Differential Equations for Machine Learning

Çağatay Yıldız
Differential Equations for Machine Learning

Çağatay Yıldız

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, https://aalto.zoom.us/j/61873808631, on 18 February 2022 at 12:15.

Aalto University
School of Science
Department of Computer Science
Computational Systems Biology
Supervising professor
Professor Harri Lähdesmäki, Aalto University, Finland

Thesis advisors
Professor Harri Lähdesmäki, Aalto University, Finland
Doctor Markus Heinonen, Aalto University, Finland

Preliminary examiners
Professor Arto Klami, University of Helsinki, Finland
Professor Niklas Wahlström, Uppsala University, Sweden

Opponent
Professor Carl Henrik Ek, University of Cambridge, United Kingdom

Aalto University publication series
DOCTORAL THESES 7/2022

© 2022 Çağatay Yıldız

ISBN 978-952-64-0665-7 (printed)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Unigrafia Oy
Helsinki 2022

Finland
Mechanistic models express novel hypotheses for an observed phenomenon by constructing mathematical formulations of causal mechanisms. As opposed to this modeling paradigm, machine learning approaches learn input-output mappings by complicated and often non-interpretable models. While requiring large chunks of data for successful training and downstream performance, the resulting models can come with universal approximation guarantees. Historically, differential equations (DEs) developed in physics, economics, engineering, and numerous other fields have relied on the principles of mechanistic modeling. Despite providing causality and interpretability that machine learning approaches usually lack, mechanistic differential equation models tend to carry oversimplified assumptions. In this dissertation, we aim to bring these two worlds together by demonstrating how machine learning problems can be tackled by means of differential equations, and how differential equation models can benefit from modern machine learning tools.

First, we examine the problems in which mechanistic modeling becomes too difficult, which include the cases with partial knowledge about the observed system and with an excessive number of interactions. Such limitations complicate the process of constructing mathematical descriptions of the phenomenon of interest. To bypass this, we propose to place Gaussian process priors to the time differential and diffusion functions of unknown ordinary (ODEs) and stochastic differential equations (SDEs), and approximate the resulting intractable posterior distribution. We demonstrate that the model can estimate unknown dynamics from sparse and noisy observations.

Motivated by the fact that our proposed approach is unable to learn sequences obtained by transforming the ODE states, we develop a new technique that can simultaneously embed the observations into a latent space, and learn an ODE system in the embedding space. Our new model infers the dynamics using Bayesian neural networks for uncertainty handling and more expressive power. We furthermore explicitly decompose the latent space into momentum and position components, which leads to increased predictive performance on a variety of physical tasks.

Our next task concerns another problem involving DEs, namely, non-convex optimization. By carefully crafting the drift and diffusion functions of an SDE, we first obtain a stochastic gradient MCMC algorithm. Tuning a temperature variable in the proposed algorithm allows the chain to converge to the global minimum of a non-convex loss surface. We significantly speed up the convergence by using second-order Hessian information in an asynchronous parallel framework.

Lastly, we explore how reinforcement learning problems can benefit from neural network based ODE models. In particular, we propose to learn dynamical systems controlled by external actions by a novel, uncertainty-aware neural ODE model. The inferred model, in turn, is utilized for learning optimal policy functions. We illustrate that our method is robust to both noisy and irregularly sampled data sequences, which poses major challenges to traditional methods.

Keywords Machine learning, differential equations, neural networks, Gaussian processes
Preface

This journey started when Taha asked me about applying to Aalto University, a place that I had barely heard about. The second northernmost capital in the world did not seem like the perfect destination at first but that quickly changed after contacting Harri Lähdesmäki. Looking back four years later, I could not have asked for a better supervisor. Your guidance and support have helped me (who knows how many times) find my way through seemingly impossible-to-escape-from situations. I will truly miss those eureka moments I had during our discussions. Thank you for showing me research is enjoyable and letting me develop myself as an independent researcher.

Next, I would like to thank Markus Heinonen for his immense support throughout my journey. Our day-to-day, sometimes hours-long conversations helped me leap over countless obstacles. I have learned so much from you, not just the theory but also how to ask questions and conduct research. I am grateful particularly for your patience in my early times, answering all my questions regardless of how silly they are.

I would also like to express my gratitude to many colleagues, starting with my coauthors Henrik Mannerström and Jukka Intosalmi. My first two publications would not be possible without your ideas and contributions, particularly the sensitivity equations. Thanks Pashupati Hegde for our ODE discussions and code exchanges. Charles Gadd, our regular RL whinings assisted me to gain an understanding of the whole discipline and kept me going; thanks for sharing those moments with me. It has really been a privilege to brainstorm with excellent researchers at Aalto; in particular, Alexander Ilin, Arno Solin, and Ville Kyrki. Thank you Umut Şimşekli for your supervision and lead in our asynchronous L-BFGS project; you showed me how rewarding such difficult projects can be. Thomas Schön, I am very thankful to have visited your group through the end of my doctoral studies. Finally, I would like to thank the pre-examiners Arto Klami and Niklas Wahlström for their valuable feedback.

In my years at the Computational Systems Biology research group, I have been lucky enough to meet many great colleagues. Juho Timonen,
Preface

Maria Osmala, Siddharth Ramchandran, Sini Rautio, Emmi Jokinen, Viivi Halla-aho, Juhi Somani, Lu Cheng, Valerii Iakovlev, and Gleb Tikhonov; thanks for creating this inspiring and supportive work environment, fun conference trips, lunches, and after-work beers. Thank you Betül Güvenç, Tolou Shadbahr, Yogesh Kumar, Santosh Hiremath, Kunal Ghosh, Eero Siivola, Pradeep Eranti, Tianyu Cui, Topi Paananen, Reza Ashrafi and other fellow PhD students not mentioned here for occasional, stimulating chats. Aija Kukkala, Noora Suominen de Rios, Richard Darst, Jaakko Kotimäki, Simo Tuomisto and other members of the administrative and Science-IT team, I am thankful for your support with all my daily issues.

Mine Öğretir and Onur Poyraz; my life in Finland and PhD experience changed for the better right upon your arrival. I count myself very fortunate to have your company, especially through this spring. Çağlar Hızlı and Ashutosh Vaishnav; getting you on board was also lovely, I will surely miss our suddenly but comfortably deep conversations. Daniel Monsivais-Velazquez; having you as my first flatmate and friend here was gratifying; I will definitely keep alive the memory of your awkwardly funny jokes. My dear friends Taha Ceritli, Berkant Kepez, Emre Erdoğan, Barış Kurt, Beyza Ermiş, Şebnem Gelmedi, and also the other member of the “Börekçi” team, Muhammed Tel, Tuna Kırabah, Orkun Uçkunlar, Ahmet Duran Arslan, and Ümit Yapıcı; those little get-together breaks, endless remote talks and football chitchats animated me a zillion times. Thank you all for adding so much color to my life.

Finally, I wholeheartedly thank my family for their never-ending encouragement, unconditional love and support in my endeavour to become a scientist, for which I will always be indebted. Dear Tuğçe Çetin; meeting you has completely turned my life around. Thank you for constant presence, providing me with the safest possible environment to explore my own map and open new doors no matter how scary they are.

Helsinki, December 27, 2021,

Çağatay Yıldız
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>i</td>
</tr>
<tr>
<td>Contents</td>
<td>iii</td>
</tr>
<tr>
<td>List of Publications</td>
<td>vii</td>
</tr>
<tr>
<td>Author's Contribution</td>
<td>ix</td>
</tr>
<tr>
<td>Abbreviations</td>
<td>xi</td>
</tr>
<tr>
<td><strong>1. Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Contributions</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Structure of the Dissertation</td>
<td>4</td>
</tr>
<tr>
<td><strong>2. Theoretical Foundations</strong></td>
<td>5</td>
</tr>
<tr>
<td>2.1 Function Approximation</td>
<td>6</td>
</tr>
<tr>
<td>2.1.1 Neural Networks</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2 Gaussian Processes</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Differential Equations</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 Ordinary Differential Equations</td>
<td>9</td>
</tr>
<tr>
<td>2.2.2 Stochastic Differential Equations</td>
<td>10</td>
</tr>
<tr>
<td>2.2.3 Numerical Solvers</td>
<td>12</td>
</tr>
<tr>
<td>2.2.4 Gradient Computation</td>
<td>15</td>
</tr>
<tr>
<td>2.3 A Primer on Parameter Inference</td>
<td>17</td>
</tr>
<tr>
<td>2.3.1 Variational Inference</td>
<td>18</td>
</tr>
<tr>
<td>2.3.2 Stochastic Gradient Markov Chain Monte Carlo</td>
<td>21</td>
</tr>
<tr>
<td>2.3.3 Numerical Optimization</td>
<td>23</td>
</tr>
<tr>
<td><strong>3. Gaussian Processes for Learning Unknown Differential Equations</strong></td>
<td>25</td>
</tr>
<tr>
<td>3.1 Auto-Regressive Dynamics Learning</td>
<td>26</td>
</tr>
<tr>
<td>3.1.1 Gradient Matching</td>
<td>26</td>
</tr>
<tr>
<td>3.1.2 Related Work</td>
<td>28</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.2</td>
<td>Learning ODEs</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Time Differential Function</td>
</tr>
<tr>
<td>3.2.2</td>
<td>MAP Inference for npODE</td>
</tr>
<tr>
<td>3.3</td>
<td>Learning SDEs</td>
</tr>
<tr>
<td>3.4</td>
<td>Experiments with npODE and npSDE</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Simulated ODE Systems</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Simulated SDE System</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Motion Capture Experiments</td>
</tr>
<tr>
<td>4.</td>
<td>Continuous-Time Bayesian Modelling for High Dimensional Sequences</td>
</tr>
<tr>
<td>4.1</td>
<td>Limitations of npODE</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Modeling Space</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Learning Second-Order Systems</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Scalability of the Dynamics</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Overfitting</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Scalability of the Optimization</td>
</tr>
<tr>
<td>4.2</td>
<td>Proposed Model</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Approximate Posterior</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Evidence Lower Bound</td>
</tr>
<tr>
<td>4.3</td>
<td>Experiments</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Robustness against Overfitting</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Generalization to Unseen Data Points</td>
</tr>
<tr>
<td>4.3.3</td>
<td>The Virtue of Second-Order Modeling</td>
</tr>
<tr>
<td>5.</td>
<td>SDEs for Stochastic Asynchronous Non-convex Optimization</td>
</tr>
<tr>
<td>5.1</td>
<td>Background</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Stochastic L-BFGS</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Tempered SGLD</td>
</tr>
<tr>
<td>5.2</td>
<td>Asynchronous Stochastic L-BFGS</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Designing the Correct SDE</td>
</tr>
<tr>
<td>5.2.2</td>
<td>A Practical Distributed Algorithm</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Main Theoretical Result</td>
</tr>
<tr>
<td>5.3</td>
<td>Large-scale Matrix Factorization Experiment</td>
</tr>
<tr>
<td>6.</td>
<td>Continuous-Time Model-Based Reinforcement Learning</td>
</tr>
<tr>
<td>6.1</td>
<td>Problem Formulation</td>
</tr>
<tr>
<td>6.2</td>
<td>Dynamics Learning</td>
</tr>
<tr>
<td>6.3</td>
<td>Continuous-time Actor-Critic</td>
</tr>
<tr>
<td>6.3.1</td>
<td>State-Value Approximation (Critic)</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Estimating the Actor and Critic</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Connection to Temporal Difference</td>
</tr>
<tr>
<td>6.4</td>
<td>Previous Work</td>
</tr>
<tr>
<td>6.5</td>
<td>A New CTRL Evaluation Environment</td>
</tr>
</tbody>
</table>
This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.


Author’s Contribution

Publication I: “Learning Unknown ODE Models with Gaussian Processes”

The initial idea of Gaussian process based ODE modeling was proposed by Markus Heinonen and Harri Lähdesmäki. The final version of the model was decided by discussions including all authors. Jukka Intosalmi suggested using sensitivity equations and provided an example code. I improved the toy model implemented by Markus Heinonen, and unified it with sensitivity equations. I performed all the experiments, which were designed by Markus and myself. The article was written by all authors.

Publication II: “Learning Stochastic Differential Equations with Gaussian Processes without Gradient Matching”

The idea of extending our previous method to SDEs was discussed among all authors. Markus Heinonen and I worked on MATLAB and TensorFlow implementations in parallel. I designed and executed the experiments. Markus Heinonen, Harri Lähdesmäki and I wrote the publication while other coauthors helped us with proofreading and corrections.

Publication III: “ODE³VAE: Deep Generative Second Order ODEs with Bayesian Neural Networks”

The original idea was proposed by Markus Heinonen and Harri Lähdesmäki. I performed the literature survey, derived the theoretical aspects, implemented the method, designed and executed the experiments. All authors jointly wrote the article.

Umut Şimşekli came up with the idea of building an asynchronous L-BFGS method. Umut Şimşekli and I discussed the underlying mathematical construction and experiments. I developed the main asynchronous algorithm while Umut Şimşekli held the responsibility of writing the article and proving our theorems.

Publication V: “Continuous-Time Model-Based Reinforcement Learning”

Markus Heinonen and Harri Lähdesmäki proposed the idea of building a continuous-time reinforcement learning method. In conjunction with our discussions, I developed new variants of the standard continuous-time approach and derived a novel policy learning method. I implemented the method and executed all the experiments. The article was written jointly.
Abbreviations

ACA  adaptive checkpoint algorithm
AR  auto-regressive models
a-SGD  asynchronous SGD
BFGS  Broyden–Fletcher–Goldfarb–Shanno
BNN  Bayesian neural network
CNN  convolutional neural network
CT  continuous-time
CTRL  continuous-time reinforcement learning
DT  discrete-time
EM  Euler-Maruyama
GP  Gaussian process
HAMCMC  Hessian approximated MCMC
HMC  Hamiltonian Monte Carlo
iid  independently and identically distributed
KL  Kullback-Leibler
L-BFGS  limited memory BFGS
MAP  maximum a posteriori
mb-L-BFGS  multi-batch LBFGS
MBRL  model-based reinforcement learning
MCMC  Markov chain Monte Carlo
Abbreviations

ML  maximum likelihood
MLP  multi-layer perceptron
MSE  mean squared error
NN  neural network
NODE  neural ODE
NPETS  non-linear PETS
npODE  non-parametric ODE
npSDE  non-parametric SDE
ODE  ordinary differential equation
PCA  principle component analysis
PETS  probabilistic ensembles with trajectory sampling
PILCO  probabilistic inference for learning control
RBF  radial basis function
RL  reinforcement learning
RMSE  root mean squared error
RNN  recurrent neural network
SDE  stochastic differential equation
SGD  stochastic gradient descent
SGLD  stochastic gradient Langevin dynamics
SGMCMC  stochastic gradient MCMC
SGP  sparse Gaussian process
VAE  variational auto-encoder

xii
1. Introduction

Time series forecasting has been one of the core problems in statistics and machine learning. In the most basic form, the goal is to make predictions based on historical measurements, which typically requires learning how the measured variables evolve over time (Barber et al., 2011; Durbin and Koopman, 2012; Särkkä, 2013). For decades, researchers have addressed this problem by discrete dynamical systems, where the variable of interest jumps from one value to another as the system moves from “one point in time” to the other. Perhaps the most well-known example of such discrete dynamical systems is the Kalman filter (Kalman, 1960), which has been successfully applied in a wide range of fields (Barber et al., 2011). Nevertheless, variables in certain problems such as radioactive decay, the flow of current in a conductor and the growth of a population necessitate treating time as a continuum, meaning that the variables can only have a particular value for only an infinitesimally short amount of time. Such problems are broadly referred to as continuous-time problems.

Since discrete dynamical systems are rather straightforward to grasp and implement, they have been often applied to continuous-time problems (Barber et al., 2011; Murphy, 2013). This is also nicely complemented by the fact that continuous signals in modern day computers are almost always represented in a discrete manner, e.g., by a finite collection of samples. However, as demonstrated later in this dissertation, the success of discrete systems on truly continuous-time problems is restricted by factors such as high sampling frequency and low measurement noise. This natural mismatch can be addressed by turning to differential equations, which can be defined as equations that involve functions and their derivatives (Tenenbaum, 1985). Dating back to the discovery of calculus by Leibniz and Newton, differential equations were first used to explain the laws of motion and have played a vital role in many physical theories ever since (see Robinson (2004) for a discussion).

Ordinary differential equations (ODEs) form a special subgroup of differential equations in which the unknown function depends on a single independent variable. When an ODE involves a stochastic process, the
resulting system is referred to as *stochastic* differential equation (SDE). We focus our attention on ODEs and SDEs, without loss of generality, with the independent variable being *time*; hence, differential equation systems of our interest describe the time evolution of dependent variables. In essence, our goal is to build mathematical models for the functions appearing in such differential equation systems. Accurate models can then be used for predicting the system’s future behavior.

This problem has been studied for decades in the contexts of dynamical systems learning and system identification. Conventional ODE approaches relying on *white-box* modeling assume that a parametric model for the process of interest is given (Ljung, 2010). Consequently, the ODE inference problem reduces to parameter estimation, which can be tackled using optimization and statistical inference tools (Butcher and Goodwin, 2008). We refer to the methods following this traditional ODE modeling approach as “parametric (ODE) models”.

Successful applications of parametric ODE models necessitate accurate functional forms describing the system dynamics. Nonetheless, researchers often have partial knowledge about the dynamics which leads to an incomplete representation of observed phenomena. Similarly, formulating complex dynamics directly from data that possibly consist of hundreds of observations, is a tedious task, and parametric models may only provide too simplistic explanations (Äijö and Lähdesmäki, 2009). Therefore, accurate descriptions of the real-world system dynamics could easily become an impractical task, if not impossible.

This dissertation concerns an alternative approach to conventional techniques, namely, *non-parametric*, or *black-box*, ODE modeling. Leveraging the advances in function estimators, we propose to infer unknown system dynamics with Gaussian processes (GP) or neural networks (NN). Thanks to the universal function approximation theorems (Cybenko, 1989; Hammer and Gersmann, 2003), the resulting black-box ODE systems would theoretically be able to learn unknown differential equations from arbitrary data sequences. We experimentally demonstrate that the proposed models can indeed learn a variety of dynamical systems, even from image sequences or with external control signals.

### 1.1 Contributions

Before explaining the proposed approaches in detail, we give a general overview of the research questions this dissertation addresses and the solutions devised. Please see Figure 1.1 for a schematic illustration of how the developed methods are related.

- Publication I presents npODE, our first attempt to model unknown ODE systems in a black-box manner. Due to their inherent ability
Figure 1.1. A diagram of the methods presented in this dissertation. We first develop a black-box ODE model based on Gaussian processes, which is later extended to an SDE model. Next we investigate how SDE formulations can be used for large-scale optimization (Publication IV). Publication III describes a technique to learn ODEs from video sequences. Finally, Publication V applies black-box ODE modelling approach to reinforcement learning problems.

To learn smooth function, we opt for inducing point approximations of GPs to approximate the unknown time differentials. To efficiently compute the gradients of the system, we propose to use another set of differential equations, named sensitivity equations. We demonstrated the model’s ability to learn arbitrary systems on three different parametric ODE systems without seeing the functional forms as well as the dynamics of real walking sequences.

- The above technique is adapted to SDEs in Publication II (hence called $npSDE$). Our methodology is based on simulating path distributions by forward integration to match observed sequences. We also propose a practical, sampling-based likelihood model in connection with kernel density estimation. Experimental evaluations demonstrate that arbitrary drift functions as well as state-dependent and constant diffusion functions can be accurately estimated.

- Kernel methods, including GPs, tend to suffer from curse of dimensionality (Evangelista et al., 2006), which limits the applicability of our previous GP-based ODE and SDE approaches to small and medium dimensional problems. Motivated by the fact that NNs excel at high-dimensional tasks such as high-resolution image generation and classification (LeCun et al., 2015), we propose to approximate unknown differentials with Bayesian neural networks (BNN) to ac-
count for scalability as well as uncertainty. Furthermore, we describe how to embed high dimensional trajectories, such as video sequences, into a low dimensional latent space in which ODE system is defined. We also explicitly decompose the latent space into momentum and position components, and name our method $ODE^2VAE$ (Publication III).

- In Publication IV, we shift our attention from unknown differential equations to numerical optimization. Our technique, named as-LBFGS, relies on the numerical simulation of a specific type of SDE known as Langevin dynamics. We first show that the dynamics, when coupled with an additional temperature parameter, can be used to reach the global optimum of a target function. Furthermore, we propose to include local geometric information through an approximation to the Hessian matrix and show how to perform the optimization procedure in an asynchronous manner. The experiments show that our algorithm provides a significant speedup over existing synchronous distributed algorithms.

- The majority of physical systems and control tasks are described in form of differential equations; however, existing reinforcement techniques are inherently designed for discrete setups. We address this mismatch in Publication V by introducing $ODE-RL$, a continuous-time reinforcement learning (RL) algorithm. Similar to Publication III, we propose a BNN-based ODE model that involve control signals, and also develop a novel continuous-time actor-critic method to learn optimal policies. We demonstrate that our method is robust to both noisy and irregularly sampled data sequences, which poses major challenges to traditional discrete-time methods.

1.2 Structure of the Dissertation

The next chapter outlines the technical background needed to understand the remaining of the dissertation. Chapter 3 presents our GP-based ODE and SDE methods (Publication I-II). $ODE^2VAE$ is described in the following chapter. We then explain as-LBFGS in Chapter 5 and Chapter 6 is devoted to $ODE-RL$. In each chapter, we give a summary of the proposed methods and experimental findings. Hence, readers can study each chapter separately after related technical background is surveyed. The dissertation is concluded with a discussion of the presented techniques and future directions.
In this chapter, we briefly explain the concepts on which our proposed models are built:

- The first section gives an overview of function approximation in machine learning. In particular, we discuss Gaussian processes and different types of neural networks.

- Our Publication I and Publication II entail background knowledge on ordinary and stochastic differential equations, both of which are formally defined in Section 2.2. Afterwards, two rather practical topics are examined: numerical approximations to compute state solutions and gradient computation in differential equation systems. These practical aspects are often overlooked by researchers; however, they are crucial to understand how differential equation modeling differs from standard machine learning tasks such as regression or classification.

- The last section briefly covers parameter learning methods in machine learning in general. After an introduction to point estimation, we touch upon variational inference as it is employed in Publication III and Publication V. This subsection is followed by stochastic gradient Markov chain Monte Carlo techniques on which our Publication IV is built. We conclude by explaining gradient-descent based numerical optimization, including second-order techniques.

Since the above-mentioned topics have been studied for decades, the rest of this chapter focuses only on what is relevant to our publications. Please note that an in-depth overview of reinforcement learning (RL) is presented in the respective chapter.
2.1 Function Approximation

This section reviews two commonly used function approximators in machine learning: neural networks and Gaussian processes. Here, we introduce only the basic concepts and advances that have enabled our proposed models. A thorough investigation of neural networks can be found in Goodfellow et al. (2016) and please see Schmidhuber (2015) for a timeline of the advances in deep learning. We refer the readers to Rasmussen and Williams (2006) to find out more about Gaussian processes and Liu et al. (2020) for a summary of recent and more scalable approaches.

2.1.1 Neural Networks

A standard neural network (NN) consists of simple processing units called neurons, which are connected to each other in a structured way. The application at hand often dictates internal organization of neurons. Regardless of the architecture, the network parameters are typically inferred by backpropagation, a technique to compute the gradient of a loss function with respect to each parameter by chain rule (Linnainmaa, 1970; Rumelhart et al., 1986). Thanks to their auto-differentiation feature, modern machine learning frameworks such as PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2016) require only the forward model to perform backpropagation. In the following, we describe specific types of NNs that are relevant to our publications.

Multi-Layer Perceptron (MLP)

Multi-layer perceptron, also known as feedforward neural network or fully connected layers, is a class of NNs comprised of an input layer, an output layer and at least one hidden layer (McCulloch and Pitts, 1943; Rosenblatt, 1961). Each layer $l$ transforms an input vector $x_l$ into an output vector $x_{l+1}$ by performing affine matrix operations and by applying a possibly nonlinear function to the output:

$$x_{l+1} = f^{(l)}(x_l) = \sigma_l(W_l x_l + b_l).$$

Here, $W_l$, $b_l$, and $\sigma_l$ denote the weight, bias, and activation at layer $l$. Note that often the collection $(W_l, b_l)$ is called “weights”. Finally, a simple MLP performs a chain of operations $f^{(n)}(f^{(n-1)}(\ldots f^{(0)}(x)))$, where $x$ is the original input.

Convolutional Neural Network (CNN)

Convolutional neural networks are another class of NNs designed particularly for extracting patterns in a hierarchical manner (LeCun et al., 1989). Inspired by the neurons in human visual system, CNN layers perform the “convolution operation”. In the context of image processing, convolutions
operate on 3D input tensors, where two indices are reserved for the spatial coordinates and one index for the channel. Typically, each layer of a CNN stores 4D kernel parameters \( K \), where \( K_{ijmn} \) gives the connection strength between a unit in the \( n \)'th channel of the output and a unit in the \( m \)'th channel of the input, and \( i \) and \( j \) denotes the row and column offsets. Given a 3D input \( X^{(l)} \), which could be the original input image or the output of the previous layer, each layer performs the following convolution operation:

\[
X^{(l+1)}_{p,r,n} = \sum_{i,j,m} X^{(l)}_{p+i,r+j,m} K_{ijmn}.
\]

Similar to MLPs, the convolutional layers are very conveniently stacked to extract features with an increasing complexity. Computer vision applications typically enrich the above operation by padding the input image and by shifting the filter by different amounts, called stride. Also, nonlinear activation functions applied at layer outputs and pooling layers to downsample feature maps are known to boost model performance. We refer the reader to Goodfellow et al. (2016) for more details and recent advances in CNNs.

**Bayesian Neural Network (BNN)**

Despite the universal function approximation guarantees, standard NNs suffer from a number of limitations. First, since deeper and wider NNs with more parameters almost always achieve better fits to the training data, blindly minimizing an error function with respect to model parameters prevents us from determining the appropriate level of complexity. To tackle this problem, researchers typically resort to ad-hoc regularization terms (Hanson and Pratt, 1988; Krogh and Hertz, 1992), terminate the optimization procedure prematurely (Prechelt, 1998) or use validation sets. Second, computing point estimates does not allow us to quantify the uncertainty on the unknown parameters, which makes risk assessment and decision making difficult in safety-critical applications such as autonomous driving.

Bayesian modeling provides an alternative and automated way of handling model uncertainty by penalizing complex models and by maintaining posterior beliefs on the parameters. The pioneering work by MacKay (1992) describes the first Bayesian treatment of neural networks by placing a Gaussian prior distribution on the weights. Later, Neal (1994) specifies different types of priors and also infers NN parameters by hybrid Monte Carlo (Duane et al., 1987; Neal et al., 2011). We revisit the parameter learning and Bayesian perspective in Subsection 2.3.1.

### 2.1.2 Gaussian Processes

Gaussian processes (GPs) are stochastic processes such that any finite set of its random variables follows a multivariate Gaussian distribution.
(Rasmussen and Williams, 2006). In machine learning, GPs are typically used to define priors on the functions, denoted by

$$f(x) \sim \mathcal{GP}(\mu(x), k(x,x')),$$

where $f : \mathbb{R}^d \rightarrow \mathbb{R}$ maps $d$-dimensional inputs into one-dimensional outputs (Please see, e.g., (Boyle and Frean, 2005; Williams et al., 2007) for the generalization of this framework to multi-output functions). A GP is fully specified in terms of its mean

$$\mathbb{E}[f(x)] = \mu(x)$$

and covariance function

$$\text{cov}[f(x), f(x')] = k(x, x'),$$

where $x$ and $x'$ denote input points. In this dissertation, we abide by the very typical zero mean assumption: $\mu(x) = 0$. The covariance between two function outputs is determined solely by the kernel function $k(\cdot, \cdot)$, which simply gives a measure of similarity between two inputs.

For any set of input points $X = [x_1, x_2, \ldots, x_N]^T \in \mathbb{R}^{N \times d}$, a GP prior defines a multivariate Gaussian distribution over function outputs $F = [f(x_1), f(x_2), \ldots, f(x_N)]^T \in \mathbb{R}^N:

$$
\begin{bmatrix}
  f(x_1) \\
  \vdots \\
  f(x_N)
\end{bmatrix}
\sim \mathcal{N}
\left(
\begin{bmatrix}
  0 \\
  \vdots \\
  0
\end{bmatrix},
\begin{bmatrix}
  k(x_1, x_1) & \cdots & k(x_1, x_N) \\
  \vdots & \ddots & \vdots \\
  k(x_N, x_1) & \cdots & k(x_N, x_N)
\end{bmatrix}
\right),
$$

which can be written more compactly as follows:

$$p(F) = \mathcal{N}(F | 0, K).$$

Here, $K$ is the covariance matrix with entries $K_{ij} = k(x_i, x_j)$. After observing a collection of noise-free data points, usually in form of input/output pairs $(X, F)$, a GP specifies a Gaussian posterior predictive distribution at any test point $x^*$:

$$p(f(x^*)|X, F) = \mathcal{N}(f(x^*)|k(x^*, X)k(X,X)^{-1}F, k(x^*, x^*) - k(x^*, X)k(X,X)^{-1}k(X,x^*)),$$

Informally speaking, posterior mean would have similar values to the outputs of those states that are close to $x^*$, and the variance grows as $x^*$ goes away from the data points.

The computational bottleneck in GP inference is the matrix inversion in (2.1), which scales cubically $O(N^3)$ with the size of the input set. A widely used technique to enable GP models in big data regimes is the inducing variable approach (Seeger et al., 2003; Quiñonero-Candela and Rasmussen,
The idea is built on augmenting the GP with a set of inducing points \( Z = [z_1, z_2, \ldots, z_M]^T \) that live in the same space as \( X \), and corresponding outputs \( U = [u_1, u_2, \ldots, u_M]^T \) with \( u_m \equiv f(x_m) \). In turn, the predictive posterior distribution is computed conditioned on the inducing set:

\[
p(f(x^*)|Z, U) = \mathcal{N}(f(x^*)|k(x^*, Z)k(Z, Z)^{-1}U, k(x^*, x^*) - k(x^*, Z)k(Z, Z)^{-1}k(Z, x^*)).
\]

The resulting model is a low-rank inducing point approximation of GPs, also called sparse GP (SGP). With inducing approximation, computational cost of predictive posterior becomes \( O(NM^2) \), which would give a significant improvement when \( M \ll N \). Inducing points are typically learned by variational inference (Titsias, 2009; Hensman et al., 2013) in which inducing locations are treated as hyperparameters. Also, the original exact GP formulation can be retrieved by setting \( Z = X \) and \( U = F \).

### 2.2 Differential Equations

We now introduce ordinary and stochastic differential equations. Without loss of generality, this section treats time \( t \) as the independent variable on which state solutions \( x(t) \) depends. After giving formal definitions, we illustrate the concepts on Van der Pol oscillator, a parametric differential equation system describing the motion of a non-conservative oscillator. The section is concluded by the numerical approximations to state solutions and gradient computation.

#### 2.2.1 Ordinary Differential Equations

Ordinary differential equation (ODE) systems are defined as

\[
x(t) = \frac{dx(t)}{dt} = f(x(t), a(t), t),
\]

where \( x(t) \in \mathcal{X} \subseteq \mathbb{R}^d \) is the state vector of a \( d \)-dimensional dynamical system at time \( t \), \( a(t) \in \mathcal{A} \subseteq \mathbb{R}^m \) is the external control signal and \( \dot{x}(t) \in \dot{\mathcal{X}} \subseteq \mathbb{R}^d \) is the first order time derivative of \( x(t) \). The vector-valued and continuous (time) differential function \( f : \mathcal{X} \times \mathcal{A} \times \mathbb{R}_+ \to \dot{\mathcal{X}} \) describes the system’s evolution over time with \( \mathbb{R}_+ \) denoting non-negative real numbers (Tenenbaum, 1985). The control signals are typically bounded within a fixed range \( a_i^{\text{min}} \leq a_i(t) \leq a_i^{\text{max}} \).

**Computing ODE solutions** An ODE solution \( x(t) \) at time \( t \in \mathbb{R}_+ \) is a function given by

\[
x(t) = x_0 + \int_0^t f(x(\tau), a(\tau), \tau) \, d\tau,
\]
where $x_0$ denotes the initial value and $\tau \in \mathbb{R}_+$ is an auxiliary time variable. The above equation implies $\dot{x}(\tau) = f(x(\tau), a(\tau), t) \quad \forall \tau \in [0, t]$ and thus $x(t)$ has a continuous derivative everywhere.

The celebrated Picard’s existence and uniqueness theorem states that an initial value problem given by $\dot{x}(t) = f(x(t), a(t), t)$ and $x(0) = x_0$ has a unique solution $x(t)$ if $f(x(t), a(t), t)$ satisfies the Lipschitz condition:

$$|f(x_1, t) - f(x_2, t)| \leq K |x_1 - x_2|, \quad \forall (x_1, t), (x_2, t).$$

Despite the uniqueness guarantee, there is no general recipe to analytically compute the solution $x(t)$. We examine the numerical methods to compute ODE solutions in Section 2.2.3.

**Higher dimensional ODEs** Our discussion so far has concentrated on first-order ODE systems. In general, the order of an ODE system is determined by the highest order time-derivative involved in the equation. For instance, a $p$’th order ODE is defined as

$$x^{(p)}(t) = F(x(t), x^{(1)}(t), \ldots, x^{(p-1)}(t), a(t), t)$$

$$x^{(r)}(t) \equiv \frac{d^r x(t)}{dt^r} \in \mathbb{R}^d, \quad \forall r \in [1, \ldots, p],$$

where $x^{(r)}(t)$ stands for the $r$’th order derivative and $F$ is the $p$’th order time derivative function that takes as input all lower order derivatives. Note that any higher order ODE appears as a special case of a first-order system defined in an augmented space:

$$z \equiv [x, x^{(1)}, \ldots, x^{(p-1)}] \in \mathbb{R}^{dp}$$

$$f(z(t), a(t), t) = [x^{(1)}, x^{(2)}, \ldots, x^{(p-1)}, F(z, a, t)] \in \mathbb{R}^{dp},$$

where $[\cdot]$ column stacks vectors and we drop $(t)$ from our state notation for clarity.

### 2.2.2 Stochastic Differential Equations

Stochastic differential equations (SDEs) are informally defined as noise-driven ODEs (Øksendal, 2014; Särkkä and Solin, 2019):

$$\frac{dx(t)}{dt} = f(x(t)) + \sigma(x(t))W_t. \tag{2.5}$$

We first note that without loss of generality, we drop any explicit dependence on time or control signals as they are not relevant to any of our publications. Above $f(x)$ and $\sigma(x)$ are referred to as drift and diffusion functions. $W_t$ denotes a multivariate continuous-time stochastic process, called Wiener process or white noise. The Wiener process always starts from a zero initial state $W_0 = 0$, and the increments $W_{t+\Delta t} - W_t \sim \mathcal{N}(0, \Delta t I)$
Theoretical Foundations

Figure 2.1. Two Brownian motion illustrations. The left panel demonstrates 100 different realizations of the one dimensional Brownian motion plotted over time. On the right are two examples of 2D motions.

follow a Gaussian distribution with standard deviation $\sqrt{\Delta t}$, i.e., are also independent from the past values $W_s$, $s \leq t$. Due to historical connections, Wiener process is also called Brownian motion, which was first proposed to study the random motions of particles suspended in fluid. More formally, Brownian motion $B(t)$ is the solution to the following stochastic differential equation:

$$\frac{dB_t}{dt} = W_t. \quad (2.6)$$

We illustrate one and two dimensional Brownian motions in Figure 2.1.

The theory of ODEs requires that the right hand side of a differential system must be continuous whereas the white noise process in (2.5) is discontinuous. Since the differential does not exist in the ordinary sense, we turn our attention to Itô calculus which allows us to compute integrals with respect to stochastic processes. By multiplying both sides of (2.5) by $dt$ and plugging in the definition of Brownian motion (2.6), we obtain the following differential:

$$dx(t) = f(x(t))dt + \sigma(x(t))dB_t.$$ 

Integrating both sides of above equation gives us the stochastic state solution:

$$x(t) = x_0 + \int_0^t f(x(\tau))d\tau + \int_0^t \sigma(x(\tau))dB_\tau,$$

where the second integral is an Itô integral. We refer the readers to Øksendal (2014) and Särkkä and Solin (2019) for a more detailed derivation of SDE state solutions.

**Example: Van der Pol Oscillator**

As an example, we examine Van der Pol (VDP) oscillator, a parametric second-order time-invariant ODE system that evolves according to the following differential equation:

$$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x = 0, \quad (2.7)$$
where we set the only parameter $\mu = 1$. By defining $y \triangleq \frac{dx}{dt}$ and organizing the terms, we obtain the following two dimensional first-order system:

$$\frac{d}{dt} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ y (1 - x^2) y - x \end{bmatrix}. \quad (2.8)$$

To illustrate how Brownian motion impacts the deterministic ODE trajectory, we contrast ODE trajectories evolving according to VDP dynamics (2.8) with SDE trajectories evolving according to the following differential:

$$\frac{d}{dt} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} y (1 - x^2) y - x \end{bmatrix} dt + \sigma \begin{bmatrix} x \\ y \end{bmatrix} \cdot dB_t, \quad (2.9)$$

where $I$ denotes the identity matrix. For notational convenience, we combine the concatenated states into a single variable: $x = [x, y]^T$. We consider two constant diffusion functions $\sigma(x, t) = 0.05$ and $\sigma(x, t) = 0.25$, as well as a state-dependent diffusion $\sigma(x, t) = 3N(x; [-2, 0]^T, I)$ with $N$ denoting the Gaussian distribution. We integrate all systems with initial condition $x_0 = [1, 0]^T$ for $t \in [0, 30]$ (the details of integration are presented in Section 2.2.3). Figure 2.2 illustrates the trajectories in state space and also over time, please see the figure caption for more details.

### 2.2.3 Numerical Solvers

We now give a quick recap of numerical approximations to ODE and SDE state solutions. Note that this section places more emphasis on ODE approximations since they form the basis for their SDE counterparts.

**Numerical Approximations to ODEs**

The literature on numerical approximations to the solutions of ODEs dates back to 18th century when Euler discovered the simplest numerical integration technique. A thorough overview of the numerical approximations is given in Butcher and Goodwin (2008) and Atkinson et al. (2011). In this subsection, we informally discuss numerical approximations relevant to our publications on a generic one-dimensional ODE system.

The simplest and least efficient numerical method is known as **Euler’s method**. Relying on the standard derivative approximation, Euler’s method computes the state solution $x(t_{n+1})$ starting from the previous solution $x(t_n)$ as follows:

$$x(t_{n+1}) = x(t_n) + hf(x_n, t_n), \quad h \equiv t_{n+1} - t_n, \quad (2.10)$$

where we use the shorthand notation $x_n \equiv x(t_n)$. Here, $h$ represents the step size and controls the local truncation error, i.e., the error made in a single step, which can be obtained via the Taylor approximation of the state solution (Butcher and Goodwin, 2008):

$$x^{\text{true}}(t_{n+1}) - x(t_{n+1}) = \frac{1}{2} h^2 \ddot{x}(t) + O(h^3).$$
Theoretical Foundations

Figure 2.2. An illustration of Van der Pol oscillator. The ODE plots are generated using (2.8) while SDE trajectories follow (2.9). (a) Phase portraits in which each axis corresponds to one state. The empty circles and red curves show the initial value and ODE state trajectory, which are identical across all four figures. The streamplots on the background demonstrate the vector field, i.e., the time derivative (2.8). Streams get darker as the magnitude of the vector field grows. Colored non-smooth curves are 25 trajectory samples from each SDE system. (a2-a3) As expected, variance of the sampled trajectories grow with diffusion and the trajectories are corrupted everywhere due to constant diffusion. (a4) While the dynamics are highly chaotic near the diffusion hotspot $[-2, 0]$, they become almost deterministic and even converge to the limit cycle far away. (b) The same state trajectories plotted over time. (b1) The system already reaches the limit cycle near $t = 7$. (b2-b4) The phase difference between the ODE state solution and SDE state distribution $p(x_t)$ grows as integration proceeds.
The first implication of the above equation is that decreasing step size $h$ improves the quality of approximation, leading to the classical trade-off between speed and accuracy. More importantly, Euler’s method becomes less accurate as the local gradient changes more abruptly, which is due to the first term on the right hand side (rhs) of the equality. Such equations are sometimes called *stiff* equations, although there is no consensus on the formal definition of stiffness (Butcher and Goodwin, 2008).

Stability of Euler’s method can be improved by simply performing more computations in a single step. For instance, taking an additional step to compute an averaged slope leads to the so-called *Heun’s method* (Heun, 1900):

$$\tilde{x}(t_{n+1}) = x(t_n) + hf(x_n, t_n)$$

$$x(t_{n+1}) = x(t_n) + \frac{h}{2} \left[ f(x_n, t_n) + f(\tilde{x}_{n+1}, t_{n+1}) \right].$$

Being a second order technique, Heun’s method is one of the simplest example of a class of numerical algorithms known as *Runge-Kutta (RK) methods* (Runge, 1895; Kutta, 1901). Due to their ease of implementation in digital computers, the literature on higher order RK solvers quickly advanced (Butcher and Goodwin, 2008). In our publications, we often utilize RK4 solver, a popular fourth-order procedure with $O(h^5)$ local truncation error given by the following set of equations:

$$z_1 = x_n$$
$$z_2 = x_n + \frac{h}{2} f(z_1, t_n)$$
$$z_3 = x_n + \frac{h}{2} f(z_2, t_n + \frac{h}{2})$$
$$z_4 = x_n + hf(z_3, t_n + \frac{h}{2})$$

$$x(t_{n+1}) = x(t_n) + \frac{h}{6} \left[ f(z_1, t_n) + 2f(z_2, t_n + \frac{h}{2}) + 2f(z_3, t_n + \frac{h}{2}) + f(z_4, t_n + h) \right].$$

A clever direction to further improve higher-order RK methods is to set the step size $h$ dynamically, leading to adaptive-step ODE solvers (also known as *Fehlberg methods*) (Fehlberg, 1969). These solvers simultaneously compute $x(t_{n+1}) - x(t_n)$ by using two methods of different orders, and then the higher-order formula is used to estimate the local approximation error of the lower-order formula. In turn, the error is used to tune the step size. Despite computing two formulae, such methods are highly compute friendly. For instance, one of the most commonly used adaptive-step solver, *RK45 (dopri5)*, uses only six function evaluations to compute forth and fifth order approximations to $x(t_{n+1}) - x(t_n)$ (Dormand and Prince, 1980).

**Numerical Approximations to SDEs**

Numerical approximations to SDE state solutions are typically built upon their ODE counterparts. A detailed categorization of numerical SDE
approximations are presented in Särkkä and Solin (2019). Here, we explain the only solver involved in our publications, namely, Euler-Maruyama (EM) method.

Given a set of time points \([t_0, \ldots, t_N]\), we first divide the interval \([t_0, t_N]\) into \(M \gg N\) steps \([\tilde{t}_0, \ldots, \tilde{t}_M]\) of fixed length \(h = (t_N - t_0)/M\). We then sample the Wiener increments as \(\Delta B_m \sim N(0, h \cdot I)\), \(\forall m \in [0, \ldots, M]\), and iteratively compute the following:

\[
x(\tilde{t}_{m+1}) = x(\tilde{t}_m) + f(x_m)h + \sigma(x_m)\Delta B_m.
\] (2.11)

The strong order of convergence of EM is given as follows:

\[
\mathbb{E}[|x^{\text{true}}(\tilde{t}_{m+1}) - x(\tilde{t}_{m+1})|] \leq K\sqrt{h}
\]

for a constant \(K\). Therefore, similar to Euler’s method, the accuracy of state solutions increase with \(M\).

2.2.4 Gradient Computation

All our publications except IV require approximating unknown time differentials by black-box function approximators. In the following we describe different strategies to learn the parameters of an ODE system driven forward by \(f_\theta\) with \(\theta \in \mathbb{R}^P\) denoting the parameters. Please see Publication II for a discussion on SDE parameter learning.

In the most general sense, the learning objectives of our interest can be described as minimizing the integral of an arbitrary loss \(\ell(x(t); D)\) that involves forward simulated states \(x(t) = x_0 + \int_0^tf_\theta(x_\tau)d\tau\) and a dataset \(D\):

\[
\min_\theta L = \int_0^T \ell(x(t); D)dt,
\]

where we omit the control signals and the explicit dependence on time for clarity. Any optimization algorithm based on gradient descent would need to compute the gradient of forward simulated states with respect to model parameters: \(\frac{dx(t)}{d\theta} \in \mathbb{R}^{d \times P}\). The majority of machine learning problems requires countably many tensor operations to compute the loss, allowing the gradients to be computed trivially by the chain rule. Nonetheless, forward simulated states are given by continuous-time integrals, which makes the gradient computation non-trivial. Next, we summarize three strategies to compute \(\frac{dx(t)}{d\theta}\).

**Backpropogate through Numerical Solver**

The first and simplest approach is to store the entire computational graph built by the numerical integration, and backpropogate through it. Naturally, this technique yields the correct gradients given an accurate automatic differentiation framework. Nonetheless, a long backpropogation tree
may easily cause vanishing or exploding gradients issues that are commonly observed in recurrent neural networks (Pascanu et al., 2013). Also, as pointed out in Zhuang et al. (2020), the backward computational graph would unnecessarily include the intermediate calculations performed during the step size search in case of an adaptive solver.

**Sensitivity Equations**

The goal of sensitivity analysis is to understand how the model responses change with variations in the input variables (Confalonieri et al., 2010; Weber et al., 2018). The local sensitivity of ODE models was introduced in a seminal work by Petar and Heller (1967) and recently revisited in the context of mechanistic modeling of biochemical reaction networks (Fröhlich et al., 2017). The analysis starts with considering the time derivative of the target gradient denoted by \( S(t) \) below:

\[
\frac{d}{dt} \frac{d}{\theta} x(t, \theta) = \frac{d}{\theta} f_\theta(x(t, \theta)) \in \mathbb{R}^{d \times P},
\]

where we denote a state by \( x(t, \theta) \) to explicitly show its dependence on the parameters \( \theta \). Since both inputs to \( f_\theta \), namely \( \theta \) and \( x(t, \theta) \), are functions of \( \theta \), the rhs term can be computed by total derivation:

\[
\frac{d}{dt} \frac{d}{\theta} x(t, \theta) = J(t) S(t) + R(t). \tag{2.12}
\]

The terms \( R(t) \in \mathbb{R}^{d \times P} \) and \( J(t) \in \mathbb{R}^{d \times d} \) can easily be obtained by auto differentiating the output of \( f_\theta \) with respect to its inputs. The target gradient is then given by the following ODE, hence also called forward sensitivities:

\[
S(t) = S_0 + \int_0^t J(\tau) S(\tau) + R(\tau)d\tau. \tag{2.13}
\]

If the initial state \( x_0 \) is known, \( S_0 = \frac{dx_0}{\theta} \) becomes a zero-matrix. Alternatively, one could define \( x_0 \) as a free parameter and the sensitivity equations would also give the gradient \( \frac{dL}{dx_0} \).

The state solutions (2.4) and the sensitivity ODE (2.13) can be computed together in a single call to numerical ODE solver, which leads to an \( d(1+P) \) dimensional ODE system. If the loss \( J \) is computed over a sequence of length \( T \), the memory footprint becomes \( O(TdP) \). Therefore, this technique is not suitable for computing the gradients of overparameterized functions such as neural networks.

**Adjoint Method**

The adjoint equation provides a computationally efficient alternative to forward sensitivities. Even though the method antedate the first works
on optimal control (Pontryagin et al., 1962; Pontryagin, 2018), the neural ODE (NODE) breakthrough (Chen et al., 2018b) introduced the method to the machine learning community. The method starts by considering the gradient $\nabla_\theta L$ of a Lagrangian involving both the loss function and ODE constraint:

$$
L = \int_0^T \ell(x(t); D) + \lambda(t)^T (x(t) - f_\theta(x(t))) dt,
$$

where $\lambda(t)$ denotes the Lagrange multiplier, or costate, at time $t$. After a series of algebraic manipulations, we obtain the dynamics of the multipliers (Pontryagin, 2018; Bradley, 2013):

$$
\lambda(t) = -\lambda(t)^T \frac{\partial f_\theta(x(t))}{\partial x(t)} + \frac{\partial \ell(x(t); D)}{\partial x(t)}, \quad (2.14)
$$

where $\lambda(T) = 0$ and the integral is computed in the backward direction. Our target gradient also follows an integral:

$$
\frac{dL}{d\theta} = \int_0^T -\lambda(t)^T \frac{\partial f_\theta(x(t))}{\partial \theta} + \frac{\partial \ell(x(t); D)}{\partial \theta} dt \quad (2.15)
$$

Similar to the forward sensitivities, the costates (2.14) and the loss integral (2.15) are executed concurrently. Since the gradient of the loss is computed in the backward direction, the adjoints are viewed as the continuous-time analog of backpropagation. Also, the memory consumption of the loss integral (2.15) is independent of $T$, hence improving the resource usage in orders of magnitude.

In their recent presentation of the adjoint method, Chen et al. (2018b) propose to discard the forward states $x(t)$ given by (2.4), and re-compute them during backpropagation (2.15) for memory considerations. In a more recent work, it was demonstrated that the forward and backward state trajectories are not identical (Zhuang et al., 2020), which leads to numerical inaccuracies. The authors resolve this problem by their adaptive checkpoint algorithm (ACA) that records the forward trajectory while also controlling memory cost. We utilized ACA in our Publication V, and experimentally verify its superiority over the standard adjoint method.

2.3 A Primer on Parameter Inference

We now examine parameter inference in machine learning models. Given a dataset $D = \{x_i\}_{i=1}^N$ consisting of $N$ data points, the maximum likelihood (ML) approach aims to find a set of parameters $\theta \in \mathbb{R}^P$ that maximizes the likelihood (Bishop, 2006):

$$
\max_{\theta} p(D|\theta). \quad (2.16)
$$
Theoretical Foundations

The data points are typically assumed to be independently and identically distributed (iid), which allows the likelihood to be factorized. We can further simplify the likelihood by maximizing its logarithm instead:

$$\max_{\theta} \sum_{i=1}^{N} \log p(x_i | \theta).$$  \hspace{1cm} (2.17)

Above optimization problem (2.17) has the same optima as the original formulation (2.16) since \(\log(\cdot)\) is a monotonic function. This ML approach leads to a so-called point estimate for \(\theta\), which does not provide any measure of uncertainty. Nonetheless, many applications require to know how much we can trust a model in its predictions. Bayes’ rule provides a natural framework to express our degree of belief in form of probability distributions. Given a prior distribution \(p(\theta)\) on the unknown quantities (parameters \(\theta\) here), Bayes’ theorem gives the formula for the posterior distribution, which reflects our belief after seeing data:

$$p(\theta | D) = \frac{p(D | \theta) p(\theta)}{p(D)},$$

where the denominator is independent of \(\theta\). When the likelihood function is complex or the prior is not conjugate to likelihood, deriving a closed form expression for the posterior distribution \(p(\theta | D)\) becomes too difficult. For such cases, we could directly maximize the posterior distribution (or its logarithm), a technique called maximum a posteriori (MAP) estimation:

$$\max_{\theta} \log p(\theta | D) = \log p(\theta) + \sum_{i=1}^{N} \log p(x_i | \theta) + c,$$  \hspace{1cm} (2.18)

where the constant \(c = -\log p(D)\) appears due to the denominator in (2.17). The only difference between ML (2.17) and MAP (2.18) estimates is the prior term, which acts as a regularizer (Bishop, 2006). Despite the regularization effect, MAP estimation still prevents us from maintaining a measure of uncertainty over the unknown parameters \(\theta\). Next, we discuss two alternative inference schemes that approximate intractable posteriors. This section is concluded by a brief introduction to numerical optimization.

2.3.1 Variational Inference

Variational inference (VI) aims to approximate an intractable posterior \(p(\theta | D)\) with a family of tractable densities \(\mathcal{D}\) (Jordan et al., 1999; Blei et al., 2016). Once \(\mathcal{D}\) is specified, we try to find the member \(q(\theta)\) of the family closest to the true posterior

$$\arg \min_{q(\theta) \in \mathcal{D}} \text{KL}[q(\theta) \mid \mid p(\theta | D)],$$  \hspace{1cm} (2.19)
where KL denotes the KL divergence that measures the discrepancy between two distributions. The density \( q(\theta) \) is often referred to as variational posterior. Expanding the KL divergence (2.19) gives

\[
\text{KL}[q(\theta) \| p(\theta|D)] = E[\log q(\theta)] - E[\log p(\theta, D)] + \log p(D),
\]

where the expectations are with respect to \( q(\theta) \). This objective is again impossible to compute due to the unknown marginal log likelihood \( \log p(D) \). However, since \( \log p(D) \) does not depend on the variational posterior, we simply discard it and maximize (the minus of) the remaining terms, leading to evidence lower bound (ELBO):

\[
\text{ELBO}(q) = E[\log p(D, \theta)] - E[\log q(\theta)] = E[\log p(D|\theta)] - \text{KL}[q(\theta) \| p(\theta)].
\]

The first term above denotes the expected log likelihood, which favors distributions \( q(\theta) \) that better fits the data. The second term computes the KL divergence between the approximate posterior and a user-specified prior distribution, which is typically chosen to inject our a priori knowledge about \( \theta \). Next we introduce two families of tractable approximate densities \( \mathcal{D} \).

**Mean-Field Variational Inference**

The simplest class of approximate densities assumes that variational factors \( \{\theta_p\}_{p=1}^P \) are independent of each other, which gives the following factorization:

\[
q(\theta) = \prod_{p=1}^P q_p(\theta_p; \alpha_p),
\]

with \( q_p(\cdot) \) and \( \alpha_p \) denoting a factor-specific distribution and corresponding variational parameters. As an example application of mean-field VI, we consider Bayesian neural networks, which typically lead to analytically intractable posterior densities (Graves, 2011). A mean-field VI approximation places a Gaussian distribution on each individual weight and bias parameter:

\[
q_p(\theta_p; \alpha_p) = \mathcal{N}(\theta_p; \mu_p, \sigma^2_p).
\]

Typically, the prior distribution \( p(\theta) \) is chosen to be an isotropic Gaussian, allowing a closed form expression for the KL term in ELBO (2.20). Also, the expected log likelihood term can be approximated by Monte Carlo sampling:

\[
E[\log p(D|\theta)] \approx \frac{1}{L} \sum_{\ell=1}^L \log p(D|\theta^\ell), \quad \theta^\ell \sim \mathcal{N}(\theta; \mu, \sigma^2)
\]

with \( \mu = [\mu_1, \ldots, \mu_P]^T \in \mathbb{R}^P \) and \( \sigma^2 = \text{diag}([\sigma^2_1, \ldots, \sigma^2_P]) \in \mathbb{R}^{P \times P} \) denoting the collection of variational mean and variance parameters, noting that \( \text{diag}(\cdot) \) forms a diagonal matrix from its argument.
Amortized Variational Inference

Mean-field VI has so-called local parameters for each unknown, which does not constitute a problem when the unknowns remain the same during training and testing. However, probabilistic models often associate data points with latent variables, implying the need for mechanisms to infer latent variables for (unseen) test points. Also, the number of variables increases with the dataset size, which makes large-scale applications difficult. To that end, amortized VI replaces free local parameters with a flexible and powerful global approximation, e.g., neural network. Below we consider one of the most popular models utilizing amortized inference, namely, variational auto-encoders (VAEs) (Kingma and Welling, 2014; Rezende et al., 2014).

The VAE framework assumes a data point \( x \) is generated by an unknown process that involves an unobserved continuous random variable \( z \). The unknown generative process is approximated by a non-linear decoder neural network \( p_\gamma(x|z) \) parameterized by \( \gamma \), which makes the posterior distribution \( p(z|x) \) intractable. As an approximation to the posterior, a recognition model \( q_\psi(z|x) \) is proposed, which leads to the following expression for ELBO:

\[
\text{ELBO}(x; \psi, \gamma) = \mathbb{E}_{q_\psi(z|x)} \left[ \log p_\gamma(x|z) \right] - \text{KL}[q_\psi(z|x) \| p(z)].
\]

In VAEs, the recognition model is chosen to be a neural network \( g_\psi \) that outputs the parameters of the variational posterior distribution. This neural network is also known as encoder or inference network. The approximate posterior \( q_\psi(z|x) \) is typically chosen to allow reparameterization \( z \sim g_\psi(\epsilon, x) \) using an auxiliary noise variable \( \epsilon \), which in turns leads to the convenient Monte Carlo integration shown below:

\[
z^{(l)} = g_\psi(\epsilon^{(l)}, x), \quad \epsilon^{(l)} \sim p(\epsilon)
\]

\[
\mathbb{E}_{q_\psi(z|x)} \left[ \log p_\gamma(x|z) \right] = \mathbb{E}_{p(\epsilon)} \left[ \log p_\gamma(x|g_\psi(\epsilon, x)) \right] \approx \frac{1}{L} \sum_{l=1}^{L} \log p_\gamma(x|g_\psi(\epsilon^{(l)}, x)).
\]

The original VAE paper proposes to set the approximate posterior to a multivariate Gaussian distribution with a diagonal covariance

\[
\log q_\psi(z|x) = \log \mathcal{N}(z; \mu, \sigma I)
\]

and prior to a centered isotropic multivariate Gaussian:

\[
p(z) = \mathcal{N}(z; 0, I).
\]

Putting all pieces together, we obtain the following approximation to the
ELBO:

\[
\text{ELBO}(\mathbf{x}; \psi, \gamma) \approx \frac{1}{L} \sum_{\ell=1}^{L} \log p_\gamma(\mathbf{x}|z^{(\ell)}) - \text{KL}[\mathcal{N}(\mu, \sigma I) \| \mathcal{N}(0, I)]
\]

\[
z^{(\ell)} = g_\psi(\epsilon^{(\ell)}, \mathbf{x}) = \mu + \epsilon^{(\ell)} \sigma I
\]

\[
\epsilon^{(\ell)} \sim \mathcal{N}(0, I).
\]

The bound is maximized with respect to the global parameters \(\psi\) and \(\gamma\). The gradients are computed by reparameterization trick (Kingma and Welling, 2014), which is also known as stochastic backpropagation (Rezende et al., 2014).

### 2.3.2 Stochastic Gradient Markov Chain Monte Carlo

In Markov chain Monte Carlo (MCMC), the goal is to construct a Markov chain whose stationary distribution is the target density we wish to sample from. The stationary distribution is fully specified by the transition kernel, which describes the evolution of the Markov chain in probabilistic terms. Thus, the main research question in MCMC is how to construct the kernel given a target density that can be evaluated up to a normalizing constant. In this subsection, we focus on stochastic gradient MCMC (SGMCMC) methods, which utilize continuous stochastic dynamics to define transition kernels. For a rigorous treatment of transition kernels and a comprehensive survey of MCMC methods, please see (Andrieu et al., 2003; Bishop, 2006; Cemgil, 2014).

We start with the assumption that the target posterior density of our interest can be written as

\[
p(\theta|\mathcal{D}) \propto \exp(-U(\theta))
\]

where \(U(\theta)\) is known as a potential energy function. In complicated models, often times direct sampling from \(p(\theta|\mathcal{D})\) is difficult while evaluating \(U(\theta)\) for any input \(\theta\) is easy. Ma et al. (2015) showed that an SDE of the form

\[
d\theta = f(\theta)dt + \sqrt{2D(\theta)}dW(t) \quad (2.21)
\]

with a special drift function that involves the gradient of \(U(\theta)\)

\[
f(\theta) = -[D(\theta) + Q(\theta)] \nabla U(\theta) + \Gamma(\theta), \quad \Gamma_i(\theta) = \sum_{j=1}^{P} \frac{\partial}{\partial \theta_j} [D_{ij}(\theta) + Q_{ij}(\theta)]
\]

\[
(2.22)
\]

has a stationary distribution proportional to our target density \(p(\theta|\mathcal{D})\). Above, \(Q(\theta)\) and \(D(\theta)\) are respectively skew-symmetric and positive semi-definite matrices with \((\cdot)_{ij}\) denoting the \(i\)’th row and \(j\)’th column. Typically,
Theoretical Foundations

\(Q(\theta)\) and \(D(\theta)\) are chosen to reflect our choices in the traversing effects and the strength of the diffusion process. The rest of this subsection examines two special cases of this general framework for sampling from intractable posterior distributions.

**Stochastic Gradient Langevin Dynamics (SGLD)**

Since we are free to choose the matrices \(D(\cdot)\) and \(Q(\cdot)\), we simplify the above framework by setting \(D(\theta)\) to the identity matrix and \(Q(\theta)\) to a zero-matrix, which causes \(\Gamma(\theta)\) to disappear as a by-product. In turn, Euler-Maruyama discretization of the dynamics (2.21) gives a straightforward update equation representing the transition kernel:

\[
\theta_{t+1} = \theta_t + \epsilon_t \nabla U(\theta) + \eta_t \tag{2.23}
\]

\[
\eta_t \sim \mathcal{N}(0, 2\epsilon_t \cdot I).
\]

The above equation is identical to gradient ascent update up to the last noise term. As shown in Publication IV, a tempered version of (2.23) connects SGMCMC approaches with optimization. Furthermore, Welling and Teh (2011) showed that the gradient term \(\nabla U(\theta)\) can be replaced by its unbiased estimate computed on mini-batches of data, which significantly improves the overall execution time.

**Hamiltonian Monte Carlo**

Hamiltonian (hybrid) Monte Carlo has been proven one of the most prominent MCMC methods that can efficiently explore posterior distributions (Duane et al., 1987; Neal et al., 2011; Ma et al., 2015). As suggested by its name, HMC simulates approximate Hamiltonian dynamics described by so-called generalized position \(\theta\) and momentum \(\rho\) variables. In turn, the Hamiltonian function gives the total energy of a closed system:

\[
\mathcal{H}(\theta, \rho) = U(\theta) + g(\rho).
\]

where \(U(\theta)\) was already defined as the potential energy. Analogously, \(g(\rho)\) is known as the kinetic energy function and usually set as \(g(\rho) = \frac{1}{2} \rho^T M^{-1} \rho\) with \(M\) being a mass matrix. The time evolution of this system is expressed by Hamilton’s equations:

\[
\frac{d\theta}{dt} = \frac{\partial \mathcal{H}}{\partial \rho} = M^{-1} \rho
\]

\[
\frac{d\rho}{dt} = -\frac{\partial \mathcal{H}}{\partial \theta} = -\nabla U(\theta)
\]

which is typically approximated by a symplectic solver such as the leapfrog integrator (Neal et al., 2011). HMC can be seen as a special case of the SGMCMC framework defined in the extended space \((\theta, \rho)\). More specifically, setting \(p(\theta, \rho | \mathcal{D}) \propto \exp(g(\rho) - U(\theta))\), \(D(\theta, \rho) = 0\), and \(Q(\theta, \rho) = \begin{pmatrix} 0 & -I \\ I & 0 \end{pmatrix}\) leads to the dynamics equations given above.
2.3.3 Numerical Optimization

This subsection introduces several concepts and ideas for solving unconstrained optimization problems of the following sort:

$$\min_{\theta} f(\theta),$$  \hspace{1cm} (2.24)

where $f : \mathbb{R}^P \rightarrow \mathbb{R}$ is an infinitely differentiable objective function to be optimized with respect to $\theta \in \mathbb{R}^P$. Descent methods that we examine here produce a sequence of iterates $\theta_k$ to solve (2.24) (Boyd and Vandenberghe, 2004):

$$\theta_{k+1} = \theta_k - \alpha_k \Delta \theta_k$$  \hspace{1cm} (2.25)

with the step size $\alpha_k > 0$ such that $f(\theta_{k+1}) < f(\theta_k)$. To choose the update direction $\Delta \theta$, we consider the Taylor approximation of $f$ around $\theta_k$:

$$f(\theta_k - \Delta \theta) = f(\theta_k) - \nabla f_{\theta_k} \Delta \theta + \nabla^2 f_{\theta_k} \frac{(\Delta \theta)^2}{2!} - \cdots - \nabla^m f_{\theta_k} \frac{(\Delta \theta)^m}{m!} - \cdots$$

with $\nabla^m f_{\theta_k} = \frac{d^m f(\theta_k)}{d\theta^m}$ denotes the $m$th order derivative at $\theta_k$. If we consider only the first-order Taylor expansion, the maximum leverage out of moving along $\Delta \theta$ can be obtained by choosing it to be in the same direction as the gradient:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla f_{\theta_k}$$

This update technique is thus called gradient descent. Since the Taylor approximation holds in a small neighborhood of $\theta_k$, we choose step size $\alpha_k$ to be small in magnitude. Typically, too small step sizes slow down the convergence whereas setting them too big may cause diverging from the local optima (Nocedal and Wright, 2006).

Higher order Taylor expansions would naturally give better approximations to $f(\theta_k - \Delta \theta)$. We can maximize the second-order approximation

$$f(\theta_k + \Delta \theta) \approx f(\theta_k) - \nabla f_{\theta_k} \Delta \theta + \nabla^2 f_{\theta_k} \frac{(\Delta \theta)^2}{2}$$

by setting the derivative to zero:

$$\frac{d}{d\Delta \theta} f(\theta_k + \Delta \theta) = -\nabla f_{\theta_k} + \nabla^2 f_{\theta_k} \Delta \theta = 0$$

$$\Delta \theta = (\nabla^2 f_{\theta_k})^{-1} \nabla f_{\theta_k}.$$  \hspace{1cm} (2.26)

This method is known as Newton’s method since it depends on the iterative root finding technique developed by Newton and Raphson (Nocedal and Wright, 2006). Newton’s method is more favorable to gradient descent thanks to its higher rate of converge. Nonetheless, in high dimensions, even storing the $P \times P$ Hessian matrix $\nabla^2 f_{\theta_k}$ becomes very expensive. For
Theoretical Foundations

Figure 2.3. An illustration of gradient descent and L-BFGS methods on a convex problem. Each row shows the first 10 and 100 iterations of the same optimization procedure. The contours and curves show the optimization surface and iterations $\theta_1:T$ with different fixed step sizes $\alpha$. (a-b) Gradient descent with high step sizes (red) can quickly reach near the minimum but oscillates around it instead of converging. Small step sizes require more iterations but optimization finally converges. (c) Typical L-BFGS step sizes are much smaller compared to gradient descent and only few iterations are sufficient for convergence due to the convexity of the problem. (d) Fixed step sizes may cause strange jumps (pink curve); therefore, it is recommended to choose $\alpha$ with line search (Nocedal and Wright, 2006).

such cases, *Quasi-Newton methods* provide an attractive alternative as they achieve similar convergence rates without computing the Hessian (Nocedal and Wright, 2006).

One widely used Hessian approximation is BFGS (due to its inventors Broyden, Fletcher, Goldfarb, and Shanno). The method iteratively updates an approximation to Hessian using the gradients $\nabla f_{\theta_k}$ that are computed as optimization proceeds. A computationally more efficient BFGS variant, named limited memory BFGS (L-BFGS), uses only the most recent $L$ iterations to build an approximation to Hessian. Also, L-BFGS computes the matrix-gradient product in (2.26) using the *two-loop recursion*, which has $O(LP)$ time and space complexity (Nocedal and Wright, 2006). We contrast the gradient descent and L-BFGS on a convex optimization problem in Figure 2.3.
3. **Gaussian Processes for Learning Unknown Differential Equations**

We start this section by a brief review of discrete dynamical systems. Given a sequence of $T$ observations, $Y \equiv y_{1:T}$, discrete dynamical systems, also known as *state-space models*, are characterized by a first-order Markovian state evolution and an observation equation:

$$
\begin{align*}
x_{t+1} &= A(t)x_t + n(t) \\
y_t &= B(t)x_t + v(t),
\end{align*}
$$

where $x_t$ is the $t$'th state of the process of our interest, $A(t)$ is a time-varying state transition matrix, $B(t)$ is the linear observation mapping, and $n(t)$ and $v(t)$ stand for noise processes. Although the term state-space originates in the area of control engineering (Kalman, 1960; Shumway et al., 2000), we omit any control inputs for clarity.

A variety of statistical tools such as auto-regressive (AR) models, hidden Markov models and Kalman filters appear as special cases of the above framework. However, due to its linearity and state independence, the transitions $A(t)x_t$ significantly restrict the expressiveness of the dynamical system. This problem can be alleviated by means of nonlinear, black-box function approximations:

$$
\begin{align*}
x_{t+1} &= f(x_t, t) + n(t) \\
y_t &= g(x_t, t) + v(t).
\end{align*}
$$

Above system is also known as *difference equation* (Jensen, 2011). Using neural networks for $f$ and $g$ leads to so-called *recurrent neural networks (RNNs)* whose well-known examples include long short-term memory networks (Hochreiter and Schmidhuber, 1997) and gated recurrent units (Chung et al., 2014) (Note that RNNs typically exclude the noise terms but include observations $y_t$ in the dynamics). The unknown $f$ and $g$ functions can also be approximated by Gaussian processes, resulting in a series of models (Wang et al., 2006, 2008; Damianou et al., 2011), which are shown to be capable of learning high dimensional video sequences.
We now give a description of how time series forecasting problem is tackled with AR models and then draw connections with differential equations. Throughout the rest of this chapter, we assume that any dataset of our interest consists of \( N \) observed state trajectories \( D = \{ (y^{(n)}(t_{0:T}^{(n)}), t^{(n)}_{0:T}) \}_{n=1}^{N} \) where \( y_i := y(t_i) \) stands for the \( i \)'th observation within a trajectory observed at time \( t_i \). We denote the state difference by \( \Delta y_i = y_{i+1} - y_i \) and the time until the next observation by \( \Delta t_i = t_{i+1} - t_i \). Here, we focus on the more generic case in which observations may arrive irregularly in time.

### 3.1 Auto-Regressive Dynamics Learning

In order to learn the state evolution with function approximators, AR models first construct input-output pairs for training. A well-established technique is to extract state and state difference pairs \( (y_i, \Delta y_i) \) and solve the regression problem (Dorffner, 1996; Wang et al., 2006):

\[
y_{i+1} - y_i \approx f(y_i)
\]  

(3.1)

with a time-invariant dynamics function. Observe that unlike the formulations introduced earlier, the auro-regressive dynamics are defined in the data space. Learning the dynamics this way implicitly relies on uniform sampling assumption. To see that, imagine the system visits some state twice \( y_i = y_j = y \) at timepoints \( t_i \) and \( t_j \). If the observations arrive non-uniformly, i.e., \( \Delta t_i \neq \Delta t_j \), and the system dynamics are different at \( y_i \) and \( y_j \), then the next states would be different: \( y_{i+1} \neq y_{j+1} \). Consequently, resulting data pairs \( (y_i, \Delta y_i) \) and \( (y_j, \Delta y_j) \) would have the same input but different targets, which would lead to training issues.

A simple yet effective workaround is “correcting” the state difference with the time increment:

\[
\frac{y_{i+1} - y_i}{t_{i+1} - t_i} \approx f(y_i).
\]  

(3.2)

We illustrate how time correction resolves the irregular sampling issue on a simulated example, see Figure 3.1 for details.

### 3.1.1 Gradient Matching

A careful examination of (3.2) would reveal that the seemingly ad-hoc time correction term converges to the time differential in the limit. To see this, we rewrite (3.2) by explicitly showing the time dependency:

\[
\frac{y_{i+1} - y_i}{t_{i+1} - t_i} = \frac{y(t_i + \Delta t) - y(t_i)}{\Delta t} \xrightarrow{\Delta t \to 0} \frac{dy(t_i)}{dt_i}.
\]

This approach is known as gradient matching since it aims to approximate the unknown state evolution function via the empirical gradients obtained
Figure 3.1. An illustration of AR dynamics learning with a neural network approximation $f$. We generate ten Van der Pol sequences (2.8) for training and five sequences for testing. We consider two scenarios with uniformly ($\Delta t = 0.1$) and irregularly ($\Delta t \sim U[0, 0.4]$) sampled data sequences ($U$ denoting uniform distribution). We also demonstrate the impact of time correction by training the system with update equations (3.1) and (3.2) on irregularly sampled data. The dashed orange curve shows that the AR model fails to capture the dynamics when trained with irregularly sampled data without time correction. Overlapping solid curves indicate that time correction resolves the issue.

from data pairs (Varah, 1982). Gradient matching has manifested itself as one of the most common strategies to learn ODE systems as it skips the computationally intensive forward integration step (Varah, 1982; Ellner et al., 2002; Ramsay et al., 2007). Nevertheless, gradient matching comes with two fundamental issues: (i) it inevitably leads to wrong differential unless the time difference $\Delta t$ approaches zero, and (ii) noisy observations imply incorrect data pairs $(y_i, \Delta y_i)$ for dynamics training. Hence, in our Publication I and Publication II, we have taken an alternative approach by computing the costly integration. Before explaining our method, we briefly visit some well-known ODE inference techniques.
3.1.2 Related Work

Several authors have already proposed embedding a parametric differential function within a Bayesian or GP framework (Graepel, 2003; Calderhead et al., 2009; Dondelinger et al., 2013; Wang and Barber, 2014; Macdonald, 2017) (see Macdonald et al. (2015) for a review). GPs have been successfully applied to model linear differential equations as they are analytically tractable (Gao et al., 2008; Raissi et al., 2017). However, conventional ODE modelling can only proceed if a parametric form of the driving function $f(\cdot)$ is known. Initial work to handle unknown or non-parametric ODE models have been proposed with various limiting approximations. Early works include spline-based smoothing and additive functions $\sum_{j}^{D} f_j(x_j)$ to infer gene regulatory networks (De Hoon et al., 2002; Henderson and Michailidis, 2014). Äijö and Lähdesmäki (2009) proposed estimating the unknown nonlinear function with GPs using either finite time differences, or analytically solving the derivative function as a function of only time, $\dot{x}(t) = f(t)$ (Äijö et al., 2013). In the technical report of Heinonen and d’Alche Buc (2014) a full vector-valued kernel model $f(y)$ was proposed, however using a gradient matching approximation.

3.2 Learning ODEs

We now explain how unknown time differentials can be learned with GPs. We start by placing a $d > 1$ dimensional GP prior on the unknown function $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ (Rasmussen and Williams, 2006)

$$f(x) \sim \mathcal{GP}(0, K(x, x')),$$

where $K(x, x') \in \mathbb{R}^{d \times d}$ is a kernel function. For any set of input points $X = [x_1, x_2, \ldots, x_N]^T \in \mathbb{R}^{N \times d}$, this multi-output GP prior defines a Gaussian distribution over function outputs $F = [f(x_1), f(x_2), \ldots, f(x_N)]^T \in \mathbb{R}^{N \times d}$:

$$p(F) = \mathcal{N}(\text{vec}(F)|0, K(X, X)),$$

where $K(X, X) = (K(x_i, x_j))_{i,j=1}^{N} \in \mathbb{R}^{ND \times ND}$ is a block matrix and vec($\cdot$) operator stacks the rows of its argument into a column matrix.

Our goal is to compute the posterior distribution over $f$ with the following likelihood model

$$x(t) = x_0 + \int_{0}^{t} f(x(\tau))d\tau$$

$$y_t = x(t) + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \Omega),$$

where $x(t)$ and $y_t$ denote a forward simulated state and observation at time $t$, and $\Omega \in \mathbb{R}^{d}$ is the observation noise parameter. As noted by Barber (2014),
a closed form expression for the state solutions $x(t)$ is available only if $x(t)$ follows a Gaussian process and $f$ is linear with respect to $x$. Consequently, the majority of existing approaches rely on approximate solutions and sampling state trajectories. In this work, we resort to the sparse GP framework by introducing a set of inducing points $Z = [z_1, z_2, \ldots, z_M]^T \in \mathbb{R}^{M \times d}$ that live in the same space as $x$, and corresponding output vectors $U = [u_1, u_2, \ldots, u_M]^T \in \mathbb{R}^{M \times d}$ with $u_m \equiv f(z_m)$. This approximation leads to a conditional distribution on the function value at any input point, which in turn facilitates explicit forward integration (3.3).

### 3.2.1 Time Differential Function

Due to the formal definition of ODEs, the time differential function in (3.3) needs to be continuous whereas we introduced a (conditional) stochastic process over the function, i.e., $p(f(x^*)|Z, U)$. To resolve this, we propose to discard the variance induced by the distribution $p(f(x^*)|Z, U)$:

$$f(x^*)|Z, U \triangleq K_\theta(x^*, Z)K_\theta(Z, Z)^{-1}\text{vec}(U),$$

where $\theta$ denotes the kernel parameters. We opt for the simplest identity decomposable kernel as our kernel function:

$$K_\theta(x, x') \triangleq k_\theta(x, x') \cdot I_d,$$

where $I_d$ denotes a $d \times d$ identity matrix. Diagonal elements of the kernel matrix are given by a the following scalar-valued squared exponential kernel, the de-facto standard kernel for GPs:

$$k_\theta(x, x') = \sigma_f^2 \exp \left( -\frac{1}{2} \sum_{j=1}^{d} \frac{(x_j - x'_j)^2}{\ell_j^2} \right)$$

where $\sigma_f$ is known as signal variance and $\ell_j$ denotes dimension-wise length-scale parameter, i.e., $\theta = \{\sigma_f, \ell_1, \ldots, \ell_d\}$. We name our model npODE, abbreviating non-parametric ODEs, and illustrate it in Figures 3.2 and 3.3. Please note that non-parametric refers to our model’s ability to approximate any ODE system without having access to its parameters or functional form of the time differential.

### 3.2.2 MAP Inference for npODE

In this section, we describe the inference task for a single input data sequence $Y \equiv y_{0:T}$ without loss of generality. As described in (3.3-3.4), the likelihood is given by a Gaussian observation model with mean being the forward simulated states $x(t)$. For simplicity, we assume a diagonal noise covariance $\Omega = \text{diag}(\omega_1, \ldots, \omega_d)$ to be learned. Due to the noisy observation
Figure 3.2. 2D illustration of npODE on Van der Pol oscillator (the figure is taken from Publication I). (a) Above panel demonstrates the model in the phase space, where each axis corresponds to an observed dimension. Black circles and arrows denote the inducing locations $Z$ and vectors $U$. Grey arrows in the background are computed by interpolating via (3.5). The red trajectory is computed by integrating the resulting system starting from the red circle. (b) The states plotted over time.

assumption, the initial value $x_0$ also appears as a free parameter. Overall, the posterior distribution becomes

$$p(Z, U, x_0, \theta, \Omega|Y) \propto p(Y|x_0, U, \Omega, Z, \theta)p(U|Z, \theta)p(Z, x_0, \theta, \Omega)$$

(3.6)

$$\propto \prod_{i=0}^{T} \mathcal{N}(y_i; x(t_i), \Omega) \mathcal{N}(\text{vec}(U)|0, K_\theta(Z, Z)),$$

(3.7)

where we assume a Gaussian process prior on the inducing vectors and non-informative priors for the remaining variables.

**Inducing locations** To simplify the learning, we propose to place inducing points $Z$ on a fixed and sufficiently dense grid. The grid locations are chosen to cover the training data sequences. We have experimentally observed that learning the locations complicates gradient descent based optimization
routines and usually leads to worse test performance compared to learning with a fixed grid.

**Whitened inducing vectors** As another measure to stabilize learning, we aim to reduce the correlation between inducing points by an invertible transformation known as whitening Kuss and Rasmussen (2005). Whitening relies on transforming the inducing variables using the Cholesky decomposition of the covariance matrix $L_\theta L_\theta^T = K_\theta(Z, Z)$ as follows:

$$
\text{vec}(U) = L_\theta \text{vec}(\tilde{U}), \quad \text{vec}(\tilde{U}) = L_\theta^{-1} \text{vec}(U).
$$

In turn, $\tilde{U}$ appears as the model parameters instead of $U$, and the prior distribution becomes $p(\tilde{U}) = N(0, I)$. The whitened variables $\tilde{U}$ can easily be projected on the kernel manifold $L_\theta$ to obtain the inducing vectors $U$.

**MAP estimation** Since full posterior inference of our model is difficult, we opt for computing the MAP estimates for the initial state $x_0$, whitened inducing vectors $\tilde{U}$, and noise variances $\Omega$ instead of inferring the full posterior:

$$
\arg\max_{x_0, \tilde{U}, \Omega} \sum_{i=0}^{T} \log N(y_i; x(t_i), \Omega) + \log N(\text{vec}(\tilde{U}); 0, I).
$$

Note that we choose the kernel hyperparameters $\theta$ by cross-validation, which helps alleviating the overfitting problem. The gradients of the above system are computed via the sensitivity equations (2.13). In all forward simulations, we use RK45 solver (Dormand and Prince, 1980).

### 3.3 Learning SDEs

We now turn out attention to the cases where the dynamics are inherently stochastic. Perhaps the simplest example of such cases is the flow of particles suspended in liquid, which is explained using the friction and random forces caused by random collisions of the particles (Langevin, 1908; Lemons and Gythiel, 1997). The randomness is typically expressed via the non-differentiable Brownian motion, making ODEs an unsuitable approach. Therefore, in this section, we extent npODE to handle such non-differentiable cases by means of stochastic differential equations (SDEs).

For conciseness, we first re-introduce the Itô integral which was defined in (2.2.2):

$$
x(t) = x_0 + \int_0^t f(x(\tau))d\tau + \int_0^t \sigma(x(\tau))dB_\tau.
$$

Above differential equation differs from the ODE definition (3.3) due to the second integral. Thus, we propose to build upon the npODE framework by
Gaussian Processes for Learning Unknown Differential Equations

Figure 3.3. We illustrate how a single inducing vector influences the resulting system on the Van der Pol oscillator. Across all panels, only the inducing vector shown by red vary. The gray streams denote the vector field given by the kernel interpolation (3.5), which substantially differ near the red arrow. This locally different behaviour immediately translates into unique blue state trajectories that are computed by forward simulating the ODE system by (3.3).
introducing a second set of inducing vectors to approximate the diffusion function \( \sigma(\cdot) \):

\[
\sigma(x^*)|Z, U \doteq k(x^*, Z) k(Z, Z)^{-1} U_\sigma. \tag{3.10}
\]

The new set of “diffusion” inducing vectors, \( U_\sigma = [u_1^\sigma, \ldots, u_M^\sigma] \in \mathbb{R}^M \) with
\[
u_m^\sigma \equiv \sigma(z_m),
\]
share the same inducing locations \( Z \) with the “drift” inducing vectors, which we rename as \( U_f \) for clarity. Observe that the proposed diffusion function estimator is one-dimensional; however, it can be easily extended to higher dimensions by defining multiple GPs as in previous section. We name our approach npSDE.

**MAP estimation** Similar to npODE, our goal is to compute the MAP estimate of the parameters \( U_f, U_\sigma \) and \( \Omega \):

\[
\arg \max_{U_f, U_\sigma, \Omega} p(U_f, U_\sigma, \Omega|Y) \propto p(Y|U_f, U_\sigma, \Omega), p(U_f)p(U_\sigma)p(\Omega)
\]

where the GP priors defined in (3.7) are placed over \( U_f \) and \( U_\sigma \), and a non-informative prior over \( \Omega \). Kernel hyperparameters \( \theta \) and inducing locations \( Z \) are chosen as described in the previous section (we drop them from the conditioning notation for conciseness). Finally, we opt for setting the initial value of the integral to the first observation: \( x_0 \equiv y_0 \). Note that this simple choice works well in practice only if the data contains little amount of noise. Alternatively, our framework can be extended to include the initial value \( x_0 \) as a parameter.

**Likelihood** Since each different realization of the stochastic process leads to a different state solution, our likelihood model should be expressed in terms of the state solution distributions \( p(x(t_i)|U_f, U_\sigma) \). Thus, we propose to learn the underlying system to induce state distributions with high expected likelihood:

\[
p(Y|U_f, U_\sigma, \Omega) = \prod_{i=0}^T \mathbb{E}_{p(x(t_i)|U_f, U_\sigma)}[\mathcal{N}(y_i|x(t_i), \Omega)].
\]

**Monte Carlo sampling** The state solutions \( p(x(t_i); U_f, U_\sigma) \) do not admit tractable closed-form expressions with nonlinear drift and diffusion functions. A simple yet efficient approximation can be obtained by Monte Carlo sampling:

\[
p(Y|U_f, U_\sigma, \Omega) \approx \prod_{i=0}^T \frac{1}{L} \sum_{l=1}^L \mathcal{N}(y_i|x_i^{(l)}, \Omega), \quad x_i^{(l)} \sim p(x(t_i)|U_f, U_\sigma) \tag{3.11}
\]

where we use the shorthand notation \( x_i \equiv x(t_i) \). State solution samples \( x_i^{(l)} \) can be obtained by first drawing a Brownian motion sample and then numerically integrating (3.9). We used the simplest Euler-Maruyama solver given in (2.11) for numerical integration. Note that above stochastic likelihood estimate turns out to be a kernel density estimator with
Gaussian bases. We experimentally observed that the simple Monte Carlo approximation to state solution distributions \( p(x(t_i); U, U_\sigma) \) works well in practice. Alternatively, the expectation can be approximated using the sequential importance sampling method presented in Sarkka (2006) if the diffusion is constant or using more complicated particle based MCMC techniques described in Botha et al. (2021).

**Gradient computation** The gradient of the Monte Carlo approximated log-likelihood (3.11) is

\[
\frac{d}{dU} \sum_{i=0}^{T} \sum_{l=1}^{L} \log \frac{1}{L} \sum_{l=1}^{L} \mathcal{N}(y_i|x_i^{(l)}, \Omega) = \sum_{i=0}^{T} \sum_{l=1}^{L} \frac{\partial \mathcal{N}(y_i|x_i^{(l)}, \Omega)}{\partial x} \frac{dx_i^{(l)}}{dU} \sum_{l=1}^{L} \mathcal{N}(y_i|x_i^{(l)}, \Omega),
\]

(3.12)

where \( x_i^{(l)} \) follows numerical integration and \( U \) represents either set of inducing vectors. Above, the only difficult term \( \frac{dx_i^{(l)}}{dU} \) can be computed by slightly modifying forward sensitivities, see Publication II for a detailed description. Also, note that modern automatic differentiation frameworks conveniently utilize the same Brownian motion (or random numbers, in general) during the forward and backward pass, which leads to accurate gradients.

To sum up, our npODE and npSDE rely on four main principles: estimating the unknown functions with black-box function approximators, MAP estimation of the parameters, explicit integration to compute state solutions, and sensitivity equations for gradient computation. Unlike the npODE approach, in which whitened transformations of inducing points appear as parameters, we choose simplify npSDE framework by optimizing the inducing values themselves. Finally, npSDE integration starts from the initial observation \( y_0 \) while npODE treats the initial values as unknowns.

### 3.4 Experiments with npODE and npSDE

In this section, we illustrate our non-parametric differential equation systems on simulated and real data experiments. For simulated experiments, we consider three well-known ODE systems and one SDE system whose details are presented in the respective subsections. Our goal is to show that black-box modeling of parametric systems leads to accurate estimation of the dynamics. The real data experiment is conducted on a benchmark dataset of human motion capture data from the Carnegie Mellon University motion capture (mocap) database \(^1\). Since no parametric differential equation exists for human motion, this experiment demonstrates how our model can estimate realistic, unknown dynamics from noisy observations. Please note that additional results can be found in Publication I and II.

\(^1\)The data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.
3.4.1 Simulated ODE Systems

We consider three simulated differential systems: the Van der Pol (VDP), FitzHugh-Nagumo (FHN) and Lotka-Volterra (LV) oscillators of the following form:

\begin{align*}
\text{VDP} & : \quad \dot{x}_1 = x_2 \\
& \quad \dot{x}_2 = (1 - x_1^2)x_2 - x_1 \\
\text{FHN} & : \quad \dot{x}_1 = 3(x_1 - \frac{x_1^3}{3} + x_2) \\
& \quad \dot{x}_2 = \frac{0.2 - 3x_1 - 0.2x_2}{3} \\
\text{LV} & : \quad \dot{x}_1 = 1.5x_1 - x_1x_2 \\
& \quad \dot{x}_2 = -3x_2 + x_1x_2.
\end{align*}

We generate five training sequences of length 25 for each oscillator. Each VDP and FHN trajectory contains one cycle whereas LV trajectories make up 1.7 cycles. We also add observation noise with variance $\sigma^2_n = 0.1^2$. The inducing points are placed on a $6 \times 6$ fixed grid. Figure 3.4 demonstrates our findings. We observe that npODE is able to accurately estimate the vector field, especially near the data points. Consequently, it can successfully forecast up to 8 future cycles.

3.4.2 Simulated SDE System

We now illustrate the merits of our npSDE on a simulated SDE system in which the drift function is given by the Van der Pol equations and the diffusion is proportional to a Gaussian density. The true and unknown
vector field and the diffusion “hot-spot” is illustrated in Figure 3.5a. The blue contour plot in (b) represents the state density \( p(x(t); U_f, U_d) \) at \( t = 7.2 \). (c) The same trajectory samples plotted over time. Blue curves denote the marginal distribution of each state component at \( t = 7.2 \). (d) Estimated drift and diffusion functions. (e) Three noisy input trajectories for training (red) and forward simulated paths from the estimated model (black). (f) Data trajectories (red) and estimated path samples plotted over time (black) along with the same state marginals in (c) (blue). Note that the figure is taken from Publication II.

### 3.4.3 Motion Capture Experiments

Next, we evaluate our model on a benchmark real-world dataset of walking sequences. The dataset consists of 50-dimensional pose measurements from walking people, where each measurement comes from a sensor attached to subjects’ bodies (Wang et al., 2008). After pre-processing the data as described in Wang et al. (2008), we obtain a training dataset of 43 trajectories, each containing roughly 100 measurements. Following Wang et al. (2006) and Damianou et al. (2011), the dataset is projected into a three dimensional latent space by principle component analysis (PCA), which helps eliminating the highly-correlated features.

For ODE learning task, PCA embedding of each training sequence is fitted separately since walking sequences follow subject-specific styles.
Figure 3.6. An illustration of npODE and competing methods on mocap experiment (the figure is taken from Publication I). (a) Three dimensional latent space in which dynamics learning is performed. Black points represent the training data and colored lines are future forecasts. (b) Latent sequences are reconstructed into the original fifty dimensional space via inverse PCA mapping. The grey region shows the training data.

We use the first half of a sequence for training and cross validation, and compute the prediction error on the second half. As baselines, we consider GP based state space models GPDM (Wang et al., 2006) and variational GPLVM (Damianou et al., 2011). Overall, our method’s root mean squared error (RMSE) on future time points is 4.52, while GPDM and VGPLVM achieve 4.94 and 8.74 RMSE, respectively. A comparison of the three methods on a single sequence is presented in Figure 3.6.

To demonstrate npSDE on real-world data scenarios, we follow the same experiment setup with Wang et al. (2008), and collect four walking sequences from different subjects into a single dataset. Our goal is to learn a drift function that models the walking dynamics and a diffusion function that explains the discrepancies among walking styles. We visualize the inferred drift fit and the density of the sample paths in Figure 3.7. We conclude that our model is capable of learning drift and diffusion functions that match arbitrary data sequences.
Figure 3.7. npSDE model fits on mocap data (the figure is taken from Publication II). (a) Shared drift estimate learned from walking data of four subjects. (b) Estimated sample paths. (c) Density plots of the sample paths. Four observed trajectories are shown as black lines in (b-c), with red circles denoting the initial state.
4. Continuous-Time Bayesian Modelling for High Dimensional Sequences

In the previous chapter, we explained how continuous-time phenomena can be learned by means of Gaussian processes. As demonstrated by the simulated system experiments, the framework is capable of inferring unknown systems given noisy state measurements. Nevertheless, the model suffers from certain limitations which we aimed to address in ODE²VAE, the method proposed in our Publication III. In the next section, we examine the limitations of npODE, which naturally motivate the ideas behind ODE²VAE. We then describe our proposed variational framework that relies on variational auto-encoders and Bayesian neural networks. The chapter is concluded with the experimental evaluations.

4.1 Limitations of npODE

Since the methodologies presented in the previous chapter are analogous, the discussion in this section is limited to npODE (while the same arguments apply to npSDE as well). In the following, we describe five shortcomings of npODE. A complete list of differences between npODE and ODE²VAE is given in Table 4.1.

4.1.1 Modeling Space

npODE dynamics specified in (3.3-3.4) are chosen to live in the same space as the observations. Nonetheless, as opposed to the model description, walking dynamics were learned in a 3D latent space onto which the dataset was projected by PCA. npODE's success on this task can be partly attributed to the fact that sensor measurements are highly correlated, making a linear projection sufficient for our purposes. However, in more complicated problems, linear mappings may not reflect the underlying structure in the data. Lawrence and Hyvärinen (2005) indeed show that a non-linear variant of PCA, known as Gaussian process latent variable model, better structure the latent space, which in turn leads to more
Table 4.1. A summary of the differences between npODE and ODE\textsuperscript{2}VAE.

<table>
<thead>
<tr>
<th></th>
<th>npODE</th>
<th>ODE\textsuperscript{2}VAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling space</td>
<td>Data space</td>
<td>Latent space</td>
</tr>
<tr>
<td>Dynamics approximator</td>
<td>Gaussian process</td>
<td>Bayesian neural net.</td>
</tr>
<tr>
<td>Inference</td>
<td>MAP inference</td>
<td>Variational inference</td>
</tr>
<tr>
<td>Order of the dynamics</td>
<td>First</td>
<td>Second</td>
</tr>
<tr>
<td>Gradient computation</td>
<td>Sensitivity equations</td>
<td>Adjoint method</td>
</tr>
<tr>
<td>Data modality</td>
<td>States</td>
<td>States/images</td>
</tr>
</tbody>
</table>

interpretable embeddings and much smaller latent classification error.

4.1.2 Learning Second-Order Systems

Classical mechanics describes the motions of everyday objects by the language of differentials and integrals. Not only the simplest mechanics formulations, e.g. Newton’s laws, but also more sophisticated ones such as Lagrangian and Hamiltonian mechanics rely on higher order derivatives. Without diving into the details of such formulations, we define a second-order ODE system in terms of position \( s(t) \) and velocity \( v(t) \) components:

\[
\begin{align*}
\dot{s}(t) &= \frac{ds(t)}{dt} = v(t) \\
\dot{v}(t) &= \frac{dv(t)}{dt} = f(s(t), v(t), u(t), t, C)
\end{align*}
\]

where the equations represent velocity and acceleration, respectively. Being the time derivative of velocity, acceleration (4.2) is generally an arbitrary function of time \( t \), position \( s(t) \), velocity \( v(t) \), force \( u(t) \), and possibly other system-specific variables \( C \) such as mass and friction. We conjecture that above formulation is more suitable for modeling systems that are expressed via the laws of motion.

4.1.3 Scalability of the Dynamics

Being a kernel method, sparse Gaussian processes compute function outputs as a locally weighted average of other data points, or inducing values. When the kernel function is stationary, as the squared exponential kernel used in npODE, learning non-stationarities becomes too difficult. The problem may be relieved by placing a large number of inducing points near the non-stationary regions; however, npODE framework assumes a fixed grid, and knowing such regions rather than inferring would be a restrictive assumption. Second, kernel values quickly diminish as the state dimensionality increases, famously known as curse of dimensionality.
This necessitates a growing set of inducing points, which would in turn significantly slow down the computations.

### 4.1.4 Overfitting

Learning in npODE framework relies on MAP estimation instead of marginalizing out the unknown quantities. Since the marginal likelihood is not available, choosing the correct model complexity (without overfitting) was attained by cross-validation, which requires repeating the optimization procedure several times from scratch. Furthermore, MAP estimates consist of likelihood and prior terms, which are responsible for data fitting and regularization, respectively. Nonetheless, npODE posterior distribution (3.6) does not place a prior on the inducing locations and initial value, making the model vulnerable to overfitting.

### 4.1.5 Scalability of the Optimization

The gradient of npODE loss was computed using forward sensitivities (2.12). The method requires solving a $dP$-dimensional ODE system, where $d$ and $P$ denote the state dimensionality and number of parameters in the differential function. Therefore, forward sensitivities can handle systems with $P \leq 500$ (Rackauckas et al., 2018). Scalability was not a problem in the context of npODEs since $P$ typically varies in [50, 300]. However, with larger scale problems and over-parameterized time differential approximations such as neural networks, gradient computation becomes a huge computational bottleneck.

### 4.2 Proposed Model

Building upon the observations in the previous section, we propose to infer continuous-time, second-order latent trajectories that live in a smaller space. For this, we consider a generative model with three components:

(i) Second order dynamics defined by an acceleration field

(ii) A distribution for the initial position $p(s_0)$ and velocity $p(v_0)$ in the latent space

(iii) A likelihood model $p(x|s)$ that maps latent positions into high-dimensional observations,

which can be written more concretely as follows:

\[ s_0 \sim p(s_0) \]
\[ v_0 \sim p(v_0) \]
\[
\begin{align*}
\mathbf{s}_t &= \mathbf{s}_0 + \int_0^t \mathbf{v}_\tau d\tau \\
\mathbf{v}_t &= \mathbf{v}_0 + \int_0^t f_{\text{true}}(s_\tau, \mathbf{v}_\tau) d\tau \\
x_i &\sim p(x_i|s_i) \quad i \in [0, T]
\end{align*}
\]

As in npODE, we consider a parametric approximation to the unknown latent differential function \(f_{\text{true}}\). Due to the scalability issues discussed before, we opt for a neural network (NN) approximation instead of GPs. As in other generative models, our primary optimization objective is to compute the marginal likelihood by integrating out the latent position and velocity variables. With non-linear time differential or likelihood functions, marginalization becomes intractable and we resort to variational approximation. In particular, we turn our attention to VAE, which is proven useful for learning highly complicated latent mappings (Kingma and Welling, 2014; Rezende et al., 2014), and Bayesian neural networks.

**Variational auto-encoder** We first shortly remind the idea behind VAE (please see Subsection 2.3.1 for a more rigorous treatment). Given a data point \(x\) and an intractable posterior density \(p(z|x)\) over the corresponding latent variable \(z\), the encoder objective is to approximate the posterior with a variational family \(q_\psi(z|x)\) while the decoder learns the inverse mapping \(p_\gamma(x|z)\), which corresponds to the likelihood model. Typically, both the encoder and decoder are parameterized by deep neural networks (usually CNNs in image based applications and MLPs otherwise).

Although VAE is a rather recently popularized tool presented mainly in the context of variational inference, encoder-decoder type architectures have been known for decades, e.g., a non-probabilistic VAE variant called autoassociator was first proposed for dimensionality reduction (Cottrell, 1985). In the context of latent ODE modeling, we utilize the encoder to extract a latent representation only for the initial value of a trajectory while the decoder maps all forward simulated latent states into data space.

**Bayesian neural networks** As mentioned in the previous section, npODE lacks a principled approach to deal with overfitting. When the GP time differential approximation is replaced with a NN, the resulting model would still be susceptible to overfitting since vanilla multi-layer perceptron described in Section 2.1.1 provides no means for handling uncertainty. Therefore, we choose to treat the NN parameters \(W\) as random variables, which in turn corresponds to modeling unknown dynamics with a BNN \(f_W(s_t, \mathbf{v}_t)\).
4.2.1 Approximate Posterior

We now describe our variational framework from the first principles. For notational clarity, we combine the latent position and velocity components into a single vector \( z_t := (s_t, v_t) \) and propose the following approximation for the unknown quantities:

\[
q(W, z_{0:T} | x_{0:T}) = q(W)q_{\text{enc}}(z_0 | x_{0:T})q_{\text{ode}}(z_{1:T} | z_0, W).
\] (4.3)

The first term on the rhs defines the mean-field approximate posterior for the dynamics approximation:

\[
q(W) = \mathcal{N}(W | \mu, sI).
\]

The second term is based on the amortized VI idea of VAE, where \( q_{\text{enc}} \) is given by NN encoders. Similar to original VAE formulation (Kingma and Welling, 2014), latent representations have a diagonal covariance matrix:

\[
q_{\text{enc}}(z_0 | x_{0:T}) \equiv q_{\text{enc}} \left( \begin{array}{c} s_0 \\ v_0 \end{array} \mid x_{0:T} \right) \\
\mathcal{N} \left( \begin{array}{c} \mu_s(x_0) \\ \mu_v(x_{0;m}) \end{array} \right), \begin{pmatrix} \text{diag}(\sigma_s(x_0)) & 0 \\ 0 & \text{diag}(\sigma_v(x_{0;m})) \end{pmatrix},
\]

where \( \mu_s, \mu_v, \sigma_s, \sigma_v \) are the encoding NNs. Note that the latent position representation is computed conditioned only on the initial data point while the higher-order velocity information is extracted from the first \( 1 < m \leq T \) data points.

**Instantaneous change of variables** The last term in (4.3) describes how the density evolves as the initial latent point \( z_0 \) is forward integrated. For this, we first express our second-order system as a first order ODE as described in Section 2.2.1:

\[
z_N = z_0 + \int_0^{t_N} \hat{f}_W(z_\tau) d\tau
\]

\[
\hat{f}_W(z_t) \equiv \begin{pmatrix} v_t \\ f_{WV}(s_t, v_t) \end{pmatrix}.
\]

Using the instantaneous change of variable theorem in Chen et al. (2018a), we obtain the following expression for the evolution of latent density:

\[
\frac{\partial \log q(z_t | W)}{\partial t} = -\text{Tr} \left( \frac{d\hat{f}_W(z_t)}{dz_t} \right) = -\text{Tr} \left( \frac{\partial v_t}{\partial s_t} \frac{\partial f_{WV}(s_t, v_t)}{\partial s_t} + \frac{\partial v_t}{\partial v_t} \frac{\partial f_{WV}(s_t, v_t)}{\partial v_t} \right) \\
= -\text{Tr} \left( \frac{\partial f_{WV}(s_t, v_t)}{\partial v_t} \right),
\]
which results in the log densities over time

$$\log q(z_N|z_0, W) = \log q(z_0|W) - \int_0^{t_N} \text{Tr} \left( \frac{\partial f_W(s_t, v_t)}{\partial v_t} \right) dt.$$  

The model components are visualized in Figure 4.1. Note that we set $T = m = 3$ in this illustration. The first three decoded frames match the input sequence while the following four frames are unseen.

### 4.2.2 Evidence Lower Bound

In this section, we derive the ELBO and show how the loss term can be augmented with additional terms our model benefits from. For brevity, we denote $X = x_{0:T}$ and $Z = z_{0:T}$, and derive the evidence lower bound as follows (Blei et al., 2016):

$$\log p(X) \geq -KL[q(W, Z|X)||p(W, Z)] + \mathbb{E}_{q(W, Z|X)}[\log p(X|W, Z)]$$

$$= -\mathbb{E}_{q(W, Z|X)} \left[ \log \frac{q(W)q(Z|W, X)}{p(W)p(Z)} \right] + \mathbb{E}_{q(W, Z|X)}[\log p(X|W, Z)]$$

$$= -KL[q(W)||p(W)] + \mathbb{E}_{q(W, Z|X)} \left[ -\log \frac{q(Z|W, X)}{p(Z)} + \log p(X|W, Z) \right]$$

$$= -KL[q(W)||p(W)] + \mathbb{E}_{q\text{enc}(z_0|X)} \left[ -\log \frac{q_{\text{enc}}(z_0|X)}{p(z_0)} + \log p(x_0|z_0) \right]$$
\[ + \sum_{i=1}^{T} \mathbb{E}_{q_{\text{enc}}(z_0|X)} \mathbb{E}_{q(W)} \left[ \mathbb{E}_{q_{\text{ode}}(z_i|z_0, W)} \left[ - \log \frac{q_{\text{ode}}(z_i|W, z_0)}{p(z_i)} \right] \right] \]

\text{dynamic regularization}

\[ + \sum_{i=1}^{T} \mathbb{E}_{q_{\text{enc}}(z_0|X)} \mathbb{E}_{q(W)} \left[ \mathbb{E}_{q_{\text{ode}}(z_i|W, z_0)} \left[ \log p(x_i|z_i) \right] \right] , \]

\text{reconstruction}

where we set the prior distribution \( p(z_t) \) to isotropic Gaussian distribution. A state sample \( z_i^{(l)} \sim q_{\text{ode}}(z_i|z_0, W) \) is drawn by first sampling from the approximate posteriors and then forward simulating the dynamics model:

\[ W^{(l)} \sim q(W) \]
\[ z_0^{(l)} \sim q_{\text{enc}}(z_0|X) \]
\[ z_i^{(l)} = z_0^{(l)} + \int_0^{t_i} \hat{f}_{W^{(l)}}(z_t^{(l)}) d\tau. \]

The first term in ELBO (4.4) regularizes the dynamics model, and the second term is identical to original VAE bound. The third term penalizes complex latent state distributions while the last term is responsible for mapping the states into observations. We maximize the ELBO with respect to dynamics, encoder and decoder parameters. We compute the ODE regularization term in closed form while the other expectations are approximated by Monte Carlo sampling.

**Penalized loss** A well-known pitfall of VAE models is that optimizing the ELBO objective does not necessarily result in accurate inference (Alemi et al., 2018). Borrowing the ideas from Higgins et al. (2017), we weight the \( \text{KL}[q(W)||p(W)] \) term resulting from the BNN with a constant factor \( \beta \). We choose to fix \( \beta \) to the ratio between the latent space dimensionality and number of BNN parameters, \( \beta = q/|W| \), in order to counter-balance the penalties on latent variables \( W \) and \( z_i \).

**Augmented loss** Our variational model utilizes encoders only for obtaining the initial latent distribution. When the input sequences are long, encoder’s contribution to ELBO would be much smaller compared to dynamic regularization and reconstruction terms, which are summations over time. Similarly, near-fixed initial distributions \( p(z_0) \) would reduce the effective number of data points the encoder is trained with. To overcome these issues, we augment ELBO with a term that matches the encoder distribution and the distribution induced by the ODE flow, leading to the
Continuous-Time Bayesian Modelling for High Dimensional Sequences

following penalized optimization objective:

\[
\mathcal{L}_{\text{ODE}^2\text{VAE}} = -\beta \text{KL}[q(W)||p(W)] + \\
\mathbb{E}_{q(W,Z|X)} \left[ -\log \frac{q(Z|W,X)}{p(Z)} + \log p(X|W,Z) \right] + \\
-\gamma \mathbb{E}_{q(W)} \left[ \text{KL}[q_{\text{ode}}(Z|X)||q_{\text{enc}}(Z|W,X)] \right].
\] (4.5)

We choose the constant \(\gamma\) by cross-validation. In practice, we found out that an annealing scheme in which \(\gamma\) is gradually increased helps optimization, which is also used in (Karl et al., 2016; Rezende and Mohamed, 2015).

**Gradient computation** Being an alternative to forward sensitivities, adjoint method (2.14-2.15) efficiently computes the loss gradients using Jacobian-vector products. In ODE\(^2\)VAE, we utilize the `torchdiffeq` package that implements adjoints method (Chen et al., 2018b).

### 4.3 Experiments

The model is evaluated on three different datasets: Motion capture, rotating MNIST (Casale et al., 2018) and bouncing balls (Sutskever et al., 2009).

- **Motion capture datasets:** In addition to the dataset described in Section 3.4.3, we consider a second dataset consisting of 23 walking sequences of subject 35 (Gan et al., 2015), which is partitioned into 16 training, 3 validation and 4 test sequences.

- **Rotating MNIST:** We repeat the experiment in Casale et al. (2018) by constructing a dataset by rotating the images of handwritten “3” digits (16 rotation angles). Overall, the dataset consists of 360 training and 40 validation sequences of 28 \(\times\) 28 frames. We furthermore randomly remove four rotation angles from each sequence to introduce non-uniformity and missing data.

- **Bouncing balls:** This experiment concerns learning the physical laws of particle collision. For this, we simulate three balls bouncing within a frictionless rectangular box and also colliding with each other using the implementation provided with Sutskever et al. (2009). The dataset consists of video frames of 10000 training and 500 test sequences of length 20, where each frame is 32x32 and pixel values vary between 0 and 1.

In all experiments, our method consistently outperforms the discrete transition baselines. We furthermore highlight different merits of ODE\(^2\)VAE in the subsequent subsections.
Table 4.2. Average MSE on future frames on mocap datasets

<table>
<thead>
<tr>
<th>Model</th>
<th>Mocap-1</th>
<th>Mocap-2</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPDM</td>
<td>126.46 ± 34</td>
<td>N/A</td>
<td>Wang et al. (2006)</td>
</tr>
<tr>
<td>VGPLVM</td>
<td>142.18 ± 1.92</td>
<td>N/A</td>
<td>Damianou et al. (2011)</td>
</tr>
<tr>
<td>NPODE</td>
<td>45.74</td>
<td>22.96</td>
<td>Heinonen et al. (2018)</td>
</tr>
<tr>
<td>Neural ODE</td>
<td>87.23 ± 0.02</td>
<td>22.49 ± 0.88</td>
<td>Chen et al. (2018b)</td>
</tr>
<tr>
<td>ODE(^2)VAE</td>
<td><strong>15.99 ± 4.16</strong></td>
<td><strong>8.09 ± 1.95</strong></td>
<td>Publication III</td>
</tr>
</tbody>
</table>

4.3.1 Robustness against Overfitting

We compare our model against the methods considered in Section 3.4.3 and vanilla latent neural ODE (Chen et al., 2018b) on mocap datasets. As previously, GPDM, VGPLVM and npODE learn the dynamics in a latent space induced by the PCA mapping. In contrast, a VAE governs the neural ODE latent space similar to our method. The results are presented in Table 4.2. Most strikingly, ODE\(^2\)VAE is superior to other methods (including npODE) on both datasets, justifying our transition from GP-based differential equation modeling to latent neural ODEs. Secondly, while all models achieve smaller test error with more data (second column), the improvement is more drastic for npODE and neural ODE. We conjecture that ML/MAP estimation based parameter inference of these methods leads to overfitting in low-data regimes whereas our variational inference framework is more robust.

4.3.2 Generalization to Unseen Data Points

In the rotating MNIST experiment, we aim to illustrate how our model generalizes to interpolate an unseen rotation angle from a sequence of rotating digits. The first rows of Figure 4.2a and 4.2b demonstrate an example training sequence with missing frames and the left-out test angle, respectively. We contrast our method against Gaussian process prior VAE (GPPVAE) (Casale et al., 2018), which replaces the commonly used iid Gaussian prior with a GP and thus performs latent regression. GPPVAE achieves a test mean squared error (MSE) of 0.0309 on the unseen test angle while our ODE\(^2\)VAE reaches a smaller test error (0.0188). For a visual illustration of a rotation sequence and different hand-writing styles ODE\(^2\)VAE generates, please see Figure 4.2.
Figure 4.2. Panel (a) shows a training sequence with missing values (first row) and its reconstruction (second row). First row in panel (b) demonstrates test angles from different sequences, i.e., hand-writing styles, and below are model predictions (the figure is taken from Publication III).

Figure 4.3. Bouncing balls test errors, taken from Publication III.

4.3.3 The Virtue of Second-Order Modeling

The competing techniques in our largest-scale bouncing ball experiments are deep temporal sigmoid belief networks (DTSBN-S) (Gan et al., 2015) and decompositional disentangled predictive auto-encoder (DDPAE) (Hsieh et al., 2018). Both methods assume discrete transitions, learn the dynamics in a latent space and performs the same experiment on the respective papers. As shown by the test MSEs in Figure 4.3, our method achieves a much smaller error.

To further investigate ODE$^2$VAE modeling choices, we test two variants of our model on the same dataset, where the first variant approximates the dynamics with a NN, and the second variant has the latent space governed by a first-order ODE system (thus named ODE$^1$VAE). The test MSEs are given in Table 4.3. As expected, NN dynamics approximation leads to comparable performance since the large number of training sequences prevent the model from overfitting. More interestingly and perhaps not surprisingly, second-order dynamics help with more accurate future predictions, which we attribute to the fact that the underlying physical system is defined in terms of higher order dynamics.
Table 4.3. Comparison of neural network (NN) and Bayesian neural network (BNN) ODE’s with different latent dimensionalities on BOUNCING BALL experiment. Adding 2nd order momentum achieves better performance, while BNN's have a smaller impact.

<table>
<thead>
<tr>
<th>Model</th>
<th>Latent dimensions $d$</th>
<th>Test MSE</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st-order state</td>
<td>2nd-order momentum</td>
<td>NN</td>
<td>BNN</td>
</tr>
<tr>
<td>ODE$^1$VAE</td>
<td>25</td>
<td>-</td>
<td>45</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>-</td>
<td>36</td>
<td>35</td>
</tr>
<tr>
<td>ODE$^2$VAE</td>
<td>25</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
</tbody>
</table>
5. SDEs for Stochastic Asynchronous Non-convex Optimization

In the previous chapters, we have described how unknown differential equation systems can be learned using GPs and NNs, which in turn allows accurate long-term predictions. We also demonstrated how ODEs can be coupled with VAEs to handle high dimensional sequences. Thus, the line of research so far has focused only on learning dynamical systems.

We now turn our attention to another problem involving differential equations, namely, stochastic optimization. As briefly touched upon in Section 2.3.2, discretization of SDE systems with carefully crafted drift and diffusion functions leads to stochastic gradient Markov chain Monte Carlo (SGMCMC) algorithms. As an example, we saw that the simplest SGMCMC algorithm, stochastic gradient Langevin dynamics (SGLD), generates samples from a target density by Euler-Maruyama discretization of an SDE system. In an algorithmic sense, SGLD is identical to stochastic gradient descent (SGD), except that it injects a Gaussian noise at each iteration. As shown later, this noise term is the bridge connecting SGMCMC methods (hence SDEs) to optimization, which was also observed in a series of works by Dalalyan (2017); Raginsky et al. (2017); Zhang et al. (2017).

In a related study, Şimşekli et al. (2016a) proposed Hessian approximated MCMC (HAMCMC), an L-BFGS-based SGLD algorithm that significantly speeds up the convergence thanks to the use of second-order Hessian information. Nonetheless, since the standard L-BFGS algorithm requires gradients to be computed on the whole dataset, large scale applications of L-BFGS are quite limited. As a remedy, Berahas et al. (2016) proposed a parallel stochastic L-BFGS algorithm, called multi-batch L-BFGS (mb-L-BFGS). The algorithm relies on a synchronized communication between a master node and all workers, each of which operates on a mini-batch of data. This work illustrated that carrying out L-BFGS in a distributed setting introduces further theoretical and practical challenges; however, if these challenges are addressed, stochastic L-BFGS can be powerful in a distributed setting as well.

Although synchronous parallel algorithms have clear advantages over serial optimization algorithms, the computational efficiency of synchronous
algorithms is often limited by the overhead induced by the synchronization and coordination among the worker processes. Inspired by asynchronous parallel stochastic optimization techniques (Agarwal and Duchi, 2011; Lian et al., 2015; Zhang et al., 2015; Zhao and Li, 2016; Zheng et al., 2017), we propose an asynchronous parallel stochastic L-BFGS algorithm for large-scale non-convex optimization problems. Our proposed approach aims at speeding up the synchronous algorithm presented in Berahas et al. (2016) by allowing all the workers work independently from each other and circumvent the inefficiencies caused by synchronization and coordination. We present a taxonomy of the said methods in Figure 5.1.

In the following section, we describe the problem and challenges of stochastic L-BFGS, followed by how tempering bridges the gap between optimization and sampling. Next, we explain our proposed SDE system that generates samples from a target density and show how this SDE can be adapted to a distributed optimization setting. We also repeat the main theoretical finding of our Publication IV for completeness. Section 5.3 details our large scale matrix factorization experiment. Please note that the following sections only aim to convey the ideas, and we refer the reader to Publication IV for the proofs and more detail.

5.1 Background

The main problem of interest in this chapter is MAP estimation, which was first presented in the context of numerical optimization (Section 2.3.3). As previously, we are given a set of \( N \) data points \( D \equiv \{x_1, \ldots, x_N\} \) and \( \theta \in \mathbb{R}^P \) denotes our model parameters. \( \theta^* \) stands for the optimum of the posterior density \( p(\theta|D) \propto \exp(-U(\theta)) \), where \( U(\theta) \) is often called the potential energy function. A stochastic unbiased estimate of the gradient
\nabla U(\theta) \text{ is given by }

\nabla \hat{U}(\theta) = - \left[ \nabla \log p(\theta) + \frac{N}{N_\Omega} \sum_{i \in \Omega} \nabla \log p(x_i | \theta) \right], \quad (5.1)

where \( \Omega \subset \{1, \ldots, N\} \) denotes a random data subsample that is drawn with replacement and \( N_\Omega = |\Omega| \) is the cardinality of \( \Omega \).

### 5.1.1 Stochastic L-BFGS

The L-BFGS algorithm iteratively applies the following equation in order to find the MAP estimate given in (5.1):

\[ \theta_n = \theta_{n-1} - h H_n \nabla U(\theta_{n-1}) , \]

where \( n \) denotes the iterations and \( h \) is the step size. Here, \( H_n \) is an approximation to the inverse Hessian at \( \theta_{n-1} \) and is computed by using the \( M \) past values of the “iterate differences” \( s_n \triangleq \theta_n - \theta_{n-1} \), and “gradient differences” \( y_n \triangleq \nabla U(\theta_n) - \nabla U(\theta_{n-1}) \) (Nocedal and Wright, 2006). The collection of the iterate and gradient differences is called the L-BFGS memory.

In order to achieve computational scalability, stochastic L-BFGS algorithms replace \( \nabla U \) with \( \nabla \hat{U} \). This turns out to be problematic since the gradient differences \( y_n \) would be inconsistent, meaning that the stochastic gradients in different iterations will be computed on different data sub-samples, i.e. \( \Omega_{n-1} \neq \Omega_n \). Furthermore, in the presence of the stochastic gradients, L-BFGS is no longer guaranteed to produce positive definite approximations even in convex problems, therefore more considerations should be taken in order to make sure that \( H_n \) is positive definite.

### 5.1.2 Tempered SGLD

The connection between SGLD and SGD can be clearly seen by the following tempered version of SGLD:

\[ d\theta_t = \nabla U(\theta_t) dt + \sqrt{2/\beta} W_t , \]

where \( W_t \) denotes the standard Brownian motion in \( \mathbb{R}^P \). Under mild regularity conditions on \( U \), the solution process \( (\theta_t)_{t \geq 0} \) attains a unique stationary distribution with a density that is proportional to \( \exp(-\beta U(\theta)) \) (Roberts and Stramer, 2002). An important property of this distribution is that, as \( \beta \) goes to infinity, this density concentrates around the global maximum of \( U(\theta) \) (Hwang, 1980; Gelfand and Mitter, 1991). Therefore, for large enough \( \beta \), a random sample that is drawn for the stationary distribution of \( (\theta_t)_{t \geq 0} \) would be close to \( \theta^* \).

Building upon this property, Chen et al. (2016) developed an annealed SGMCMC algorithm for non-convex optimization, which was later extended by Ye et al. (2017). Raginsky et al. (2017) and Xu et al. (2018)
provided finite-time guarantees for SGLD to find an approximate global minimizer that is close to $\theta^\star$. Yet, none of these SGMCMC-based techniques describes how to incorporate Hessian information within an asynchronous framework, which we aim to answer in the following.

### 5.2 Asynchronous Stochastic L-BFGS

In this section, we describe our asynchronous L-BFGS-based (tempered) SGMCMC algorithm that aims to provide an approximate optimum that is close to $\theta^\star$ by generating samples from a distribution that has a density that is proportional to $\exp(-\beta U(\theta))$. For this, we carefully choose the matrices in (2.21-2.22) so that the resulting system

- utilizes the Hessian information
- allows for tempering to reach the optimum
- becomes computationally feasible and suitable for asynchronous settings

We call the proposed algorithm asynchronous parallel stochastic L-BFGS (as-L-BFGS).

#### 5.2.1 Designing the Correct SDE

Similar to Hamiltonian Monte Carlo sampler presented in Section 2.3.2, we consider an augmented setting $X_t = [\theta_t^T, p_t^T]^T \in \mathbb{R}^{2P}$ where $p_t \in \mathbb{R}^P$ is called the momentum variable. Given an energy function of the form

$$ E(X) \triangleq \beta U(\theta) + \frac{\beta}{2} p^T p, $$

the following SDE has a stationary distribution proportional to $\exp(-E(X))$:

$$
\begin{align*}
  dX_t &= \left\{-\begin{pmatrix}
0 & 0 \\
0 & \frac{\gamma}{\beta} I \\
\end{pmatrix} + \begin{pmatrix}
0 & -\frac{H_t(\theta_t)}{\beta} \\
\frac{H_t(\theta_t)}{\beta} & 0 \\
\end{pmatrix}
\end{pmatrix}
\begin{pmatrix}
\beta \nabla \theta U(\theta_t) \\
\beta p_t \\
\nabla_X E(X_t) \\
\Gamma_t(X_t) \\
\end{pmatrix}
  + \begin{pmatrix}
0 \\
\frac{1}{\beta} \Gamma_t(\theta_t) \\
\end{pmatrix}
\right\} dt \\
  &\quad + \sqrt{2D} dw_t, \\
\end{align*}
$$

(5.2)

where $\Gamma_t(X)$ was defined in (2.22). Here, $D$ is positive semi-definite and $Q$ is skew-symmetric as required in Section 2.3.2. Extending the terms in (5.2) yields the following simplified SDE system:

$$
\begin{align*}
  dp_t &= \left[\frac{1}{\beta} \Gamma_t(\theta_t) - H_t(\theta_t) \nabla \theta U(\theta_t) - \gamma p_t\right] dt + \sqrt{\frac{2\gamma}{\beta}} dw_t \\
  d\theta_t &= H_t(\theta_t) p_t dt
\end{align*}
$$

(5.3)
with

$$\left[ \Gamma_t(\theta) \right]_i \triangleq \sum_{j=1}^P \frac{\partial [H_t(\theta)]_{ij}}{\partial \theta_j}$$

where $[v]_i$ denotes the $i$'th component of a vector $v$ and similarly $[M]_{ij}$ denotes a single element of a matrix $M$. Here, $H_t(\theta_t)$ is an arbitrary symmetric matrix, which we choose to be the L-BFGS matrix. Consequently, (5.3) becomes a dynamical system that uses local geometry information to update the momentum component. Thanks to the convergence property of the SDE, for large enough $\beta$ and $t$, $\theta_t$ would be close to the global optimum $\theta^*$ as desired.

5.2.2 A Practical Distributed Algorithm

In order to obtain a practical algorithm, we consider Euler-Maruyama discretization of the SDE system (5.3) with a fixed step size $h$. We also define $u_n \triangleq h p_n$, $\gamma' \triangleq h \gamma$, $h' \triangleq h^2$, and discard the term $\Gamma_n$ as described in Şimşekli et al. (2016b). These simplifications lead to the following update equations:

$$u_{n+1} = u_n - h' H_n(\theta_n) \nabla_{\theta} U(\theta_n) - \gamma' u_n + \sqrt{2h' \gamma' / \beta} Z_{n+1}$$

$$\theta_{n+1} = \theta_n + H_n(\theta_n) u_n \quad (5.4)$$

Our Distributed Framework

Next we describe how to adapt above update equations to a distributed setting. We consider a classical asynchronous optimization architecture, which is composed of (i) a master node, (ii) several worker nodes, and a (iii) a data server.

**Master node** The main task of the master node is to maintain the newest iterate of the algorithm. At each iteration, the master node receives an additive update vector from a worker node, adds this vector to the current iterate in order to obtain the next iterate, and then sends the new iterate to the worker node which has sent the update vector.

**Worker node** The worker nodes work in a completely asynchronous manner. A worker node receives the iterate from the master node, computes an update vector, and sends the update vector to the master node. However, since the iterate would be possibly modified by another worker node which runs asynchronously in the mean time, the update vector that is sent to the server will thus be computed on an old iterate, which causes both practical and theoretical challenges. Such updates are aptly called "delayed" or "stale".
Data server The full data is kept in the data server and we assume that all the workers have access to the data server.

In a distributed scenario, the gradient $\nabla U$ in (5.4) needs to be replaced with its stochastic counterpart (5.1). Furthermore, the variables $\theta_n$ and $u_n$ are replaced with stale ones $\theta_{n-l_n}$ and $p_{n-l_n}$ due to the asynchronous nature of our algorithm. Here, $l_n \geq 0$ denotes the “staleness” of a particular update and measures the delay between the current update and the up-to-date iterate that is stored in the master node. We assume that the delays are bounded, i.e. $\max n l_n \leq l_{\text{max}} < \infty$.

Update Equations

Master updates The proposed algorithm iteratively applies the following update equations in the master node:

$$
\begin{align*}
    u_{n+1} &= u_n + \Delta u_{n+1}, \\
    \theta_{n+1} &= \theta_n + \Delta \theta_{n+1},
\end{align*}
$$

where $n$ is the iteration index, and $\Delta u_{n+1}$ and $\Delta \theta_{n+1}$ are the update vectors that are computed by the worker nodes (see below).

Worker updates A worker node runs the following equations in order to compute the update vectors:

$$
\begin{align*}
    \Delta u_{n+1} &\triangleq -h' H_{n+1}(\theta_{n-l_n}) \nabla \hat{U}_{n+1}(\theta_{n-l_n}) - \gamma' u_{n-l_n} + \sqrt{2h'/\beta} Z_{n+1} \\
    \Delta \theta_{n+1} &\triangleq H_{n+1}(\theta_{n-l_n}) u_{n-l_n},
\end{align*}
$$

where $\nabla \hat{U}_{n+1}$ denotes the stochastic gradient computed at iteration $n$.

Handling stochastic L-BFGS The master node in mb-L-BFGS algorithm stores a central L-BFGS memory (i.e. the collection of the gradient and iterate differences), and delivers the stored information to the workers in synchronously. However, we let each worker have their own local L-BFGS memories since the master node would not be able to keep track of the gradient and iterate differences due to asynchrony. In our strategy, each worker updates its own L-BFGS memory right after sending the update vector to the master node.

5.2.3 Main Theoretical Result

For completeness, we repeat the main theoretical result of the paper (Theorem 1 in Publication IV).

Theorem 1. Assuming all the conditions presented in Publication IV hold,
the ergodic error of the proposed algorithm is bounded as follows:

\[ |\mathbb{E}[\hat{U}_N - U^*]| = \mathcal{O}\left(\frac{1}{Nh} + \max(1, l_{\text{max}})h + \frac{1}{\beta}\right) \] (5.5)

\[ \hat{U}_N \triangleq \frac{1}{N} \sum_{n=1}^{N} U(\theta_n) \]

\[ U^* \triangleq U(\theta^*) \]

Theorem 1 provides a non-asymptotic guarantee for convergence to a point that is close to the global optimizer \( \theta^* \) even when \( U \) is non-convex. The suggested bound on the optimal rate of convergence is in line with the rate of non-convex asynchronous gradient descent algorithm (Lian et al., 2015). Finally, as expected, the error grows linearly with the staleness of the updates \( l_{\text{max}} \).

**The temperature parameter \( \beta \)** As shown in the above theorem, the convergence bound is inversely proportional to \( \beta \). Hence, we propose to set \( \beta \) to large values to speed up the convergence. Note that the inverse of \( \beta \) appears as a multiplicative factor of \( \Gamma_n \), a term that we discarded (see (5.3) again). Consequently, high values of \( \beta \) help us diminish the error caused by our negligence, as well. In the following experiment, we set \( \beta = 10^3 \) by cross-validation.

### 5.3 Large-scale Matrix Factorization Experiment

The performance of asynchronous stochastic gradient methods has been evaluated in several studies, where the advantages and limitations have been illustrated in various scenarios (Dean et al., 2012; Zhang et al., 2015; Zheng et al., 2017). In this study, in order to illustrate the advantages of asynchrony, we compare as-L-BFGS with mb-L-BFGS Berahas et al. (2016); and in order to illustrate the advantages that are brought by using higher-order geometric information, we compare as-L-BFGS to asynchronous SGD (a-SGD) Lian et al. (2015). We also explore the speedup behavior of as-L-BFGS for increasing the number of workers \( W \).

**Experiment setup** We showcase our method on a large-scale matrix factorization problem (Gemulla et al., 2011), where the goal is to obtain the MAP solution of the following probabilistic generative model:

\[ F_{rk} \sim \mathcal{N}(0, 1) \]

\[ G_{ks} \sim \mathcal{N}(0, 1) \]

\[ Y_{rs} | F, G \sim \mathcal{N}\left( \sum_k F_{rk} G_{ks}, 1 \right) \]
Table 5.1. A summary of the datasets.

<table>
<thead>
<tr>
<th></th>
<th># NONZEROS (N)</th>
<th># MOVIES (R)</th>
<th># USERS (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML-1M</td>
<td>1 million</td>
<td>3883</td>
<td>6040</td>
</tr>
<tr>
<td>ML-10M</td>
<td>10 million</td>
<td>10681</td>
<td>71567</td>
</tr>
<tr>
<td>ML-20M</td>
<td>20 million</td>
<td>27278</td>
<td>138493</td>
</tr>
</tbody>
</table>

Here, \( Y \in \mathbb{R}^{R \times S} \) is the data matrix, and \( F \in \mathbb{R}^{R \times K} \) and \( G \in \mathbb{R}^{K \times S} \) are the factor matrices to be estimated.

In this context, we evaluate the algorithms on three large-scale movie ratings datasets, namely MovieLens (grouplens.org). The properties of the datasets are given in Table 5.1. We have conducted the experiments on a cluster consisting of more than 500 interconnected computers, each of which is equipped with variable quality CPUs and memories. All the three algorithms are implemented in C++ by using a low-level message passing protocol for parallelism, namely the OpenMPI library. This code can be used both in a distributed environment or a single computer with multiprocessors.

**Ablations** First, we investigate the speedup properties of as-L-BFGS. For this, we only consider the ML-1M dataset and run the as-L-BFGS algorithm for different number of workers. Figure 5.2 illustrates the results of this experiment. As we increase the number of workers \( W \) from 1 to 10, we obtain a decent speedup that is close to linear. However, when we set \( W = 20 \) the algorithm becomes unstable since the term \( l_{\text{max}} \) in (5.5) dominates the error. Therefore, for \( W = 20 \) we need to decrease the step-size \( h \), which requires the algorithm to be run for a longer amount of time in order to achieve the same error as achieved when \( W \) was smaller.

**Comparisons** Figure 5.3 shows the performance of the three algorithms on the MovieLens datasets in terms of the root-mean-squared-error (RMSE), which is a standard metric for recommendation systems, and the norm

![Figure 5.2. The convergence behavior of as-L-BFGS on the ML-1M dataset for increasing number of workers (taken from Publication IV). The “time speedup” is computed as the ratio of the running time with 1 worker to the running time of \( W \) workers.](image-url)
Figure 5.3. The convergence behavior of the algorithms on the MovieLens datasets for $W = 10$ (taken from Publication IV).

of the gradients through iterations to measure convergence. In these experiments, we set $K = 5$ for all the three datasets and we set the number of workers to $W = 10$. The results show that, in all datasets, as-L-BFGS provides a significant speedup over mb-L-BFGS thanks to asynchrony. We can observe that even when the speed of convergence of mb-L-BFGS is comparable to a-SGD and as-L-BFGS (cf. the plots showing the norm of the gradients), the final RMSE yielded by mb-L-BFGS is poorer than the two other methods, which is an indicator that the asynchronous algorithms are able to find a better local minimum. On the other hand, the asynchrony causes more fluctuations in as-L-BFGS when compared to a-SGD.

We also observe that as-L-BFGS provides a slight improvement in the convergence speed when compared to a-SGD. This indicates that a-SGD is able to achieve a comparable convergence speed by taking more steps while as-L-BFGS is computing the matrix-vector products. However, this gap can be made larger by considering a more efficient, yet more sophisticated implementation for L-BFGS computations Chen et al. (2014).
The goal in reinforcement learning (RL) is to build intelligent agents that achieve a goal by learning through experience. In the most general sense, the agent interacts with the outside world, which we refer to as environment. Throughout this chapter, we assume that the environment can be represented by a state vector \( s \in S \subseteq \mathbb{R}^n \) with \( S \) denoting the set of all possible states. The agent interacts with the environment through actions \( a \in A \subseteq \mathbb{R}^m \), where \( A \) is a bounded action set. The environment also gives rise to rewards \( r \in \mathbb{R} \) to be maximized over a long run (Sutton and Barto, 2018).

The properties of the environment are typically determined by the task of interest. While some of the commonly used benchmark tasks such as chess or Atari games (Schrittwieser et al., 2020) have countably many states and actions, classical RL tasks such as pole balancing (Sutton and Barto, 2018) are naturally defined in continuous state and action spaces. Similarly, the environment may be inherently stochastic or completely deterministic. In certain tasks, the reward is obtained at a fixed frequency while other environments might return the reward once the task terminates (known as sparse rewards). In this work, we focus our attention on continuous-time, state and action problems, where the dynamics are defined by ODEs and the immediate reward is always available. The setting is formally described in Section 6.1.

The particular approach we take to solve RL problems is known as model-based RL (MBRL). The idea behind MBRL is to infer an approximate dynamics model that can be used to simulate the environment. In turn, the simulations allow the agent to learn a policy that specifies how to act. The vast majority of MBRL methods are based on auto-regressive, discrete-time transition functions that take as input the current state and action and output a new state - thus are infeasible to learn ODEs. To resolve this, we propose to adopt the ideas presented in previous chapters to learn unknown continuous-time dynamical systems involving actions (Section 6.2). We also describe a novel and theoretically consistent continuous-time policy learning algorithm that generalizes its discrete-time counterparts.
Table 6.1. A comparison of existing MBRL methods: PILCO (Deisenroth and Rasmussen, 2011), GP-MPC (Kamthe and Deisenroth, 2018), PETS (Chua et al., 2018), RSSM (Hafner et al., 2018), Dreamer (Hafner et al., 2020), DeLaN (Lutter et al., 2019), SymODEN (Zhong et al., 2020). “Model predictive control” and “proportional-derivative” are abbreviated as MPC and PD.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>DYNAMICS</th>
<th>APPROXIMATION</th>
<th>CONTROL/RL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PILCO</td>
<td>Discrete</td>
<td>GP</td>
<td>Policy</td>
</tr>
<tr>
<td>GP-MPC</td>
<td>Discrete</td>
<td>GP</td>
<td>MPC</td>
</tr>
<tr>
<td>PETS</td>
<td>Discrete</td>
<td>Ensembles</td>
<td>MPC</td>
</tr>
<tr>
<td>RSSM</td>
<td>Discrete</td>
<td>RNN</td>
<td>MPC</td>
</tr>
<tr>
<td>Dreamer</td>
<td>Discrete</td>
<td>RNN</td>
<td>Policy</td>
</tr>
<tr>
<td>DeLaN</td>
<td>Lagrangian</td>
<td>NN</td>
<td>PD-controller</td>
</tr>
<tr>
<td>SymODEN</td>
<td>Hamiltonian</td>
<td>NN</td>
<td>Energy shaping</td>
</tr>
<tr>
<td>Publication V</td>
<td>Continuous</td>
<td>Ensembles</td>
<td>Policy</td>
</tr>
</tbody>
</table>

(Section 6.3). A detailed description of existing approaches, which also highlight the novel aspects of our method, is presented in Section 6.4. Please see Table 6.1 for a summary of the most recent MBRL approaches.

Ever-increasing number of RL methods are typically benchmarked on simulation frameworks such as OpenAI Gym (Brockman et al., 2016) and DeepMind Control Suite (Tassa et al., 2018). Both frameworks by-pass the need to contact with the physical world thanks to the MuJoCo physics engine (Todorov et al., 2012). MuJoCo represents multi-joint dynamics in generalized coordinates and performs numerical integration to simulate real-world systems. Consequently, the accuracy of the engine is inherently restricted by the underlying numerical approaches. This often overlooked property of MuJoCo turns out to be highly crucial from a continuous-time modeling standpoint. We provide experimental evidence demonstrating the inaccuracies of existing benchmarks, and present a new framework in Section 6.5. We conclude by demonstrating our method on three different RL problems and show its robustness to both noisy and irregular data, which pose major challenges to conventional discrete-time approaches.

### 6.1 Problem Formulation

A continuous-time and state RL problem is defined by an ODE (Doya, 1996):

\[
\dot{s}(t) = f(s(t), a(t)),
\]

which was first given in (2.4). The state \(s(t)\) at real-valued time \(t \in \mathbb{R}_+\) depends on the initial state \(s_0\) as well as the infinitesimal sequence of past
actions $a(0, t)$, and can be solved with

$$s(t) = s_0 + \int_0^t f(s_\tau, a_\tau) d\tau,$$

(6.2)

where $\tau \in \mathbb{R}_+$ is an auxiliary time variable. Our goal is to learn a policy function parameterized by $\phi$ that maps states into actions

$$a_t = \pi_\phi(s_t)$$

such that the following discounted infinite horizon reward integral, or value function, is maximized (Sutton and Barto, 2018):

$$\max_\phi V(s_t) = \int_t^\infty e^{-\frac{\tau-t}{\eta}} r(s_\tau, a_\tau) d\tau.$$  

(6.3)

Above, the state evolution $s(t)$ is given by the ODE dynamics (6.2). We also assume a given discount term $\eta > 0$ and the instantaneous reward function $r(s_\tau, a_\tau) : \mathbb{R}^{n+m} \to \mathbb{R}$ which can be differentiated wrt $s_\tau$ and $a_\tau$. Figure 6.1 contrasts the discrete and continuous-time modeling approaches.

Maximizing $V(s_t)$ wrt policy parameters $\phi$ brings two challenges: (i) computing the derivatives along the path $s[t, \infty)$ governed by the unknown dynamics $f$ (6.1) and (ii) computing the infinite integral (6.3). We circumvent the first problem by learning neural network dynamics approximation $f_W$, which allows differentiation via forward sensitivities or adjoints as described in Section 2.2.4. The second problem is tackled by a novel dynamic programming approach to approximate $V(s_t)$. The following two sections describe our method in detail.

### 6.2 Dynamics Learning

Throughout this section, we assume that the dataset consists of $N$ observed state-action trajectories $D = \{(s_0^{(n)}, a_0^{(n)})\}_{n=1}^N$ collected from a continuous-time environment we aim to model or, in case of simulated environments, obtained by solving the true ODE system at observation time points $t_{0:T}$. The $i$'th observation within a trajectory is denoted by $s_i := s(t_i)$ and $\Delta t_i = t_{i+1} - t_i$ stands for the time until the next observation $s_{i+1}$. We note that observations may arrive irregularly in time.

We consider deep neural ODE models $f_W$ to learn the observed trajectories, where $W$ denotes the network parameters (Chen et al., 2018b). As pointed out in Section 4.2, non-linear neural networks lead to intractable posterior distributions, and we resort to variational inference by introducing an approximate posterior distribution $q(W)$ and evidence lower bound (ELBO):

$$\log p(D) \geq \mathbb{E}_{q(W)} \left[ \log p(D|W) \right] - \text{KL} \left[ q(W) \parallel p(W) \right],$$

(6.4)
where $p(W)$ is the parameter prior. Note that the likelihood $p(D|W)$ is evaluated only on the state-trajectories, i.e., $p(D|W) = p(D_s|D_a, W)$, where $D_s = \{s_{0:T}^{(n)}\}_{n=1}^N$ and $D_a = \{a_{0:T}^{(n)}\}_{n=1}^N$. Similar to ODE$^2$VAE, the first expectation above is approximated by Monte Carlo sampling:

$$
\mathbb{E}_{q(W)}[\log p(D|W)] \approx \frac{1}{L} \sum_{l=1}^{L} \sum_{n=1}^{N} \sum_{i=0}^{T} \log \mathcal{N}\left(s_i^{(n)} | \hat{s}_{i|l}^{(n,l)}, \Sigma \right), \quad (6.5)
$$

where $\Sigma$ denotes a diagonal matrix of trainable observation noise variances. A trajectory sample $s^{(n,l)}_{0:T}$ is obtained by first sampling from the posterior $W^{(l)} \sim q(W)$ and then forward simulating the dynamics model $f_{W^{(l)}}(s, a)$.

**Ensemble of neural ODEs (ENODEs)** A series of MBRL works (Kurutach et al., 2018; Chua et al., 2018; Janner et al., 2019; Pan et al., 2020) showed that deep ensembles typically excel at learning multi-modal posterior
Continuous-Time Model-Based Reinforcement Learning

distributions. In turn, this provides means for epistemic uncertainty modeling, which can be defined as our lack of knowledge about the dynamics in certain regions in the state space due to lack of data. Since epistemic uncertainty handling lies at the heart of successful MBRL applications (Chua et al., 2018), we choose our variational posterior to be a mixture of Dirac densities on the weights, resulting in an ensemble of neural ODEs approximation (ENODEs). As demonstrated in Figure 6.2, each ensemble member indeed leads to slightly different vector fields.

Comparison with our previous models The presented approach is in large part built upon the ideas in Publication I and Publication III. Similar to npODE, we consider a first-order dynamical system (6.5) that is defined in the same (state) space as the observations. As in ODE$^2$VAE, we opt for a neural network differential function approximation and variational inference for the model parameters. Finally, our approximate posterior distribution is a mixture of Dirac densities since the MAP estimation in Publication I and mean-field approximation in Publication III only provide unimodal approximations to a complicated, multimodal posterior (Izmailov et al., 2021), which prevents us from handling epistemic uncertainty.

Time-continuous actions Any state trajectory is fully determined by the initial state $s_0$ and the infinitesimal actions $a_t(0, t)$ applied during integration. Since the numerical integration methods may evaluate the integral at arbitrary time points, the actions must be continuous in time, whereas they are assumed to be known only at observation time points. A simple yet convenient mean to obtain time-continuous actions is interpolating between observed discrete actions. We opt for kernel interpolation with a squared exponential kernel function whose length-scale parameter can be set depending on the smoothness of the action sequence.

6.3 Continuous-time Actor-Critic

Actor-critic methods are among the most prominent policy learning techniques (Sutton and Barto, 2018). As implied by its name, actor-critic methods consist of two modules: the actor (policy) that is used to select the actions, and the critic that estimates the value function (6.3). To the best of our knowledge, the actor-critic methods have been presented solely in the context of discrete-transitions previously (Sutton and Barto, 2018). In the following, we describe a novel algorithm that is fully compatible with our continuous-time framework, and the standard discrete-time actor-critic methods (Schulman et al., 2015; Hafner et al., 2020) appear as special cases of our algorithm.

Our main policy learning goal is to maximize the discounted cumulative reward (6.3) when the policy is executed in the real-world. Because the
true differential function is unknown, we leverage our proxy dynamics $f_W$ to maximize a surrogate objective:

$$V(s_0) \approx \hat{V}(s_0) = \mathbb{E}_q(W) \left[ \int_0^\infty e^{-\tau} r(s_\tau, \hat{a}_\tau) d\tau \right],$$  \hspace{1cm} (6.6)

where $\hat{s}[0, t]$ and $\hat{a}[0, t]$ are referred to as \textit{imagined trajectory} and \textit{imagined actions}\(^1\) and $s_0$ denotes an initial value. The imagination is controlled by a policy function $\pi_\phi(\cdot)$ with parameters $\phi$:

$$\hat{s}(t) = s_0 + \int_0^t f_W(\hat{s}_\tau, \hat{a}_\tau) d\tau$$  \hspace{1cm} (6.7)

$$\hat{a}_t = \pi_\phi(\hat{s}_t).$$  \hspace{1cm} (6.8)

Note that for the dynamics differential $\dot{s}(t)$ to exist, the function $f_W$ needs to be differentiable over time. When the actions are given by a policy function as in our imagination framework, both $f_W$ and $\pi_\phi$ need to be deterministic. Also, the initial state $s_0$ generally follows a distribution $s_0 \sim p(s_0)$. In such cases, the optimization target becomes $\mathbb{E}_{p(s_0)}[\hat{V}(s_0)]$, which is usually approximated by Monte-Carlo averaging.

### 6.3.1 State-Value Approximation (Critic)

The optimization objective in (6.6) requires computing an infinite integral. To circumvent this problem, we first re-write the value function in a recursive manner:

$$\hat{V}^H(s_0) = \mathbb{E}_q(W) \left[ \int_0^H e^{-\frac{H}{\eta}} r(\hat{s}_\tau, \hat{a}_\tau) d\tau + e^{-\frac{H}{\eta}} \hat{V}(\hat{s}_H) \right],$$

where $H$ is known as \textit{horizon}, $\hat{s}_H := \hat{s}(H)$ is the end-state of the integral, and $\hat{V}(\hat{s}_H)$ is \textit{future reward}. The first term inside the expectation represents the imagined reward, which can be computed by evaluating the reward function along the trajectory given by (6.7-6.8), whereas the second term is still an infinite integral. Next we introduce the critic, or state-value approximation, that replaces intractable future reward term:

$$\hat{V}(s) \approx \nu_\xi(s),$$

where $\nu_\xi$ is a neural network with parameters $\xi$. With a slight abuse of notation, we rewrite the value function as follows:

$$\hat{V}^H(s_0) = \mathbb{E}_q(W) \left[ \int_0^H e^{-\frac{H}{\eta}} r(\hat{s}_\tau, \hat{a}_\tau) d\tau + e^{-\frac{H}{\eta}} \nu_\xi(\hat{s}_H) \right].$$  \hspace{1cm} (6.9)

Consequently, our new objective seeks to learn a policy function $\pi_\phi$ that maximizes the reward integral over horizon $H$ and ends up at a high-value state.

---

\(^1\)The actions are applied to imagined states, rendering the actions imagined as well.
6.3.2 Estimating the Actor and Critic

**Actor** The actor aims to maximize the value estimates (6.9) for a batch of initial values. In order to learn the optimal policy everywhere, initial values are drawn from the previous experience. More formally, we solve the following maximization problem:

\[
\max_\phi \mathbb{E}_{q(s)} \left[ \hat{V}^H(s) \right], \quad q(s) = \frac{1}{|D|} \sum_{n,i} \delta \left( s - s^{(n)}_i \right), \quad \text{(6.10)}
\]

where \(|D|\) denotes the cardinality of the dataset. Observe that both terms in (6.9) depend on the imagined state trajectory, which in turn depends on the actions. Therefore, the gradient of \(\hat{V}^H(s)\) wrt \(\phi\) can easily be computed in auto-differentiation frameworks that support ODE gradients.

**Critic** Similar to discrete-time frameworks, our method utilizes temporal difference learning to estimate the state-value function. In particular, the value approximation \(\nu_{\xi}(\cdot)\) is regressed on the sum of the finite reward integral and future reward given in (6.9). Since the value approximation holds for any horizon length \(H\), we propose to average it out:

\[
\min_{\xi} \mathbb{E}_{q(s)} \left[ \left( \nu_{\xi}(s) - \mathbb{E}_{q(h)} \left[ \hat{V}^h(s) \right] \right)^2 \right], \quad q(h) = \mathcal{U}[0, H], \quad \text{(6.11)}
\]

where the outer expectation is wrt \(q(s)\) as shown in (6.10), \(\mathcal{U}\) denotes the uniform distribution, and the inner expectation is approximated by Monte-Carlo sampling. We use a separate target network that generates the future rewards, an idea that is known to stabilize the critic training (Mnih et al., 2015). We update the target network with a copy of the critic every 100 optimization iterations.

Both actor and critic loss functions require trajectory sampling from the dynamics model. In practice, we forward integrate the dynamics model once and utilize the resulting state-action trajectories to optimize both objectives. The inner expectation in (6.11) can be evaluated using the intermediate states without incurring any additional overhead.

6.3.3 Connection to Temporal Difference

Discrete state-value formulation can be retrieved by discretizing the reward integral (6.3) with a predetermined step size \(\Delta t_i \equiv t_i - t_{i-1}\) (Sutton and Barto, 2018):

\[
\bar{V}(s_t) = \sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l})
\]

where the overbar sign denotes discrete formulation and \(\gamma\) is the discount factor. In such a scenario, \(\hat{V}^h(s)\) would reduce to so-called \(h\)-step return:
\[ \nabla^h(s_t) = \sum_{t=0}^{h-1} \gamma^t r_{t+t} + \nabla(s_{t+h}). \]

Marginalizing the horizon would correspond to computing the mean of the \( h \)-step returns, a commonly used technique to reduce the gradient variance (Schulman et al., 2016). Finally, TD-\( h \)-learning would be retrieved by simply keeping the horizon fixed at \( h \), i.e., with \( q(h) \) being a Dirac distribution.

### 6.4 Previous Work

**Model-based RL** The majority of model-based reinforcement learning methods assume auto-regressive transitions, which effectively learn a distribution over the next state given the current state and action. Unknown transitions are typically approximated by a Gaussian process (Kocijan et al., 2004; Deisenroth and Rasmussen, 2011; Kamthe and Deisenroth, 2018; Levine et al., 2011), multi-layer perceptron (Gal et al., 2016; Depeweg et al., 2017; Nagabandi et al., 2018; Chua et al., 2018) or recurrent neural network (Ha and Schmidhuber, 2018; Hafner et al., 2018, 2020). Such models are typically developed in conjunction with model predictive control (Richards, 2005) used for planning or with a parametric policy.

**Continuous-time RL (CTRL)** The first model-free CTRL approach, which was proposed by Baird (1993), simultaneously learns value and advantage functions. Bradtke and Duff (1994) developed Q-functions and temporal different learning in the context of semi-Markov decision processes (continuous time, discrete state). Later, Doya (1996) proposed a continuous-time and state version of the temporal difference algorithm. In their seminal paper, Doya (2000) approximated the continuous-time value function by a neural network and derived a greedy policy formulation based on value gradients. Abu-Khalaf and Lewis (2005) proposed a policy-iteration algorithm for the optimal control of continuous-time systems with constrained controllers. An online version of this algorithm was derived in Vrabie and Lewis (2009), which was extended in a series of papers (Luo et al., 2014; Modares et al., 2016; Zhu et al., 2016; Lee and Sutton, 2019). A direct least-squares solution to Hamilton-Jacobi-Bellman equation was studied in Tassa and Erez (2007), requiring no forward integration for value estimation but a bag of tricks to deal with numerical instabilities. In a related work, Mehta and Meyn (2009) proposed an adaptive controller for nonlinear continuous-time systems via a continuous-time analog of the Q-function. Above-mentioned methods are either built upon known dynamics or they are model-free. In either case, the dynamics are assumed to be linear with respect to the action, a premise needed for closed-form optimal policies.

**Neural CTRL** The neural ODE breakthrough (Chen et al., 2018b) has opened a new research avenue in CTRL. In particular, physics-informed...
Continuous-time dynamical systems have gained popularity. For example, Lagrangian mechanics are imposed on the architecture presented in Lutter et al. (2019), which results in near-perfect real-time control of a robot with seven degrees of freedom. Hamiltonian framework, coupled with symplectic solvers, is proven useful for inferring controls from generalized coordinates and momenta (Zhong et al., 2020). Later in Zhong and Leonard (2020), an interpretable latent Lagrangian dynamical system and controller are trained from images. Note that above-mentioned methods describe model-specific recipes for learning controls.

**Continuous-time Value Functions** In one of the earliest CTRL studies, Baird (1994) showed that the Q-function that appears in Q-learning collapses to the value function in continuous-time, which prevents the use of off-the-shelf discrete-time policy learning methods. Continuous-time counterpart of the Q-function proposed by Baird (1994) and later extended by Tallec et al. (2019) is known as **advantage updating**. Doya (2000) considered the Euler discretization of the infinite value integral, which leads to an update equation that coincides with the discrete temporal difference update. Later, Tassa and Erez (2007) described an iterative least squares method that directly minimizes Hamiltonian-Jacobi-Bellman (HJB) estimation error, where HJB famously gives the necessary and sufficient condition for optimality.

### 6.5 A New CTRL Evaluation Environment

We experimentally evaluate our model on three CTRL benchmark environments: Pendulum, CartPole and Acrobot, which are already implemented in standard RL frameworks such as OpenAI Gym (Brockman et al., 2016) and DeepMind Control Suite (Tassa et al., 2018). Existing implementations either perform discrete transitions or inaccurate numerical integration routines, such as Euler’s and Runge-Kutta’s methods with too big discretization step sizes. Thus, we implement a new RL framework which takes as input a policy function \( \pi(\cdot) \) and observation time points, performs numerically-stable forward integration, and returns the state trajectory at corresponding time points:

\[
s(T) = s_0 + \int_0^T f(s_\tau, \pi(s_\tau)) d\tau.
\]

Note that the environment-specific initial state \( s_0 \) and the true (hidden) time differential \( f(s, a) \) are specified in the supplementary material of Publication V.

Next, we ask two questions in relation with the simulation environment and numerical integration: (i) which numerical ODE solver one should use, (ii) to what extent our continuous framework differ from its discrete
counterparts? To answer, we have built a simple experiment on CartPole environment where several ODE solvers are compared: three adaptive step solvers (dopri5 (RK45), RK23 and RK12), five fixed step solvers (RK4 with 1/10 intermediate steps, and Euler with 10/100/1000 intermediate steps), as well as discrete transitions. Due to the lack of a closed-form ODE solution, true ODE solutions are obtained by Runge-Kutta 7(8) solver, the numerical integrator which achieves the smallest local error to the best of our knowledge (Prince and Dormand, 1981). Each ODE solver takes as input the same set of initial values and policy functions.

Figure 6.3 demonstrates the distance between the true state solutions and those given by different ODE solvers. The most striking observation is that discrete transitions of the form $s_{t+1} - s_t = h \cdot f(s_t, a_t)$ are highly erroneous. Moreover, adaptive solvers as well as fixed-step solvers with sufficiently many intermediate steps attain practically zero error. Unsurprisingly, approximate state solutions deteriorate over time since the error accumulates. In our experiments, we use RK78 to mimic the interactions with the real world, and dopri5 to forward simulate model dynamics.

6.6 Experimental Results

In all three environments, our goal is to bring the pole from downright to upright position (also at the origin in CartPole task). We define the differentiable reward functions as follows:

$$r(s, a) = \exp(||s - s_{\text{goal}}||_2) - \lambda ||a||_2,$$

which favors the states that are close to the goal state and the actions small in magnitude. Environment-specific goal states, constant $\lambda$, and lower/upper bounds of allowed actions can also be found in Publication V.
Learning procedure  To learn optimum policy functions, MBRL techniques typically start with a dataset $D$ of transitions collected by executing a random policy in the real world (Deisenroth and Rasmussen, 2011; Janner et al., 2019). Afterwards, the following procedure is performed: (i) fitting the dynamics model $f_{W}$ to the dataset $D$, (ii) optimizing the policy $\pi_{\phi}$ using the dynamics model, (iii) collecting new sequences by executing the current policy $\pi_{\phi}$ in the real world, and (iv) stop if the problem is solved, else go to step (i). We refer to each iteration of this procedure as a “round”.

Observation times and noise  We consider three scenarios in which observation arrival times are (i) constant $\kappa$, (ii) uniformly random within a range $\Delta t \sim U(0, 2\kappa)$, and (iii) exponentially distributed $\text{Exp}(1/\kappa)$ with $\kappa$ being the rate parameter. Observe that all three cases have the same mean arrival times $\kappa$. We also add an observation noise with standard deviation $\sigma = 0.025$.

Compared methods  We compare our model with ensemble neural ODE (ENODE) approximation against the state-of-the-art probabilistic ensembles with trajectory sampling (PETS) by Chua et al. (2018). As pointed out in Section 3.1, auto-regressive methods such as PETS are unable to learn irregularly sampled time sequences. In the following, we propose to parameterize the transition with the time increments as follows:

$$s_{i+1} = s_i + f(s_i, a_i, \Delta t_i).$$

We rename this new model variant modified PETS (MPETS).

Results  The results are presented in Figure 6.4. Each panel demonstrates the performance on a single environment with a fixed mean arrival time $\kappa$. $x$-axes correspond to rounds and $y$-axes illustrate the mean values $V(s_0)$ obtained by executing policies in the real world. We repeat each experiment 20 times and report the mean rewards and 90/10 percent quantiles. As expected, our continuous-time model is rather robust against arrival frequency while small $\kappa$ values are particularly problematic for MPETS. In another ablation study, we observed that both models perform comparably in noise-free scenarios. Therefore, we conclude that our model is more suitable than its discrete counterparts for noisy datasets, and also provide a fundamentally sound approach to handle irregularly sampled sequences.
Figure 6.4. ENODE vs. MPETS on noisy environments ($\sigma = 0.025$), taken from Publication V.
Thanks to their solid theoretical foundations and decades of previous research, differential equations arise in a wide range of statistical learning problems. This dissertation develops a series of methods in which differential equations played a key role for solving time series prediction, optimization and reinforcement learning problems. Our most significant findings are as follows:

- Parametric ODE and SDE systems can be inferred in a completely data-driven way using black-box function approximations such as GPs and NNs.

- Inferring unknown continuous-time systems with black-box ODEs leads to better predictive performance compared to discrete-time approaches.

- Latent ODE systems can be inferred simultaneously with observation mapping from video sequences.

- An SDE perspective to SGMCMC allows us to conveniently utilize Hessian information and paves the way for an asynchronous and distributed optimization algorithm.

- When coupled with a suitable actor-critic framework, continuous-time dynamical systems provide an alternative avenue for MBRL.

Further research has demonstrated that differential equation formulations have applications in variational inference (Grathwohl et al., 2018), node classification (Poli et al., 2019), generative modeling (Song et al., 2020), speech recognition (Kidger et al., 2020) and so on. Similar to our proposed methods, these techniques utilize differential equations to develop novel strategies for solving distinct statistical learning problems. We conjecture that this very recent marriage between differential equations and machine learning will lead to more exciting and theoretically grounded works and believe that contributions in this dissertation will aid researchers to develop the field further.
Conclusion and Future Directions

We conclude this dissertation by briefly reminding the proposed ideas, their shortcomings and by suggesting future directions.

**GPs for learning unknown differential equations** Publication I-II present GP-based estimation schemes for unknown ODEs and SDEs. Our completely black-box approach does not require any prior knowledge about the system dynamics; thus, it presents an alternative paradigm to mechanistic modeling. We experimentally demonstrated that the resulting npODE model is capable of learning the dynamics evolution of human walking sequences for which there is no mechanistic model, and parametric ODE systems without seeing the form of the time differential. Likewise, npSDE model is proven useful for inferring the underlying dynamics of both simulated and real-world sequences such as double-well potential and climate records.

As mentioned in Chapter 3, we compute the MAP estimate of the inducing parameters, which makes the model prone to overfitting. The inference can alternatively be performed by sampling from the posteriors of drift and diffusion functions (Wilson et al., 2020). Similarly, MCMC sampling of the inducing point posterior would lead to trajectory samples instead of a single estimate. Time-varying functions and investigation of different kernel choices reflecting our prior knowledge about the dynamics might also increase model accuracy.

**Continuous-time Bayesian modelling for high dimensional sequences** In Publication III, we presented ODE$^2$VAE, a neural ODE system governing the dynamics evolution in a latent space induced by a VAE. Unlike our previous approach, we address the uncertainty of the dynamics by computing an approximate posterior over the ODE parameters. Based on the observation that many Newtonian systems are expressed in terms of generalized position and velocity components (Montague, 1995), we explicitly decompose the latent space to account for these components. As shown empirically, the model achieves state-of-the-art performance in long-term forecasting and imputation of high-dimensional image sequences.

Since both the latent mapping and the dynamical system can be arbitrarily expressive, identifying these components becomes a challenging task. The final optimization objective contains a KL term in addition to ELBO, which aims to minimize the distance between the encoder distribution and the distribution induced by the ODE flow. A more principled approach to handle this mismatch might provide new insights. The scope of the experiments was limited to gray-scale images; therefore, demonstrating the model capabilities or extensions to learn dynamics from colored images as well as other high-dimensional data sequences would be a fruitful area for future work. An alternative SDE formulation of the dynamics might further allow the model to learn more complicated scenarios.
Conclusion and Future Directions

SDEs for stochastic asynchronous non-convex optimization  Building upon the recent findings in SGMCMC literature, we propose a stochastic asynchronous non-convex optimization algorithm that converges to a point that is close to the global optimum. Our methodology relies on tempering the SGMCMC target posterior to ensure that the tempered density concentrates around the optimum. We also describe the first asynchronous L-BFGS framework, allowing faster convergence as demonstrated by the experiments.

Despite the non-asymptotic guarantee for convergence to a point that is close to the global optimizer, the experiments show that asynchrony causes a great deal of fluctuations in the loss term. A thorough investigation of how the fluctuations impact the Hessian matrix (and thus optimization) could shed more light on the methodology. Although presented as a non-convex optimization method, the experiments were performed on convex problems. Further experiments demonstrating the technique on more complicated non-convex problems would be useful to show the merits of the technique.

Continuous-time model-based reinforcement learning  Publication V describes a novel continuous-time perspective to MBRL by replacing the discrete-time dynamics approximations with ensemble of neural ODEs, which can learn arbitrarily complicated controlled ODE systems in an uncertainty aware way. We furthermore present the first actor-critic algorithm that operates in continuous time and space and develop a continuous-time RL benchmark that addresses the inaccuracies of standard benchmark frameworks. We experimentally demonstrate that our method is robust to environment changes such as observation noise, temporal sparsity and irregularity.

Although continuous-time formulation achieves a smaller long-term dynamics error, discrete-time baseline still requires a comparable number of trials to solve the problem. Further research should be undertaken to explore how dynamics error impacts the overall performance. Uncertain interpolations between discrete action points could also be tested for better uncertainty handling. Similar to ODE$^2$VAE, coupling the model with a VAE might allow learning controls from image sequences.


References


Rainer Gemulla, Erik Nijkamp, Haas. Peter J., and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *ACM Special Interest Group on Knowledge Discovery and Data Mining*, 2011.


References


Arne Jensen. Lecture notes on difference equations. Department of mathematical science, Aalborg University, Denmark, July, 18, 2011.
References


References


Feiyang Pan, Jia He, Dandan Tu, and Qing He. Trust the model when it is confident: Masked model-based actor-critic. In Advances in Neural Information Processing Systems, 2020.


References


