choose will affect our results, and some curve $\boldsymbol{x}$ may be a local minimum of J under one norm but not under another.

We now distinguish between two notions of minima – *weak* and *strong* minima – commonly encountered in calculus of variations and optimal control.

Let us consider for the moment that our curve $\boldsymbol{x}$ is C^1. Moreover, let us simplify things and consider one spatial dimension, so that $x(u) \in \mathbb{R}$ for $u \in [a, b]$. Now there are two natural choices of norm that we can use

- 0-Norm: $\|\boldsymbol{x}\|_0 = \sup_{u\in[a,b]} |x(u)|$.
- 1-Norm: $\|\boldsymbol{x}\|_1 = \|\boldsymbol{x}\|_0 + \sup_{u\in(a,b)} |x'(u)|$.

Each of these norms then allows us define the notion of minimum.

Definition 2.4: Strong and Weak Minima

Let $J : C^1([a, b], \mathbb{R}) \to \mathbb{R}$ be a functional and $\boldsymbol{x}^* \in C^1([a, b], \mathbb{R})$. We say that $\boldsymbol{x}^*$ is a strong local minimum if there exists a $\delta > 0$ such that $J[\boldsymbol{x}^*] \leq J[\boldsymbol{x}]$ for all $\|\boldsymbol{x} - \boldsymbol{x}^*\|_0 \leq \delta$. We say that $\boldsymbol{x}^*$ is a weak local minimum if we place the norm $\|\cdot\|_0$ by $\|\cdot\|_1$. The global versions are defined similarly.

Now, it is easy to see that any strong minima must be a weak minima, but the converse is not true. Moreover, observe that the Euler-Lagrange equations (Thm. 2.2) apply to weak minima, whereas we need more advanced tools to handle strong minima. We now consider a simple example below where a weak minima simply do not exist – but we will see later that this does not prevent the existence of a strong minima. All of these reasons motivate us to go past the setting of Euler and Lagrange and into the realm of optimal control.

Example 2.5: Piece-wise C^1 Minimizer

[2] Let $\|\cdot\|_A$ and $\|\cdot\|_B$ be two norms on $\mathbb{R}^d$, then there exists $c \in (0, 1]$ such that $c\|x\|_A \leq \|x\|_B \leq \frac{1}{c}\|x\|$ for all $x \in \mathbb{R}^d$.

Consider the problem of minimizing the functional

$$J[\boldsymbol{x}] = \int_{-1}^{1} [x(u)]^2 [x'(u) - 1]^2 du, \tag{2.8}$$

subject to the boundary conditions $x(-1) = 0$ and $x(1) = 1$. Clearly, for all $\boldsymbol{x} \in C^1$ we have $J[\boldsymbol{x}] > 0$. But the curve

$$x(u) = \begin{cases} 0 & -1 \leq u < 0 \\ x & 0 \leq u \leq 1 \end{cases} \tag{2.9}$$

achieves $J[\boldsymbol{x}] = 0$, but is only piece-wise C^1. In fact, C^1 curves can get closer and closer to $x(u)$ with lower and lower cost, thus a C^1 global minimizer does not exist.

2.1.4 A Dynamical View on the Calculus of Variations

Optimal control is another way to look at calculus of variations problems, in that we view things in a dynamical nature. Concretely, we may re-parameterize the curves $x(u)$ considered via infinitesimal changes in it, in the form of a control. Let us motivate this approach in the context of the Brachistochrone problem.

Example 2.6: Control Formulation of Brachistochrone

Consider the Brachistochrone problem 2.1, but this time we parameterize the ramp by a parametric form from the outset, i.e. $(u(t), x(t))$ where t is time. Then, the speed at time t is $s(u(t)) = s(t) = \sqrt{\dot{u}(t)^2 + \dot{x}(t)^2}$. Then, conservation of energy leads to

$$2gx(t) = \dot{x}(t)^2 + \dot{u}(t)^2. \tag{2.10}$$

Now, we imagine the reverse scenario treating the velocities $\dot{x}, \dot{u}$ as *controls*, by setting

$$\theta_1(t) = \dot{u}(t)/\sqrt{2gx(t)} \qquad \theta_2(t) = \dot{x}(t)/\sqrt{2gx(t)}. \tag{2.11}$$

Then, we end up with a control system that defines the equation of the ramp

$$\begin{aligned} \dot{u}(t) &= \theta_1(t)\sqrt{2gx(t)} \\ \dot{x}(t) &= \theta_2(t)\sqrt{2gx(t)} \\ \theta_1(t)^2 &+ \theta_2(t)^2 = 1 \\ (u(t_0), x(t_0)) &= (a, 0), \qquad u(t_1) = b \end{aligned} \tag{2.12}$$

The cost function in this case is the time taken, so $J = \int_{t_0}^{t_1} 1 dt = t_1 - t_0$.

It is worth noting that by formulating the original calculus of variations problem as a control problem, we actually gained some generality:

- It is no longer assumed that x can be written as a function of u
- It is not necessary for x to be differentiable with respect to u

2.1.5 The Optimal Control Formulation

Now, let us formulate precisely the optimal control problem in the general setting.

The Dynamics. Consider the ordinary differential equation

$$\dot{x}(t) = f(t, x(t), \theta(t)), \qquad t \in [t_0, t_1], \qquad x(t_0) = x_0. \tag{2.13}$$

Here $x(t) \in \mathbb{R}^d$ is the *state*, $\theta(t) \in \Theta \subset \mathbb{R}^m$ is the *control*, with Θ the *control set*. We will assume that the control set is closed (but it need not be bounded).

We will assume that the following conditions on f holds, unless otherwise stated:

- $f(t, x, \theta)$ is continuous in t and θ for all x
- $f(t, x, \theta)$ is continuously differentiable in x for all t, θ

These conditions are sufficient to ensure that (2.13) is well-posed by a similar result as in Theorem 1.8. See [BP07].

Remark. *The conditions outlined above are certainly not the weakest possible to imply local well-posedness of solutions, and they can be weakened in various ways (See e.g. [BP07] Ch.2).*

*We also emphasize two crucial points **not** assumed*

- *We did not assume that f is differentiable with respect to θ*
- *We did not assume that $t \mapsto \theta(t)$ is regular. In fact, in the general case we can consider θ to be a essentially bounded function of t*

The Cost Functional Let us now define the objective functionals. We will consider functionals of the form

$$J[\boldsymbol{\theta}] = \int_{t_0}^{t_1} L(t, x(t), \theta(t))dt + \Phi(t_1, x(t_1)) \tag{2.14}$$

- $L : \mathbb{R} \times \mathbb{R}^d \times \Theta \to \mathbb{R}$ is called the *running cost*
- $\Phi : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ is called the *terminal cost*

The Bolza Problem of Optimal Control Now, we state the *Bolza* problem of optimal control, which will be the primary object of analysis in these notes.

$$\begin{aligned} &\inf_{\boldsymbol{\theta}} J[\boldsymbol{\theta}] = \int_{t_0}^{t_1} L(t, x(t), \theta(t))dt + \Phi(t_1, x(t_1)) \\ &\text{subject to} \\ &\dot{x}(t) = f(t, x(t), \theta(t)), \qquad t \in [t_0, t_1], \qquad x(t_0) = x_0. \end{aligned} \tag{2.15}$$

For historical reasons, the case where $\Phi = 0$ (no terminal cost) is called a *Lagrange* problem, where as the case with $L = 0$ (no running cost) is called a *Mayer* problem. In optimal control theory, we often consider x_0 (initial condition) and t_0 (initial time) to be fixed. However, the terminal time t_1 can either be fixed or it can vary. Moreover, there can be a constraint set placed on the terminal state $x(t_1)$. We will mostly consider the case where the final time t_1 is fixed (so that we can neglect the t_1 dependence of Φ), and there is no constraint on the terminal state, and we will discuss how the various results may change if we consider the general case.

As with classical optimization problems, the primary object of study is optimality conditions. One differentiates between *necessary* and *sufficient* conditions for optimality. The former asks what conditions must any local/global optimum satisfy, and the latter concerns a condition that is enough to guarantee optimality. In the following sections, we will investigate each of these aspects in turn.

2.2 Pontryagin's Maximum Principle

In this section, we discuss a necessary condition for optimality – the Pontryagin's Maximum Principle (PMP) – that is a hallmark result in optimal control theory and the calculus of variations. It greatly generalizes the Euler Lagrange equations in highly nontrivial ways, and forms a natural bridge between optimal control theory and deep learning, as we will subsequently investigate.

We will present the proof of the PMP in the case of fixed end time, without constraints on the terminal state. In this case, the problem is

$$\begin{aligned} &\inf_{\boldsymbol{\theta}} J[\boldsymbol{\theta}] = \int_{t_0}^{t_1} L(t, x(t), \theta(t))dt + \Phi(x(t_1)) \\ &\text{subject to} \\ &\dot{x}(t) = f(t, x(t), \theta(t)), \qquad t \in [t_0, t_1], \qquad x(t_0) = x_0. \end{aligned} \tag{2.16}$$

The proof of the PMP for this case is quite accessible, and hence we will present it in full. We will discuss the PMP for other variants of the basic formulation, but we will omit the proofs as they can be significantly more involved.

2.2.1 The Maximum Principle

To state the Pontryagin's maximum principle, we need some definitions. Let us define the *Hamiltonian*

$$\begin{aligned} &H : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^d \times \Theta \to \mathbb{R}, \\ &H(t, x, p, \theta) = p^\top f(t, x, \theta) - L(t, x, \theta). \end{aligned} \tag{2.17}$$

For a control $\boldsymbol{\theta} = \{\theta(t) : t \in [t_0, t_1]\}$, we say it is *admissable* if $\theta(t) \in \Theta$ for all $t \in [t_0, t_1]$.

Theorem 2.7: Pontryagin's Maximum Principle

Let $\boldsymbol{\theta}^*$ be a bounded, measurable and admissable control that optimizes (2.16), and $\boldsymbol{x}^*$ be its corresponding state trajectory. Then, there exists an absolutely continuous process $\boldsymbol{p} = \{p(t) : t \in [t_0, t_1]\}$ such that

$$\dot{x}^*(t) = \nabla_p H(t, x^*(t), p^*(t), \theta^*(t)), \qquad x^*(t_0) = x_0 \tag{2.18}$$

$$\dot{p}^*(t) = -\nabla_x H(t, x^*(t), p^*(t), \theta^*(t)), \qquad p^*(t_1) = -\nabla_x \Phi(x^*(t_1)) \tag{2.19}$$

$$\begin{aligned} &H(t, x^*(t), p^*(t), \theta^*(t)) \geq H(t, x^*(t), p^*(t), \theta) \\ &\qquad \forall \theta \in \Theta \text{ and } a.e.\ t \in [t_0, t_1] \end{aligned} \tag{2.20}$$

Proof 2.7: Proof of the PMP (Theorem 2.7)

The proof proceeds in several steps. To make the proof instructive, we will first assume that the function $t \mapsto \theta^*(t)$ is continuous, and we will relax this assumption at the end.

Step 1: Convert to Mayer Problem. Define an auxiliary scalar variable $x^0(t)$, with

$$\dot{x}^0(t) = L(t, x(t), \theta(t)), \qquad x^0(t_0) = 0. \tag{2.21}$$

Then, by going one dimension higher and setting $\tilde{x} = (x^0, x)$, $\tilde{f} = (L, f)$, and $\tilde{\Phi}(\tilde{x}) = \Phi(x) + x^0$ we can rewrite (2.16) as one without running cost in the new augmented coordinates. Hence, we will hereafter drop the tildes and assume without loss of generality that $L \equiv 0$.

Step 2: Needle Perturbation. Fix $\tau > 0$ and an admissible $s \in \Theta$. Define the *needle perturbation* to the optimal control

$$\theta_\varepsilon(t) = \begin{cases} s & \text{if } t \in [\tau - \varepsilon, \tau] \\ \theta^*(t) & \text{otherwise} \end{cases} \tag{2.22}$$

Let $\boldsymbol{x}_\varepsilon$ be the corresponding controlled trajectory, i.e. the solution of

$$\dot{x}_\varepsilon(t) = f(t, x_\varepsilon(t), \theta_\varepsilon(t)), \qquad x_\varepsilon(t_0) = x_0. \tag{2.23}$$

Our goal is to derive necessary conditions for which any such needle perturbation will be sub-optimal, thus resulting in a necessary condition for a strong minima in the cost functional.

Step 3: Variational Equation It is clear that $x_\varepsilon(t) = x^*(t)$ for $t \le \tau - \varepsilon$. Let us define for $t \ge \tau$

$$v(t) := \lim_{\varepsilon \to 0+} \frac{x_\varepsilon(t) - x^*(t)}{\epsilon}. \tag{2.24}$$

This measures the propagation of the effect of the needle perturbation as time increases. In particular, at $t = \tau$, $v(\tau)$ is the tangent vector of the curve $\epsilon \mapsto x_\varepsilon(\tau)$, given by

$$\begin{aligned} v(\tau) &= \lim_{\varepsilon \to 0+} \left\{ \frac{1}{\varepsilon} \int_{\tau-\varepsilon}^{\tau} f\left(t, x_\varepsilon(t), s\right) dt - \frac{1}{\varepsilon} \int_{\tau-\varepsilon}^{\tau} f\left(t, x^*(t), \theta^*(t)\right) dt \right\} \\ &= f\left(\tau, x^*(\tau), s\right) - f\left(\tau, x^*(\tau), \theta^*(\tau)\right). \end{aligned} \tag{2.25}$$

For the remaining time $t \in [\tau, T]$, x_ε follows the same ODE (2.23). Thus, by Theorem 1.9 $v(t)$ is well-defined and solves the linear variational equation

$$\dot{v}(t) = \nabla_x f\left(t, x^*(t), \theta^*(t)\right) v(t), \qquad t \in [\tau, t_1], \tag{2.26}$$

with initial condition given by (2.25). In particular, the vector $v(t_1)$ describes the variation in the end point $x_\varepsilon(t_1)$ due to the needle perturbation.

Step 4: Optimality Condition at End Point. By our assumption, the control $\boldsymbol{\theta}^*$ is optimal, hence we must have

$$\Phi(x^*(t_1)) \le \Phi(x_\varepsilon(t_1)). \tag{2.27}$$

Thus, we have

$$0 \le \lim_{\varepsilon \to 0+} \frac{\Phi\left(x_\varepsilon(t_1)\right) - \Phi\left(x^*(t_1)\right)}{\varepsilon} = \frac{d}{d\varepsilon} \Phi\left(x_\varepsilon(t_1)\right)\Big|_{\varepsilon=0+} = \nabla\Phi\left(x^*(t_1)\right) \cdot v(t_1) \tag{2.28}$$

In fact, the inequality (2.28) holds for any τ and s that characterizes the needle perturbation.

Step 5: The Adjoint Equation and the Maximum Principle. The idea is now to derive consequence that the end-point optimality condition have on each τ. To this end, we define $\boldsymbol{p}^*(t)$ as the solution of the backward Cauchy problem

$$\dot{p}^*(t) = -\nabla_x f(t, x^*(t), \theta^*(t))^\top p^*(t), \qquad p^*(t_1) = -\nabla\Phi(x^*(t_1)). \tag{2.29}$$

Then, observe that we indeed have

$$\frac{d}{dt}[p^*(t)^\top v(t)] = 0 \quad \forall t \in [\tau, t_1] \qquad \Rightarrow \qquad p^*(\tau)^\top v(\tau) = p^*(t_1)^\top v(t_1) \leq 0, \tag{2.30}$$

which implies that for any $\tau \in (t_0, t_1]$ we have

$$[p^*(\tau)]^\top f(\tau, x^*(\tau), \theta^*(\tau)) \geq [p^*(\tau)]^\top f(\tau, x^*(\tau), s) \tag{2.31}$$

for any $s \in \Theta$. By continuity this also holds for $t = t_0$.

By undoing the conversion in in Step 1, we can back to a general Bolza problem by sending $\boldsymbol{p}^* \to (\boldsymbol{p}^0, \boldsymbol{p}^*)$. In particular, observe that $\dot{p}^0(t) = 0$ and $p^0(t_1) = -1$. Hence, $p^0(t) \equiv -1$. Hence, we get from the optimality condition (2.31) that

$$\underbrace{p^*(\tau)^\top f(\tau, x^*(\tau), \theta^*(\tau)) - L(\tau, x^*(\tau), \theta^*(\tau))}_{H(\tau, x^*(\tau), p^*(\tau), \theta^*(\tau))} \geq \underbrace{p^*(\tau)^\top f(\tau, x^*(\tau), s) - L(\tau, x^*(\tau), s)}_{H(\tau, x^*(\tau), p^*(\tau), s)}, \tag{2.32}$$

where $\boldsymbol{p}^*$ satisfies the adjoint equation

$$\dot{p}^*(t) = -\nabla_x H(t, x^*(t), p^*(t), \theta^*(t)), \qquad p^*(t_1) = -\nabla\Phi(x^*(t_1)). \tag{2.33}$$

Step 6: Extending to Measurable Controls. The last step is purely of technical interest, where we relax the assumption that $t \mapsto \theta^*(t)$ is continuous. By the Lebesgue differentiation theorem, we have for almost every $\tau \in (t_0, t_1)$,

$$\lim_{\varepsilon \to 0+} \frac{1}{\varepsilon} \int_{\tau-\varepsilon}^{\tau+\varepsilon} |f(t, x^*(t), \theta^*(t)) - f(\tau, x^*(\tau), \theta^*(\tau))| \, dt = 0, \tag{2.34}$$

that is, the measurable function $t \mapsto f(t, x^*(t), \theta^*(t))$ is *quasi-continuous*. Hence, the proof steps 1-5 proceeds exactly as before, only that τ is required to be a Lebesgue point, and hence the solutions of the state and adjoint equations are now only absolutely continuous, and the maximization condition (2.32) now only holds at Lebesgue points, which is almost every $t \in [t_0, t_1]$. This concludes the proof of the maximum principle. □

Let us make some remarks on the maximum principle.

- The equation (2.18) is called the *state equation*, and it is simply

$$\dot{x}^*(t) = f(t, x^*(t), \theta^*(t)), \tag{2.35}$$

and it describes the evolution of the state under the optimal control.

- The equation (2.19) is called the *co-state equation*, with $\boldsymbol{p}^*$ being the *co-state*. As evidenced in the proof of the PMP, the role of the co-state equation is to propagate back the optimality condition and is the adjoint of the variational equation. In fact, one can also connect $\boldsymbol{p}^*$ formally to a Lagrange multiplier enforcing the constraint of the ODE. However, this approach can only derive the weaker optimality condition that H is stationary at the optimum.
- The maximization condition (2.20) is the heart of the maximum principle. It says that an optimal control must *globally* maximize the Hamiltonian. One can regard this as a nontrivial generalization of the Euler-Lagrange equations to handle strong extrema (See [BP07], Theorem 6.5.2), as well as a generalization of the KKT conditions to non-smooth settings.

2.2.2 Other Forms of the Maximum Principle

The reason why we called the result (2.7) a maximum *principle* is to emphasize that it is not just one result, but a class of results of similar nature. Indeed, there are many variants of the maximum principle, and we state one of them below, which is for a *fixed-end-point* variant of the Bolza problem (variation highlighted)

$$\begin{aligned} &\inf_{\boldsymbol{\theta}} J[\boldsymbol{\theta}] = \int_{t_0}^{t_1} L(t, x(t), \theta(t))dt \cancel{+\Phi(x(t_1))} \\ &\text{subject to} \\ &\dot{x}(t) = f(t, x(t), \theta(t)), \qquad t \in [t_0, t_1], \qquad x(t_0) = x_0, \qquad x(t_1) = x_1. \end{aligned} \tag{2.36}$$

In this case, the maximum principle now reads

$$\dot{x}^*(t) = \nabla_p H(t, x^*(t), p^*(t), \theta^*(t)), \qquad x^*(t_0) = x_0 \quad x^*(t_1) = x_1 \tag{2.37}$$

$$\dot{p}^*(t) = -\nabla_x H(t, x^*(t), p^*(t), \theta^*(t)), \qquad \cancel{p^*(t_1) = -\nabla_x \Phi(x^*(t_1))} \tag{2.38}$$

$$\begin{aligned} &H(t, x^*(t), p^*(t), \theta^*(t)) \geq H(t, x^*(t), p^*(t), \theta) \\ &\qquad \forall \theta \in \Theta \text{ and } a.e.\ t \in [t_0, t_1] \end{aligned} \tag{2.39}$$

Example 2.8: Piece-wise C^1 Minimizer Revisted

Let us consider the problem in Example 2.5 and we now show that the piece-wise C^1 minimizer satisfies the PMP (2.37). Notice that we can convert the problem into a fixed-

end-point problem

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \int_{-1}^{1} x(t)^2(\theta(t)-1)^2 dt \\ &\text{subject to} \\ &\dot{x}(t) = \theta(t), \qquad t \in [-1,1], \qquad x(-1) = 0, \qquad x(1) = 1. \end{aligned} \tag{2.40}$$

That is, $f(t,x,\theta) = \theta$ and running cost is $L(t,x,\theta) = x^2(\theta-1)^2$. Writing out the PMP equations for an optimal $\boldsymbol{\theta}^*$, we get $H(t,x,p,\theta) = p\theta - x^2(1-\theta)^2$

$$\dot{x}^*(t) = \theta^*(t), \qquad x^*(-1) = 0, \quad x^*(1) = 1, \tag{2.41}$$

$$\dot{p}^*(t) = 2x^*(t)(1-\theta^*(t))^2, \tag{2.42}$$

$$\theta^*(t) \in \arg\max_{\theta\in\mathbb{R}} \{p^*(t)\theta - [x^*(t)]^2(1-\theta^2)\}. \tag{2.43}$$

One can then check that the control

$$\theta^*(t) = \begin{cases} 0 & -1 \le t < 0 \\ 1 & 0 \le t \le 1 \end{cases} \tag{2.44}$$

satisfies the PMP above with $x^*(t)$ given by (2.9) and $p^*(t) = 0$.

Example 2.9: Driving a Car

Suppose we are driving a car on a straight road for $t \in [0,T]$. Let $x(t)$ denote the position of the car at time t. We suppose that we are initially at rest at the origin, and we want to drive forwards on the road. We have control over an accelerator, which we can use to accelerate or brake, but acceleration costs fuel. The problem statement is, suppose we want to drive far yet save fuel, how should we drive?

This problem can be formulated as a Bolza problem with fixed end time and free end point (2.16) as follows:

$$\begin{aligned} &\inf_{\boldsymbol{\theta}} J[\boldsymbol{\theta}] = \int_0^T \frac{1}{2}\max(0,\theta(t))^2 dt - x(T) \\ &\text{subject to} \\ &\dot{x}(t) = v(t), \qquad x(0) = 0, \\ &\dot{v}(t) = \theta(t), \qquad v(0) = 0, \\ &\theta(t) \in [-1,1] \text{ for all } t. \end{aligned} \tag{2.45}$$

Here, the fuel cost is related to the acceleration by $\frac{1}{2}\max(0,\theta)^2$ (braking spends no fuel).

Let us now apply the PMP (2.7) to derive a solution. In this case, the Hamiltonian is

$$H(t, x, v, p_x, p_v, \theta) = p_x v + p_v \theta - \frac{1}{2}\max(0, \theta)^2. \tag{2.46}$$

Thus, the PMP equations are

$$\dot{x}^*(t) = v^*(t), \qquad x^*(0) = 0, \tag{2.47}$$
$$\dot{v}^*(t) = \theta^*(t), \qquad v^*(0) = 0, \tag{2.48}$$
$$\dot{p}_x^*(t) = 0, \qquad p_x^*(T) = 1, \tag{2.49}$$
$$\dot{p}_v^*(t) = -p_x^*(t), \qquad p_v^*(T) = 0, \tag{2.50}$$

and hence $p_x^*(t) = 1$, $p_v^*(t) = T - t$. Therefore, the optimal control is found by maximizing the Hamiltonian:

$$\begin{aligned}\theta^*(t) &\in \arg\max_{\theta \in [-1,1]} H(t, x^*(t), v^*(t), p_x^*(t), p_v^*(t), \theta^*(t)) \\ &\in \arg\max_{\theta \in [-1,1]} v^*(t) + (T-t)\theta - \frac{1}{2}\max(0,\theta)^2 \\ &= \min(T - t, 1).\end{aligned} \tag{2.51}$$

Thus, we should drive at maximum acceleration, and then ease off on the accelerator linearly.

Exercise 2.10: Driving a Better Car

As an extension of Example 2.9, we can consider the following scenario: the car has been upgraded so that the fuel cost now scales linearly with acceleration, i.e. the running cost is now $\max(0, \theta)$ instead of $\max(0, \theta)^2$. What is the optimal way to drive in this case?

2.2.3 Further Reading

Besides the basic fixed end time setting considered in the previous part, other variants of the PMP can be derived for different scenarios, including: variable end time, general set constraints on initial and final states. The proofs of these results are more involved than what is proposed above, requiring some machinery from functional analysis. For the purpose of the application cases in these notes, the previous formulation is enough. However, the interested reader is encouraged to consult optimal control references for various generalizations, or proofs under weaker assumptions e.g. [Lib12, BP07].

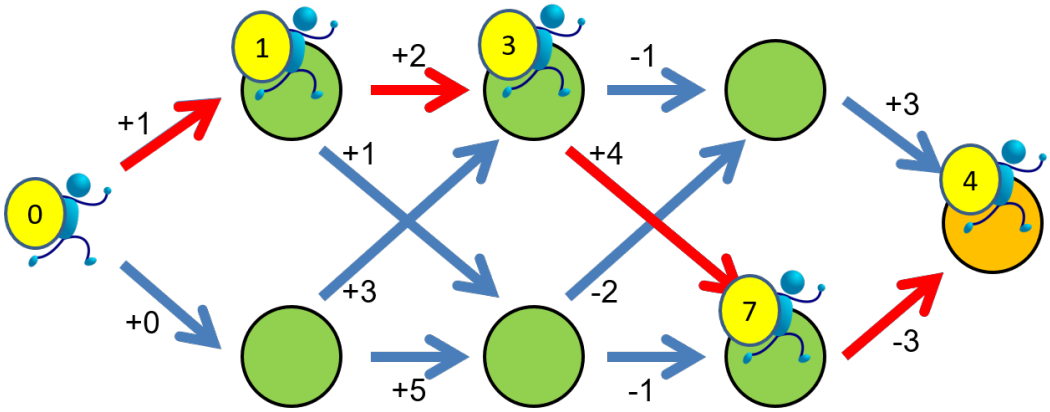
2.3 Hamilton-Jacobi-Bellman Equations

As a key alternative to the maximum principle, we now discuss another line of work that establishes necessary and sufficient conditions for optimality for optimal control problems. This presents another approach to optimal control theory that is important in its own right, as it depends on the very general idea of *dynamic programming* [Bel66].

2.3.1 Motivating Example of Dynamic Programming

Example 2.11: A Toy Maze

Consider the following maze where we want to get to the orange circle while maximizing the reward obtained along the way. When we cross each arrow, we gain a reward equal to the number attached to that arrow. The red path shows an example path with a final reward of 4.



Suppose that there are N circles to choose from per step and T steps in total. Then, the total number of paths is N^T and grows exponentially with T. This is known as the *curse of dimensionality*.

Instead of a brute force search over all paths, we can use the principle of dynamic programming to find a solution much more efficiently. To do this, let us introduce some notation. We will index each time step in the maze by $t = 0, 1, \ldots, T$. Also, we denote by S_t the circle we step on at the t^{th} step, and R_t the reward we obtain at the t^{th} step.

Define the function

$$V(t, x) = \max \left\{ \sum_{s=t+1}^{T} R_s : S_t = x \right\}. \tag{2.52}$$

In other words, $V(t, x)$ is the best possible reward we can get starting from state x at time t. Then, we can work backwards easily!

Let us just consider the case in Example 2.11, where $S_t = 1$ or 2 for $t = 1, 2, 3$. Here, $S_t = 1$ denotes the top circle and $S_t = 2$ is the bottom circle. The initial state is $S_0 = 0$. Then, clearly we have

$$V(3, 1) = +3, \qquad V(3, 2) = -3, \tag{2.53}$$

since both cases we only have one choice – and this is the best we can do. Now, let us consider $t = 2$. Given we are at $S_2 = 1$, then there are two choices, either we go to $S_3 = 1$ or $S_3 = -1$. If we go to $S_3 = 1$ we get a reward of -1 and then, the best we can do from there would be $V(3, 1) = +3$. Similarly, if we take $S_3 = 2$ then we get $+4$ reward and the best we can do from $S_3 = 2$ is $V(3, 2) = -3$. Hence,

$$V(2, 1) = \max\{-1 + V(3, 1), +4 + V(3, 2)\} = +2. \tag{2.54}$$

A similar calculation shows that $V(2, 2) = +1$. Once we know these values we can then compute $V(1, \cdot)$ and so on. This allows us to calculate backwards to obtain $V(0, 0) = +6$. This is the best possible reward we can get, and we have obtained it without resorting to brute force search over all the paths! Moreover, once we have solved for $V(t, x)$ for all t, x, we can also easily find the optimal policy to navigate this maze. We simply proceed *greedily* with respect to the value function: at time t we always go the circle in the next step with the highest $V(t, x)$ plus the immediate reward.

In fact, the above methodology is known as *dynamic programming* [Bel66]. Let us look at the computational complexity of dynamic programming versus a brute force search, which takes N^T steps. In dynamic programming, we simply have to traverse the time steps once, starting the from the end. For each time step, we have to compute N values of $V(t, x)$, each depends on a linear combination of $V(t + 1, s)$. Hence, for each time step we incur a computation overhead of N^2. Therefore, the entire dynamical programming procedure solves the problem in $N^2 T$ steps. This is *much* less than N^T!.

The key idea behind dynamic programming is defining the so called cost-to-go $V(t, x)$ (2.52), which allows us to derive a recursion in $V(t, x)$ that gives a solution to our original problem. The function $V(t, x)$ is also known as the *value function*, emphasizing the fact that it represents the "value" of a given state. This understanding will motivate the alternative approach we present next on optimal control.

2.3.2 The Dynamic Programming Principle

Now, let us state and prove the dynamic programming principle as applied to optimal control problems. We recall the Bolza problem with fixed end time:

$$\begin{aligned} &\inf_{\boldsymbol{\theta}} J[\boldsymbol{\theta}] = \int_{t_0}^{t_1} L(t, x(t), \theta(t))dt + \Phi(x(t_1)) \\ &\text{subject to} \\ &\dot{x}(t) = f(t, x(t), \theta(t)), \qquad t \in [t_0, t_1], \qquad x(t_0) = x_0. \end{aligned} \tag{2.55}$$

Following the idea of dynamic programming, we embed this problem in a *bigger* class of problems:

$$\begin{aligned} &V(s,z) := \inf_{\boldsymbol{\theta}} \int_s^{t_1} L(t,x(t),\theta(t))dt + \Phi(x(t_1)) \\ &\text{subject to} \\ &\dot{x}(t) = f(t,x(t),\theta(t)), \qquad t \in [s,t_1], \qquad x(s) = z. \end{aligned} \tag{2.56}$$

The function $V : [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}$ is called the *value function*. In words, it is the *minimum cost attainable starting from the initial condition z at time t*. Observe that $V(t_0, x_0)$ is the optimal cost of (2.55).

It may appear that we have made the problem more difficult, since we are not considering a much larger class of optimal control problems. However, it turns out that we can derive a recursion on V in terms of a partial differential equation, thereby deriving an elegant characterization of optimal controls.

Now, let us state and prove the dynamic programming principle concerning the value function for the optimal control problem.

Theorem 2.12: Dynamic Programming Principle

For every $\tau, s \in [t_0, t_1]$, $s \le \tau$, and $z \in \mathbb{R}^d$, we have

$$V(s,z) = \inf_{\boldsymbol{\theta}} \left\{ \int_s^{\tau} L(t,x(t),\theta(t))dt + V(\tau, x(\tau)) \right\}, \tag{2.57}$$

where on the right hand side, $\boldsymbol{x}$ solves $\dot{x}(t) = f(t,x(t),\theta(t))$ on $t \in [s,\tau]$ with $x(s) = z$.

The meaning of the dynamic programming principle is that the optimization problem defining $V(s,z)$ can be split into two parts:

- First, solve the optimization problem on $[\tau, t_1]$ with the usual running cost L and terminal cost Φ, but for all initial conditions $z' \in \mathbb{R}^d$. This gives us the value function $V(\tau, \cdot)$
- Next, we solve the optimization problem on $[s,\tau]$ with running cost L and terminal cost $V(\tau,\cdot)$ given by the step before.

Proof 2.12: Dynamic Programming Principle

Let us denote the right hand side of (2.57) as J^τ. We first show that $J^\tau \le V(s,z)$. We fix $\varepsilon > 0$ and choose a control $\boldsymbol{\theta} : [s,t_1] \to \Theta$ such that

$$J[\boldsymbol{\theta}] \le V(s,z) + \varepsilon. \tag{2.58}$$

This $\boldsymbol{\theta}$ always exists since $V(s,z)$ is defined as the infimum of such $J[\boldsymbol{\theta}]$. Under this control,

we have again by the definition of the value function

$$V(\tau, x(\tau)) \leq \int_{\tau}^{t_1} L(t, x(t), \theta(t))dt + \Phi(x(t_1)). \tag{2.59}$$

Then, we have

$$\begin{aligned} J^{\tau} &\leq \int_{s}^{\tau} L(t, x(t), \theta(t))dt + V(\tau, x(\tau)) && (2.60)\\ &\leq \int_{s}^{t_1} L(t, x(t), \theta(t))dt + \Phi(x(t_1)) && (2.61)\\ &= J[\boldsymbol{\theta}] \leq V(s, z) + \varepsilon. && (2.62) \end{aligned}$$

Since $\varepsilon > 0$ is arbitrary, we have $J^{\tau} \leq V(s, z)$.

Next, we show the reverse inequality. Fix $\varepsilon > 0$. Then, there exists a control $\boldsymbol{\theta}_1 : [s, \tau] \to \Theta$ such that

$$\int_{s}^{\tau} L(t, x(t), \theta_1(t))dt + V(\tau, x(\tau)) \leq J^{\tau} + \varepsilon. \tag{2.63}$$

Now, similarly there exists a control $\boldsymbol{\theta}_2 : [\tau, t_1] \to \Theta$ such that

$$\int_{\tau}^{t_1} L(t, x(t), \theta_2(t))dt + \Phi(x(t_1)) \leq V(\tau, x(\tau)) + \varepsilon. \tag{2.64}$$

This allows us to concatenate the two controls together to define

$$\theta(t) = \begin{cases} \theta_1(t) & t \in [s, \tau], \\ \theta_2(t) & t \in (\tau, t_1]. \end{cases} \tag{2.65}$$

Then, combining (2.63) and (2.65) we have

$$V(s, z) \leq J[\boldsymbol{\theta}] \leq J^{\tau} + 2\varepsilon, \tag{2.66}$$

and since $\varepsilon > 0$ is arbitrary, we obtain the desired result. □

2.3.3 Hamilton-Jacobi-Bellman Equations

In this section, we will derive the key result from the dynamic programming approach to optimal control problems, which establishes connections with partial differential equations, in particular the Hamilton-Jacobi equations. As defining the right sort of solutions for these equations turns out to be a slightly involved problem, we will proceed mostly formally in this section, but we will discuss at the end the key ideas in making these steps rigorous.

The basic motivation here is to derive an *infinitesimal* version of the dynamic programming

principle (Theorem 2.12). To this end, we will make extensive use of Taylor expansions by assuming that $\tau = s + \Delta s$ with $\Delta s \ll 1$ in Eq. (2.57), giving the infinitesimal dynamic programming principle

$$V(s,z) = \inf_{\boldsymbol{\theta}} \left\{ \int_s^{s+\Delta s} L(t, x(t), \theta(t))dt + V(s+\Delta s, x(s+\Delta s)) \right\}, \tag{2.67}$$

where again on the right hand side $\boldsymbol{x}$ follows the ODE

$$\dot{x}(t) = f(t, x(t), \theta(t)), \qquad t \in [s, s+\Delta s], \qquad x(s) = z. \tag{2.68}$$

Applying Taylor's expansion on the ODE, we have

$$x(s+\Delta s) = z + \int_s^{s+\Delta s} f(t, x(t), \theta(t))dt = z + f(s, z, \theta(s))\Delta s + o(\Delta s), \tag{2.69}$$

Furthermore, assuming that V is sufficiently regular, we have

$$V(s+\Delta s, x(s+\Delta s)) = V(s,z) + \partial_s V(s,z)\Delta s + [\nabla_z V(s,z)]^\top f(s,z,\theta(s))\Delta s + o(\Delta s). \tag{2.70}$$

Similarly, we can also expand the running cost

$$\int_s^{s+\Delta s} L(t, x(t), \theta(t))dt = L(s, z, \theta(s))\Delta s + o(\Delta s). \tag{2.71}$$

Combining (2.67), (2.70) and (2.71), we have

$$V(s,z) = \inf_{\boldsymbol{\theta}} \left\{ L(s,z,\theta(s))\Delta s + V(s,z) + \partial_s V(s,z)\Delta s + [\nabla_z V(s,z)]^\top f(s,z,\theta(s))\Delta s + o(\Delta s) \right\}. \tag{2.72}$$

Cancelling the term $V(s,z)$ on both sides and taking the limit $\Delta s \to 0$, the infimum over paths $\boldsymbol{\theta}$ on $t \in [s, s+\Delta s]$ becomes an infimum over a scalar $\theta = \theta(s) \in \Theta$, thus we obtain:

$$\partial_s V(s,z) + \inf_{\theta \in \Theta} \{ L(s,z,\theta) + [\nabla_z V(s,z)]^\top f(s,z,\theta) \} = 0. \tag{2.73}$$

This is known as the *Hamilton-Jacobi-Bellman* (HJB) equation for the value function. It remains to specify the boundary conditions. One can quickly observe that at time $s = t_1$, we in fact have by definition, $V(t_1, z) = \Phi(z)$.

Now, we note that the derivations above are purely formal for at least two reasons:

- We do not know if $V(s,z)$ is sufficiently regular to admit Taylor expansions.
- We do not know if the partial differential equation (2.73) is well-posed, i.e. whether it admits a unique solution, and in what sense should a solution be defined.

This is a common difficulty faced by many nonlinear partial differential equations. In this case, the Hamilton-Jacobi structure allows one to use the concept of *viscosity solutions* [CL83] as an appropriate notion of solution. Loosely speaking, viscosity solutions are a class of weak

solutions to nonlinear PDEs defined by being some sense of an extremum of a sequence of smooth functions that satisfy an inequality corresponding to the PDE. One can also see them as limits of solutions of the original PDE regularized with a diffusive term (hence the term "viscosity"). For more information on viscosity solutions, the reader is referred to [FS06]. With the notion of viscosity solutions, we in fact can put the HJB equations on a rigorous footing. Let us now state the main theorem in this section, whose proof we omit (but see [BP07], Theorem 8.7.1). For convenience we will replace (s, z) by (t, x) in the following.

Theorem 2.13: Hamilton-Jacobi-Bellman Equation

Let $V : [t_0, t_1] \times \mathbb{R}^d \to \mathbb{R}$ be the value function defined by (2.56). Then, V is the unique viscosity solution of the Hamilton-Jacobi-Bellman equation

$$\begin{aligned} &\partial_t V(t,x) + \inf_{\theta \in \Theta} \left\{ L(t,x,\theta) + [\nabla_x V(t,x)]^\top f(t,x,\theta) \right\} \qquad (t,x) \in (t_0,t_1) \times \mathbb{R}^d \\ &V(t_1,x) = \Phi(x) \end{aligned} \tag{2.74}$$

2.3.4 Implications for Optimal Control

Recall that we have the correspondence

$$V(t_0, x_0) = \inf_{\boldsymbol{\theta}} J[\boldsymbol{\theta}], \tag{2.75}$$

hence the solution of the HJB equations will give us the optimal cost that we can obtain for the Bolza problem. In fact, we will see that this gives us much more.

A Necessary Condition. It should be clear from our discussions so far that what we have formally derived is that the HJB constitutes a necessary condition for global optimality. Indeed, suppose we have a family of optimal controls $\{\boldsymbol{\theta}^*_{s,z} : s \in [t_0, t_1], z \in \mathbb{R}^d\}$ and define

$$\begin{aligned} &\widehat{V}(s,z) = \Phi(x^*_{s,z}(t_1)) + \int_s^{t_1} L(t, x^*_{s,z}(t), \theta^*_{s,z}(t))dt, \\ &\text{where } \dot{x}^*_{s,z}(t) = f(t, x^*_{s,z}(t), \theta^*_{s,z}(t)), \qquad t \in [s, t_1], \qquad x^*_{s,z}(s) = z. \end{aligned} \tag{2.76}$$

Then, by Theorem 2.13 $\widehat{V} \equiv V$ satisfies the HJB equation.

In fact, let us fix $s, \tau \in [t_0, t_1)$ and $z \in \mathbb{R}^d$. By the assumption of global optimality we can rewrite the dynamic programming principle (2.57) as

$$\begin{aligned} V(s,z) &= \inf_{\boldsymbol{\theta}} \left\{ \int_s^\tau L(t, x(t), \theta(t))dt + V(\tau, x(\tau)) \right\} \\ &= \int_s^\tau L(t, x^*_{s,z}(t), \theta^*_{s,z}(t))dt + V(\tau, x^*_{s,z}(\tau)). \end{aligned} \tag{2.77}$$

We may now proceed as before using Taylor expansions to derive an infinitesimal version of the above. Let us call $\boldsymbol{\theta}^* = \boldsymbol{\theta}^*_{t_0,x_0}$ the optimal control for our original problem, and $\boldsymbol{x}^*$ is corresponding controlled state trajectory. Then, Taylor expanding and comparing with the usual dynamic programming principle we obtain the equality

$$\begin{aligned} -\partial_s V(s, x^*(t)) &= \min_{\theta\in\Theta} \left\{ L(t, x^*(t), \theta(t)) + [\nabla_x V(t, x^*(t))]^\top f(t, x^*(t), \theta) \right\}, \\ &= L(t, x^*(t), \theta^*(t)) + [\nabla_x V(t, x^*(t))]^\top f(t, x^*(t), \theta^*(t)), \end{aligned} \tag{2.78}$$

which we can rewrite as

$$H(t, x^*(t), -\nabla_x V(t, x^*(t)), \theta^*(t)) = \max_{\theta\in\Theta} H(t, x^*(t), -\nabla_x V(t, x^*(t)), \theta) \tag{2.79}$$

where the Hamiltonian is defined exactly as in the case of the PMP (2.17)

$$H(t, x, p, \theta) = p^\top f(t, x, \theta) - L(t, x, \theta). \tag{2.80}$$

Thus, this is similar to the statement of the PMP, except that the co-state $p^*(t)$ is now replaced by $-\nabla_x V(t, x^*(t))$. However, there is a nontrivial difference in that now, this is also a sufficient condition for global optimality, as we now show.

A Sufficient Condition. Let us now assume that a continuously differentiable function V satisfies the HJB (2.74) and moreover that a control $\widehat{\boldsymbol{\theta}} : [t_0, t_1] \to \Theta$ satisfies

$$H(t, \widehat{x}(t), -\nabla_x V(t, \widehat{x}(t)), \widehat{\theta}(t)) = \max_{\theta\in\Theta} H(t, \widehat{x}(t), -\nabla_x V(t, \widehat{x}(t)), \theta), \tag{2.81}$$

for all $t \in [t_0, t_1]$, where $\widehat{\boldsymbol{x}}$ is the state process corresponding to the control $\widehat{\boldsymbol{\theta}}$, then $\widehat{\boldsymbol{\theta}}$ is a globally optimal control that solves (2.55) with optimal cost $V(t_0, x_0)$.

To show this, observe that if we set $x = \widehat{x}(t)$ in the HJB equation for V, noting the condition (2.81), we have

$$\partial_t V(t, \widehat{x}(t)) + [\nabla_x V(t, \widehat{x}(t))]^\top f(t, \widehat{x}(t), \widehat{\theta}(t)) + L(t, \widehat{x}(t), \widehat{\theta}(t)) = 0, \tag{2.82}$$

which means

$$\frac{d}{dt} V(t, \widehat{x}(t)) + L(t, \widehat{x}(t), \widehat{\theta}(t)) = 0. \tag{2.83}$$

Integrating from t_0 to t_1 and using the boundary condition $V(t_1, x) = \Phi(x)$, we have

$$V(t_0, x_0) = \int_{t_0}^{t_1} L(t, \widehat{x}(t), \widehat{\theta}(t))dt + \Phi(\widehat{x}(t_1)) = J[\widehat{\boldsymbol{\theta}}]. \tag{2.84}$$

On the other hand, if $\boldsymbol{\theta}$ be any other control whose trajectory is $\boldsymbol{x}$, we would have

$$\partial_t V(t, x(t)) + [\nabla_x V(t, x(t))]^\top f(t, x(t), \theta(t)) + L(t, x(t), \theta(t)) \geq 0, \tag{2.85}$$

which yields

$$0 \leq \underbrace{\int_{t_0}^{t_1} L(t, x(t), \theta(t))dt + V(t_1, x(t_1))}_{J[\boldsymbol{\theta}],\ \text{since } V(t_1,x(t_1))=\Phi(x(t_1))} - V(t_0, x_0), \tag{2.86}$$

or

$$J[\widehat{\boldsymbol{\theta}}] = V(t_0, x_0) \leq J[\boldsymbol{\theta}]. \tag{2.87}$$

This shows that $\widehat{\boldsymbol{\theta}}$ is globally optimal, with cost $V(t_0, x_0)$.

Example 2.14: Nondifferentiable Value Function ([Lib12], Example 5.2.1)

Consider the scalar control system

$$\dot{x}(t) = x(t)\theta(t), \qquad t \in [0, T], \qquad x(0) = x_0 \in \mathbb{R}, \qquad \theta(t) \in \Theta \equiv [-1, 1]. \tag{2.88}$$

We set running cost $L \equiv 0$ and terminal cost $\Phi(x) = x$. The optimal control is just $-\text{Sign}(x_0)$ if $x_0 \neq 0$, and if $x_0 = 0$ the cost is always 0. Hence, the value function is simply

$$V(t, x) = \begin{cases} e^{-(T-t)}x & \text{if } x > 0 \\ e^{T-t}x & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases} \tag{2.89}$$

Observe that it is not differentiable at $x = 0$.

Let us now check that the value function satisfies the HJB, which is now

$$\partial_t V(t, x) - |x\partial_x V(t, x)| = 0, \qquad V(T, x) = x. \tag{2.90}$$

Clearly, this is the case. In fact, we can derive the value function from the HJB by applying the method of characteristics (See [Eva98], Ch. 3).

Remark. *We end this section with a remark on the HJB solution. Recall that we can write the optimal control as*

$$\theta^*(t) = u(t, x^*(t)) := \min_{\theta}\{L(t, x^*(t), \theta) + [\nabla_x V(t, x^*(t))]^\top f(t, x^*(t), \theta)\}. \tag{2.91}$$

In other words, provided we can solve the HJB, the optimal control solution is of feed-back *or* closed-loop *form, meaning that it tells how to steer the system by just observing the state trajectory $\boldsymbol{x}^*$. We can contrast with the PMP, where we obtain* open-loop *controls that are pre-computed (since it also depends on the co-state) and cannot be applied on-the-fly. This is an important distinction.*

2.3.5 Further Reading

The principle of optimality has been referenced in different manners throughout the development of calculus of variations, dating back to the solution of the Brachistochrone problem of Jacob Bernoulli in 1697. The building of the Hamilton-Jacobi-Bellman theory for optimal control rests on important works of Carathéodory, Bellman and Kalman in the early 1900s. The theory is first put on rigorous footing via the introduction of viscosity solutions by Crandall and Lions [CL83]. See also [FS06] for a general exposition of viscosity solutions. Here we also omitted the interesting topic of how the HJB and the PMP are related. In fact, they can be related via the method of characteristics ([Eva98] Ch. 3): the PMP equations can be interpreted, at least formally, as characteristic equations associated with the HJB. See [Lib12], Ch. 5.2.

3 Dynamical Systems Meets Deep Learning

In this chapter, we enter the main application topic of these notes, which is the connections between dynamical systems and deep learning, first proposed in [E17]. The key connecting piece turns out to be the theory of optimal control that we spent a significant amount of time introducing. We will first start with the formulation of the training problem in deep residual networks as a variant of the classical optimal control problem. We will go on to discuss some applications of these ideas, including novel algorithms, model architectures and other mathematical results.

3.1 A Mean-field Optimal Control Formulation of Deep Learning

Review: Supervised Learning with Residual Networks. We start by reviewing the basic residual network architecture. Let x denote the input data and y its corresponding label. Let us recall the form of the deep residual networks that transforms the input x via

$$\begin{aligned} &x(k+1) = x(k) + f(x(k), \theta(k)), \qquad k = 0, \dots, K-1 \qquad x(0) = x, \qquad \theta(k) \in \Theta, \\ &\widehat{y} = g(x(K)). \end{aligned} \tag{3.1}$$

where we got rid of explicit k dependence in f via the usual trick, and x_K is final output of the network, which is then transformed by another function $g : \mathbb{R}^d \to \mathbb{R}^m$ (the latter being the space of outputs/labels). The goal of learning is to minimize the distance between the true label y and the predicted label $\widehat{y} = g(x(K))$. This is usually measured through some loss function $\Phi(g(x(K)), y)$. When discussing optimization problems, we will simplify things by absorbing g into Φ (by redefining Φ if necessary) so that the loss is $\Phi(x(K), y)$. Otherwise, we can consider the case where $g \in \mathcal{G}$, the latter a family of "terminal" functions.

Here, technically the parameter set Θ can vary with k, but usually these sets just Euclidean spaces so one can embed them into a common one of the largest dimension. Recall also from Sec. 1.2 that the input-output pair (x, y) is not a fixed value, but rather drawn according to a distribution μ. Hence, putting things together, the supervised learning problem seeks the

solution of the following problem

$$\begin{aligned} &\inf_{(\theta(0),\ldots,\theta(K-1))\in\Theta^K} \mathbb{E}_{(x,y)\sim\mu}\left[\Phi(x(K),y)+\sum_{k=0}^{K-1} L(x(k),\theta(k))\right] \\ &\text{subject to} \\ &x(k+1)=x(k)+f(x(k),\theta(k)), \qquad k=0,\ldots,K-1, \qquad x(0)=x. \end{aligned} \tag{3.2}$$

Here, we have introduced a regularization term $L : \mathbb{R}^d \times \Theta \to \mathbb{R}$. For example, a simple ℓ^2 regularizer often applied in practice translates to $L(x,\theta) = \lambda\|\theta\|^2$, but we will consider the general case of potentially complicated regularizers.

The key difficulty in analyzing (3.2) is the repeated compositional structure present in the difference equation. The main idea in circumventing this is to introduce the dynamical systems viewpoint, where we replace the discrete dynamics (3.1) by a continuous dynamical system.

The Continuous-time Idealization and Mean-field Optimal Control Problem. The central approach in these notes is the continuous-time (or continuum) idealization of the discrete dynamical system represented by (1.33). In other words, we replace it with a continuous-time dynamics

$$\dot{x}(t) = f(x(t),\theta(t)), \qquad t\in[0,T], \qquad x(0)=x. \tag{3.3}$$

This approximation is of conceptual importance, as it passes from studying the intricate issues in a deep neural network to the study of continuous-time dynamical systems. For the latter, there are many more tools available to us.

From the approximation viewpoint, the current hypothesis is built from flow maps. An interesting problem of approximation is when this hypothesis space has the universal approximation property. We will revisit this point later in Sec. 3.5.1, but for now we will focus on the optimization problem.

The bridge that connects (3.1) and (3.3) is the forward Euler method for numerically solving ODEs, as we introduced in Sec. 1.3.3. Here, a factor of Δt is missing, but this can be justified by observing that in a trained deep ResNet, the transformation in each residual block is close to the identity [VWB16].

Hence, in the continuous-time case, we may replace the original problem (3.2) by (absorbing the final layer g into Φ)

$$\begin{aligned} &\inf_{\boldsymbol{\theta}\in L^\infty([0,T],\Theta)} \mathbb{E}_{(x,y)\sim\mu}\left[\Phi(x(T),y)+\int_0^T L(x(t),\theta(t))dt\right] \\ &\text{subject to} \\ &\dot{x}(t)=f(x(t),\theta(t)), \qquad t\in[0,T], \qquad x(0)=x. \end{aligned} \tag{3.4}$$

This is *almost* like a standard Bolza problem (2.16), except for the following:

- Initial condition to the ODE is random
- Dependence of Φ on y (random)
- Expectation over μ in the cost

We call (3.4) a *mean-field* optimal control problem. The choice of name in fact requires justification, as there is no explicit mean-field dynamics (c.f. Mckean-Vlasov systems [Szn91]) involved here. Rather, we call it mean-field to emphasize the fact that we seek an optimal control that is shared with all input-label pairs (x, y) that are jointly distributed according to μ.

Going from (3.2) to (3.4), we can see that the dynamical systems and control viewpoint highlights one important aspect of deep learning: *whereas the learning problem for shallow models can be framed as an optimization problem, the learning problem for deep residual networks can be viewed as an optimal control problem.*

3.2 Optimality Conditions

As with usual calculus of variations problems, we can ask for the optimality conditions associated with the deep learning problem recast as mean-field optimal control (3.4). In this section, we will state two recent results in this direction, entirely analogous to the development of classical optimal control.

3.2.1 Mean-field Pontryagin's Maximum Principle

We start with a result analogous to the Pontryagin's maximum principle (Thm. 2.7) gives general necessary conditions for optimality for our problem (3.4).

Theorem 3.1: Mean-field Pontryagin's Maximum Principle

Let f be bounded, f, L be continuous in θ, and f, L, Φ be continuously differentiable with respect to x. Assume further that μ has bounded support in $\mathbb{R}^d \times \mathbb{R}^m$. Suppose $\boldsymbol{\theta}^* \in L^\infty([0,T], \Theta)$ be an optimal control. Then, there exists absolutely continuous stochastic processes $\boldsymbol{x}^*, \boldsymbol{p}^*$ such that

$$\dot{x}^*(t) = f(x^*(t), \theta^*(t)), \qquad x^*(t) = x, \tag{3.5}$$

$$\dot{p}^*(t) = -\nabla_x H(x^*(t), p^*(t), \theta^*(t)), \qquad p_T^* = -\nabla_x \Phi(x^*(T), y), \tag{3.6}$$

$$\mathbb{E}_\mu H(x^*(t), p^*(t), \theta^*(t)) \geq \mathbb{E}_\mu H(x^*(t), p^*(t), \theta), \quad \forall\, \theta \in \Theta, \quad a.e.\ t \in [0,T], \tag{3.7}$$
$$(x, y) \sim \mu$$

The proof of this results follows closely that of Theorem 2.7, and hence we omit here. The proof can be found in [EHL19]. This result has algorithmic implications and is intricately connected

with back-propagation, as we will see later.

3.2.2 Mean-field Hamilton-Jacobi-Bellman Equations

Just like the PMP, the HJB also has a natural extension to the mean-field case. In this case, we may obtain a necessary and sufficient condition for optimality, which is not something that has been well explored in a machine learning context. The statement and a proof of the result is slightly more involved, and we will try to state its key elements here.

- First, one needs to define a value function. This should not be hard as we simply adapt the usual *cost to go*

$$\begin{aligned} V(s,\rho) = \inf_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim\rho} \left[\int_t^T L(x(t),\theta(t))dt + \Phi(x(T),y) \right] \\ \text{subject to} \\ \dot{x}(t) = f(x(t),\theta(t)), \qquad t \in [s,T], \qquad x(s) = x. \end{aligned} \tag{3.8}$$

- Next, recall in (2.74) that we need to take the gradient with respect to the state. Here the state is ρ, a probability measure, so we need to define a notion of derivative. It turns out a convenient technique to use is *lifting* [Car10]. Concretely, let $u(\rho)$ be a real-valued function(al) on the space of square-integrable measures (say on $\mathbb{R}^d$). We can lift it to a function on square-integrable random variables

$$u(\rho) = U(Z) \text{ where } \mathbb{P}_Z = \rho \ (\mathbb{P}_Z \text{ denotes the law of } Z)\ . \tag{3.9}$$

  Now, $U(Z)$ is a functional on a Hilbert space, and hence we can define its Fréchet derivative $DU(Z)$. One can further show that $DU(Z)$'s law only depends on the law of Z and not on Z itself. Thus we can view $X \mapsto DU(X)$ a mapping from $\mathbb{R}^d$ to $\mathbb{R}^d$. Therefore, we may simply define $\partial_\rho u(\rho)$ to be the function $x \mapsto DU(x)$.

Theorem 3.2: Mean-field Hamilton-Jacobi-Bellman Equation

The value function (3.8) is the unique viscosity solution[a] to the following mean-field Hamilton-Jacobi-Bellman equation

$$\begin{aligned} &\partial_t V(t,\rho) + \inf_{\theta\in\Theta} \int_{\mathbb{R}^{d+m}} L(x,\theta) + [\partial_\rho V(t,\rho)(x,y)]^\top [f(x,\theta),0] d\rho(x,y) = 0, \\ &V(T,\rho) = \int_{\mathbb{R}^{d+m}} \Phi(x,y) d\rho(x,y). \end{aligned} \tag{3.10}$$

[a] This is a natural extension of the classical definition of viscosity solution

The proof of Theorem 3.2 and auxiliary results can be found in [EHL19].

3.3 Control Inspired Learning Algorithms

Recall that the mean-field Pontryagin's maximum principle is a general necessary condition of optimality for mean-field optimal formulation of deep learning. Hence, it should give rise to training algorithms just like how the condition $\nabla\Phi(\theta^*) = 0$ gives rise to gradient descent. Moreover, the dynamical structure of the problem also hints at adapting other methodologies in numerical analysis of differential equations to deep learning. In this section, we survey some work in this direction. For simplicity of presentation, we will only consider empirical risk (and often we will only consider the 1-sample case for ease of writing). In this case, the mean-field PMP reduces to its classical counter-part 2.7.

3.3.1 Method of Successive Approximations

The first is the so-called *method of successive approximations* (MSA) [CL82] or the *sweeping method*, which is perhaps the simplest method for finding a solution of the PMP equations. The PMP equations are 3 coupled equations in 3 unknowns $\boldsymbol{x}^*, \boldsymbol{p}^*, \boldsymbol{\theta}^*$. Moreover, given any two of them, computing the third is straightforward, at least conceptually. Hence, the basic MSA method proceeds as follows.

We start with an initial guess $\boldsymbol{\theta}^0$ for the optimal control. At the n^{th} iteration we solve

$$\dot{x}^n(t) = f(x^n(t), \theta^n(t)) \qquad x^n(0) = x \tag{3.11}$$

$$\dot{p}^n(t) = -\nabla_x H(x^n(t), p^n(t), \theta^n(t)) \qquad p^n(T) = -\nabla_x \Phi(x^n(T), y) \tag{3.12}$$

$$\theta^{n+1}(t) = \arg\max_{\theta\in\Theta} H(x^n(t), p^n(t), \theta). \tag{3.13}$$

If $(\boldsymbol{x}^n, \boldsymbol{p}^n, \boldsymbol{\theta}^n)$ converges, then the limit must be a solution of the PMP. In the N-sample case, according to the mean-field formulation (with μ the empirical measure), we would then solve the MSA

$$\dot{x}_i^n(t) = f(x_i^n(t), \theta^n(t)) \qquad x_i^n(0) = x_i \tag{3.14}$$

$$\dot{p}_i^n(t) = -\nabla_x H(x_i^n(t), p_i^n(t), \theta^n(t)) \qquad p_i^n(T) = -\nabla_x \Phi(x_i^n(T), y_i) \tag{3.15}$$

$$i = 1, \dots, N$$

$$\theta^{n+1}(t) = \arg\max_{\theta\in\Theta} \frac{1}{N}\sum_{i=1}^{N} H(x_i^n(t), p_i^n(t), \theta). \tag{3.16}$$

We now show that this method includes as a special case the classical back-propagation algorithm for deep learning by a discretization argument.

MSA and Back-propagation. Consider for simplicity $L \equiv 0$. We will drop the superscript n, assume the sample size $N = 1$, and let $\boldsymbol{x}$ be the state under some control $\boldsymbol{\theta}$. Let us now discretize (3.11) in time using a forward Euler discretization to obtain

$$\widehat{x}(k+1) = \widehat{x}(k) + \Delta t f(\widehat{x}(k), \widehat{\theta}(k)), \qquad k = 0, \dots, K-1 \quad (K\Delta t = T) \tag{3.17}$$

Now fix $k < K$ and regard $\widehat{x}(K)$ as a function of $\widehat{x}(k)$, we have by the chain rule (D denotes total derivative)

$$\begin{aligned} D_{\widehat{x}(k)}\Phi(\widehat{x}(K)) &= [D_{\widehat{x}(k)}x(k+1)]^\top D_{x(k+1)}\Phi(\widehat{x}(K)) \\ &= [I + \Delta t \nabla_x f(\widehat{x}(k), \widehat{\theta}(k))]^\top D_{x(k+1)}\Phi(\widehat{x}(K)). \end{aligned} \tag{3.18}$$

Letting $\widehat{p}(k) := -D_{\widehat{x}(k)}\Phi(\widehat{x}(K))$, we see that

$$\widehat{p}_{k+1} = \widehat{p}_k - \Delta t[\nabla_x f(\widehat{x}(k), \widehat{\theta}(k))]^\top \widehat{p}_{k+1}, \qquad k = 0, \dots, K-2, \qquad \widehat{p}_K = -\nabla\Phi(\widehat{x}(K)), \tag{3.19}$$

which one recognizes as the (backward) Euler discretization of the co-state equation. In other words, the costate here tells us the sensitivity of the loss function with respect to the state.

A further application of chain rule shows that

$$-D_{\widehat{\theta}(k)}\Phi(\widehat{x}(K)) = \nabla_\theta H(\widehat{x}(k), \widehat{p}(k+1), \widehat{\theta}(k)). \tag{3.20}$$

Thus, we have shown that applying gradient descent on the loss function Φ is equivalent to performing gradient ascent on H, i.e. we are approximately maximizing H when we perform gradient descent.

Thus, back-propagation with gradient descent is equivalent to the MSA if we replace the step (3.13) by

$$\theta^{n+1}(t) = \theta^n(t) + \eta \nabla_\theta H(x^n(t), p^n(t), \theta). \tag{3.21}$$

In other words, we can view MSA as a generalization of the back-propagation algorithm. This leads to the question of whether we can derive other algorithms from the MSA directly, without replacing this step. It turns out that this is possible [LCTE17], and furthermore one can derive algorithms to train quantized neural networks owing to the property that the PMP generally applies to arbitrary parameter sets Θ [LH18]. Other applications of the MSA/adjoint method includes adversarial training [ZZL+19] and generative models [CRBD19].

3.3.2 Layer Parallel Training Algorithms

Besides the Hamiltonian maximization property which leads to different types of algorithms to GD/SGD, there is another aspect of the PMP that leads to other algorithmic innovations, namely layer-parallel training. This refers to learning algorithms that can use many processors to train a deep neural network in parallel, not in terms of data splitting but layer splitting.

One way to motivate such methods is to look at the PMP equations, which consist of two components:

1. Solving the dynamical equations for $\boldsymbol{x}^*$ and $\boldsymbol{p}^*$
2. Solving the Hamiltonian maximization problem for $\boldsymbol{\theta}^*$

A crucial observation is that given component 1, component 2 can be computed in parallel for all t, i.e. this step is naturally layer-parallel and each layer need not talk to one another. Thus, to achieve full layer parallelism, it is enough to make the solution of $\boldsymbol{x}^*$ and $\boldsymbol{p}^*$ parallel. There are at least two may ways of doing so, each relying on different aspects of the Hamilton's equations.

Exploiting the Two-Point BVP Form. Notice that given $\boldsymbol{\theta}$, the equations

$$\dot{x}(t) = f(x(t), \theta(t)) \qquad x(0) = x \tag{3.22}$$

$$\dot{p}(t) = -\nabla_x H(x(t), p(t), \theta(t)) \qquad p(T) = -\nabla_x \Phi(x(T), y) \tag{3.23}$$

constitute a *two-point boundary value problem (2P-BVP)*, in that the equation for $\boldsymbol{x}$ has a initial condition whereas the equation for $\boldsymbol{p}$ has a terminal condition. Hence, one can think of breaking this down into two sub-problems: let $S = T/2$ and consider

$$P1 \quad (t \in [0, S]): \qquad \begin{aligned} \dot{p}^n(t) &= -\nabla_x H(x^{n-1}(t), p^n(t), \theta^n(t)) \qquad p^n(S) = p^{n-1}(S) \\ \dot{x}^n(t) &= f(x^n(t), \theta^n(t)) \qquad x^n(0) = x \end{aligned} \tag{3.24}$$

$$P2 \quad (t \in [S, T]): \qquad \begin{aligned} \dot{x}^n(t) &= f(x^n(t), \theta^n(t)) \qquad x^n(S) = x^{n-1}(S) \\ \dot{p}^n(t) &= -\nabla_x H(x^n(t), p^n(t), \theta^n(t)) \qquad p^n(T) = -\nabla_x \Phi(x^n(T), y) \end{aligned} \tag{3.25}$$

The two problems P_1 and P_2 can now be run in parallel on two processors, and at the end of each run the boundary values are passed between them in a communication round. This the main idea of the algorithm proposed in [PM19], but there are additional correction steps that we do not discuss here in detail. We remark here that the method here requires synchronization between P1 and P2, and extension to more than two processors requires additional work.

Exploiting Continuity in time. An alternative approach is the multi-grid approach directly on the state and co-state ODEs. This is based on the multi-grid in time idea in numerical analysis [FFK+14], which is an adaptation of the classical spatial (non-linear) multi-grid method for solving boundary-value problems.

The basic observation in this case is that for a discretized (say forward Euler) ODE (we absorbed $\widehat{\theta}(k)$ and $k\Delta t$ into the definition of f)

$$\widehat{x}(k+1) = \widehat{x}(k) + \Delta t f(k, \widehat{x}(k)), \qquad k = 0, \dots, K-1, \qquad \widehat{x}(0) = x_0, \tag{3.26}$$

we can define a vector X and a vector-valued function A so that

$$X := \begin{pmatrix} \widehat{x}(0) \\ \widehat{x}(1) \\ \vdots \\ \widehat{x}(K) \end{pmatrix}, \qquad A(X) := \begin{pmatrix} X_0 - x_0 \\ X_1 - f(0, X_0) \\ \vdots \\ X_K - f(K-1, X_{K-1}) \end{pmatrix}. \tag{3.27}$$

Then, we can write (3.26) as

$$A(X) = 0. \tag{3.28}$$

The feed-forward method (3.26) solves (3.28) by sequential substitution, since $A(X)$ has a "lower triangular" form, in the sense that the k^{th} row of its output does not depend on $X_{k+1}, \ldots, X_K$.

Alternatively, one can also attempt to solve (3.28) iteratively, say by minimizing $\min_X \|A(X)\|^2$. In this case, one can start with initial guess X^0 and refine this guess – and the crucial point is that the relaxation is now no longer sequential in time and can be parallelized. The multi-grid method builds on this idea, where instead of solving this minimization problem in one go, it solves it by iterating a sequence of grids of varying sizes, and transferring the information of the solution across the grids.

For this to be sensible, one should expect some regularity in time in the solution X, so that restriction/interpolation operations between grids of different resolution give meaningful initializations for their respective temporal scales. This is ensured by the ODE structure of the dynamics that naturally builds temporal regularity in the states and co-states. Recent work based on this approach include [CMH+18b, GRS+20].

3.3.3 Summary and Outlook

Let us summarize the discussion so far on control inspired learning algorithm. From the big picture, they are all derived by picturing the original learning problem in terms of a control problem of a dynamical system. By appealing to the form of the maximum principle, new optimization algorithms can be developed. On the other hand, by appealing to the dynamical evolution of forward and adjoint equations, one can derive layer-parallel methods based on traditional numerical analysis.

There are many future avenues for development when it comes to numerical algorithms for deep learning from the control viewpoint. Here we motivate some possible directions that has not been thoroughly explored

1. Some important methods from numerical optimal control has not been tested on deep learning. Notable examples include collocation methods (See survey [Rao10]) and multi-scale needle-perturbation type of methods [CL82].
2. The training methods developed so far are generic, in the sense that it tries to apply to all forms of the dynamics modelling the network architecture. It is plausible to ask, can we get more efficient training method if we change the feed-forward dynamics? This is partially explored in [LH18], and this topic is also related to the work that relies on building architectures that we will discuss later
3. It should be apparent from the discussion that none of these algorithm relies on the Hamilton-Jacobi-Bellman approach. Numerical algorithms based on the HJB is an interesting point of exploration

4. Lastly, the reverse question is also meaningful. Since learning and control can be connected, it is possible to go the reverse direction and ask what learning algorithms we have can help solve control, or PDEs. This has been pursued in many recent works and further developments are expected

3.4 Control Inspired Architectures

Besides training algorithms, the connection between continuous-time dynamical systems and deep learning also gives rise to novel model architectures. Classical numerical analysis now acts as the bridge between dynamics and architectures. Indeed, recall from Sec. 1.2.5 that the ResNet architecture is a forward Euler discretization of the ODE

$$\dot{x}(t) = f(x(t), \theta(t)) \qquad \rightarrow \qquad \widehat{x}(k+1) = \widehat{x}(k) + \Delta t f(\widehat{x}(k), \widehat{\theta}(k)). \tag{3.29}$$

Of course, one do not have to restrict to such a discretization and can consider general schemes. In the following we will regard $\widehat{x}(k)$ as the iterates in a general finite-difference discretization scheme – each of which will correspond to a deep ResNet architecture.

Finite-difference discretization of differential equations is a well-studied topic in numerical analysis. There, one is interested in several types of questions:

1. Stability: Do the iterates $\widehat{x}(k)$ evolve stably? Do perturbations to the state get amplified or damped as it proceeds?
2. Consistency: Does the continuous time solution satisfy the discretized equation to leading order?
3. Convergence: Do we have $\|\widehat{x}(k) - x(k\Delta t)\| \to 0$? Does it converge uniformly? What is the convergence rate?

Since each discretization scheme corresponds to an architecture, our knowledge on these questions can lead to novel ways to build neural networks. Let us now discuss some works in these directions.

3.4.1 Constraining Weights to Guarantee Stability

We first discuss the issue of stability. We start with a motivating example.

Example 3.3: Stability of Linear Systems

Consider the simple linear ODE

$$\dot{x}(t) = Ax(t), \quad x(0) = x_0 \qquad \Rightarrow \qquad x(t) = e^{tA}x_0. \tag{3.30}$$

Let $x_\varepsilon(t) = e^{tA}x_\varepsilon$, i.e. solution of the ODE with a different initial condition x_ε where

$\|x_\varepsilon - x_0\| \leq \varepsilon$. Now, suppose that A is diagonalizable with eigenvalues $\lambda_1, \ldots, \lambda_d \in \mathbb{C}$ and corresponding unit eigenvectors $v_1, \ldots, v_d \in \mathbb{C}^d$. Moreover, suppose $x_0 = \sum_{i=1}^d \alpha_i v_i$ and $x_\varepsilon = \sum_{i=1}^d \beta_i v_i$. Then,

$$x(t) - x_\varepsilon(t) = \sum_{i=1}^{d} (\alpha_i - \beta_i) e^{\lambda_i t}. \tag{3.31}$$

One can see that: 1) If $\Re(\lambda_i) \leq 0$ for all i then $\|x(t) - x_\varepsilon(t)\|$ is bounded; and 2) If $\Re(\lambda_i) > 0$ for some i and $\alpha_i \neq \beta_i$ then $\|x(t) - x_\varepsilon(t)\| \to \infty$ as $t \to \infty$.

From the example above we see that not all ODE solutions are stable to small perturbations. We have not defined what stability means precisely. The following is a simple definition, but we note that there are many other definitions of stability depending on the application scenario.

Definition 3.4: Stability of ODE

Let $\boldsymbol{x}$ be the solution of a well-posed initial value problem $\dot{x}(t) = f(t, x(t))$, $x(0) = x_0$. We say that it is stable if for every $\varepsilon > 0$ and every $\boldsymbol{x}_\epsilon$ which is a solution of the same problem with initial condition x_ε such that $\|x_\varepsilon - x_0\| \leq \varepsilon$, we have a $\delta > 0$ such that

$$\|x(t) - x_\varepsilon(t)\| \leq \delta \text{ for all } t \geq 0. \tag{3.32}$$

Example 3.3 gives conditions for a linear system to be stable. For nonlinear systems, a similar condition of stability can be derived, at least locally around some reference point, by replacing A with the jacobian $J = \nabla_x f$. In this case the local stability condition will become

$$\Re[\lambda_i(J)] \leq 0 \text{ for all } i = 1, \ldots, d. \tag{3.33}$$

Finally, for the discrete iterations a similar notion of and condition for stability can be defined.

Why do we want stability in neural networks? Recall that the dynamical system (continuous or discrete) is there to transform the input x to a final form $x(T)$ so as to match some output y. A basic requirement for such a network is continuity: if we vary the input by a very small amount, the output should only vary by a small amount as well. This is necessary for the model to be robust to noise, e.g. sampling noise due to finite training sample. This can be directly related to the issue of generalization.

With this observation in mind, one can then construct neural network architectures that have stability guarantees. For example, suppose that A is skew-symmetric, i.e. $A^T = -A$. Then, one can show that its eigenvalues must have zero real part. One can thus guarantee the stability of a linear system by forcing A to be skew-symmetric. A simple way to do so is to write

$$A = B - B^\top \tag{3.34}$$

where B is an unconstrained matrix. One can check that $A^\top = -A$ for any B. Hence, one can reparameterize the neural network in a similar way:

$$\dot{x}(t) = \sigma(W(t)x(t) + b(t)) \qquad \rightarrow \qquad \dot{x}(t) = \sigma([V(t) - V(t)^\top]x(t) + b(t)). \tag{3.35}$$

Note that in the nonlinear case the skew-symmetry of the Jacobian does not strictly hold, unless $\mathrm{Diag}\left[\sigma'\left(([V(t) - V(t)^\top]x(t) + b(t))\right)\right]$ commutes with $V(t) - V(t)^\top$. This approach is explored in [HR17], where the authors also explored other means to ensure stability, including symplectic integration, giving rise to yet another type of architecture based on Hamiltonian dynamics. These ideas are also applied to recurrent networks [CCHC19].

In a different vein, [ZS19] considers the *post-activation* form of the ResNet, which is

$$\widehat{x}(k+1) = \sigma_2\left(\widehat{x}(k) + \tau\widehat{W}_2(k)\sigma_1(\widehat{W}_1(k)\widehat{x}(k) + \widehat{b}_1(k)) + \tau\widehat{b}_2(k)\right) \tag{3.36}$$

Assuming that $\sigma_2(z) = \max(0, z)$, one can then show that (3.36) is a consistent discretization of the differential inclusion

$$-\dot{x}(t) + W_2(t)\sigma(W_1(t)x(t) + b_1(t)) + b_2(t) \in \partial\mathbb{1}_{\mathbb{R}^d_+}(x). \tag{3.37}$$

The last term is the sub-differential of the indicator function of $\mathbb{R}^d_+$, which corresponds to the normal cone of $\mathbb{R}^d_+$. By imposing sign constraints on $W_2(t)$ (forcing it to be negative), one can derive stability estimates of the type in Def. 3.4.

Lastly, we also mention a recent work [YLBS20], which explores architectures based on a continuous model that interpolates between residual and non-residual networks. There, the authors consider

$$\dot{x}(t) = -\lambda x(t) + \rho(\lambda)f(t, x(t), \theta(t)), \tag{3.38}$$

where $\rho : [0, \infty) \rightarrow [0, \infty)$ with $\rho(\lambda) \rightarrow 1, \lambda \rightarrow 0^+$and $\rho(\lambda) \sim \lambda, \lambda \rightarrow +\infty$. By adjusting the weight λ and discretizing, one can interpolate between a residual and non-residual network. It is shown that this approach also improves stability for appropriate choices of λ, due to the damping effect of the first term. This is similar to the differential inclusion approach where W_2 is negated.

3.4.2 Architectures from Other Finite Difference Discretization Schemes

Instead of constraining weights to achieve stability, another direction is to derive novel architectures based on different finite-difference discretization schemes. In this viewpoint, one keeps in mind an underlying dynamical equation (in the form of an ODE) that is the "true" model of the feed-forward dynamics. Thus, in this viewpoint one can view architectural design of the neural net as a way to find a good discretization scheme, either to improve accuracy or stability.

To this end, suppose we have a underlying deep model in the form of an ODE

$$\dot{x}(t) = f(t, x(t), \theta(t)) =: \tilde{f}(t, x(t)), \tag{3.39}$$

The usual ResNet then corresponds to the forward Euler discretization of the above. Are there other architectures that we have invented that corresponds to other discretization schemes?

Indeed, this is the case as observed in [LZLD18]. For example, the *backward* Euler discretization of (3.39) reads

$$\widehat{x}(k+1) = \widehat{x}(k) + \Delta t f((k+1)\Delta t, \widehat{x}(k+1)) \tag{3.40}$$

$$\Downarrow$$

$$\widehat{x}(k+1) = (I - \Delta t f((k+1)\Delta t, \cdot))^{-1}(\widehat{x}(k)) \tag{3.41}$$

For linear f one can expand (3.41) in Δt as $\widehat{x}(k+1) \approx \widehat{x}(k) + \Delta t f((k+1)\Delta t, \widehat{x}(k)) + \Delta t^2[f((k+1)\Delta t, \cdot)^2](\widehat{x}(k))$. This iteration rule in fact is equivalent to PolyNet [ZLCLL17] (Poly-2) where it was observed that better performance can be obtained when using (3.41) in place of the usual ResNet. We note that in numerical analysis, backward Euler methods enjoy better stability than forward Euler method, and is particularly useful for stiff equations [SG79]. It is further remarked in [LZLD18] that other architectures also have similar interpretations as different discretization schemes. The authors then go on to propose another family of networks based on linear multi-step discretization, which is

$$\widehat{x}(k+1) = (1-\alpha_k)\widehat{x}(k) + \alpha_k \widehat{x}(k-1) + \Delta t f(k\Delta t, \widehat{x}(k)), \tag{3.42}$$

with the weight α_k as a trainable parameter. This effectively adds a second order time derivative term in tho the dynamics, which changes the dynamical properties of the evolution. Such effect can be quantified by deriving the modified equation [WH74]. Similar methods are also used to understand noise injection for residual networks, see [STD18]. We note that the well-known DenseNet [HLVDMW17] can be thought of as an extreme case of such a multi-step method. Multi-step methods are also used to construct reversible architectures, which have memory saving properties on top of stability guarantees [CMH+18a]. Runge-Kutta method inspired architectures are recently explored in [ZCF19], but we also remark that such approaches has been applied to learn dynamical data in very early works [RKK+92]. Other variants of differencing scheme inspired architectures include [XBMW19, YWL+19].

3.4.3 Architectures from PDE Theory

So far, we focused our discussion on architectures derived from ODE theory, either from enforcing stability conditions or from other forms of discretization, including implicit, symplectic and multi-step methods. We mention that there are also work that interpret feed-forward networks as certain reductions from PDEs. These come in two categories. First, for convolutional networks acting on images, there is a natural spatial dimension in addition to the temporal one, and hence this can be immediately connected with evolution equations in the form of PDEs (see e.g. [RH19]). On the other hand, there are also interpretations via the method of characteristics. In fact, we mentioned that in the mean-field formulation, the PMP equations (for which the forward equation is the feed-forward network) are indeed characteristics of PDEs on distribution spaces (Theorem 3.2). A related approach is to interpret the networks as

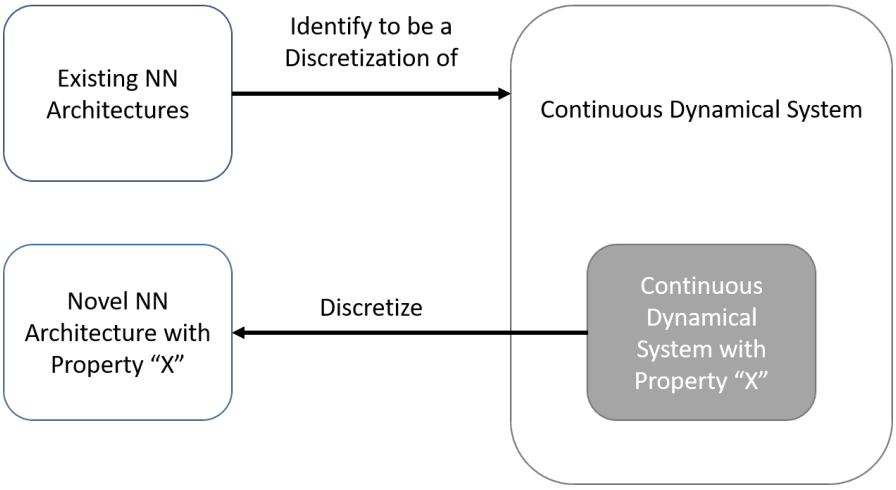


Figure 3.1: The overall approach of architectural design using dynamical systems viewpoint.

characteristics of a transport PDE [LS18], from which one can again derive new models and methods [WLL+18, WYSO19, WLZ+20].

3.4.4 Summary and Outlook

Let us now give a quick summary. In fact, the flavor of most of the aforementioned work on architectural design can be understood in Figure 3.4.4. There, the property "X" may be stability, robustness, reversibility or any other dynamical property. On the other hand the property "X" may also be not having certain limitations, e.g. not being a homeomorphism, which heuristically motivates embedding higher dimensions [DDT19]. While this approach has enjoyed many successes, the type of properties "X" can be further expanded in future work. For example, most of the properties that have been considered so far are properties that was studied mainly in the feed-forward step (i.e. stability, reversibility). In some sense, they are necessary for good performance but not sufficient. After all, the performance measure that one is interested in a (mean-field) control problem is the convergence rate of the iterative step to find an optimal control, as well as the generalization of that control to unknown data samples. These properties and their precise relation to discretization requires an extension of traditional numerical analysis techniques to combine with optimization/control theory in order to obtain a holistic understanding of the effect of discretization on optimal control problems.

3.5 Mathematical Results from the Dynamical Systems Approach

In this section, we discuss some mathematical results relevant to the dynamical systems approach to deep learning. Recall in Sec. 3.2 that we have introduced the basic optimality results that one can prove for the learning setting – namely the mean-field Pontryagin's maximum principle and the mean-field Hamilton-Jacobi-Bellman equation. These however are only one facet of the theory (see Fig. 1.1) and we are still far from a complete understanding of this continuous-time approach to learning. In particular, the grand goal of the theoretical development is to arrive at insights that currently hinders our understanding of deep learning itself. One key question whose comprehensive answer remain elusive is: why and when are deep networks better than shallow ones?

Let us now summarize some mathematical results on various aspects of the dynamical systems viewpoint.

3.5.1 Approximation Theory

One of the most basic results in the theory of neural networks is the *universal approximation theorem* (1.3). There, it is required that the width of the neural network go to infinity in order to approximate functions to arbitrary accuracy. This question can also be posed in the continuous-time dynamical systems setting. However, the key question one would like to answer now is whether universal approximation can be achieved by a sufficiently deep (instead of wide) neural network, corresponding to a dynamical system with large terminal time T. Let us formulate this problem more precisely.

Let us consider an controlled ODE

$$\dot{x}(t) = f(x(t), \theta(t)), \qquad \theta(t) \in \Theta \tag{3.43}$$

We can write this without explicit parameterization as

$$\dot{x}(t) = f_t(x(t)), \qquad f_t \in \mathcal{F} \tag{3.44}$$

where $\mathcal{F}$ is some family of control functions. Then, we may define the family of flow maps

$$\Phi(\mathcal{F}, T) := \{x \mapsto x(T) : \dot{x}(t) = f_t(x(t)), f_t \in \mathcal{F}, t \in [0, T], x(0) = x\}. \tag{3.45}$$

To match output dimensions, we will need a family $\mathcal{G}$ of functions (final classification/regression layers) from $\mathbb{R}^d$ (input space) to $\mathbb{R}^m$ (output space). Hence, the resulting hypothesis space induced by flow maps takes the form

$$\mathcal{H} = \cup_{T \geq 0} \{g \circ \varphi : g \in \mathcal{G}, \varphi \in \Phi(\mathcal{F}, T)\}. \tag{3.46}$$

The primary approximation question is thus: *What conditions on $\mathcal{F}$ and $\mathcal{G}$ is sufficient to guarantee the universal approximation property (UAP)*?

First, if $\mathcal{F}$ is itself a family of functions having the UAT, then it is not hard to show that then $\mathcal{H}$ also has this property [ZGUA20]. However, this result does not capture the effect of the dynamical flow in function approximation. In [LLS19], a general sufficient condition for universal approximation for $\mathcal{H}$ is given that allows for $\mathcal{F}$ to be simple, and many existing architectures are shown to satisfy the properties proved there. Furthermore, some approximation rates in terms of T can be analyzed in limited settings. This partially resolves the approximation problem, but a general understanding of how to efficiently approximate functions using dynamical flows remains an open problem in need of further investigations. We also mention [CLT19] where the authors study an alternative property of *universal interpolation*, i.e. mapping finite number of distinct points onto another finite set of target points. There it is proved that such a property holds for simple control families consisting of linear combinations of vector fields that satisfy some controllability conditions. Again, rate estimates are currently not established.

Recently, work on proving efficient approximations – say those that avoid the curse of dimensionality – have been investigated for continuous-time systems under the Barron space framework [EMW19, MW19b, MW19c, MW19a]. However, the control family $\mathcal{F}$ there is still very large and such results have so far not been generalized for simple $\mathcal{F}$, and in particular, the depth dependence in terms of approximation rates remain unclear. In fact, the control family there is a continuum limit in the width, in that one considers (ignoring bias for simplicity)

$$\dot{x}(t) = \sum_i v_i(t)\sigma(w_i(t)^\top x(t)) \qquad \to \qquad \dot{x}(t) = \int v\sigma(w^\top x(t))d\rho(t)(v,w), \tag{3.47}$$

with $\rho(t)$ being a probability measure. This has close relationship to the study of double continuum limits (in depth and in width) and its relationship to optimal transport and ridgelet transform, see [SM17, SM19], and also a kinetic theory formulation that relies on both limits [HTV20]. Lastly, a rigorous formulation of the optimal control and optimality conditions similar to the mean-field PMP/HJB above, but directly on this double continuum limit view of the dynamical process, can be found in [LM19].

3.5.2 Generalization

The problem of generalization is a very important aspect of learning algorithms. There are at least two types of questions one can ask:

- Does $|R_{\text{pop}}(F) - R_{\text{emp}}(F)| \to 0$ as $N \to 0$ uniformly in F, and at what rate?
- Does a minimizer of R_{emp} converge to that of R_{pop} as $N \to \infty$, and at what rate?

The first question is well-studied in classical statistical learning [FHT01], where generalization bounds come in the form of a guarantee of the difference between empirical risk and population/expected risk, e.g.

$$R_{\text{pop}}(F) \leq R_{\text{emp}}(F) + \frac{C(F)}{N^\alpha}. \tag{3.48}$$

Here, $C(F)$ is a complexity measure of the model. In the language of numerical analysis, this is a *a posteriori* estimate, in the sense that the estimate depends on the learned model F, instead of being dependent purely on the ground truth F^*.

Although such bounds are useful in classical settings, in deep learning it is found that they are mostly vacuous, because the F in this case are deep neural networks for which most traditional measures of complexity gives a very big estimate so that the right hand side of (3.48) is large. In fact, it is suggested that deep networks do generalize despite having the ability to over-fit to the training set if forced to [ZBH+16].

The second question has a more numerical analysis flavor, if one views R_{emp} as a "discretization" of R_{pop} (in fact, it *is* a discretization of the data distribution μ into an empirical measure). Indeed, in this viewpoint, question 1 asks if the discretization is consistent, and question 2 asks if it is convergent. Answers to 2 implies those to 1 under the assumption that the risk functions are well-behaved. Note that with respect to variational problems (of which an optimal control problem can be thought of as a special case), 2 is known as a Γ-convergence result.

There are relatively few results from the dynamical systems and the optimal control viewpoint on the generalization question. In [EHL19], a basic relationship between the solution of the empirical control problem and the population mean-field problem is established. In particular, it is proved that under a concavity assumption on the Hamiltonian, a solution of the PMP corresponding to the empirical control problem converges to that of the mean-field control problem. However, this does not imply the optimal solution converges to the population optimal solution in general.

More recently, [BCL19] proves the Γ-convergence of the sampled optimal control problem to the mean-field optimal control problem with a relaxed control set, i.e. the control is now a joint law of the control and the state parameters, instead of directly on the controls. In fact, a more general forward dynamics is considered in [BCL19] where explicit mean-field interactions is included that includes batch-normalization as a special case. As before, the rate of convergence, and in particular its dependence on problem dimension, remain largely unexplored at the time of writing of these notes.

3.5.3 Connection between Continuous and Discrete Time

Finally, we will discuss the connection of results between continuous time and discrete time. The viewpoint in these notes are primarily of the continuous-time nature, but in practice the problems we can solve on a computer are discrete. Hence, it is an important question to ask what the continuous-time results imply about discrete time. We will look at two angles:

- Approximation: in approximation theory, this relationship is precisely the connection between evolution equations and their discretizations. Here, the problem of convergence is well-studied, and under quite general conditions, if the continuous-time flow map has a density property, then that property will transfer to that of a discretized map (feed-forward NN) as long as the discretization is convergent. This has been discussed

in [LLS19]. However, we point out that if we are after convergence rates, then one must be more careful, in that one needs the continuous-time dynamics to satisfy certain regularity conditions before we can transfer a rate of approximation on T (the time horizon of the continuous-time evolution equation) and $K = T/\Delta t$ (the depth of the discretized network). In fact, if the continuous-time system is extremely ill-behaved in time, the fact that T is small does not imply K is small, since a smaller step size Δt may be required to achieve the same level of truncation error.

- Optimization: the optimization issue is more delicate as in the case of [BCL19]. In short, one not only have to prove that the temporal discretization is consistent, but one also have to show that the solution of the discretized version (optimal parameters of the neural network) converges to the solution of the continuous-time optimal control problem in the limit of infinitely many layers. This has been partially resolved in [TvGv18], where the authors prove a Γ-convergence result of the control problem for discretized dynamics (training ResNets) to the continuous-time control problem. However, one limitation there is that the proof required the introduction a regularizer on trainable parameters akin penalizing its temporal derivative. In other words, the convergence is established under a condition that the traininable parameters are regular in time, something that may not be satisfied by control problems in continuous-time. We also remark that in [AN20], a similar convergence result in the limit of infinite depth is presented for the case where the control do not vary in time.

3.5.4 Summary and Outlook

In this part we discussed some existing mathematical work on the dynamical systems approach to deep learning from the three usual paradigms, approximation, optimization, and generalization. We see many promising progress, yet there are many questions that await, making this an interesting area of research. Here, we discuss some problems in the theoretical aspect of the dynamical systems approach, most of which are also important open problems for deep learning at large, and it is our hope that the dynamical systems approach can provide useful insights into these problems.

- Approximation: The key issue is approximation rates, and in particular, answer the question of what types of functions can be *efficiently* approximated by a dynamical system. This requires one to introduce a sensible notion of complexity of a target function F^*, that is expected to be very different from usual ones in approximation theory, such as smoothness. In some sense, this is the most important obstacle to understanding why "deep" networks work so well in applications.
- Optimization: While various Γ-convergence results have been established, it remains to develop a useful theory to understand the dynamics of solving optimal control problems. For example, what is the relationship between the hardness of an optimal control problem with respect to the form of the dynamical systems or the choice of discretization schemes (i.e. architecture)? A useful first step is explored in [HKR19] in this direction.

- Generalization: The curse of dimensionality remains unresolved, especially when we do not take the continuum limit in width. In particular, what properties of the dynamical system are sufficient for good generalization? This requires further thought than necessary conditions such as stability, which often are generic and limits expressivity [HR17]. A useful theory of generalization and its interaction with dimension d and depth T (or K) is an important future goal of generalization theory for the current setting.

4 Summary

In these notes, we presented a pedagogical introduction of classical optimal control theory. Then, we discussed in a survey form various recent work on the development of the dynamical systems and optimal control approach to deep learning, including algorithm development, architecture selection and theoretical results. Some very interesting topics that the dynamical systems viewpoint have contributed to have not been discussed at length, e.g. stochastic aspects [TR19, TKS19, JB19, LWCD20], generative modelling [ZW18, GCB+18, BGC+19, CBDJ19, QGMK19, YHL19, ZYG+19, MPC+19, FJNO20], robustness studies [ZHW+19, HJVJ19, LXS+19, RW19], modelling training dynamics [LTE17, LTE19] and various applications to image processing and 3D vision [LLZS10, CYP15, ZLLD18, JLFZ19, HCZ20]. Thus, these notes are not meant to be a comprehensive survey of research on the intersection of machine learning and dynamical systems, and many recent and interesting work may also be missed due to unfamiliarity, ignorance or time-irreversibility. These will be updated in due time as these notes evolve and any email updates on latest research are always welcome.

Bibliography

[AF13] Michael Athans and Peter L Falb. *Optimal Control: An Introduction to the Theory and Its Applications*. Courier Corporation, 2013.

[AN20] Benny Avelin and Kaj Nyström. Neural ODEs as the Deep Limit of ResNets with constant weights. *arXiv:1906.12183 [cs, math, stat]*, January 2020.

[Arn12] Vladimir Igorevich Arnold. *Geometrical Methods in the Theory of Ordinary Differential Equations*, volume 250. Springer Science & Business Media, 2012.

[BCL19] Lijun Bo, Agostino Capponi, and Huafu Liao. Deep Residual Learning via Large Sample Mean-Field Optimization: Relaxed Control and Gamma-Convergence. *arXiv:1906.08894 [math]*, June 2019.

[BCN18] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization Methods for Large-Scale Machine Learning. *SIAM Review*, 2018.

[Bel66] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[BGC+19] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582, 2019.

[Bir27] George David Birkhoff. *Dynamical Systems*, volume 9. American Mathematical Soc., 1927.

[BO06] Christopher M Bishop and Others. *Pattern Recognition and Machine Learning*, volume 4. springer New York, 2006.

[BP07] Alberto Bressan and Benedetto Piccoli. *Introduction to the Mathematical Theory of Control*, volume 2. American institute of mathematical sciences Springfield, 2007.

[Car10] Pierre Cardaliaguet. Notes on mean field games. Technical report, Technical report, 2010.

[CBDJ19] Ricky TQ Chen, Jens Behrmann, David K. Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*, pages 9916–9926, 2019.

[CCHC19] Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.

[CL82] F L Chernousko and A A Lyubushin. Method of successive approximations for solution of optimal control problems. *Optimal Control Applications and Methods*, 3(2):101–114, 1982.

[CL83] Michael G. Crandall and Pierre-Louis Lions. Viscosity solutions of Hamilton-Jacobi equations. *Trans. Amer. Math. Soc.*, 277(1):1–42, 1983.

[CLT19] Christa Cuchiero, Martin Larsson, and Josef Teichmann. Deep neural networks, generic universal interpolation, and controlled ODEs. *arXiv:1908.07838 [math]*, August 2019.

[CMH+18a] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[CMH+18b] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level Residual Networks from Dynamical Systems View. *arXiv:1710.10348 [cs, stat]*, February 2018.

[Cod12] Earl A. Coddington. *An Introduction to Ordinary Differential Equations*. Courier Corporation, 2012.

[CRBD19] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. *arXiv:1806.07366 [cs, stat]*, December 2019.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[CYP15] Yunjin Chen, Wei Yu, and Thomas Pock. On learning optimized reaction diffusion processes for effective image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5261–5269, 2015.

[DBRDB78] Carl De Boor, J. R., and Carl De Boor. A practical guide to splines. In *Mathematics of Computation*, volume 27. springer-verlag New York, 1978.

[DDT19] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs. *arXiv:1904.01681 [cs, stat]*, October 2019.

[Don06] David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, pages 1–34, 2006.

[DP07] Alan J. Davies and M. J. D. Powell. Approximation Theory and Methods. In *The Mathematical Gazette*. 2007.

[E17] Weinan E. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[EHL19] Weinan E, Jiequn Han, and Qianxiao Li. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6(1):10, 2019.

[EMW19] Weinan E, Chao Ma, and Lei Wu. Barron Spaces and the Compositional Function Spaces for Neural Network Models. *arXiv preprint arXiv:1906.08039*, 2019.

[Eva98] L C Evans. *Partial Differential Equations*. American Mathematical Society, 1998.

[FFK+14] R. D. Falgout, S. Friedhoff, Tz. V. Kolev, S. P. MacLachlan, and J. B. Schroder. Parallel Time Integration with Multigrid. *SIAM J. Sci. Comput.*, 36(6):C635–C661, January 2014.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer series in statistics New York, 2001.

[FJNO20] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M. Oberman. How to train your neural ode. *arXiv preprint arXiv:2002.02798*, 2020.

[FS06] Wendell H. Fleming and Halil Mete Soner. *Controlled Markov Processes and Viscosity Solutions*, volume 25. Springer Science & Business Media, 2006.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016.

[GCB+18] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[GRS+20] Stefanie Günther, Lars Ruthotto, Jacob B. Schroder, Eric C. Cyr, and Nicolas R. Gauger. Layer-parallel training of deep residual neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):1–23, 2020.

[GS00] Izrail Moiseevitch Gelfand and Richard A. Silverman. *Calculus of Variations*. Courier Corporation, 2000.

[GVL96] G Golub and C Van Loan. Matrix computations, 3rd edn. J. *Hopkins, London*, 1996.

[HCZ20] Xingzhe He, Helen Lu Cao, and Bo Zhu. AdvectiveNet: An Eulerian-Lagrangian Fluidic reservoir for Point Cloud Processing. *arXiv preprint arXiv:2002.00118*, 2020.

[HJVJ19] Y. A. N. Hanshu, D. U. Jiawei, T. A. N. Vincent, and FENG Jiashi. On robustness of neural ordinary differential equations. In *International Conference on Learning Representations*, 2019.

[HKR19] Kaitong Hu, Anna Kazeykina, and Zhenjie Ren. Mean-field Langevin System, Optimal Control and Deep Neural Networks. *arXiv:1909.07278 [cs, math]*, October 2019.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[HR17] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):14004, 2017.

[HTV20] M. Herty, T. Trimborn, and G. Visconti. Kinetic Theory for Residual Neural Networks. *arXiv:2001.04294 [cs, math]*, January 2020.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.0, 2015.

[JB19] Junteng Jia and Austin R. Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 9847–9858, 2019.

[JLFZ19] Xixi Jia, Sanyang Liu, Xiangchu Feng, and Lei Zhang. FOCNet: A fractional optimal control network for image denoising. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6054–6063, 2019.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LCTE17] Qianxiao Li, Long Chen, Cheng Tai, and Weinan E. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research*, 18(1):5998–6026, 2017.

[LDP07] Michael Lustig, David Donoho, and John M. Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine*, 2007.

[LH18] Qianxiao Li and Shuji Hao. An optimal control approach to deep learning and applications to discrete-weight neural networks. *arXiv preprint arXiv:1803.01299*, 2018.

[Lib12] Daniel Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, Princeton ; Oxford, 2012.

[LLS19] Qianxiao Li, Ting Lin, and Zuowei Shen. Deep Learning via Dynamical Systems: An Approximation Perspective. *arXiv:1912.10382 [cs, math, stat]*, December 2019.

[LLZS10] Risheng Liu, Zhouchen Lin, Wei Zhang, and Zhixun Su. Learning PDEs for image restoration via optimal control. In *European Conference on Computer Vision*, pages 115–128. Springer, 2010.

[LM19] Hailiang Liu and Peter Markowich. Selection dynamics for deep neural networks. *arXiv preprint arXiv:1905.09076*, 2019.

[LS18] Zhen Li and Zuoqiang Shi. Deep Residual Learning and PDEs on Manifold. *arXiv:1708.05115 [cs, math]*, January 2018.

[LTE17] Qianxiao Li, Cheng Tai, and Weinan E. Stochastic modified equations and adaptive stochastic gradient algorithms. In *International Conference on Machine Learning*, 2017.

[LTE19] Qianxiao Li, Cheng Tai, and Weinan E. Stochastic Modified Equations and Dynamics of Stochastic Gradient Algorithms I: Mathematical Foundations. *Journal of Machine Learning Research*, 20(40):1–47, 2019.

[LWCD20] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *arXiv preprint arXiv:2001.01328*, 2020.

[LXS+19] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

[LZLD18] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3276–3285, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

[Mal09] Stephane Mallat. A Wavelet Tour of Signal Processing. In *A Wavelet Tour of Signal Processing*. 2009.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[MPC+19] Stefano Massaroli, Michael Poli, Federico Califano, Angela Faragasso, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Port–Hamiltonian Approach to Neural Network Training. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 6799–6806. IEEE, 2019.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT press, 2018.

[MW19a] Chao Ma and Lei Wu. Machine Learning from a Continuous Viewpoint. *arXiv preprint arXiv:1912.12777*, 2019.

[MW19b] Chao Ma and Lei Wu. On the Generalization Properties of Minimum-norm Solutions for Over-parameterized Neural Network Models. *arXiv preprint arXiv:1912.06987*, 2019.

[MW19c] Chao Ma and Lei Wu. A priori estimates of the population risk for two-layer neural networks. *Communications in Mathematical Sciences*, 17(5):1407–1425, 2019.

[Nes04] Yurii Nesterov. *Introductory Lectures on Convex Optimization*. 2004.

[PM19] Panos Parpas and Corey Muir. Predict globally, correct locally: Parallel-in-time optimal control of neural networks. *arXiv preprint arXiv:1902.02542*, 2019.

[QGMK19] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutnık. Accelerating neural odes with spectral elements. *arXiv preprint arXiv:1906.07038*, 2019.

[Rao10] Anil Rao. A Survey of Numerical Methods for Optimal Control. *Advances in the Astronautical Sciences*, 135, January 2010.

[RH19] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[RKK+92] R. Rico-Martínez, K. Krischer, I.G. Kevrekidis, M.C. Kube, and J.L. Hudson. DISCRETE- vs. CONTINUOUS-TIME NONLINEAR SIGNAL PROCESSING OF Cu ELECTRODISSOLUTION DATA. *Chemical Engineering Communications*, 118(1):25–48, November 1992.

[RW19] Viktor Reshniak and Clayton Webster. Robust learning with implicit residual networks. *arXiv preprint arXiv:1905.10479*, 2019.

[SG79] Lawrence F. Shampine and Charles William Gear. A user's view of solving stiff ordinary differential equations. *SIAM review*, 21(1):1–17, 1979.

[SM17] Sho Sonoda and Noboru Murata. Double continuum limit of deep neural networks. In *ICML Workshop Principled Approaches to Deep Learning*, 2017.

[SM19] Sho Sonoda and Noboru Murata. Transport analysis of infinitely deep neural network. *The Journal of Machine Learning Research*, 20(1):31–82, 2019.

[STD18] Qi Sun, Yunzhe Tao, and Qiang Du. Stochastic Training of Residual Networks: A Differential Equation Viewpoint. *arXiv:1812.00174 [cs, stat]*, December 2018.

[Szn91] Alain-Sol Sznitman. *Topics in Propagation of Chaos*, volume 1464, pages 165–251. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[TKS19] Niall Twomey, Micha\l Koz\lowski, and Raúl Santos-Rodríguez. Neural ODEs with stochastic vector field mixtures. *arXiv preprint arXiv:1905.09905*, 2019.

[TR19] Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

[TvGv18] Matthew Thorpe, Yves van Gennip, Yves Van Gennip, and Yves van Gennip. Deep limits of residual neural networks. *arXiv preprint arXiv:1810.11741*, 2018.

[VWB16] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[WH60] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[WH74] R. F Warming and B. J Hyett. The modified equation approach to the stability and accuracy analysis of finite-difference methods. *Journal of Computational Physics*, 14(2):159–179, February 1974.

[WLL+18] Bao Wang, Xiyang Luo, Zhen Li, Wei Zhu, Zuoqiang Shi, and Stanley J. Osher. Deep Neural Nets with Interpolating Function as Output Activation. *arXiv:1802.00168 [cs, stat]*, June 2018.

[WLZ+20] Bao Wang, Alex T. Lin, Wei Zhu, Penghang Yin, Andrea L. Bertozzi, and Stanley J. Osher. Adversarial Defense via Data Dependent Activation Function and Total Variation Minimization. *arXiv:1809.08516 [cs, math, stat]*, April 2020.

[WYSO19] Bao Wang, Binjie Yuan, Zuoqiang Shi, and Stanley J. Osher. ResNets Ensemble via the Feynman-Kac Formalism to Improve Natural and Robust Accuracies. *arXiv:1811.10745 [cs, math, stat]*, June 2019.

[XBMW19] Xuping Xie, Feng Bao, Thomas Maier, and Clayton Webster. Analytic Continuation of Noisy Data Using Adams Bashforth ResNet. *arXiv:1905.10430 [physics]*, May 2019.

[YHL19] Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. ODE2VAE: Deep generative second order ODEs with Bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 13412–13421, 2019.

[YLBS20] Zonghan Yang, Yang Liu, Chenglong Bao, and Zuoqiang Shi. Interpolation between Residual and Non-Residual Networks. *arXiv:2006.05749 [cs, stat]*, June 2020.

[YWL+19] Yibo Yang, Jianlong Wu, Hongyang Li, Xia Li, Tiancheng Shen, and Zhouchen Lin. Dynamical System Inspired Adaptive Time Stepping Controller for Residual Network Families. *arXiv:1911.10305 [cs, stat]*, November 2019.

[ZBH+16] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[ZCF19] Mai Zhu, Bo Chang, and Chong Fu. Convolutional Neural Networks combined with Runge-Kutta Methods. *arXiv:1802.08831 [cs]*, January 2019.

[ZGUA20] Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation Capabilities of Neural ODEs and Invertible Residual Networks. *arXiv:1907.12998 [cs, stat]*, February 2020.

[ZHW+19] Jingfeng Zhang, Bo Han, Laura Wynter, Kian Hsiang Low, and Mohan Kankanhalli. Towards robust resnet: A small step but a giant leap. *arXiv preprint arXiv:1902.10887*, 2019.

[ZLCLL17] Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. Polynet: A pursuit of structural diversity in very deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 718–726, 2017.

[ZLLD18] Xiaoshuai Zhang, Yiping Lu, Jiaying Liu, and Bin Dong. Dynamically unfolding recurrent restorer: A moving endpoint control method for image restoration. *arXiv preprint arXiv:1805.07709*, 2018.

[ZS19] Linan Zhang and Hayden Schaeffer. Forward Stability of ResNet and Its Variants. *Journal of Mathematical Imaging and Vision*, pages 1–24, 2019.

[ZW18] Linfeng Zhang and Lei Wang. Monge-amp$\backslash$ere flow for generative modeling. *arXiv preprint arXiv:1809.10188*, 2018.

[ZYG+19] Tianjun Zhang, Zhewei Yao, Amir Gholami, Kurt Keutzer, Joseph Gonzalez, George Biros, and Michael Mahoney. Anodev2: A coupled neural ode evolution framework. *arXiv preprint arXiv:1906.04596*, 2019.

[ZZL+19] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You Only Propagate Once: Accelerating Adversarial Training via Maximal Principle. *arXiv preprint arXiv:1905.00877*, 2019.