STEINDÓR SÆMUNDSSON

CONSERVATION LAWS AS INDUCTIVE BIASES

A thesis presented for the degree of
DOCTOR OF PHILOSOPHY

Supervised by
MARC DEISENROTH
KATJA HOFMANN

DPT. OF COMPUTING
IMPERIAL COLLEGE LONDON

AUGUST, 2021

Statement of Originality

The work presented in this thesis is my own, except where otherwise indicated.

Copyright Notice

Acknowledgements

For the innumerable ways in which others have generously contributed to and supported this thesis...

I have been particularly fortunate to do my PhD with the guidance of Marc Deisenroth as my supervisor. Throughout, he has gone over and beyond in supporting me both academically and personally. I will always be grateful to him—especially for his patience of my (sometimes unreasonable) desire for exploration.

I have also had the great joy and luck to be supervised by Katja Hofmann. Much of my growth as a researcher can be attributed to her ever-insightful questions and astute feedback. For this, and all her support during my PhD/internship, I am truly grateful.

A number of students in the Statistical Machine Learning group at Imperial/UCL have been important for this thesis. Conversations with Hugh Salimbeni and Sanket Kamthe were always informative and clearly important to my progression; key components of the thesis would not have been possible without my collaboration with Alexander Terenin; and James Wilson often pointed me towards useful work in the literature.

Outside the group, I had the pleasure of supervising/collaborating with Jean Kaddour, producing results contained in the thesis. I would also like to thank Nick Pawlowski and Sebastian Popescu for always being a source of interesting conversation; Andreas Hochlehnert for his excellent work as a supervisee; and Amani El-Kholy for her continued support during my time at Imperial.

This thesis was supported by Microsoft Research (MSR) through its PhD Scholarship Programme. I would like to thank Sebastian Tschiatschek for his support as a supervisor during my time as an intern, and Richard E. Turner for his great advice.

On that note, I would like to thank the people that shaped my internship at X (the moonshot factory): Julian Green, Raffi Hotter, Albin Jones, Isaac Kauvar, Owen Lewis and Marius Wiggert; all of whom introduced me to new and interesting perspectives on life and research.

I also wish to express my deepest gratitude to Emma and Andrew Dunn for their generosity and kindness, and Max Dunn for his unrivalled spirit and infectious drive. Undoubtedly, this thesis is shaped by the inspiring environment I have enjoyed in their company.

Finally, I would like to thank Grace Dunn for infusing the last two years with meaning—for her creativity, kindness, and insight—and for her time and attention throughout my writing of this thesis.

Abstract

A basic pattern in nature is invariance: the notion that properties (e.g. energy) of a system remain unchanged through a transformation (e.g. time). However, learning such patterns from data can be challenging since they are often non-trivially disguised as variation in observed phenomena. The motivation for the work in the thesis is improved data efficiency when learning predictive models of physical dynamical systems. Building on ideas from machine learning and physics, it explores learning algorithms using conserved quantities and conservation laws as general purpose model components, with the aim of more efficient learning.

Chapter 2 develops learning algorithms for task structured problems where the notion of a task is identified with an unobserved conserved quantity to be learned from data. The main contribution is a model that accounts for globally invariant sources of variation (e.g. the laws of physics) and task-specific sources of variation (e.g. system parameters). The idea is to encourage modularity: a separation of reusable components of the model from task-specific ones. The chapter empirically studies the model in the context of learning predictive models of dynamical systems. It is found to be useful as an inductive bias for modularity, as measured by data efficiency in multi-task, transfer- and meta-learning settings.

Chapter 3 develops expressive function classes with inbuilt physical geometry such as conservation laws. The main contribution is a tying together of the theory of variational integrators and neural networks. This produces a scheme for deriving symplectic and momentum-preserving architectures (variational integrator networks). The architectures are studied empirically in the context of noisy, and image observations, of physical systems. In the former, they are found to be efficient and flexible learners. In the latter, they are found to learn physically meaningful geometric representations, enabling accurate long-term forecasts in image space.

Learning modular task representations is potentially important for developing practically useful meta-learning algorithms. In chapter 2 the representations are non-hierarchical and require labelling at the task-level. Extending the idea to hierarchical and unsupervised settings is an interesting future direction. Physical geometry is an elegant example of compact general-purpose representations. Extending chapter 3 to different and more general physics/manifolds, building on the literature on variational integrators, is also an interesting direction.

Thesis Outline

▸ **Chapter 1**—Introduction. Contextualises chapters 2–3 and discusses the motivation: data efficiency, modular task representations, and physical structure as inductive biases. It also introduces the basic theory used in subsequent chapters and gives an overview of the problem setting considered in the thesis.

▸ **Chapter 2**—Representing Tasks as Conserved Quantities. Develops a model for task structured learning problems where the aim is to share and transfer knowledge between tasks. The content in this chapter is based on two published works:

1. S. Sæmundsson, K. Hofmann and M. Deisenroth.
   Meta Reinforcement Learning with Latent Variable Gaussian Processes, UAI 2018;
2. J. Kaddour, S. Sæmundsson and M. Deisenroth.
   Probabilistic Active Meta-Learning, NeurIPS 2020.

Compared to the published texts, the thesis uses a revised notation and places more emphasis on modularity for task structured learning problems.

▸ **Chapter 3**—Variational Integrator Networks. Bridges the theory of variational integrators and neural networks and develops function classes with inbuilt conservation laws. The content in the chapter is based on:

1. S. Sæmundsson, A. Terenin, K. Hofmann and M. Deisenroth.
   Variational Integrator Networks for Physically Structured Embeddings, AISTATS 2020.[1]

Different from the paper, the thesis interprets the networks as being a part of the likelihood function. The experimental results in the noisy regime are also extended with more random seeds, an additional baseline, and further analysis.

▸ **Chapter 4**—Conclusion. This chapter summarises the main findings from chapters 2–3, and discusses the limitations and potential future work.

[1] See also A. Hochlehnert, A. Terenin, S. Sæmundsson and M. Deisenroth. Learning contact dynamics using physically structured neural networks, AISTATS 2021; for related follow up work.

Contents

List of Figures

List of Tables

List of Algorithms

Notation

Notation 1 (Variables, functions, and distributions)**.** Unless otherwise stated, scalar variables $\{x, y, z\}$ are denoted by lowercase italics, (column) vector variables $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ by boldface lowercase italics, and matrices $\{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}$ by boldface uppercase symbols. Boldface uppercase sans serif symbols will denote higher dimensional arrays $\{\mathsf{X}, \mathsf{Y}, \mathsf{Z}\}$, or datasets $\mathsf{D} = \{\mathbf{X}, \mathbf{Y}\}$.

Notation 2 (Scalar sample vectors)**.** It will be useful to represent collections of scalars corresponding to samples or a specific set of indices. To distinguish these from multi-dimensional vectors $\boldsymbol{y}$ (typically representing output features),

$$\boldsymbol{y}_{(n)} = (y_n), \quad n = 1, \ldots, N$$

is used to represent a column vector of scalar samples or indices.

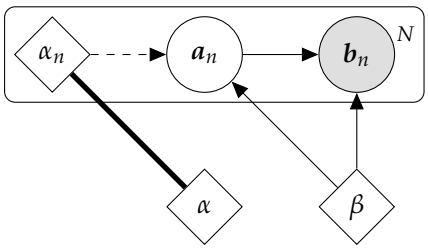
Notation 3 (Model and variational parameters θ, ϕ)**.** The set θ contains all the parameters that define a statistical model of data. Distributions p_θ or functions f_θ that depend on θ are subscripted, even if they only depend on a subset. The set ϕ analogously contains parameters that are used to define approximate posterior distributions.[2]

[2] Specifically, variational distributions (section 1.4).

Notation 4 (Identity matrix)**.** Identity matrices are denoted by $\mathbf{1}_A$, where A is the number of rows and columns.

Notation 5 (Shorthand for distributions.)**.** The superscript notation $p^{\boldsymbol{a}|\boldsymbol{b}}$ denotes the conditional distribution over $\boldsymbol{a}$ given $\boldsymbol{b}$, i.e. $p^{\boldsymbol{a}|\boldsymbol{b}} = p(\boldsymbol{a}|\boldsymbol{b})$, and analogously $p^{\boldsymbol{a}}$ denotes the marginal $p(\boldsymbol{a})$.

Notation 6 (Graphical models/Inference graphs.)**.** The thesis uses graphical models and inference graphs throughout. The figure below illustrates the key components.



Circles represent random variables where shading denotes observed instances (otherwise latent). Diamonds denote deterministic variables. Solid arrows denote dependencies in a statistical model. Dashed arrows denote dependencies in a variational distribution. Thick lines denote that one variable is a subset of the other (depending on context).

The figure defines a model $p(\boldsymbol{b}_n|\boldsymbol{a}_n, \beta)p(\boldsymbol{a}_n|\beta)$ for $n = 1, \ldots, N$; and a variational distribution $q(\boldsymbol{a}_n|\alpha_n)$, where $\alpha_n \in \alpha$.

1

Introduction

Now symmetry and consistency are convertible terms: — thus Poetry and Truth are one.

—Edgar Allan Poe

Conservation laws are fundamental to our understanding of the *nature of nature*. Intuitively, they are (relatively) simple statements about invariant properties of physical systems that nonetheless explain remarkably rich phenomena. For example, rotational invariance of the laws of electromagnetism can be seen to produce the irregular shape of a snowflake in addition to the round shape of a raindrop [Council et al., 2001].

Richness in observed phenomena means that learning these structures from data is a challenge. A reoccurring example in the thesis is learning to predict the dynamics of systems that conserve energy. This conservation law (invariant structure) is hidden in the evolution of the system due to the nonlinear relationship between energy and evolution. The problem is further complicated by how the data is represented and the presence of noise in the observations.[1]

One way to address the problem—the one considered in the thesis—is to build these structures into the learning algorithms in the form of an inductive bias. Assuming the structure is present in the data distribution, this will generally improve efficiency and generalisation [Wolpert and Macready, 1996, Baxter, 2000]. Canonical examples of invariant structures as inductive biases are found in existing neural network architectures.[2] In fact, invariance has been considered a key concept for understanding neural networks since at least Minsky and Papert [1972], and remains a core element in a recent unified perspective on neural networks [Bronstein et al., 2021].

[1] A more general example from machine learning is given by images of the same object (the invariant) from different perspectives: Even for small changes in perspective, the variation in the images may be large due to the complicated nature of the interactions between light and the object's shape and material.

[2] E.g. approximate translation invariance due to pooling in convolutional neural networks (CNNs) [Fukushima, 1980, LeCun et al., 1999]; time invariance in recurrent neural networks (RNNs) [Rumelhart et al., 1986, Hochreiter and Schmidhuber, 1997]; or permutation invariance in graph neural networks (GNNs) [Scarselli et al., 2009].

1.1 Motivation: Data Efficiency

The core motivation for the thesis is improved data efficiency when learning from physical dynamical systems. Data efficiency makes explicit the practical concern of performance as a function of the available data or cost of collecting data. Since the focus of the thesis is on the design of inductive biases, data efficiency is also one of the key measurements that are made in the experiments in chapters 2–3.

A common criticism of deep neural networks (DNNs) is the need for large datasets [Lucic et al., 2018, Adadi, 2021]. Requiring less data for equal (or better) generalisation performance is for practical purposes an important goal in its own right [Riedmiller, 2005, Deisenroth, 2010, Kamthe and Deisenroth, 2018a]. In many applications, data collection is either expensive or impossible. For example, in robotics due to wear-and-tear, drug design due to cost of running experiments, or in medical applications where it might be impossible to run experiments for ethical or practical reasons [Deisenroth et al., 2015, Liu et al., 2021, Esteva et al., 2021].

In chapter 2, data efficiency is pursued using ideas from multi-task learning [Ruder, 2017], transfer learning [Pan and Yang, 2010, Weiss et al., 2016], meta-learning [Lemke et al., 2015], and active (task) learning [Settles, 2009]. Crucially, the aim is to learn a model that can share information between different but related physical dynamical systems, each one being related to the notion of a task.[3] A key motivation for the developments in chapter 2 is to separate task-specific sources of variation from global ones. This view is shared by work on disentangled representations [Bengio et al., 2013, Higgins et al., 2018], which aims to learn separated representations from data automatically. In contrast, the work in the thesis is more focused on building this structure into the model in the form of an inductive bias.

[3] In the thesis, each task is also identified by conserved quantities representing the system parameters (e.g. length or mass).

In chapter 3, data efficiency is pursued from the standpoint of encoding geometric mechanics as inductive bias in neural networks. A key motivation for the developments in the chapter is the observation that without this structure, neural networks struggle to learn properties of physical systems such as the conservation of energy [Greydanus et al., 2019]. A number of recent works have considered this problem [Toth et al., 2019, Cranmer et al., 2020b, Zhong and Leonard, 2020],[4] which also shares commonalities with works relating differential equations and neural networks [Haber and Ruthotto, 2017, E, 2017, Chen et al., 2018, Scholkopf, 2019] and scientific machine learning [Raissi et al., 2019, Rackauckas et al., 2020].

[4] See Botev et al. [2021] for an overview.

1.2 Problem Setting: Physical Dynamical Systems

The learning problems considered in the thesis consist of observations generated by simulated physical dynamical systems, such as the pendulum example in fig. 1.1. Practically, this is achieved by numerical simulation of ordinary differential equations (ODEs), producing observed samples in discrete-time. Importantly, ODEs are also used to reason about the inductive biases of the model. Inductive bias is defined generally as the set of assumptions (implicit and explicit) that enable generalisation [Hullermeier et al., 2013]. In the thesis, it refers to assumptions about:

1. The form of the ODE used to define the model;
2. The strategy for representing a continuous-time system in discrete time;
3. Statistical relationships between variables of the resulting model.

The rest of this section introduces the key concepts of dynamical systems that are relevant to the thesis. In the context of the thesis, continuous-time dynamical systems are formal descriptions of a point evolving on a manifold. Since the full machinery of manifolds is not needed for understanding chapters 2–3, the general exposition will be limited to points evolving in Euclidean space and specific references to manifolds are addressed individually.

1.2.1 Ordinary Differential Equations

The state of a dynamical system is a Q-dimensional vector variable $\boldsymbol{h}(t) \in \mathbb{R}^Q$, and t is a parameterisation of a path such that

$$C = \{\boldsymbol{h}(t) : t \in [0, T]\}, \tag{1.1}$$

defines the path between $\boldsymbol{h}(0)$ and $\boldsymbol{h}(T)$. The path is determined by a dynamics function through an ordinary differential equation (ODE).

> **Definition 1** (Ordinary Differential Equation)**.** A nonlinear deterministic ODE is defined as:
>
> $$\frac{d\boldsymbol{h}}{dt} = f(\boldsymbol{h}, t), \tag{1.2}$$
>
> where f is a nonlinear dynamics function. Subject to initial conditions $\boldsymbol{h}(0) = \boldsymbol{h}_0$, the solution $\boldsymbol{h}(t)$ is entirely determined by the dynamics f—assuming it exists.

Note that the form of eq. (1.2) is very general. In particular, the assumptions made about what the state represents and the behaviour of the dynamics function are largely unspec-

ified. Further, although it is given as a first order ODE, it can represent an n-th order ODE by change of variables. For example, a 2nd order ODE is given by

$$\boldsymbol{h}(t) = (\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t)), \quad \dot{\boldsymbol{x}} = \frac{d\boldsymbol{x}}{dt}, \tag{1.3}$$

where $\boldsymbol{x}$ corresponds to position and $\dot{\boldsymbol{x}}$ to velocity.

1.2.2 Geometric Mechanics

The *physical* dynamical systems considered in the thesis have more structure than is apparent in definition 1. This structure is elegantly described by geometric mechanics [Holm et al., 2009]. Geometric mechanics is a formulation of Lagrangian and Hamiltonian mechanics and generally applies for a very large class of physical systems.[5] The thesis only considers finite-dimensional, time-invariant (conservative), N-particle systems to represent models. It is standard to use the notation $\boldsymbol{q}(t) \in Q$ to denote the generalised position of a system, where Q is called the configuration manifold.[6] The thesis only considers simple manifolds, namely $Q = \mathbb{R}^{ND}$ and $Q = SO(2)$.[7]

The notion of generalised coordinates points to a powerful property of geometric mechanics, namely that one can write down the equations in coordinate independent form. For example, one can choose to represent the position of a pendulum either by a single angle or by two positional coordinates in space and the equations describing the evolution would generically look different. Since the laws of physics are the same, it is useful to be able to abstract away this choice (generalised coordinates) by having coordinate independent equations (geometric mechanics).

Hamilton's Principle. Hamilton's principle of stationary action is a particularly beautiful formulation of mechanics [Hamilton, 1837]. In essence, it says that a path between two points

$$C = \{\boldsymbol{q}(t) \colon t \in [a, b], a < b\}, \tag{1.4}$$

acts according to the laws of physics, if it is a stationary point of a functional S,

$$\delta S = 0, \tag{1.5}$$

where S is called the action. Unpacking eq. (1.5) requires some definitions. The action

$$S[\boldsymbol{q}](a, b) = \int_a^b L(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))dt, \tag{1.6}$$

is a functional of the path $\boldsymbol{q}(t)$ between two points in time (a, b), that sums up the contribution of a smooth scalar function called the Lagrangian over the path.[8] Applying

[5] For example, geometric mechanics can be used to represent rigid bodies, fluids, electromagnetic and quantum systems [Holm et al., 2009].

[6] Analogously, $\dot{\boldsymbol{q}}(t) \in \mathbb{R}^{ND}$ denotes the generalised velocity.

[7] The $SO(2)$ manifold is the circle manifold. It can also be thought of in terms of polar coordinates (r, ω) with fixed radius r and varying angle ω.

[8] A functional is a 'function of a function' that returns a scalar.

Hamilton's principle on eq. (1.6), using standard results from the calculus of variation [Gelfand et al., 2000], one obtains the Euler-Lagrange (E-L) equations,

$$\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\Big) - \frac{\partial L}{\partial \boldsymbol{q}} = \mathbf{0}. \tag{1.7}$$

The Lagrangian characterises the physics of the system. The one considered in the thesis is given by,

$$L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T(\dot{\boldsymbol{q}}) - U(\boldsymbol{q}) = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}}\mathbf{M}\dot{\boldsymbol{q}} - U(\boldsymbol{q}), \tag{1.8}$$

where T is the kinetic energy of the system and U is the potential energy. In eq. (1.8), $\mathbf{M} \in \mathbb{R}^{ND\times ND}$ is a diagonal mass matrix

$$\mathbf{M} = \mathrm{diag}(\underbrace{m_1, \ldots, m_1}_{\text{D times}} \ldots, \underbrace{m_N, \ldots, m_N}_{\text{D times}}) = \mathrm{diag}(\mathbf{M}_1, \ldots, \mathbf{M}_N), \tag{1.9}$$

where m_n is the mass of particle n. Using eq. (1.8) in the E-L equations (1.7) gives

$$\mathbf{M}\ddot{\boldsymbol{q}} = \frac{\partial U}{\partial \boldsymbol{q}}, \tag{1.10}$$

which is the familiar form of Newton's second law of motion.[9]

[9] That is, force = mass × acceleration.

An important result by Noether [1918] is that continuous symmetries of the action (1.6) correspond to conservation laws (Noether's Theorem). An important corollary is that if the Lagrangian is invariant with respect to continuous transformations, the system has conservation laws. One immediate consequence is that due to the (explicit) time-invariance of L in (1.8), the energy function

$$E(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{\partial L}{\partial \dot{\boldsymbol{q}}} \cdot \dot{\boldsymbol{q}} - L(\boldsymbol{q}, \dot{\boldsymbol{q}}), \tag{1.11}$$

is a conserved quantity in time [Holm et al., 2009].

1.2.3 Conservation Laws as Inductive Biases

The kinds of problems considered in the thesis are almost entirely captured by thinking about how to learn a model that predicts the evolution of an idealised pendulum, given observations of its state (position and velocity/momentum).[10]

[10] Or multiple independent pendulums in the case of chapter 2

The idealised pendulum is a point-mass swinging from the end of a perfectly rigid rod (fig. 1.1), with no friction due to air resistance. The Lagrangian for the pendulum system is given by

$$L_{\mathrm{p}}(q, \dot{q}) = T_{\mathrm{p}}(\dot{q}) - U_{\mathrm{p}}(q) = \frac{1}{2}ml^2\dot{q}^2 + mgl(1 - \cos q), \quad q \in (-\pi, \pi), \tag{1.12}$$

where q is an angle measuring the displacement from vertical and $\dot{q}$ is the angular velocity; m is the point-mass and l is the length of the rod; and g is the magnitude of the

gravitational field. Using eq. (1.12) in eq. (1.7) gives the equations of motion (EOM) for the pendulum,[11]

$$\ddot{q} - \frac{g}{l}\sin q = 0, \tag{1.13}$$

which is a 2nd order ODE in the angle q. Finally, the energy function for the pendulum is given by eq. (1.11),

$$E_{\mathrm{p}}(q, \dot{q}) = \frac{1}{2}ml^2\dot{q}^2 - mgl(1 - \cos q), \tag{1.14}$$

which is a conserved quantity of the system.

[11] $\frac{\partial L_{\mathrm{p}}}{\partial q} = mgl\sin q,$ $\frac{d}{dt}\left(\frac{\partial L_{\mathrm{p}}}{\partial \dot{q}}\right) = ml^2\ddot{q}.$

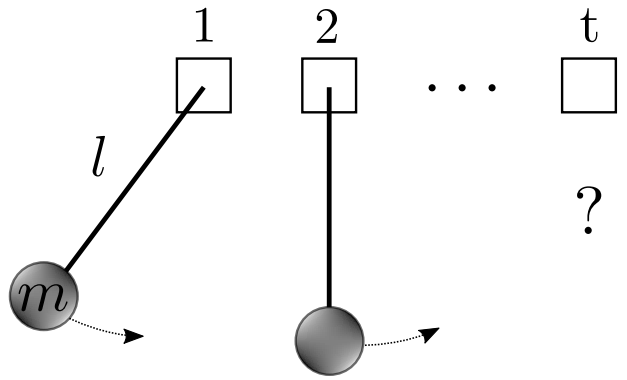


Figure 1.1: The idealised pendulum. The figure illustrates two imagined consecutive snapshots in time, denoted by $1, 2$. The idealised pendulum conserves energy (swings back and forth forever). The length and mass of the pendulum (l, m) are system parameters that affect its behaviour. The problem considered in the thesis is learning a function that can predict the position (and velocity/momentum) of the pendulum at some time t in the future.

The main properties of interest in the pendulum system are:

1. The state dynamics are nonlinear;
2. The system has physical geometric structure.

The first property implies that a learning algorithm needs to be expressive enough to represent the dynamics accurately. However, for accurate predictions beyond the training examples (generalisation), it is crucial that the model contain the geometric structure as well.[12] The structure can either be learned or built in to the predictive model in the form of an inductive bias. Motivated by data efficiency, the latter option is used in the thesis.

Chapter 2 is largely summarised by the most straightforward example of such structure. In the case of the idealised pendulum, the conservation of length and mass.[13] The key idea is to assume that there exist for each instance of a system—or task—a set of conserved system parameters. A good example of different tasks would be many pendulums with different lengths and masses. The idea is then to improve data efficiency by sharing a model of global properties (e.g. laws of physics) and only having to infer the underlying value of the conserved quantities for new tasks.

Chapter 3 is concerned with capturing more general and subtle geometric structures found in systems like the pendulum, one example being the conservation of energy.

[12] To illustrate the point, imagine a predictive model that violates the conservation of energy (geometric property) in the case of the idealised pendulum in such a way that at each step the energy is increased by a small amount. The predicted evolution would then be a pendulum increasing its velocity at each step. Analogous behaviour is demonstrated empirically to occur for standard neural networks in chapter 3.

[13] More generally, one might think of a set of unknown conserved system parameters that affect the dynamics.

More generally, it encodes geometric mechanics or classical physics into neural network architectures. The approach relies on the interpretation of neural networks as the solution to ordinary differential equations [E, 2017], specifically ones arising from classical mechanics. The key idea is to derive network architectures by discretising these physical systems using geometry preserving variational integrators.[14]

[14] Roughly speaking, geometry preserving means physics preserving in this case.

1.3 Function Approximation

A basic problem in machine learning is discovering regularities in a relationship between two vector variables

$$\boldsymbol{x} = (x_q), \quad \boldsymbol{y} = (y_d), \quad q = 1, \dots, Q, \quad d = 1, \dots, D, \tag{1.15}$$

given a collection of observed examples

$$\mathbf{D} = \{\mathbf{X}, \mathbf{Y}\} = \{(\boldsymbol{x}_n), (\boldsymbol{y}_n)\}, \quad n = 1, \dots, N. \tag{1.16}$$

In other words, what structure or pattern remains constant as the values of $\boldsymbol{x}, \boldsymbol{y}$ are varied? Naturally, this can be framed as learning a function $\boldsymbol{y} = f(\boldsymbol{x})$ that captures this regularity, where $\boldsymbol{x}$ has been chosen as the input and $\boldsymbol{y}$ as the output. If a general pattern exists, and it is captured by f, then the function can produce any value of the output given the input—that is, the function can make predictions.

Intuitively, learning a function from examples is a matter of finding the best approximation to the underlying pattern, where best is measured by how closely the learned function matches the true function for all possible inputs (or some subset of interest). The problem is fundamentally approximate in the sense of the aphorism *all models are wrong, but some are useful* [Box, 1976], and it is practically approximate since:

- Data is finite. For modestly complicated functions, there are an infinite number of solutions that perfectly match the observed values but have different behaviour for other points;
- Computational resources are finite. Even if all the points of interest were observed, a complete enumeration of interesting functions is inefficient.[15]

[15] For a binary classification task with megapixel greyscale images as inputs, a complete enumeration of any given function requires $256^{1,000,000}$ numbers [Lin et al., 2017].

This section looks at function approximation using neural networks (NNs) and Gaussian processes (GPs), since both model classes are used in chapters 2–3. Classically, the main difference between NNs and GPs is in how they represent the function f. GPs are inherently probabilistic and define a distribution over the infinite function values, whereas NNs represent the function values implicitly by a finite set of parameters.[16]

[16] This distinction is blurry. For instance, Bayesian NNs (BNNs) [MacKay, 1995] are probabilistic variants of NNs. Neural processes (NPs) sit somewhere inbetween GPs and NNs [Garnelo et al., 2018]. Further, NNs with i.i.d. Gaussian priors over the weights are equivalent to a certain class of GPs [Neal, 1996, Lee et al., 2017].

1.3.1 Probabilistic Machine Learning

Learning is framed probabilistically in the thesis. In this section, the inputs $\boldsymbol{x}$ are assumed to be observed and the outputs $\boldsymbol{y}$ are modelled as independent random variables by an observation model.

Definition 2 (Observation Model)**.** An observation model assigns a probability to any output vector $\boldsymbol{y}$. It is defined as a conditional distribution

$$p_\theta(\boldsymbol{y}|f(\boldsymbol{x})), \tag{1.17}$$

parametrised by a set of model parameters θ (notation 3) and indexed by input vectors $\boldsymbol{x} \in \mathbb{R}^Q$. The observation function in (1.17) is defined as the set

$$f(\boldsymbol{x}) = \{f(\boldsymbol{x}_i)|\boldsymbol{x}_i \in \mathbb{R}^Q\}, \tag{1.18}$$

where the vector-valued function values are defined as

$$\boldsymbol{f}_i = f(\boldsymbol{x}_i) = (f^d(\boldsymbol{x}_i)), \quad d = 1, \ldots, D. \tag{1.19}$$

The observations from eq. (1.16) are assumed to be independent samples from a set of unknown distributions

$$\boldsymbol{y}_n \sim p_{\theta^\star}(\boldsymbol{y}|f(\boldsymbol{x}_n)) = p_{\theta^\star}(\boldsymbol{y}|\boldsymbol{f}_n), \tag{1.20}$$

indexed by the inputs $\boldsymbol{x}_n \in \mathbf{X}$, where $\theta^\star$ denotes the true state of nature. That is, the data were really generated by $\theta^\star$.

[17] Some care must be taken to ensure that these are reasonably calibrated [Kuleshov et al., 2018]. This is one reason one might choose Bayesian methods such as GPs.

[18] For example, complicated simulators [Cranmer et al., 2020a] and image compression models [Ballé et al., 2018] often have intractable/ill-defined observation densities.

Assuming a distribution in (1.17) gives a natural way to model noise in the output variable explicitly, i.e. variation in the output that is not explained by the function values.[17] However, requiring an explicit form of the distribution can be restrictive.[18]

Output Dimensions. The thesis only considers two instances of the distribution in (1.17). For real-valued outputs $\boldsymbol{y} \in \mathbb{R}^D$, it will be a factorised Gaussian[19]

$$p_\theta(\boldsymbol{y}|f(\boldsymbol{x})) = \prod_{d=1}^{D} \mathcal{N}(y_d|f^d(\boldsymbol{x}), \sigma_d^2) = (2\pi)^{-\frac{D}{2}} \prod_{d=1}^{D} \sigma_d^{-1} \exp\left(-\frac{1}{2\sigma_d^2}(\boldsymbol{y} - f^d(\boldsymbol{x}))^2\right), \tag{1.21}$$

[19] $\mathcal{N}(\boldsymbol{y}|\cdot)$ denotes the Gaussian distribution evaluated at $\boldsymbol{y}$, whereas $\mathcal{N}(\cdot)$ defines a Gaussian random variable.

where $f^d(\boldsymbol{x})$ models the mean and σ_d^2 is a noise variance parameter. For binary outputs $\boldsymbol{y} \in [0,1]^D$, used to represent images in the thesis, it will be a factorised Bernoulli

$$p_\theta(\boldsymbol{y}|f(\boldsymbol{x})) = \prod_{d=1}^{D} \mathcal{B}(y_d|f^d(\boldsymbol{x})) = \prod_{d=1}^{D} \left(f^d(\boldsymbol{x})\right)^{y_d} (1 - f^d(\boldsymbol{x}))^{(1-y_d)}, \tag{1.22}$$

where $f^d(\boldsymbol{x})$ specifies the probability of $y_d = 1$.

Bayesian Learning. For probabilistic models, learning can be motivated by a Bayesian perspective.

Definition 3 (Bayesian Learning)**.** Given a dataset $\mathbf{D}$ and a model class θ, Bayes' theorem [Bayes, 1958] says

$$\underbrace{p(\theta|\mathbf{D})}_{\text{posterior}} \propto \underbrace{p(\mathbf{D}|\theta)p(\theta)}_{\text{likelihood}\times\text{prior}}, \tag{1.23}$$

where the *posterior* is the conditional distribution over models given the data (belief given data); the *likelihood* is the conditional distribution over data given a model (relating models to data); and the prior captures assumptions about the model class before observing data (belief prior to data). The constant of proportionality is the marginal likelihood

$$p(\mathbf{D}) = \int p(\mathbf{D}|\theta)p(\theta)d\theta. \tag{1.24}$$

Bayesian learning identifies the process of learning with the transformation of the prior distribution into a posterior distribution. In other words, it returns a posterior distribution over models. Two key assumptions made in Bayesian learning are:

- The data was really generated by some model $\theta^\star \in \theta$ with likelihood $p(\mathbf{D}|\theta^\star)$.
- The prior accurately reflects the prior beliefs about the model class.

Generally speaking, a Bayesian perspective on learning is both conceptually and practically attractive since it models uncertainty, and probability is the language of much of science and engineering [Ghahramani, 2015, Murphy, 2021]. Importantly, it gives a principled approach to incorporating prior knowledge (in both the prior and likelihood) and to model selection via the marginal likelihood.

At the same time, exact posterior inference is difficult in practice due to the integral in the marginal likelihood (1.24), which requires integrating over the entire model space.[20] Another issue is that incorporating prior knowledge can be challenging for practically relevant models. Using misspecified priors and/or likelihoods can result in arbitrarily bad posterior inference [Knoblauch et al., 2019].

[20] GPs are one class of models which allow for tractable Bayesian inference, assuming an independent Gaussian likelihood.

[21] Since the marginal likelihood (1.24) does not depend on θ it can be ignored. Further, the log is monotonically increasing so the maximum/minimum occurs at the same point.

Point Estimates. Alternatively, in cases where only a point estimate is desired, definition 3 also points to a form for the learning objective. The maximum a posteriori (MAP) estimate is obtained by maximising the posterior with respect to the model class. In the context of eq. (1.17), this is equivalent to minimising the negative log-posterior[21]

$$\mathcal{O}_{MAP}(\theta; \mathbf{Y}, \mathbf{X}) = -\left(\left(\sum_{n=1}^{N} \log p_\theta(\boldsymbol{y}_n|f(\boldsymbol{x}_n))\right) + \log p(\theta)\right), \tag{1.25}$$

where $\mathcal{O}_{MAP}$ is the maximum a posteriori (MAP) objective and the semicolon is used to make the implicit dependence on the data explicit. The term $p(\theta)$ is a prior distribution

over the model parameters, and the log-prior acts as a regularisation term, typically preferring simpler models.[22] The maximum likelihood (ML) estimate is obtained by assuming a uniform prior in eq. (1.25),

$$\mathcal{O}_{ML}(\theta; \mathbf{Y}, \mathbf{X}) = -\sum_{n=1}^{N} \log p_\theta(\boldsymbol{y}_n | f(\boldsymbol{x}_n)). \tag{1.26}$$

[22] By some suitable definition of simple. A canonical example is a Gaussian prior, which can be seen as the squared L2 norm. That is,

$$\log p(\theta) \propto \sum_i ||v_i||_2^2, \ \forall v_i \in \theta.$$

A similar relationship exists for the L1 norm and the Laplace prior.

Uninformative prior distributions (e.g. uniform) are known to have issues from a Bayesian perspective [Gelman and Yao, 2021], but the ML estimate has attractive asymptotic properties. If the data were really generated by a model $\theta^\star$, then the ML estimate $\widehat{\theta}$ converges (in probability) to the true value in the limit of infinite samples. In practice:

- Data is not actually generated by the model;
- Finite datasets and large number of model parameters;
- True model is unidentifiable (no unique $\theta^\star$).

Despite these issues, assuming enough data, the ML estimate can work well in practice for some or all of the parameters.[23]

[23] Highly dependent on specifics of the model and data.

Stochastic Gradient Descent. Minimising the objectives from eqs. (1.25)–(1.26) can be done with gradient descent (GD), by making small updates to the model parameters θ in the opposite direction of the gradient. Naively, this involves computing the sum over the entire dataset which is often intractable for large datasets. Stochastic (mini-batch) gradient descent (SGD) [Ruder, 2016] can be used to estimate the gradient instead. SGD requires that the objective decompose into a sum over samples, for example, due to an independence assumption like the one in definition 2. Instead of computing the full sum, the true gradient is estimated as

$$\sum_{n=1}^{N} \nabla_\theta \mathcal{O}(\theta; \boldsymbol{y}_n, \boldsymbol{x}_n) \approx \sum_{b=1}^{B} \nabla_\theta \mathcal{O}(\theta; \boldsymbol{y}_b, \boldsymbol{x}_b), \tag{1.27}$$

where ∇_θ denotes the gradient with respect to θ, and $\mathcal{O}(\theta; \boldsymbol{y}, \boldsymbol{x})$ is a per-sample loss term. Note that $(\boldsymbol{y}_b, \boldsymbol{x}_b)$ are samples from the dataset, collected in a mini-batch of size $B < N$.[24] Typically these are sampled uniformly at random from the dataset. The model parameters are then updated with a fixed learning rate using the estimate in (1.27).

[24] The size of the mini-batch B determines the amount of noise in the gradient estimate, which is balanced against the computational cost. That is, larger B means less noise but more compute.

Using a fixed learning rate has well documented problems.[25] In practice, numerous stochastic optimisation algorithms have been developed to deal with these challenges. The popular Adam algorithm [Kingma and Ba, 2017] is used in the thesis.

[25] E.g. choosing the right learning rate, or the right annealing schedule, and getting stuck in suboptimal local minima [Ruder, 2016]

1.3.2 Neural Networks

Definition 4. Neural networks (NNs) are nonlinear parametric function approximators made up of a composition of simpler functions

$$f_\theta = f_\theta^K \circ \ldots \circ f_\theta^1, \quad K \geqslant 2 \tag{1.28}$$

where $k = 1, \ldots, K$ indexes layers in the network, and the dependence on the model parameters has been made explicit. In the simplest case, $K = 2$ and (1.28) is called a shallow or single-layer neural network (NN) and otherwise it is a deep neural network (DNN). In addition to the number of layers K, a basic configuration parameter in neural networks is the width of each layer, i.e. the dimension of the output. In general, the output of a NN layer will be denoted by

$$\boldsymbol{h}_k = f_\theta^k(\boldsymbol{h}_{k-1}) \in \mathbb{R}^{W_k}, \quad \boldsymbol{h}_0 = \boldsymbol{x}, \quad \boldsymbol{h}_K = \boldsymbol{y}, \tag{1.29}$$

where $\boldsymbol{h}_k$ is the *hidden state* at layer k (excluding the input and output), and each layer will generally have a different dimension W_k.

A basic form for eq. (1.28) is the fully-connected neural network architecture (FCNN), going back at least as far as the work in [Ivakhnenko and Lapa, 1966, Berners-Lee, 1968].[26] The layers are given by a linear transform followed by a simple nonlinearity

[26] See Schmidhuber [2014], section 5.3 for more detail.

$$f_\theta^k(\boldsymbol{h}_{k-1}) = a^k\big(\mathbf{W}^k \boldsymbol{h}_{k-1} + \boldsymbol{b}^k\big), \tag{1.30}$$

where a^k is the nonlinear activation function (acting pointwise) for layer k, $\mathbf{W}^k$ is the weight matrix for layer k and $\boldsymbol{b}^k$ is its bias term. The weight matrices and bias terms for all layers are the parameters of the network (collected in θ).

A common choice of activation function is the ReLU [Nair and Hinton, 2010]

$$\mathtt{ReLU}(\boldsymbol{x}) = \max(\boldsymbol{0}, \boldsymbol{x}), \tag{1.31}$$

where the max is taken element-wise. The output activation a^K is often different from the rest, or taken to be the identity, depending on the nature of the outputs.

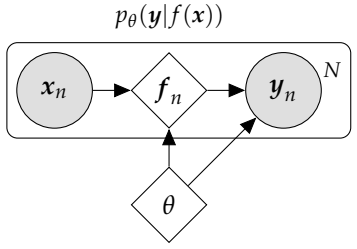


Figure 1.2: Graphical model (notation 6) for the observation model from definition 2, using a NN observation function. The model parameters θ parametrise a NN $f_\theta(\boldsymbol{x})$. The function values $\boldsymbol{f}_n = f_\theta(\boldsymbol{x}_n)$ are deterministic variables.

1.3.3 Gaussian Processes

Intuitively, a GP can be thought of as a distribution over a scalar function f.[27] Formally, GPs consist of an infinite collection of random variables

$$f(\boldsymbol{x}) = \{f(\boldsymbol{x}_i) | \boldsymbol{x}_i \in \mathcal{X}\}, \tag{1.32}$$

where $\mathcal{X}$ is an index set, and $f(\boldsymbol{x}_i)$ is a random variable assigned to each element $\boldsymbol{x}_i \in \mathcal{X}$. The defining characteristic of GPs is that any finite subset of these variables is jointly Gaussian [Rasmussen and Williams, 2006]. Rather than manipulate the infinite set in (1.32), GPs can be represented by a pair of functions on the indices $\boldsymbol{x} \in \mathcal{X}$. These are the mean function

$$\mu(\boldsymbol{x}_i) = \mathbb{E}\big[f(\boldsymbol{x}_i)\big], \tag{1.33}$$

which gives the expected value of the random variable $f(\boldsymbol{x}_i)$, and the positive definite covariance function

$$\Sigma(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbb{E}\big[(f(\boldsymbol{x}_i) - \mu(\boldsymbol{x}_i))(f(\boldsymbol{x}_j) - \mu(\boldsymbol{x}_j))\big], \tag{1.34}$$

which outputs the covariance between $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$.[28]

Throughout the thesis, the notation $p(f(\boldsymbol{x}))$ is used *as if* it were a density over the infinite set of random variables. While this is not technically correct, the computations will always reduce to a finite set, and the notation is intuitive. A proper treatment of the subject is found in [Matthews et al., 2016].

[27] They can also be seen as infinitely wide and deep neural networks assuming an i.i.d. Gaussian prior on the parameters of the network. This was shown by Neal [1996] for infinitely wide single-layer networks and Lee et al. [2017] in the infinitely wide and deep case.

[28] To be clear, both $\mu(\cdot)$ and $\Sigma(\cdot, \cdot)$ are actual functions whereas $f(\boldsymbol{x}_i)$ is a label for a random variable in the infinite set from eq. (1.32).

Definition 5 (Gaussian Process)**.** Using eqs. (1.33)–(1.34), a GP is defined as

$$p_\theta(f(\boldsymbol{x})) = \mathcal{GP}(\mu_\theta(\boldsymbol{x}), \Sigma_\theta(\boldsymbol{x}, \boldsymbol{x}')), \tag{1.35}$$

where $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ and μ_θ and Σ_θ are parametrised by θ (hyperparameters). For an arbitrary subset of indices $\mathbf{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$, eq. (1.35) says that

$$p_\theta(\boldsymbol{f}_{(n)}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}), \tag{1.36}$$

where $\boldsymbol{f}_{(n)} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N))$ and the conditioning on $\mathbf{X}$ is not included for ease of notation. This will be the convention throughout the thesis, and the index set will be clear from context. The mean vector and covariance matrix are defined as

$$\boldsymbol{\mu}_n = \mu_\theta(\boldsymbol{x}_n), \tag{1.37}$$

$$\mathbf{K}_{[m,n]} = \Sigma_\theta(\boldsymbol{x}_m, \boldsymbol{x}_n), \tag{1.38}$$

where $\boldsymbol{\mu} \in \mathbb{R}^N$, $\mathbf{K} \in \mathbb{R}^{N \times N}$, and $\boldsymbol{x}_m, \boldsymbol{x}_n \in \mathbf{X}$.

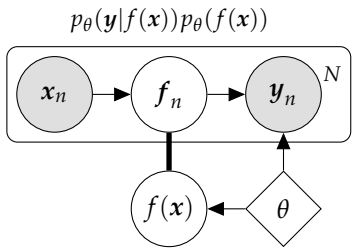


Figure 1.3: Graphical model (notation 6) for the observation model from definition 2, using a GP observation function. The undirected thick line denotes that $\boldsymbol{f}_n \in f(\boldsymbol{x})$. GPs model the function values as latent random variables.

Mean And Covariance Functions. A useful property of GPs is that global features of the function can be specified through interpretable hyperparameters in the mean and covariance functions. For the purposes of the thesis, the mean function will be taken to be the zero-function[29]

$$\mu_\theta(\boldsymbol{x}) = \mathbf{0}. \tag{1.39}$$

The covariance function, will be taken to be the exponentiated quadratic (EQ) kernel[30]

$$\Sigma_\theta(\boldsymbol{x}, \boldsymbol{x}') = \sigma_k^2 \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^{\mathrm{T}} \mathbf{L}^{-1} (\boldsymbol{x} - \boldsymbol{x}') \right) \tag{1.40}$$

$$= \sigma_k^2 \exp\left(-\frac{1}{2} \sum_{q=1}^{Q} \frac{(x_q - x_q')^2}{l_q^2} \right), \tag{1.41}$$

where σ_k^2 is a signal variance parameter, and $\mathbf{L} = \mathrm{diag}(l_1^2, \ldots, l_q^2)$ is a diagonal matrix with entries called the characteristic lengthscales.

Posterior GP. Equation (1.35) can be used as a prior over functions in a Bayesian learning setting (definition 3). In this case, learning the function is achieved by computing the posterior distribution over the function. In contrast to the notation in definition 3, it is the function values $f(\boldsymbol{x})$ that are treated as latent variables that define the model class. The model parameters θ (technically hyperparameters) will be learned by maximum likelihood. This section reviews posterior inference for GPs when the likelihood is Gaussian, eq. (1.21), such that inference is analytically tractable.

Analytic inference (and variational inference, section 1.4) takes advantage of the fact that conditioning a GP on a subset of function values results in another GP.[31]

Definition 6 (Conditional GP)**.** Given an arbitrary subset of random function values $\boldsymbol{f}_{(n)}$, and denoting the rest of the infinite set from (1.32) by $f(\boldsymbol{x}_\star)$, their joint distribution is given by

$$p_\theta(f(\boldsymbol{x}_\star), \boldsymbol{f}_{(n)}) = p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(n)}) p_\theta(\boldsymbol{f}_{(n)}), \tag{1.42}$$

[29] There are many situations where one might want to include a mean function, since it is a natural place to add prior knowledge. For the systems modelled in the thesis, the zero-mean function is used since the function models changes in state variables.

[30] Often referred to as squared exponential (SE) kernel or radial basis function (RBF) kernel

[31] The result follows from considering standard properties of multivariate Gaussian distributions. Namely, conditioning one of the variables on the rest results in a Gaussian. If this variable were indexed by $\boldsymbol{x}_\star$, then the mean and covariance would be given by equations (1.44)–(1.45), with the argument $\boldsymbol{x}$ replaced by $\boldsymbol{x}_\star$.

using the definition of conditional probability. Using standard properties of Gaussian distributions [Rasmussen and Williams, 2006], the conditional

$$p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(n)}) = \mathcal{GP}(\mu^{\mathrm{c}}_\theta(\boldsymbol{x}), \Sigma^{\mathrm{c}}_\theta(\boldsymbol{x}, \boldsymbol{x}')), \tag{1.43}$$

is a GP with mean and covariance functions given by

$$\mu^{\mathrm{c}}_\theta(\boldsymbol{x}) = \mu_\theta(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^{\mathrm{T}}\mathbf{K}^{-1}(\boldsymbol{f}_{(n)} - \boldsymbol{\mu}), \tag{1.44}$$

$$\Sigma^{\mathrm{c}}_\theta(\boldsymbol{x}, \boldsymbol{x}') = \Sigma_\theta(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}(\boldsymbol{x})^{\mathrm{T}}\mathbf{K}^{-1}\boldsymbol{k}(\boldsymbol{x}'), \tag{1.45}$$

where $[\boldsymbol{k}(\boldsymbol{x})]_n = \Sigma_\theta(\boldsymbol{x}_n, \boldsymbol{x})$ is a (column) vector of covariances between the function values at the inputs $\boldsymbol{x}_n$ and the argument $\boldsymbol{x}$.

Using the likelihood from eq. (1.21) and the partitioning from eq. (1.42), the joint distribution over the observations and the function is given by

$$p_\theta(\boldsymbol{y}_{(n)}, \boldsymbol{f}_{(n)}, f(\boldsymbol{x}_\star)) = p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(n)})p_\theta(\boldsymbol{y}_{(n)}, \boldsymbol{f}_{(n)}) \tag{1.46}$$

$$= p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(n)})p_\theta(\boldsymbol{f}_{(n)}|\boldsymbol{y}_{(n)})p_\theta(\boldsymbol{y}_{(n)}), \tag{1.47}$$

where the second step uses the definition of conditional probability to pull $p_\theta(\boldsymbol{y}_{(n)})$ out of the joint distribution. Using (1.47) in Bayes' theorem (1.23) gives[32]

$$p_\theta(f(\boldsymbol{x})|\boldsymbol{y}_{(n)}) = p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(n)})p_\theta(\boldsymbol{f}_{(n)}|\boldsymbol{y}_{(n)}), \tag{1.48}$$

where the first term on the RHS is a GP (definition 6) and the second term is a Gaussian over $\boldsymbol{f}_{(n)}$. The latter statement follows from the fact that the Gaussian family is self-conjugate and the likelihood is Gaussian.

[32] Note that: $f(\boldsymbol{x}) = f(\boldsymbol{x}_\star) \cup \boldsymbol{f}_{(n)}$

Since the mean function in eq. (1.44) for the conditional $p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(n)})$ is linear in $\boldsymbol{f}_{(n)}$, one can analytically marginalise out $\boldsymbol{f}_{(n)}$ from eq. (1.48) [Rasmussen and Williams, 2006]. When μ_θ is the zero function from eq. (1.39), the resulting posterior GP is given by

$$p(f(\boldsymbol{x})|\boldsymbol{y}_{(n)}) = \mathcal{GP}(\mu^{\mathrm{p}}_\theta(\boldsymbol{x}), \Sigma^{\mathrm{p}}_\theta(\boldsymbol{x}, \boldsymbol{x}')), \tag{1.49}$$

with mean and covariance functions given by

$$\mu^{\mathrm{p}}_\theta(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^{\mathrm{T}}\big(\mathbf{K} + \sigma^2\mathbf{1}_N\big)^{-1}\boldsymbol{y}_{(n)}, \tag{1.50}$$

$$\Sigma^{\mathrm{p}}_\theta(\boldsymbol{x}, \boldsymbol{x}') = \Sigma_\theta(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}(\boldsymbol{x})^{\mathrm{T}}\big(\mathbf{K} + \sigma^2\mathbf{1}_N\big)^{-1}\boldsymbol{k}(\boldsymbol{x}'). \tag{1.51}$$

Learning GP Hyperparameters. GPs model the relationships between different observations explicitly in the distribution over function values. Learning is partly accomplished by computing the posterior distribution (1.49), which describes how the infinite set of function values in relate to each other given the observations.

Typically, the hyperparameters that specify the mean and covariance function are learned from data also. This can be achieved by obtaining a ML estimate (1.26) of the hyperparameters, however, the function values have to be marginalised out. The objective is then the log-marginal-likelihood (LML)

$$\mathcal{O}_{GP}(\theta; \boldsymbol{y}_{(n)}) = \log p_\theta(\boldsymbol{y}_{(n)}) = \log \int p_\theta(\boldsymbol{y}_{(n)}|\boldsymbol{f}_{(n)}) p_\theta(\boldsymbol{f}_{(n)}) d\boldsymbol{f}_{(n)}. \tag{1.52}$$

In the case of i.i.d. Gaussian likelihood, eq. (1.52) is available in closed form since it is a Gaussian[33]

$$\log p_\theta(\boldsymbol{y}_{(n)}) = -\frac{1}{2}\boldsymbol{y}_{(n)}^T(\mathbf{K} + \sigma^2\mathbf{1}_N)^{-1}\boldsymbol{y}_{(n)} - \frac{1}{2}\log|\mathbf{K} + \sigma^2\mathbf{1}_N| - \frac{N}{2}\log 2\pi, \tag{1.53}$$

where $\mathbf{K}$ is given in (1.38) and σ^2 is the noise variance from (1.21).

[33] The integral in (1.52) can be done by hand using standard identities for the product of Gaussian functions. However, by the definition of the model:

$$\boldsymbol{y}_{(n)} = \boldsymbol{f}_{(n)} + \boldsymbol{\epsilon}_{(n)},$$
$$\boldsymbol{\epsilon}_{(n)} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{1}_N).$$

The result follows immediately since the sum of two independent Gaussian random variables is a Gaussian

$$\boldsymbol{y}_{(n)} \sim \mathcal{N}(\boldsymbol{f}_{(n)}, \mathbf{K} + \sigma^2\mathbf{1}_N).$$

1.4 Latent Variable Models

A more general problem than that of section 1.3 is discovering regularities and hidden structure in a vector variable $\boldsymbol{y}$, given a collection of observed examples

$$\boldsymbol{y} = (y_d), \quad d = 1, \ldots, D, \tag{1.54}$$

$$\mathbf{Y} = (\boldsymbol{y}_n), \quad n = 1, \ldots, N. \tag{1.55}$$

Many interesting learning problems are naturally framed in this way, for example: generating new observations that are similar to the observed examples but usefully different,[34] and data compression [Soliman and Omari, 2006, Ballé et al., 2018].

[34] Notable examples include: drug design [Gupta et al., 2018b] music generation [Oord et al., 2016], text generation [Collobert and Weston, 2008], image generation [Goodfellow et al., 2020].

These patterns are still usefully captured by a function $f(\boldsymbol{x})$, but generally speaking, the input space is unknown and needs to be learned by the model. The section begins by defining generative models by treating the inputs from section 1.3 as latent variables in a probabilistic model. The subsequent section introduces variational inference for approximate inference in latent variable models that have intractable posterior distributions and marginal likelihoods. Finally, variational inference is applied to the GP model introduced in section 1.3 to derive sparse variational Gaussian processes (SVGPs).

1.4.1 Latent Variable Models

The inputs $\boldsymbol{x}$ are modelled as independent and identically distributed (i.i.d.) random variables, and the outputs $\boldsymbol{y}$ are modelled by the observation model from definition 2. The joint distribution over the inputs and outputs defines a generative model.

Since the function is modelled by a GP or NN (section 1.3), the marginal distribution

over the outputs—$p_\theta(\boldsymbol{y})$—will generally be complex even if the conditional distribution over observations is simple. In the thesis, the form of the conditional distribution will be Gaussian or Bernoulli as described in equations (1.21)–(1.22).[35]

[35] This may still be a restrictive assumption for certain applications, as evidenced by the popularity of generative adversarial networks (GANs) [Goodfellow et al., 2020] which do not enforce an explicit form for the conditional.

Definition 7 (Generative Model)**.** A generative model is defined as a joint distribution over input vectors $\boldsymbol{x}$ and output vectors $\boldsymbol{y}$, indexed by a set of model parameters (notation 3),

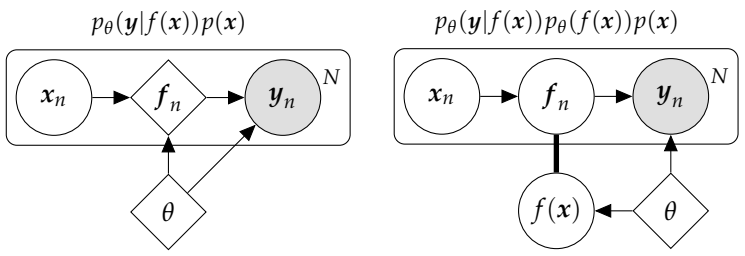
$$p_\theta(\boldsymbol{y}, \boldsymbol{x}) = p_\theta(\boldsymbol{y}|f(\boldsymbol{x}))p(\boldsymbol{x}), \tag{1.56}$$

where the conditional distribution in (1.56) is the observation model from eq. (1.17), and $p(\boldsymbol{x})$ is the prior distribution (definition 3) over the latent (unknown) inputs.

The observations from eq. (1.55) are assumed to be independent samples from a generative process described by

$$\boldsymbol{x}_n \sim p(\boldsymbol{x}), \quad \boldsymbol{y}_n \sim p_{\theta^\star}(\boldsymbol{y}|f(\boldsymbol{x}_n)) = p_{\theta^\star}(\boldsymbol{y}|\boldsymbol{f}_n), \tag{1.57}$$

where $\theta^\star$ denotes the true state of nature.



(a) Neural networks (NNs). **(b)** Gaussian processes (GPs).

Figure 1.4: Graphical models (notation 6) for the generative model in definition 7 where the observation function is modelled by (a) a NN and (b) a GP. The undirected thick line in (b) denotes that $\boldsymbol{f}_n \in f(\boldsymbol{x})$. In contrast to section 1.3, the inputs $\boldsymbol{x}_n$ are latent variables.

Learning Latent Variable Models. The generative model defined in (1.56) is also called a latent variable model (LVM) due to the latent inputs $\boldsymbol{x}$. LVMs are learned by maximising the log-marginal-likelihood (LML)

$$\mathcal{O}_{LML}(\theta; \boldsymbol{y}) = \log p_\theta(\boldsymbol{y}) = \int p_\theta(\boldsymbol{y}|f(\boldsymbol{x}))p_\theta(\boldsymbol{x})d\boldsymbol{x} \tag{1.58}$$

with respect to the model parameters θ, using for instance stochastic gradient descent (SGD) eq. (1.27). Typically, however, the LML is intractable due to the function f, which means that the posterior distribution over the inputs—needed for downstream processing, analysis or prediction—is intractable also. The posterior is given by Bayes' theorem (1.23)

$$p_\theta(\boldsymbol{x}|\boldsymbol{y}) = \frac{p_\theta(\boldsymbol{y}|f(\boldsymbol{x}))p_\theta(\boldsymbol{x})}{p_\theta(\boldsymbol{y})}, \tag{1.59}$$

which involves $p_\theta(\boldsymbol{y})$ in the denominator. Section 1.4.2 introduces variational inference

as a strategy for dealing with both eqs. (1.58)–(1.59).

Prior. The prior over the latent inputs plays an important role since it encodes prior assumptions about the structure of the latent space, and acts as regularising term in the objective for variational inference. In the thesis, the prior distribution is generally assumed to be a simple isotropic Gaussian

$$p(\boldsymbol{x}) = \mathcal{N}(\mathbf{0}, \mathbf{1}_Q), \tag{1.60}$$

where $\mathbf{1}_Q$ is an identity matrix (notation 4).

The main advantage of eq. (1.60) is that it is easy to work with, having closed form solutions for information theoretic quantities and being easy to sample. Another possible motivation is that it encourages simple, independent sources of variation due to the independence assumption.

However, it has been pointed out by numerous works that the prior in eq. (1.60) may be too restrictive for certain domains [Chen et al., 2016, Hoffman and Johnson, 2018, Lavda et al., 2020], leading to degraded performance. A partial explanation is that the regularisation effect the prior in (1.60) produces is too strong, since the true posterior is decidedly not independent in typical cases. However, this effect is more pronounced for larger latent spaces [Hoffman and Johnson, 2018], and the latent spaces considered in the thesis are relatively low dimensional.[36]

[36] From the perspective of Bayesian learning (definition 3), it would be considered bad practice to assume independence if the latent dimensions are *known* to be highly correlated a priori. For complicated generative models (expressive f), however, this can be difficult to reason about.

1.4.2 Variational Inference

This subsection introduces variational inference (VI) [Anderson and Peterson, 1987, Hinton and Van Camp, 1993, Jaakkola and Jordan, 1999], a remarkably useful tool for approximate inference, and integral to contemporary machine learning [Kingma and Welling, 2014, Jimenez Rezende et al., 2014, Blei et al., 2017, Zhang et al., 2018].

- Fundamentally, VI turns integration into optimisation—or an infinite-dimensional optimisation problem into a finite one—making otherwise intractable probabilistic learning algorithms highly scalable, particularly when combined with SGD (1.27). The approach is generally applicable to most probabilistic models and makes many of the advantages of Bayesian learning (definition 3) readily available, such as access to (a lower-bound of) the marginal likelihood for learning and model selection, and uncertainty estimates in the (approximate) posterior.
- The main limitations of VI include the bias of the approximate posterior: which is

not guaranteed to converge to the true posterior [Blei et al., 2017], is known to underestimate the variance [Giordano et al., 2017], and can bias estimation of the model parameters [Cremer et al., 2018]. Additionally, the approximate posterior can be difficult to diagnose [Yao et al., 2018]. VI is an active area of research and these limitations have not been left unaddressed.[37]

[37] Some examples include more flexible approximate posteriors [Jimenez Rezende and Mohamed, 2015], tighter lower-bounds [Burda et al., 2015] better understanding of when tigher lower-bounds are better [Rainforth et al.], new perspectives on the lower-bound [Hoffman and Johnson, 2018], to name a few. A recent review is given by Zhang et al. [2018].

Bayesian Inference as Optimisation. There are many technically equivalent derivations of VI, offering different perspectives on the learning objective. This section starts with the viewpoint in [Knoblauch et al., 2019], and subsequently explores different interpretations of the objective that exist in the literature.

Knoblauch et al. [2019] point out that work dating back to at least [Csiszár, 1975, Donsker and Varadhan, 1983] had formulated *exact* Bayesian inference as a solution to an infinite-dimensional optimisation problem. Focusing on intuition, the result is stated here in the context of the posterior (1.59) and the dataset in (1.55), where the dependence on the model parameters has been dropped. Specifically, the posterior over the latent inputs $\mathbf{X}$ (given the observations $\mathbf{Y}$) is given by the solution to

$$p(\mathbf{X}|\mathbf{Y}) = \underset{q \in \mathcal{P}(\mathcal{X})}{\arg\max} \left[\mathbb{E}_{q^{\mathbf{X}}} \Big[\log p(\mathbf{Y}|\mathbf{X}) \Big] - \mathbb{KL}\big[q^{\mathbf{X}} || p^{\mathbf{X}}\big] \right], \tag{1.61}$$

where $\mathcal{P}(\mathcal{X})$ is the space of all probability distributions on $\mathcal{X}$, $q \in \mathcal{P}(\mathcal{X})$ is the optimisation variable (a distribution), and the second term denotes the Kullback-Leibler divergence [Kullback and Leibler, 1951]

$$\mathbb{KL}\big[q^{\boldsymbol{a}} || p^{\boldsymbol{a}}\big] = \mathbb{E}_{q^{\boldsymbol{a}}} \Big[\log \frac{q(\boldsymbol{a})}{p(\boldsymbol{a})} \Big] \tag{1.62}$$

between q and p, and $q^{\mathbf{X}} = q(\mathbf{X})$ is the shorthand defined in notation 5.

The key point is this: Bayes' theorem is equivalent to the optimisation in (1.61) in the sense that a solution corresponds to the posterior given by the theorem. To be clear—no approximation has been made, it is simply a rephrasing of Bayes' theorem.

Equation (1.61) has an intuitive interpretation: it consists of a log-likelihood term that fits the observations, and a regularisation term that minimises the KL-distance between the optimised posterior $q(\mathbf{X})$ and the prior $p(\mathbf{X})$. This is analogous to Bayes' theorem, where the posterior is proportional to the product (sum in log space) of the likelihood terms and the prior.

Approximate Inference as Optimisation. The core idea of VI is to approximate the true posterior with a parametric family of distributions $\{q_\phi | \phi \in \Phi\}$, having variational

parameters ϕ (notation 3) living in a finite-dimensional space Φ. This variational posterior is denoted by

$$q_\phi^{\boldsymbol{x}} = q_\phi(\boldsymbol{x}) \approx p_\theta(\boldsymbol{x}|\boldsymbol{y}) \tag{1.63}$$

and is optimised to fit the true posterior. Intuitively, this approximation will be bias if the variational family does not contain the true posterior. However, the flexibility of the family has to be balanced with tractability, since this is the purpose of VI. A popular and simple assumption, originating from physics [Opper and Saad, 2001], is the mean-field variational family [Blei et al., 2017]

$$q_\phi(\mathbf{X}) = \prod_{n=1}^{N} q_\phi(\boldsymbol{x}_n), \tag{1.64}$$

which assumes that all the latent variables are independent and governed by unique variational parameters $\phi_n \in \phi$.[38] Equation (1.64) is a strong assumption that is unlikely to hold for the true posterior in general. This can cause significant problems, such as underestimating the variance and biasing the estimate of the model parameters θ. The advantage of the mean-field family is primarily tractability, which is the motivation in the thesis.[39] In the context of (1.61), a parametric approximation q_ϕ to the posterior immediately leads to the first perspective on VI.

[38] Technically, eq. (1.64) has not assumed independence across the dimensions of $\boldsymbol{x} \in \mathbb{R}^Q$.

[39] Later in the section, a more structured approximation is introduced. However, the independence across marginal distributions is still assumed.

Finite Dimensional Approximation. VI replaces the infinite-dimensional optimisation problem in (1.61), i.e. exact Bayesian inference, with a finite dimensional optimisation problem over the family q_ϕ. Introducing the model and variational parameters θ, ϕ, the VI learning objective for the generative model in (1.56) is given by

$$\mathcal{O}_{ELBO}(\theta, \phi; \mathbf{Y}) = \mathbb{E}_{q_\phi^{\mathbf{X}}}\Big[\log p_\theta(\mathbf{Y}|\mathbf{X})\Big] - \mathbb{KL}\big[q_\phi^{\mathbf{X}} || p_\theta^{\mathbf{X}}\big] \tag{1.65}$$

$$= \left(\sum_{n=1}^{N} \mathbb{E}_{q_\phi^{\boldsymbol{x}_n}}\big[\log p_\theta(\boldsymbol{y}_n|f(\boldsymbol{x}_n))\big] - \mathbb{KL}\big[q_\phi^{\boldsymbol{x}_n} || p_\theta^{\boldsymbol{x}_n}\big]\right), \tag{1.66}$$

where $\mathcal{O}_{ELBO}$ is commonly referred to as the evidence lower-bound (ELBO) since it is a lower-bound of the log-evidence $\log p_\theta(\mathbf{Y})$ (detailed next). Note that, the decomposition into a sum over individual data samples is a consequence of the independence assumption in eq. (1.57), and the mean-field family in eq. (1.64).

Knoblauch et al. [2019] show that the standard ELBO (1.65) is optimal with respect to a given variational family, in the sense that it converges to the best constrained solution to the original problem (1.61), however, this does not suggest that it is the optimal choice in practice.[40]

[40] Knoblauch et al. [2019] point out that key assumptions are typically violated in contemporary machine learning algorithms, and deviations from the standard ELBO can be motivated by addressing these.

Evidence Lower-Bound. VI maximises a lower-bound on the log-marginal-likelihood (log-evidence). The marginal likelihood can be expressed in terms of the variational posterior as follows,

$$p_\theta(\boldsymbol{y}) = \int p_\theta(\boldsymbol{y}|\boldsymbol{x})p_\theta(\boldsymbol{x})d\boldsymbol{x} = \int p_\theta(\boldsymbol{y}|\boldsymbol{x})\frac{p_\theta(\boldsymbol{x})}{q_\phi(\boldsymbol{x})}q_\phi(\boldsymbol{x})d\boldsymbol{x} = \mathbb{E}_{q_\phi^{\boldsymbol{x}}}\Big[p_\theta(\boldsymbol{y}|\boldsymbol{x})\frac{p_\theta(\boldsymbol{x})}{q_\phi(\boldsymbol{x})}\Big], \tag{1.67}$$

where the expectation is now with respect to the variational posterior.

Taking the log of eq. (1.67), and using Jensen's inequality to move the expectation outside the log [Jensen, 1906],

$$\begin{aligned}
\log p_\theta(\boldsymbol{y}) &= \log \mathbb{E}_{q_\phi^{\boldsymbol{x}}}\Big[\frac{p_\theta(\boldsymbol{y},\boldsymbol{x})}{q_\phi(\boldsymbol{x})}\Big] \\
&\geqslant \mathbb{E}_{q_\phi^{\boldsymbol{x}}}\Big[\log\frac{p_\theta(\boldsymbol{y},\boldsymbol{x})}{q_\phi(\boldsymbol{x})}\Big] \quad \triangleright \text{ Jensen's inequality} && (1.68)\\
&= \mathbb{E}_{q_\phi^{\boldsymbol{x}}}\big[\log p_\theta(\boldsymbol{y},\boldsymbol{x}) - \log q_\phi(\boldsymbol{x})\big] && (1.69)\\
&= \mathbb{E}_{q_\phi^{\boldsymbol{x}}}\big[\log p_\theta(\boldsymbol{y}|f(\boldsymbol{x}))\big] - \mathbb{KL}\big[q_\phi^{\boldsymbol{x}}||p_\theta^{\boldsymbol{x}}\big], && (1.70)
\end{aligned}$$

which is the objective in (1.65) written in terms of a single output vector $\boldsymbol{y}$. Note that, maximising the ELBO with respect to ϕ corresponds to approximate inference (as above), whereas maximising the ELBO with respect to θ corresponds to maximum likelihood learning of θ in (1.58).

Minimum KL. VI minimises the KL-divergence between the true posterior and the variational posterior. To see this, consider that the ELBO term from (1.68) can be rewritten as

$$\begin{aligned}
\mathbb{E}_{q_\phi^{\boldsymbol{x}}}\Big[\log\frac{p_\theta(\boldsymbol{y},\boldsymbol{x})}{q_\phi(\boldsymbol{x})}\Big] &= \mathbb{E}_{q_\phi^{\boldsymbol{x}}}\Big[\log p_\theta(\boldsymbol{y}) + \log\frac{p_\theta(\boldsymbol{x}|\boldsymbol{y})}{q_\phi(\boldsymbol{x})}\Big] \\
&= \log p_\theta(\boldsymbol{y}) + \mathbb{E}_{q_\phi^{\boldsymbol{x}}}\Big[\log\frac{p_\theta(\boldsymbol{x}|\boldsymbol{y})}{q_\phi(\boldsymbol{x})}\Big] \quad \triangleright\ p_\theta(\boldsymbol{y}) \text{ does not depend on } \boldsymbol{x} \\
&= \log p_\theta(\boldsymbol{y}) - \mathbb{KL}\big[p_\theta^{\boldsymbol{x}|\boldsymbol{y}}||q_\phi^{\boldsymbol{x}}\big], \quad \triangleright \text{ by definition of KL} && (1.71)
\end{aligned}$$

where the first step follows from the definition of conditional probability. Since the KL terms is non-negative, and $p_\theta(\boldsymbol{y})$ does not depend on ϕ, the only way to increase the ELBO in (1.70) by varying the variational parameters is by minimising the KL-divergence between the true posterior $p_\theta^{\boldsymbol{x}|\boldsymbol{y}}$ and variational posterior $q_\phi^{\boldsymbol{x}}$. Further, by properties of the KL (1.62), it is equal to zero only when $p_\theta^{\boldsymbol{x}|\boldsymbol{y}} = q_\phi^{\boldsymbol{x}}$.

1.4.3 Stochastic Variational Inference

Stochastic variational inference (SVI) [Hoffman et al., 2013, Sato, 2001, Honkela and Valpola, 2003] combines the scalability of SGD (1.27) with the VI objective in eq. (1.65). In particular, when the likelihood of a generative model has an independence structure like the one in (1.57), the ELBO decomposes into a sum over log-likelihood (LL) terms.[41] This makes the gradient of the ELBO, with respect to both the model parameters θ and variational parameters ϕ, a suitable candidate for SGD. An additional complication in the case of SVI is the stochastic nature of the objective itself:

[41] Sometimes the sum also includes the KL terms, like in (1.66), but this is not necessary for SVI.

- SGD is stochastic with respect to the dataset since it draws samples from it (typically) uniformly at random [Ruder, 2016]. The ELBO involves two terms under an expectation over the variational posterior, the LL and the log-ratio of the posterior against the prior (1.65). Both terms are potentially stochastic when the ELBO is computed in practice, since the expectations are not available in closed form generally.
- When the ELBO is estimated stochastically, the gradient is potentially subject to high variance. The approach used in the thesis is due to the work in [Kingma and Welling, 2014, Jimenez Rezende et al., 2014], which often enables low variance estimates even when using naive Monte Carlo sampling of terms in the ELBO [Zhang et al., 2018].

In the thesis, the form of each marginal in the mean-field family (1.64) is chosen as a diagonal Gaussian

$$q_\phi(\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{\mu}_n, \mathrm{diag}(\boldsymbol{\sigma}_n^2)), \tag{1.72}$$

where $\boldsymbol{\mu}_n \in \mathbb{R}^Q$ is the mean of the $n-$th latent variable, $\boldsymbol{\sigma}_n^2 \in \mathbb{R}^Q_{\geqslant 0}$ is a vector specifying the variance of the $n-$th variable, diag transforms a vector to a diagonal matrix with the vector on the diagonal. This Gaussian mean-field variational family is used in two ways:

1. The parameters $\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n^2$ are optimised directly, i.e. each latent variable has unique variational parameters.
2. The mean and variance are given as the output of an inference network.

In the second, the variational distribution is denoted

$$q(\boldsymbol{x}_n | h_\phi(\boldsymbol{y}_n)) = \mathcal{N}(\boldsymbol{\mu}_n, \mathrm{diag}(\boldsymbol{\sigma}_n^2)), \tag{1.73}$$

which is taken to imply that the mean and variance are given by an inference network h_ϕ

$$\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n^2 = h_\phi^{\boldsymbol{\mu}}(\boldsymbol{y}_n), h_\phi^{\boldsymbol{\sigma}}(\boldsymbol{y}_n), \tag{1.74}$$

where $h_\phi^{\boldsymbol{\mu}}, h_\phi^{\boldsymbol{\sigma}}$ denote different outputs from the same inference network. A sample from the approximate posterior in (1.72) is then given by [Kingma and Welling, 2014]

$$\boldsymbol{x}_n = \boldsymbol{\mu}_n + \boldsymbol{\sigma}_n \odot \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}_Q). \tag{1.75}$$

One of the main benefits of using an inference network is that the variational parameters are shared. This means that each example contributes to the approximate posterior over all latent variables, in contrast to having to optimise each set individually in (1.72). This can make the optimisation problem converge faster [Zhang et al., 2018].

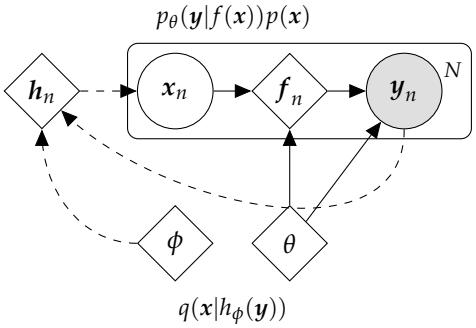


Figure 1.5: Graphical model and inference graph (notation 6) for the VAE in definition 8. The observation function f_θ is modelled by a NN, and so is the inference network h_ϕ. The vector $\boldsymbol{h}_n = h_\phi(\boldsymbol{y}_n)$ is used to denote the parameters of the distribution $q(\boldsymbol{x}_n|h_\phi(\boldsymbol{y}_n))$, e.g., the mean and variance in the Gaussian family from eq. (1.73).

Definition 8 (Variational Auto-Encoder)**.** When f_θ is a NN and the variational posterior is modelled by an inference network h_ϕ, the combination is referred to as a variational auto-encoder (VAE) [Kingma and Welling, 2014, Jimenez Rezende et al., 2014]. For example, if the observation model is Gaussian (1.21), then the composite function is given by

$$\boldsymbol{y} = f_\theta(h_\phi^{\boldsymbol{\mu}}(\boldsymbol{y}) + h_\phi^{\boldsymbol{\sigma}}(\boldsymbol{y}) \odot \boldsymbol{\epsilon}_x) + \boldsymbol{\epsilon}_y, \tag{1.76}$$

where $\boldsymbol{\epsilon}_y \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{1}_D)$ and $\boldsymbol{\epsilon}_x \sim \mathcal{N}(\mathbf{0}, \mathbf{1}_Q)$. Equation (1.76) is a deterministic function of the observation—corrupted by noise at two levels—that outputs a *reconstruction* of the observation. If the noise is taken to be zero $\boldsymbol{\epsilon}_y = \boldsymbol{\epsilon}_x = \mathbf{0}$, then the model is equivalent to deterministic auto-encoder models [Zhang et al., 2018].

Computing The ELBO. The mean-field approximation in (1.64) simplifies the ELBO significantly, since the expectation can be distributed across the terms in the sum (1.66). Since the variational family is Gaussian (1.72)–(1.73), and the prior is assumed Gaussian (1.60), the KL terms are available in closed form (in `nats`):

$$\mathbb{KL}\big[q_\phi^{\boldsymbol{x}} || p^{\boldsymbol{x}}\big] = \frac{1}{2} \sum_{q=1}^{Q} \sigma_q^2 + \mu_q^2 - 1 - 2\ln(\sigma_q), \tag{1.77}$$

where $\mu_q \in \boldsymbol{\mu}$ and $\sigma_q^2 \in \boldsymbol{\sigma}^2$. Finally, the expected LL term is computed with Monte Carlo

samples

$$\mathbb{E}_{q_\phi^x}\big[\log p_\theta(\boldsymbol{y}|f(\boldsymbol{x}))\big] \approx \frac{1}{S}\sum_{s=1}^{S}\log p_\theta(\boldsymbol{y}|f(\boldsymbol{x}_s)), \tag{1.78}$$

where $\boldsymbol{x}_s \sim q_\phi(\boldsymbol{x})$ is sample from the variational distribution. Note that, technically $\boldsymbol{x}_s = \boldsymbol{\mu} + \sigma \odot \boldsymbol{\epsilon}_s$ where $\boldsymbol{\epsilon}_s \sim \mathcal{N}(\mathbf{0}, \mathbf{1}_Q)$, but this is taken to be implied by the former.

To conclude: Given the dataset in (1.55), the full objective is given in eq. (1.66), and the model parameters θ and variational parameters ϕ are jointly optimised using SVI. Specifically, random batches of samples are drawn from the dataset and the gradient is estimated using samples according to (1.75),

$$\mathbb{E}_{\mathbf{Y}_B\sim\mathbf{D}}\big[\nabla_{\theta,\phi}\mathcal{O}_{ELBO}(\theta,\phi;\mathbf{Y})\big] \approx \frac{1}{B}\sum_{b=1}^{B}\nabla_{\theta,\phi}\mathcal{O}_{ELBO}(\theta,\phi;\boldsymbol{y}_b) \tag{1.79}$$

$$= \frac{1}{B}\nabla_{\theta,\phi}\left(\sum_{b=1}^{B}\mathbb{E}_{q_\phi^{x_b}}\big[\log p_\theta(\boldsymbol{y}_b|f(\boldsymbol{x}_b))\big] - \mathbb{KL}\big[q_\phi^{\boldsymbol{x}_b}||p^{\boldsymbol{x}_b}\big]\right) \tag{1.80}$$

$$\approx \frac{1}{B}\nabla_{\theta,\phi}\left(\sum_{b=1}^{B}\frac{1}{S}\left(\sum_{s=1}^{S}\log p_\theta(\boldsymbol{y}_b|f(\boldsymbol{x}_{[b,s]}))\right) - \mathbb{KL}\big[q_\phi^{\boldsymbol{x}_b}||p^{\boldsymbol{x}_b}\big]\right), \tag{1.81}$$

where $\boldsymbol{x}_{[b,s]} \sim q_\phi(\boldsymbol{x}_b)$ and the KL terms are computed using (1.77).

2

Representing Tasks as Conserved Quantities

If you look for a meaning, you'll miss everything that happens.

—Andrei Tarkovsky

The focus of this chapter is on inductive biases for task structured learning problems. At its core, task structure is a reflection of the hierarchical structure inherent to the physical world.[1] In our context, it refers to settings where the learning problem is broken up into related sub-problems (tasks) and the aim is to share information between them, and generalise to new ones.

We assume that tasks are defined based on prior knowledge, for example, by assigning observations from different but related robotic systems into separate tasks. Task-specific sources of variation are modelled as latent conserved (invariant) quantities, and global sources of variation are modelled by a shared function. Since the latent task variables are assumed to be invariant representations with respect to observations within a task, they are called *invariant task representations*. The motivation behind this construction is designing modular learning algorithms for meta-learning in the context of physical dynamical systems.[2] In this setting, the task representations can be seen as a way of modelling conserved quantities corresponding to system parameters.[3]

The results in this chapter are reproduced from the work in [Sæmundsson et al., 2018] and [Kaddour et al., 2020]. We run experiments on simulated robotics systems, exploring the properties of invariant task representations when learning predictive models of the dynamics (multi-task and transfer-learning) and in a model-based (meta) reinforcement learning setting; as well as in active learning of task spaces (section 2.3). We find that invariant task representations are useful for:

- Sharing knowledge between training tasks (multi-task learning);

[1] For example, galaxies contain planets, that contain molecules, that contain atoms and so on. Similarly, a division of time gives rise to sequential hierarchies.

[2] In short, a modular learning algorithm disentangles reusable (transferable) and task-specific components.

[3] For example, lengths and masses in different but related robotic systems.

- Disentangling global and task-specific sources of variation (transfer learning);
- Informed (active) learning of task spaces.

2.1 Background

A general framing of learning algorithms in task structured settings is meta-learning [Schmidhuber, 1987, Bengio et al., 1991, Baxter, 2000, Andrychowicz et al., 2016, Finn et al., 2017], although definitions of meta-learning are generally broader in scope [Lemke et al., 2015, Schmidhuber, 2020]. In the context of learning predictive models of physical dynamical systems, meta-learning is a promising strategy for improving data efficiency by sharing information between related systems. Learning such predictive models from minimal data has applications in areas such as robotics, where data collection might be expensive due to wear-and-tear of the robot [Deisenroth, 2010, Deisenroth et al., 2015, Kamthe and Deisenroth, 2018a].

2.1.1 Dynamical Systems and Task Structure

The systems considered in this chapter take the form of nonlinear ordinary differential equations (ODEs) with control forces. Control forces, denoted by $\boldsymbol{c}$, appear as additional inputs into the dynamics in contrast to eq. (1.2). Specifically,

$$\frac{d\boldsymbol{x}}{dt} = f_\psi(\boldsymbol{x}, \boldsymbol{c}), \quad \boldsymbol{x}_t = \boldsymbol{x}(t), \quad \boldsymbol{c}_t = \boldsymbol{c}(t), \tag{2.1}$$

where f_ψ characterises the dynamics of a system parametrised by physical parameters ψ, e.g. lengths and masses of components of the system. A given setting of the parameters is called the system configuration and is not observed by the learning algorithm. Varying the system configuration produces different regimes of dynamics. For example, a heavy pendulum will respond less to control signals, but it will have a larger gravitational pull compared to a lighter one—all else being equal.

The states in eq. (2.1) consist of position and velocity components $\boldsymbol{x} = [\boldsymbol{x}_{\text{pos}}, \boldsymbol{x}_{\text{vel}}]$, and the control signals are $K-$dimensional vectors $\boldsymbol{c} \in \mathbb{R}^K$ representing external forces. For ease of notation, we define the state-control inputs as

$$\tilde{\boldsymbol{x}} = \texttt{concat}(\boldsymbol{x}, \boldsymbol{c}) \in \mathbb{R}^{Q+K}. \tag{2.2}$$

Given a discrete-time control sequence

$$\mathbf{C} = (\boldsymbol{c}_t), \quad t = 1, \ldots, T, \tag{2.3}$$

the underlying ODEs are numerically solved and the states are measured at fixed intervals to produce observations in discrete time. Details about numerical solvers are given in appendix A.

The initial state (initial condition) is sampled from a Gaussian

$$\boldsymbol{x}_{[n,1]} \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) = p(\boldsymbol{x}_1), \tag{2.4}$$

where $\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1$ are experimental parameters, to produce random realisations of trajectories. In particular, given a system configuration ψ_p, a control sequence (2.3) and an initial state (2.4), each simulation produces a task trajectory

$$\mathbf{X}_{[p,n]} = (\boldsymbol{x}_{[p,n,t]}), \quad t = 1, \ldots, T, \tag{2.5}$$

where p is a task index identified with the configuration ψ and n indexes the samples.

2.1.2 Multi-Task and Transfer Learning

In the context of predictive models, multi-task learning [Caruana, 1997, Ruder, 2017] describes the setting where a single model is learned for prediction on multiple tasks. In this setting, the aim is to improve generalisation performance with respect to new observations from the same tasks.

This chapter assumes prediction tasks with observed inputs and outputs, where the tasks share a common input and output space.[4] It is also assumed (for ease of exposition) that there are a fixed number of tasks P, consisting of an equal number of observations N. A collection of tasks is represented by a pair of 3-dimensional input and output arrays

$$\mathbf{D} = \{(\mathbf{Y}_p), (\mathbf{X}_p)\}, \quad p = 1, \ldots, P, \tag{2.6}$$

where p indexes the tasks. In this case, $\mathbf{X}_p, \mathbf{Y}_p$ are matrices that collect the N input-output pairs belonging to task P. The multi-task learning approach is then concerned with learning a single model that performs well on new observations from all the tasks $p = 1, \ldots, P$. For our context, the tasks correspond to different system parameters ψ_p.

Multi-task learning can also be seen as the training phase of approaches such as transfer and meta-learning. In transfer learning [Pan and Yang, 2010, Weiss et al., 2016], the focus is generally on generalisation performance with respect to observations from new tasks.[5]

For example, having learned a single model on a set of training tasks eq. (2.6), generalisation performance is measured on a new task p^* that is not contained in dataset. In our context, this means predictive performance on a new physical system ψ_{p^*}.

[4] This assumption is discussed further in section 2.5.

[5] The distinction from multi-task learning is not always as clear as that. However, this is the usage in this chapter.

2.1.3 Meta-Learning

Meta-learning, as introduced by Schmidhuber [1987], and appearing in a number of works in subsequent years [Bengio et al., 1991, Naik and Mammone, 1992, Schmidhuber, 1995, Baxter, 2000], had a resurgence of interest following the work by Andrychowicz et al. [2016] and Finn et al. [2017]. Definitions of meta-learning are typically broad in scope and have significant overlap with related lines of research such as transfer learning and continual learning [Lemke et al., 2015, Schmidhuber, 2020].

In our context, meta-learning refers to a combination of multi-task and transfer learning of predictive models, in addition to a (meta-)learning problem that depends on the generalisation performance of the predictive models. This is explored in two ways:

1. Model-based meta reinforcement learning, where the ability of an agent to efficiently solve new tasks depends crucially on the quality of multi-task and transfer learning;
2. Active learning of tasks, where the ability to select new tasks efficiently requires ranking tasks based on how well the model predicts/expects it would transfer knowledge.

The key idea of active learning is that data efficiency can be improved by enabling the algorithm to select the training data it receives next [Settles, 2009].

In this chapter, we consider active learning of task spaces, i.e. choosing which task to learn next. Since the approach is built on top of a multi-task and transfer learning system, we refer to this is active meta-learning. Similar approaches have been developed under the rubric of automatic curriculum learning [Portelas et al., 2020] and unsupervised meta-learning [Gupta et al., 2018a].

2.1.4 Sparse Variational Gaussian Processes

Gaussian processes (GPs) (section 1.3.3) are especially data efficient models for the kinds of learning problems considered in this chapter [Deisenroth and Rasmussen, 2011, Deisenroth et al., 2015]. For this reason, GPs serve as the core building block and baselines in the experiments. A scalar output GP is given by definition 5,[6]

$$p_\theta(f(\tilde{\boldsymbol{x}})) = \mathcal{GP}(0, \Sigma_\theta(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}')), \tag{2.7}$$

assuming a zero mean function and the state-control variable from eq. (2.2) as input. The kernel used throughout the chapter is the EQ kernel from eq. (1.40)

$$\Sigma_\theta(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}') = \sigma_k^2 \exp\left(-\frac{1}{2}(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}')^{\mathrm{T}} \mathbf{L}^{-1} (\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}') \right) \tag{2.8}$$

[6] Since the vector-valued GP models used in the thesis are assumed to be independent across output dimensions, the generalisation is straightforward. That is, each dimension is modelled by a separate GP.

where σ_k^2 is the signal variance and $\mathbf{L}$ contains the characteristic lengthscales.

Given discrete-time observations from dynamical systems, the GP is used to model a single step discretisation of the ODE in eq. (2.1), with added noise. Concretely, the outputs are defined as the one-step transition

$$\boldsymbol{y}_{[p,t]} = (\boldsymbol{x}_{[p,t+1]} - \boldsymbol{x}_{[p,t]}) + \boldsymbol{\epsilon} = \boldsymbol{f}_{[p,t]} + \boldsymbol{\epsilon}, \tag{2.9}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{1}_Q)$ is independent Gaussian noise.

Sparse GPs. One of the main drawbacks of using the analytic form for the GP models from eq. (2.7) is that evaluating the posterior has a complexity cost of $\mathcal{O}(N^3)$, where N is the size of the dataset. Further, if the likelihood is non-Gaussian or the inputs are latent variables, posterior inference is no longer tractable.

This subsection introduces a variational family for approximate posterior inference in GPs, called sparse variational Gaussian processes (SVGPs) [Titsias, 2009, Hensman et al., 2013, Matthews, 2017], which address both the issue of intractable posteriors and scalability with respect to the number of data. The exposition assumes scalar observations $\boldsymbol{y}_{(n)}$, that are conditionally independent given the function values.

The key idea behind sparse approaches is to condition a GP on a set of M function values that can be optimised, instead of conditioning on the data [Snelson and Ghahramani, 2006, Titsias, 2009]. Intuitively, if $M < N$ and these function values are able to approximately represent the dataset, the evaluations that usually scale with the size of the dataset are instead proportional to M, which is a tunable hyperparameter.

The function values are not optimised directly but they are naturally related to a set of inputs, typically called *inducing inputs*, which are optimised instead. In the thesis, these are denoted by

$$\mathbf{X}_{(m)} = (\boldsymbol{x}_m), \quad \boldsymbol{f}_{(m)} = (f(\boldsymbol{x}_m)), \quad m = 1, \dots, M, \tag{2.10}$$

where $\mathbf{X}_{(m)}$ are the inducing inputs, and $\boldsymbol{f}_{(m)}$ are the corresponding function values evaluated at the inducing inputs. From the definition of a GP (definition 5), the prior over the function values is given by

$$p_\theta(\boldsymbol{f}_{(m)}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}), \tag{2.11}$$

where $\boldsymbol{\mu} \in \mathbb{R}^M$ is given by the mean function (1.37) and $\mathbf{K} \in \mathbb{R}^{M \times M}$ is given by the covariance function (1.38).

Sparse Variational GP. An important consideration is how the function values corre-

sponding to the inputs (2.10) should be optimised. Sparse variational GPs, introduced by Titsias [2009], identify the inducing inputs and corresponding function values (2.10) with a variational family over function values. The complete picture emerged with subsequent work on SVGPs [Hensman et al., 2013, Matthews, 2017], and the family is constructed as follows:

Definition 9 (Sparse Variational Gaussian Process)**.** A SVGP is defined over the infinite set of function values $f(\boldsymbol{x}) = \{f(\boldsymbol{x}_\star), \boldsymbol{f}_{(m)}\}$,

$$q_\phi(f(\boldsymbol{x}_\star), \boldsymbol{f}_{(m)}) = p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(m)}) q_\phi(\boldsymbol{f}_{(m)}), \tag{2.12}$$

where the RHS is the conditional GP from (1.42) conditioned on the function values at the inducing inputs, and $f(\boldsymbol{x}_\star)$ denotes the set of function values in $f(\boldsymbol{x})$ excluding $\boldsymbol{f}_{(m)}$. The variational distribution $q_\phi(\boldsymbol{f}_{(m)})$, referred to here as the induced variational posterior, is specified as a Gaussian

$$q_\phi(\boldsymbol{f}_{(m)}) = \mathcal{N}(\boldsymbol{m}, \mathbf{S}), \tag{2.13}$$

where $\boldsymbol{m} \in \mathbb{R}^M$ and $\mathbf{S} \in \mathbb{R}^{M\times M}$ is a full rank covariance matrix. Since the conditional $p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(m)})$ is a GP (definition 6), and its mean function is linear in $\boldsymbol{f}_{(m)}$, it is possible to marginalise over $\boldsymbol{f}_{(m)}$ when the inducing variational posterior is Gaussian. This is exactly analogous to the steps for the true posterior in section 1.3, except the conditional GP is conditioned on $\boldsymbol{f}_{(m)}$ instead of $\boldsymbol{f}_{(n)}$. Assuming a zero-mean GP prior, the result is

$$q_\phi(f(\boldsymbol{x})) = \mathcal{GP}(\mu_\phi(\boldsymbol{x}), \Sigma_\phi(\boldsymbol{x}, \boldsymbol{x}')), \tag{2.14}$$

with mean and covariance function given by

$$\mu_\phi(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^{\mathrm{T}} \mathbf{K}^{-1} \boldsymbol{m}, \tag{2.15}$$

$$\Sigma_\phi(\boldsymbol{x}, \boldsymbol{x}') = \Sigma_\theta(\boldsymbol{x}, \boldsymbol{x}') + \boldsymbol{k}(\boldsymbol{x})^{\mathrm{T}} \mathbf{K}^{-1} \big(\mathbf{S} - \mathbf{K}\big) \mathbf{K}^{-1} \boldsymbol{k}(\boldsymbol{x}'), \tag{2.16}$$

where $[\boldsymbol{k}(\boldsymbol{x})]_m = \Sigma_\theta(\boldsymbol{x}_m, \boldsymbol{x}) \in \mathbb{R}^M$ and $\mathbf{K}_{[i,j]} = \Sigma_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R}^{M\times M}$.

Equation (2.12) is intuitive: The goal is to define a variational approximation to the posterior of a GP (1.49). Using eq. (1.47), the true posterior is given by

$$p_\theta(f(\boldsymbol{x})|\boldsymbol{y}_{(n)}) = p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(n)}) p_\theta(\boldsymbol{f}_{(n)}|\boldsymbol{y}_{(n)}). \tag{2.17}$$

Comparing the SVGP (2.12) to eq. (2.17), it is clear that the induced variational distribution $q_\phi(\boldsymbol{f}_{(m)})$ plays the role of the posterior over function values at the observations $p_\theta(\boldsymbol{f}_{(n)}|\boldsymbol{y}_{(n)})$. In turn, the conditional GP is conditioned on the function values $\boldsymbol{f}_{(m)}$. When the $q_\phi(\boldsymbol{f}_{(m)})$ is Gaussian (2.13), $q_\phi(f(\boldsymbol{x}))$ is also a GP, given in eq. (2.14).

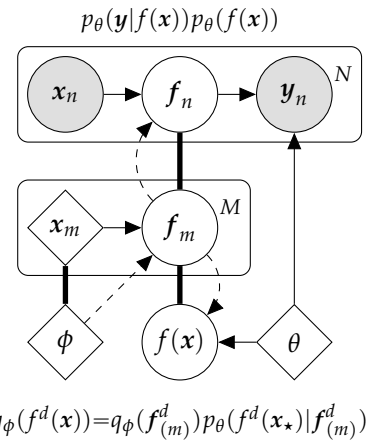


Figure 2.1: Graphical model and inference graph (notation 6) for the SVGP in definition 9. The undirected thick line denotes that $\boldsymbol{f}_n, \boldsymbol{f}_m \in f(\boldsymbol{x})$ and $\boldsymbol{x}_m \in \phi$. The dashed arrows denote dependence on variational parameters through the variational distribution q_ϕ. The SVGP defines a variational distribution over the infinite set $f(\boldsymbol{x})$, through the distribution over $\boldsymbol{f}_{(m)}$. For multi-dimensional outputs in the thesis, the variational distribution for the SVGP consists of independent GPs for each output dimension: $q_\phi(f(\boldsymbol{x})) = \prod_d q_\phi(f^d(\boldsymbol{x}))$.

SVGP ELBO. In eq. (1.65), the infinite-dimensional optimisation problem corresponding to exact Bayesian inference (1.61) was approximated with a finite-dimensional optimisation, by representing the posterior with a parametric variational family. For GPs, there is a second source of infiniteness since the distributions, defined by $p_\theta(f(\boldsymbol{x}))$ and $q_\phi(f(\boldsymbol{x}))$ are over the infinite function values.[7] By construction, the SVGP makes both sources finite. Using the arguments from [Knoblauch et al., 2019], as presented in (1.61), gives

$$p_\theta(f(\boldsymbol{x})|\boldsymbol{y}_{(n)}) \approx \arg\max_{\phi\in\Phi} \left[\mathbb{E}_{q_\phi^{f(\boldsymbol{x})}}\Big[\log p_\theta(\boldsymbol{y}_{(n)}|\boldsymbol{f}_{(n)}) \Big] - \mathbb{KL}\big[q_\phi^{f(\boldsymbol{x})} || p_\theta^{f(\boldsymbol{x})}\big] \right], \tag{2.18}$$

where the KL term contains the second source of infinity. However, using the definition of the SVGP (2.12) and conditioning the GP (1.42) on the same function values,

$$\mathbb{KL}\big[q_\phi^{f(\boldsymbol{x})} || p_\theta^{f(\boldsymbol{x})}\big] = \mathbb{E}_{q_\phi^{f(\boldsymbol{x})}}\left[\frac{p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(m)}) q_\phi(\boldsymbol{f}_{(m)})}{p_\theta(f(\boldsymbol{x}_\star)|\boldsymbol{f}_{(m)}) p_\theta(\boldsymbol{f}_{(m)})} \right] = \mathbb{KL}\big[q_\phi^{\boldsymbol{f}_{(m)}} || p_\theta^{\boldsymbol{f}_{(m)}}\big], \tag{2.19}$$

means that the infinite terms cancel. Additionally, since the likelihood is assumed to factorise,

$$\mathbb{E}_{q_\phi^{f(\boldsymbol{x})}}\Big[\log p_\theta(\boldsymbol{y}_{(n)}|\boldsymbol{f}_{(n)}) \Big] = \sum_{n=1}^{N} \mathbb{E}_{q_\phi^{f(\boldsymbol{x}_n)}}\Big[\log p_\theta(y_n|f(\boldsymbol{x}_n)) \Big], \tag{2.20}$$

the LL term decomposes into a sum and the expectation can be expressed in terms of the marginals [Salimbeni, 2019]. Combining (2.19)–(2.20), the ELBO for the SVGP is then given by

$$\mathcal{O}_{SVGP}(\theta, \phi; \boldsymbol{y}_{(n)}) = \sum_{n=1}^{N} \mathbb{E}_{q_\phi^{f(\boldsymbol{x}_n)}}\Big[\log p_\theta(y_n|f(\boldsymbol{x}_n)) \Big] - \mathbb{KL}\big[q_\phi^{\boldsymbol{f}_{(m)}} || p_\theta^{\boldsymbol{f}_{(m)}}\big]. \tag{2.21}$$

Computing The ELBO. Since both the induced prior (2.10) and the induced variational

[7] As mentioned in section 1.3 these terms are used as if $p(f(\boldsymbol{x}))$ were a density over the infinite function values. While not technically proper, the derived objects are only in terms of a finite set of function values. The infinite-dimensional analogs are treated in, e.g. [Matthews et al., 2016], but do not change the conclusion.

posterior (2.13) are Gaussian, the KL term in eq. (2.21) is computable in closed form,

$$\mathbb{KL}\big[q_\phi^{f_{(m)}} || p_\theta^{f_{(m)}}\big] = \frac{1}{2}\left(\mathrm{Tr}\big(\mathbf{K}^{-1}\mathbf{S}\big) + (\boldsymbol{\mu}-\boldsymbol{m})^{\mathrm{T}}\mathbf{K}^{-1}(\boldsymbol{\mu}-\boldsymbol{m}) - M + \ln\frac{|\mathbf{K}|}{|\mathbf{S}|}\right). \tag{2.22}$$

When the likelihood is Gaussian (1.21), the expected LL term can be computed in closed form also.[8] The result is given by [Hensman et al., 2015]

$$\mathbb{E}_{q_\phi^{f(x_n)}}\Big[\log p_\theta(y_n|f(\boldsymbol{x}_n))\Big] = \log\mathcal{N}(\boldsymbol{y}_n|\mu_\phi(\boldsymbol{x}_n),\sigma^2) - \frac{\Sigma_\phi(\boldsymbol{x}_n,\boldsymbol{x}_n)}{2\sigma^2}, \tag{2.23}$$

where μ_ϕ, Σ_ϕ are given in equations (2.15)–(2.16).

[8] This is not required using SVGPs since it is a part of a VI scheme. However, in the thesis the likelihood is Gaussian.

2.1.5 Model Predictive Control

Reinforcement learning (RL) is a framework for learning optimal policies from trial and error [Sutton and Barto, 1998]. In model-based RL, policies are learned from a simulation of an environment. In the context of this chapter, the environment is described by the unknown dynamics in eq. (2.1). Without knowledge of the underlying dynamics, a model of the dynamics can be learned from observed trajectory data, e.g. eq. (2.5). The model can then be used to simulate the environment in order to learn policies.

Require: $\pi = \mathbf{C}$
Require: $H, T, \ell, \boldsymbol{x}_1$
1: **function** MPC($p_\theta, q_\phi, S, \pi; H, T, \ell, \boldsymbol{x}_1$)
2: $\quad\mathbf{X} = [\boldsymbol{x}_1]$
3: $\quad$**for** $t = 1, \ldots, T-1$ **do**
4: $\quad\quad\mathbf{C}_{t:t+H} = \arg\min J(\mathbf{C}_{t:t+H}; p_\theta, q_\phi, \boldsymbol{x}_t)$ ▷ Plan sequence, eq. (2.25)
5: $\quad\quad\boldsymbol{x}_{t+1} = \mathcal{S}(\boldsymbol{c}_t)$ ▷ Apply only next control, $\boldsymbol{c}_t$, on system.
6: $\quad\quad\mathbf{X} = \mathbf{X} \cup \boldsymbol{x}_{t+1}$
7: $\quad$**end for**
8: $\quad\mathbf{D}, \pi = \textsc{CreateReturnArgs}(\mathbf{X}, \mathbf{C})$
9: $\quad$**return** $\mathbf{D}, \pi$
10: **end function**

Algorithm 1: Iterative MPC. Learnable control parameters are denoted by $\mathbf{C}$, $\boldsymbol{x}_1$ is an initial state, T is the trajectory horizon and H is the planning horizon. The function loops over the full trajectory horizon T and at each step it finds an optimal planning sequence by minimising the cost in eq. (2.25) [line 4]. Next it applies only the control for the current state $\boldsymbol{c}_t$ [line 5]. The CreateReturnArgs function call [line 8] is included for ease of notation, to match the output in Algorithm 4. Technically, the data consists of state transitions and the applied controls (2.9).

Given a model of the dynamics, model predictive control (MPC) aims to find an optimal sequence of controls

$$\widehat{\mathbf{C}}_{1:H} = (\widehat{\boldsymbol{c}}_t), \quad t = 1, \ldots, H, \tag{2.24}$$

for a finite horizon H, that minimises the expected long-term cost

$$J(\mathbf{C}_{1:H}; p_\theta, \boldsymbol{x}_1) = \sum_{t=1}^{H} \mathbb{E}_{p_\theta(\boldsymbol{x}_{t+1}|\boldsymbol{c}_t)}\big[\ell(\boldsymbol{x}_{t+1})\big]. \tag{2.25}$$

The expectation is over the distribution over state trajectories, specified in this case by the model p_θ. The cost function ℓ is a known function that encodes the task through the immediate cost $\ell(\boldsymbol{x}_{t+1})$, that is, the cost given the state of the system at time $t+1$ having executed the control $\boldsymbol{c}_t$ in the previous state $\boldsymbol{x}_t$. In the experiments, a variant of the iterative MPC scheme by Kamthe and Deisenroth [2018a] is used to learn the full control sequence[9]

$$\widehat{\mathbf{C}} = (\widehat{\boldsymbol{c}}_t), \quad t = 1, \ldots, T. \tag{2.26}$$

[9] A fixed horizon is not necessary for MPC.

Algorithm 1 outlines the steps of the procedure assuming a single system for ease of notation.[10] In the context of Algorithm 4, the policy π is made to contain all the required parameters for MPC, as well as the cost function ℓ and the initial state $\boldsymbol{x}_1$.

[10] For multiple systems, the same procedure is run indepedently on each system.

Given an initial state $\boldsymbol{x}_1$, the algorithm iterates through the steps $t = 1, \ldots, T-1$ in the full trajectory. At each step, it finds the optimal open-loop sequence (2.24), for the control sequence starting at step t and up to step $t+H$ [line 4]. It then executes the control signal at step t only, to give a new state $\boldsymbol{x}_{t+1}$ from the environment [line 5]. This is repeated until the desired sequence of control trajectories are obtained (2.26).

Computing The Expected Cost. Computing eq. (2.25) requires the expectation over future state trajectories. Here we give the details for computing the expectation iteratively using a SVGP model, assuming a one step transition model of the form in eq. (2.9).[11]

[11] The steps are directly analogous for the GP case. However, the notation for the SVGP is closer to that of results later in the chapter.

Assuming the state-control pair at step $\tilde{\boldsymbol{x}}_t$ is observed, the distribution over the change in state is given by the predictive distribution over the output

$$p_\theta(\boldsymbol{y}_t|\tilde{\boldsymbol{x}}_t) = \int p_\theta(\boldsymbol{y}_t|\boldsymbol{f}_t) q_\phi(\boldsymbol{f}_t|\tilde{\boldsymbol{x}}_t) d\boldsymbol{f}_t = \mathcal{N}(\mu_\phi(\tilde{\boldsymbol{x}}_t), \boldsymbol{\Sigma}_\phi(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_t) + \sigma^2 \mathbf{1}_Q), \tag{2.27}$$

where $\mu_\phi, \boldsymbol{\Sigma}_\phi$ are given in definition 9, and the integral is available in closed form since both the likelihood and the variational posterior are Gaussian. The distribution over the next state is then given by adding $\boldsymbol{x}_t$ to the mean.

Since the state $\boldsymbol{x}_t$ in the input $\tilde{\boldsymbol{x}}_t$ is a random variable for all but the initial state, we also have to consider the integral over the latent state. Generally, this integral is intractable. However, given the EQ kernel from eq. (2.8), it is possible to compute the first and second moments, assuming that the input distribution is Gaussian [Candela et al.,

2003, Deisenroth and Rasmussen, 2011, Deisenroth et al., 2015]. Specifically,

$$p_\theta(\boldsymbol{y}_t|\boldsymbol{c}_t) = \int p_\theta(\boldsymbol{y}_t|\boldsymbol{f}_t)q_\phi(\boldsymbol{f}_t|\tilde{\boldsymbol{x}}_t)p_\theta(\boldsymbol{x}_t)d\boldsymbol{f}_t d\boldsymbol{x}_t \approx \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\mu}_t^{\boldsymbol{y}}, \boldsymbol{\Sigma}_t^{\boldsymbol{y}}), \tag{2.28}$$

$\boldsymbol{\mu}_t^{\boldsymbol{y}}, \boldsymbol{\Sigma}_t^{\boldsymbol{y}}$ are obtained by computing the first two moments of eq. (2.28) and assuming a Gaussian form for the resulting distribution.[12] Since both $p_\theta(\boldsymbol{y}_t)$ and $p_\theta(\boldsymbol{x}_t)$ are Gaussian by construction, the distribution over the next state is also Gaussian

$$p_\theta(\boldsymbol{x}_{t+1}|\boldsymbol{c}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{\mu}_t^{\boldsymbol{y}} + \boldsymbol{\mu}_t^{\boldsymbol{x}}, \boldsymbol{\Sigma}_t^{\boldsymbol{y}} + \boldsymbol{\Sigma}_t^{\boldsymbol{x}} + \mathrm{cov}[\boldsymbol{y}_t, \boldsymbol{x}_t] + \mathrm{cov}[\boldsymbol{x}_t, \boldsymbol{y}_t]). \tag{2.29}$$

[12] A detailed description of moment matching for $\boldsymbol{\mu}_t^{\boldsymbol{y}}, \boldsymbol{\Sigma}_t^{\boldsymbol{y}}$ is given by Deisenroth et al. [2015].

Finally, the expected immediate cost is computed as

$$\mathbb{E}\big[\ell(\boldsymbol{x}_{t+1})|\boldsymbol{c}_t\big] \approx \int \ell(\boldsymbol{x}_{t+1})p_\theta(\boldsymbol{x}_{t+1}|\boldsymbol{c}_t)d\boldsymbol{x}_{t+1}, \tag{2.30}$$

and the cost function $\ell(\boldsymbol{x}_{t+1})$ is chosen such that the integral in eq. (2.30) is computable in closed form, e.g. a quadratic function of the state.

2.2 Invariant Task Representations

This section formulates the main contribution of the chapter. The motivating problem is multi-task, transfer and meta-learning on observations coming from dynamical systems with control forces, described by eq. (2.1).

The central idea is to identify system parameters (or other task-invariant sources of variation) with a conserved latent variable for each task, called invariant task representations. This is combined with a shared global model conditioned on the task variable. The aim is that global model captures reusable/transferable information and that the task representations can be used for more data efficient learning.

2.2.1 Latent Variable Model for Modular Task Structure

The first step is to define a generative model for a modular task structure, where globally invariant sources of variation are disentangled from task-specific sources. To accomplish this, we define a generative process

$$\boldsymbol{z}_p \sim p(\boldsymbol{z}), \quad \boldsymbol{y}_{[p,n]} \sim p_\theta(\boldsymbol{y}|f(\boldsymbol{x}_{[p,n]}, \boldsymbol{z}_p)), \tag{2.31}$$

where $\boldsymbol{y}$ are conditionally independent random variables, given the inputs $\boldsymbol{x}$ and a latent task variable $\boldsymbol{z} \in \mathbb{R}^W$. The global sources of variation are the model parameters θ and the function f, since they are shared by all the observations in eq. (2.31). The task-specific sources are $\boldsymbol{z}_p$ since they are shared by all observations belonging to the task indexed by p. This defines a modular inductive bias for task structure. For example, in the context

of the dynamical systems from eq. (2.1), z_p can be seen to model the system parameters ψ_p.

The prior over task variables in eq. (2.31) is similar to the latent variables in definition 7, and the observation model is analogous to the one in definition 2, where the function is modelled by a neural network (NN) or a Gaussian process (GP). The generative model corresponding to the generative process in eq. (2.31) is given by[13]

$$p_\theta(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}) = p_\theta(\boldsymbol{y}|f(\boldsymbol{x}, \boldsymbol{z}))p(\boldsymbol{z}). \tag{2.32}$$

[13] The indices p, n are omitted when differentiating between samples is not necessary for parsing the equations.

A key takeaway from eq. (2.32) is that the task structure is defined entirely by the grouping of observations into tasks, and modelled through the observation model $p_\theta(\boldsymbol{y}|f(\boldsymbol{x}, \boldsymbol{z}))$.

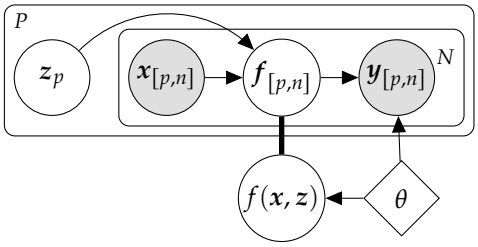


Figure 2.2: Graphical model (notation 6) for eq. (2.33), assuming that the observation function is modelled by a GP. The inputs $\boldsymbol{x}$ and model parameters θ are deterministic. The outputs $\boldsymbol{y}$ (observed) and task variables $\boldsymbol{z}$ (latent) are random variables. The undirected thick line denotes that $\boldsymbol{f}_{[p,n]} \in f(\boldsymbol{x}, \boldsymbol{z})$. P denotes the number of tasks and N the number of observations.

This clarified by looking at the joint distribution over data and task variables,

$$\prod_{p=1}^{P} p(\boldsymbol{z}_p)p_\theta(\mathbf{Y}_p|\mathbf{X}_p, \boldsymbol{z}_p) = \prod_{p=1}^{P} p(\boldsymbol{z}_p)\left(\prod_{n=1}^{N} p_\theta(\boldsymbol{y}_{[p,n]}|f(\boldsymbol{x}_{[p,n]}, \boldsymbol{z}_p))\right) \tag{2.33}$$

which is illustrated in fig. 2.2, the graphical model for (2.33). The figure further highlights the global role of the model parameters θ and the function f, in contrast to the task-specific role of the task variable $\boldsymbol{z}$.

A full specification of the generative model in (2.32) requires defining the observation model and the prior over the task variable. The observation model is a straightforward extension of the observation model from eq. (1.17). Specifically, $\boldsymbol{z}$ is included as an input to the observation function,

$$f(\boldsymbol{x}, \boldsymbol{z}) = f([\boldsymbol{x}, \boldsymbol{z}]), \quad [\boldsymbol{x}, \boldsymbol{z}] = \texttt{concat}(\boldsymbol{x}, \boldsymbol{z}) \in \mathbb{R}^{Q+W}, \tag{2.34}$$

which is achieved by concatenating the vectors $\boldsymbol{x}$ and $\boldsymbol{z}$.

To complete the specification of the model in (2.32), the prior distribution over $\boldsymbol{z}$ is assumed to be a standard Gaussian distribution

$$p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{1}_W). \tag{2.35}$$

As discussed in section 1.4, assuming independence between latent dimensions has important trade-offs. The approach taken here is motivated by ease of inference but richer distributions might improve performance.

Modular Task Structure. An important intuition into the inductive bias encoded by invariant task representations can be read off (2.33). Consider a pair of hypothetical tasks indexed by a, b. The log-likelihood given a single input-output pair from each task $(\boldsymbol{x}_{[a,m]}, \boldsymbol{y}_{[a,m]}), (\boldsymbol{x}_{[b,n]}, \boldsymbol{y}_{[b,n]})$, is given by

$$LL = \log p_\theta(\boldsymbol{y}_{[a,m]} | f(\boldsymbol{x}_{[a,m]}, \boldsymbol{z}_a)) + \log p_\theta(\boldsymbol{y}_{[b,n]} | f(\boldsymbol{x}_{[b,n]}, \boldsymbol{z}_b)). \tag{2.36}$$

- Suppose that the observations come from the same task $a = b$. Then $\boldsymbol{z}_a$ is invariant for the observations by construction. In this case, the only way to model the relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$, using the function $f(\boldsymbol{x}, \boldsymbol{z}_a)$, is through the global parameters θ.
- Suppose that the inputs have the same value $\boldsymbol{x}_{[a,m]} = \boldsymbol{x}_{[b,n]}$, but that the observations are different $\boldsymbol{y}_{[a,m]} \neq \boldsymbol{y}_{[b,n]}$. In this case, the only way to represent the difference as a signal—i.e. not as noise or uncertainty in f—is through the task variables $\boldsymbol{z}_a, \boldsymbol{z}_b$. This is illustrated in fig. 2.3 by comparing the two functions at the point $\boldsymbol{x}_{[a,m]} = \boldsymbol{x}_{[b,n]}$. Since f is a nonlinear function of both $\boldsymbol{x}, \boldsymbol{z}$, the functions can exhibit very different behaviour by varying $\boldsymbol{z}$.

Since the log-likelihood (2.36) appears in the objective (detailed in section 2.2.2), the inductive bias encoded by the model (2.32) is strongly in favour of learning a modular representation of the tasks.

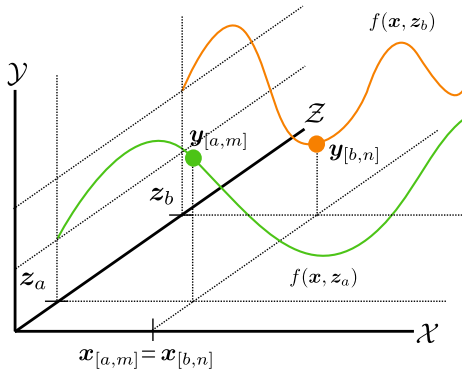


Figure 2.3: Modular task structure. The function f models global variation and the task variable $\boldsymbol{z}$ models task-specific variation. The indices a, b denote different tasks and m, n different samples within those tasks. Modularity refers to the disentangled nature of $\boldsymbol{z}$ and f. Since f is global, it is reusable. Since $\boldsymbol{z}$ is task-specific, the latent task space represents task differences.

Although the model is modular by construction, the degree to which a learned model is disentangled will depend on the training data and how well the assumed task structure maps on to the data distribution. For example, if the function class for f is flexible

enough and there is little overlap in the input spaces between tasks, the task variables might be ignored by the model despite the existence of a relationship between tasks.

2.2.2 Learning Invariant Task Representations

In this subsection, variational inference (section 1.4) is used to approach learning the model parameters θ, and doing approximate inference of the task variables. Specifically, we define an approximation to the true posterior parametrised by ϕ (variational posterior) and the corresponding evidence lower-bound (ELBO) is derived. We frame this as a multi-task learning problem, consisting of learning θ and ϕ from the tasks in eq. (2.6). The relevant conditional for posterior inference of the task variable is given by

$$p_\theta(\boldsymbol{z}|\mathbf{Y},\mathbf{X}) = \frac{p_\theta(\mathbf{Y}|\mathbf{X},\boldsymbol{z})p(\boldsymbol{z})}{p_\theta(\mathbf{Y}|\mathbf{X})}. \tag{2.37}$$

As discussed in section 1.4, this posterior is intractable in general.[14] To address this, we approximate the posterior with variational inference. We assume a Gaussian mean-field approximation (1.64)

[14] On account of the marginal likelihood $p_\theta(\mathbf{Y})$ and the nonlinear function f.

$$q_\phi^{\mathbf{Z}} = q_\phi(\mathbf{Z}) = \prod_{p=1}^{P} q_\phi(\boldsymbol{z}_p) = \prod_{p=1}^{P} \mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p), \tag{2.38}$$

where $\mathbf{Z}$ is a matrix collecting P task representations $\boldsymbol{z}_p$, and $\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p \in \phi$.

Multi-Task Learning. Using the standard form for the evidence lower bound from eq. (1.68), together with equations (2.32) and (2.38) gives

$$\log p_\theta(\mathbf{Y}|\mathbf{X}) \geqslant \mathbb{E}_{q_\phi^{\boldsymbol{z}}}\left[\log \frac{p_\theta(\mathbf{Y},\boldsymbol{z}|\mathbf{X})}{q_\phi(\boldsymbol{z})}\right] \tag{2.39}$$

$$= \mathbb{E}_{q_\phi^{\boldsymbol{z}}}\left[\log p_\theta(\mathbf{Y}|\mathbf{X},\boldsymbol{z}) + \log \frac{p(\boldsymbol{z})}{q_\phi(\boldsymbol{z})}\right] \tag{2.40}$$

$$= \mathbb{E}_{q_\phi^{\boldsymbol{z}}}\big[\log p_\theta(\mathbf{Y}|\mathbf{X},\boldsymbol{z})\big] - \mathbb{KL}\big[q_\phi^{\boldsymbol{z}}||p^{\boldsymbol{z}}\big] = \mathcal{O}_{\boldsymbol{z}}(\theta,\phi;\mathbf{Y},\mathbf{X}) \tag{2.41}$$

Analogous to eq. (1.68), the ELBO in eq. (2.41) consists of an expected log-likelihood term and a KL term. Different from the standard form, given the tasks $\mathbf{D}$ from eq. (2.6), it consists of a nested sum over tasks and observations within the task. That is, conditioned on the task variable, the log-likelihood term consists of i.i.d. terms and decomposes into a sum

$$\log p_\theta(\mathbf{Y}|\mathbf{X},\boldsymbol{z}) = \sum_{n=1}^{N} \log p_\theta(\boldsymbol{y}_n|f(\boldsymbol{x}_n,\boldsymbol{z})), \tag{2.42}$$

which mirrors the expected log-likelihood in eq. (1.70), with the addition of the task

variable as input. The full objective consists of an additional sum over tasks,

$$\log p_\theta(\mathbf{D}) \geqslant \sum_{p=1}^{P} \mathcal{O}_z(\theta, \phi; \mathbf{Y}_p, \mathbf{X}_p) = \mathcal{O}_{ITR}(\theta, \phi; \mathbf{D}). \tag{2.43}$$

Since both the task prior (2.35) and the variational posterior (2.38) are Gaussian, the KL term in (2.43) is available in closed form (1.77). The log-likelihood term can be estimated with Monte Carlo sampling (1.78). Multi-task learning is then performed by stochastic variational inference (SVI, section 1.4) to learn the model parameters θ and the variational parameters ϕ.

Predictions can then be made for a test input $\boldsymbol{x}_\star$ in a training task by approximately integrating over the latent task variable,

$$p_\theta(\boldsymbol{y}_\star|\boldsymbol{x}_\star) = \mathbb{E}_{q_\phi^z}\big[p_\theta(\boldsymbol{y}_\star|f(\boldsymbol{x}_\star, z))\big]. \tag{2.44}$$

Transfer Learning. Predictions on new tasks (transfer learning) fall into two scenarios. If there is no data available, zero-shot transfer learning is done by replacing the expectation with the prior

$$p_\theta(\boldsymbol{y}_\star|\boldsymbol{x}_\star) = \mathbb{E}_{p^z}\big[p_\theta(\boldsymbol{y}_\star|f(\boldsymbol{x}_\star, z))\big]. \tag{2.45}$$

Alternatively, if there is data available from the new task, the variational posterior over the new task variable is inferred by optimising the corresponding variational parameters with respect to (2.41). Since the model parameters θ are the reusable (global) components of the model they are kept fixed. Having obtained a new variational posterior, eq. (2.44) is used for prediction. In practice, both equations (2.44), (2.45) are approximated by sampling the latent task variable,

$$p_\theta(\boldsymbol{y}_\star|\boldsymbol{x}_\star) \approx \frac{1}{S}\sum_{s=1}^{S} p_\theta(\boldsymbol{y}_\star|f(\boldsymbol{x}_\star, \boldsymbol{z}_s)), \tag{2.46}$$

where $\boldsymbol{z}_s$ is a sample from the prior $p_\theta(\boldsymbol{z}_\star)$ or approximate posterior $q_\phi(\boldsymbol{z}_\star)$.

2.2.3 Meta-Learning Gaussian Processes

Having derived the general model, this subsection derives the meta-learning Gaussian process (ML-GP) model used in the experiments in section 2.3. Using the generative task model from (2.32) combined with the GP model from eq. (2.7), the generative model for the ML-GP is defined as

$$p_\theta(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{z}) = p_\theta(\boldsymbol{y}|f(\boldsymbol{x}, \boldsymbol{z}))p_\theta(f(\boldsymbol{x}, \boldsymbol{z}))p(\boldsymbol{z}), \tag{2.47}$$

and the graphical model is given in fig. 2.2. The observation model takes the form

$$p_\theta(\boldsymbol{y}|f(\boldsymbol{x},\boldsymbol{z})) = \mathcal{N}(f(\boldsymbol{x},\boldsymbol{z}), \sigma^2 \mathbf{1}_D), \tag{2.48}$$

which is the Gaussian from eq. (1.21), and the GP model in (2.47) consists of independent (zero-mean) GPs for each output dimension,

$$p_\theta(f^d(\boldsymbol{x},\boldsymbol{z})) = \mathcal{GP}(\mathbf{0}, \Sigma^d_\theta([\boldsymbol{x},\boldsymbol{z}],[\boldsymbol{x}',\boldsymbol{z}'])), \tag{2.49}$$

where Σ^d is a covariance function, eq. (1.34), for the d-th output dimension. Finally, the task prior is the standard Gaussian from (2.35).

For approximate inference of the ML-GP, we use the sparse variational GP from (2.12). For the task variables, we use the mean-field family from (2.38). The full variational posterior is

$$q_\phi^{f(\boldsymbol{x},\boldsymbol{z})\mathbf{Z}} = q_\phi(f(\boldsymbol{x},\boldsymbol{z}),\mathbf{Z}) = \prod_{p=1}^{P} q_\phi(\boldsymbol{z}_p) q_\phi(f(\boldsymbol{x},\boldsymbol{z}_p)) = \prod_{p=1}^{P} \left(q_\phi(\boldsymbol{z}_p) \prod_{d=1}^{D} q_\phi(f^d(\boldsymbol{x},\boldsymbol{z}_p)) \right), \tag{2.50}$$

and is illustrated by the inference graph in fig. 2.4.

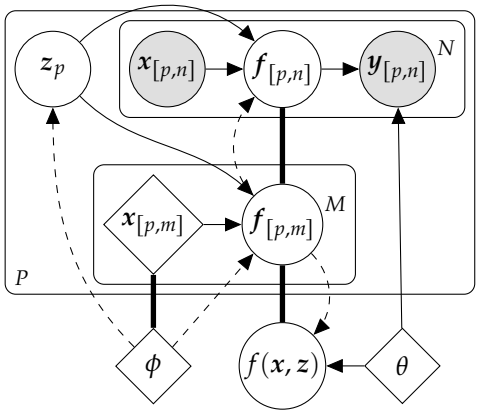


Figure 2.4: Graphical model and inference graph for the ML-GP (notation 6). The undirected thick line denotes that $\boldsymbol{f}_{[p,n]}, \boldsymbol{f}_{[p,m]} \in f(\boldsymbol{x},\boldsymbol{z})$ and $\boldsymbol{x}_{[p,m]} \in \phi$. The dashed arrows denote dependence on variational parameters through the variational distribution q_ϕ.

Using eq. (2.14), the variational posterior over each GP is given by

$$q_\phi(f^d(\boldsymbol{x},\boldsymbol{z})) = \mathcal{GP}(\mu^d_\phi([\boldsymbol{x},\boldsymbol{z}]), \Sigma^d_\phi([\boldsymbol{x},\boldsymbol{z}],[\boldsymbol{x}',\boldsymbol{z}'])), \tag{2.51}$$

where μ_ϕ, Σ_ϕ are given by equations (2.15)–(2.16) respectively.[15] Using eq. (2.41), the ELBO term due to the task variable is given by

$$\mathcal{O}_z(\theta,\phi;\mathbf{Y}) = \mathbb{E}_{q^z_\phi}\big[\log p_\theta(\mathbf{Y}|z)\big] - \mathbb{KL}\big[q^z_\phi || p^z\big], \tag{2.52}$$

and using eq. (2.21) for each of the GPs in the log-likelihood term, the ELBO for the

[15] To be clear, μ_ϕ, Σ_ϕ are functions of the variational parameters that define the SVGP family (section 2.1.4). The task variables have variational parameters $\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p \in \phi$ for each task.

ML-GP is given by

$$\log p_\theta(\mathbf{Y}|\mathbf{X}) \geqslant \mathbb{E}_{q_\phi^z}\big[\log p_\theta(\mathbf{Y}|\mathbf{X}, \boldsymbol{z})\big] - \mathbb{KL}\big[q_\phi^z||p^z\big] \tag{2.53}$$

$$\geqslant \left(\sum_{n=1}^{N} \mathbb{E}_{q_\phi^{f(\boldsymbol{x}_n,\boldsymbol{z})z}}\big[\log p_\theta(\boldsymbol{y}_n|f(\boldsymbol{x}_n,\boldsymbol{z}))\big]\right) - \mathbb{KL}\big[q_\phi^z||p^z\big] - \mathbb{KL}\big[q_\phi^{\boldsymbol{f}_{(m)}}||p_\theta^{\boldsymbol{f}_{(m)}}\big] \tag{2.54}$$

$$= \mathcal{O}_{f(\boldsymbol{x},\boldsymbol{z})z}(\theta, \phi; \mathbf{Y}, \mathbf{X}), \tag{2.55}$$

where the KL-term relating to $\boldsymbol{f}_{(m)}$ appears due to SVGP approximation, as detailed in section 2.1.4. The log-likelihood terms simplify into a sum over output dimensions,

$$\mathbb{E}_{q_\phi^{f(\boldsymbol{x}_n,\boldsymbol{z})z}}\big[\log p_\theta(\boldsymbol{y}_n|f(\boldsymbol{x}_n,\boldsymbol{z}))\big] = \sum_{d=1}^{D} \mathbb{E}_{q_\phi^{f^d(\boldsymbol{x}_n,\boldsymbol{z})z}}\big[\log p_\theta(y_{[n,d]}|f^d(\boldsymbol{x}_n,\boldsymbol{z}))\big], \tag{2.56}$$

and similarly the KL term,

$$\mathbb{KL}\big[q_\phi^{\boldsymbol{f}_{(m)}}||p_\theta^{\boldsymbol{f}_{(m)}}\big] = \sum_{d=1}^{D} \mathbb{KL}\big[q_\phi^{\boldsymbol{f}^d_{(m)}}||p_\theta^{\boldsymbol{f}^d_{(m)}}\big]. \tag{2.57}$$

The KL terms due to $\boldsymbol{z}$ and $\boldsymbol{f}_{(m)}$ are given by eq. (1.77), since all relevant distributions are Gaussian. For a collection of tasks (2.6), the objective is

$$\log p_\theta(\mathbf{D}) \geqslant \sum_{p=1}^{P} \mathcal{O}_{f(\boldsymbol{x},\boldsymbol{z}_p)\boldsymbol{z}_p}(\theta, \phi; \mathbf{Y}_p, \mathbf{X}_p) = \mathcal{O}_{\text{ML-GP}}(\theta, \phi; \mathbf{D}), \tag{2.58}$$

which is optimised with respect to θ and ϕ using SVI (1.81).

2.2.4 Task Descriptors for Active Meta-Learning

This section extends the models from the previous sections (section 2.2.1–section 2.2.3) to include task descriptors. This is done to facilitate active learning using invariant task representations.

A task descriptor might comprise (partially) observed task parameterisations, which is common for system configurations in robotics and molecular descriptors in drug design [Ramsundar et al., 2015]. In other settings, task descriptors might only indirectly contain information about the tasks, for example, a grasping robot that selects tasks based on images of objects but learns to grasp each object through tactile sensors.

In general, task descriptors are any observations that enables discriminative inference about the nature of different tasks. Importantly, from the perspective of the learning algorithm, the task descriptors resolve to a new task when selected. We denote observed task descriptors by

$$\mathbf{T} = (\boldsymbol{\tau}_r), \quad r = 1, \ldots, R, \tag{2.59}$$

where $\mathbf{T}$ is a matrix collecting R vector valued task descriptors $\boldsymbol{\tau}$. Note that these R task descriptors might correspond directly to observed tasks, or they might correspond to a bigger set of available tasks. However, we assume that we have a task descriptor for each observed task.

Generative Model. The assumed role of the task descriptors is to contain information relevant to task-specific variation in the observations. To capture this, the generative task model from eq. (2.32) is extended to include a likelihood term conditioned only on the task variables. Using the ML-GP from (2.47) as a global model, the model is defined as

$$p_\theta(\boldsymbol{\tau}, \boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{z}) = p_\theta(\boldsymbol{\tau}|\boldsymbol{z}) p_\theta(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{z}). \tag{2.60}$$

Note that, the only difference from eq. (2.47) is the task descriptor term $p_\theta(\boldsymbol{\tau}|\boldsymbol{z})$. The specific form of the task descriptor term depends on the experiment. In the first sets of experiments, the task descriptors are modelled with a second GP model,

$$p_\theta(\boldsymbol{\tau}, \boldsymbol{z}) = p_\theta(\boldsymbol{\tau}|g(\boldsymbol{z}))p_\theta(g(\boldsymbol{z})). \tag{2.61}$$

Different from the global model, this GP is not treated with variational inference.[16] Instead, the likelihood is assumed Gaussian and the function is marginalised analytically, as described in section 1.3.3, eq. (1.49). In the last set of active meta-learning experiments, the task descriptors are images and a deep neural network is used to model the task descriptors. In this case, the observation model is written

[16] If the number of training tasks is large, using the SVGP family for the task descriptors might be necessary.

$$p_\theta(\boldsymbol{\tau}|\boldsymbol{z}) = p_\theta(\boldsymbol{\tau}|g(\boldsymbol{z})). \tag{2.62}$$

Note that marginalising over the latent variables is not analytically tractable in either case. Instead they are sampled from the variational posterior.

Variational Inference. When the task descriptor term is modelled with a GP, we choose the mean field variational posterior from (1.64). However, when the task descriptors are high dimensional image observations, the variational posterior over the latent variables is defined using an inference model (1.73). Specifically,

$$q_\phi^{z|\tau} = q(\boldsymbol{z}|h_\phi(\boldsymbol{\tau})) \approx p_\theta(\boldsymbol{z}|\boldsymbol{\tau}, \mathbf{Y}), \tag{2.63}$$

where h_ϕ is an inference network used to approximate the task posterior.

For the collection of tasks $\mathbf{D}$ from (2.6), and the collection of task descriptors $\mathbf{T}$ from (2.59), the contribution to the ELBO due the observations $\mathbf{Y}$ is a minor modification of the MLGP objective in eq. (2.54). Specifically, the variational distribution is now poten-

tially conditioned on the task descriptors,

$$\log p_\theta(\mathbf{D}) \geqslant \sum_{p=1}^{P} \mathcal{O}_{f(\boldsymbol{x},\boldsymbol{z}_p)\boldsymbol{z}_p|\boldsymbol{\tau}_p}(\theta,\phi;\mathbf{Y}_p,\mathbf{X}_p) = \mathcal{O}_{A-\text{ML-GP}}(\theta,\phi;\mathbf{D}), \tag{2.64}$$

where the term in the sum is given by

$$\mathcal{O}_{f(\boldsymbol{x},\boldsymbol{z})\boldsymbol{z}_p|\boldsymbol{\tau}_p}(\theta,\phi;\mathbf{Y},\mathbf{X}) \tag{2.65}$$

$$= \left(\sum_{n=1}^{N} \mathbb{E}_{q_\phi^{f(\boldsymbol{x}_n,\boldsymbol{z})\boldsymbol{z}|\boldsymbol{\tau}_p}} \big[\log p_\theta(\boldsymbol{y}_n|f(\boldsymbol{x}_n,\boldsymbol{z})) \big] \right) - \mathbb{KL}\big[q_\phi^{\boldsymbol{z}|\boldsymbol{\tau}_p}||p^{\boldsymbol{z}}\big] - \mathbb{KL}\big[q_\phi^{\boldsymbol{f}_{(m)}}||p_\theta^{\boldsymbol{f}_{(m)}}\big], \tag{2.66}$$

where all terms are identical to eq. (2.54), except the variational distribution over the task variable is now given by $q_\phi^{\boldsymbol{z}|\boldsymbol{\tau}_p}$. The task descriptors only contribute a log-likelihood term so that the full ELBO is given by

$$\log p_\theta(\mathbf{D},\mathbf{T}) \geqslant \mathcal{O}_{A-\text{ML-GP}}(\theta,\phi;\mathbf{D}) + \sum_{r=1}^{R} \mathbb{E}_{q_\phi^{\boldsymbol{z}_r|\boldsymbol{\tau}_r}} \big[\log p_\theta(\boldsymbol{\tau}_r|\boldsymbol{z}_r) \big]. \tag{2.67}$$

For the case where the variational posterior is defined as a mean-field family, as in eq. (2.50), the form of the ELBO is the same and obtained by replacing $q_\phi^{\boldsymbol{z}_r|\boldsymbol{\tau}_r}$ with $q_\phi^{\boldsymbol{z}_r}$.

2.2.5 Probabilistic Active Meta-Learning Algorithm (PAML)

Task selection using the model in eq. (2.60) is done by defining a utility function in the latent task space. A general way of quantifying the utility of a new task, is by considering the amount of information associated with observing a particular task [Portelas et al., 2020]. To rank candidates in latent space, we use the variational posterior over (training) task representations as a mixture model with equal weights. The utility function is defined as the self-information (surprisal) [Jones, 1979] under the mixture model,

$$u(\boldsymbol{z}_\star;\phi,\mathbf{T}) = -\log \sum_{p=1}^{P} q_{\phi_p|\boldsymbol{\tau}_p}(\boldsymbol{z}_\star) + \log P, \tag{2.68}$$

where $q_{\phi_p|\boldsymbol{\tau}_p}$ denotes the variational distribution over $\boldsymbol{z}_p$. When the approximate posterior q_{ϕ_p} is an exponential family distribution, such as a Gaussian, eq. (2.68) is easy to evaluate. We assign the same weight to each component because we assume the same importance for each observed task.

Choosing A Task Descriptor. Since the ranking is done in the latent task space, the algorithm needs to map the best point $\widehat{\boldsymbol{z}}$ to a task descriptor $\widehat{\boldsymbol{\tau}}$. Each task descriptor resolves to a new task in the (meta) environment, denoted by $\mathcal{S}_{\text{meta}}$. We consider two different ways of mapping to a new task descriptor:

1. In the first, the algorithm is free to choose any value inside a bounded hypercube embedded in the space of task descriptors. For example, if the task descriptor consisted of a length parameter $\tau \in \mathbb{R}^+$, then the algorithm selects a point $\hat{\boldsymbol{z}}$ that maximises eq. (2.68), such that $g(\hat{\boldsymbol{z}})$ is within some bounds $[\tau_{\min}, \tau_{\max}]$;
2. In the second, the observed task descriptors $\mathbf{T}$ define the entire set of available tasks. In this case, the algorithm selects a point $\hat{\boldsymbol{z}}$ that maximises eq. (2.68), provided that $\hat{\boldsymbol{z}}$ corresponds to some $\hat{\boldsymbol{\tau}} \in \mathbf{T}$.

Having selected a point $\hat{\boldsymbol{\tau}}$, the meta environment produces a system $\hat{S} = \mathcal{S}_{\text{meta}}(\hat{\boldsymbol{\tau}})$. Algorithm 2 outlines the probabilistic active meta-learning (PAML) algorithm used in the experiments.

1: **function** PAML($p_\theta, q_\phi, \pi, \mathcal{S}_{\text{meta}}, \mathbf{T}$)
2: **while** learning task space **do**
3: $\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{z}} = \arg\max u(\boldsymbol{z}_\star; q_\phi)$ s.t. constrains on $\hat{\boldsymbol{\tau}}$ ▷ Active learning
4: $\hat{S} = \mathcal{S}_{\text{meta}}(\hat{\boldsymbol{\tau}})$ ▷ Get next task
5: $\mathbf{D}_i = \hat{S}(\pi)$ ▷ Sample data from task
6: $\mathbf{D} = \mathbf{D} \cup \mathbf{D}_i$
7: $\theta, \phi = \text{SVI}(\theta, \phi, \mathbf{D}, \mathbf{T})$ ▷ Learn θ, ϕ using ELBO from eq. (2.41)
8: **end while**
9: **end function**

Algorithm 2: Probabilistic active meta-learning algorithm (PAML). The inputs are the model (p_θ) and variational distribution (q_ϕ) from section 2.2.4; a policy π and task descriptors $\mathbf{T}$. PAML selects a point in latent space based on a pre-defined utility funtion, eq. (2.68) and the task descriptors $\mathbf{T}$ [line 3.]. It then uses π to sample data from the new system [line 5.] and updates the model and variational parameters [line 7.].

We refer to PAML as a meta-learning algorithm in the sense that it relies on the ability of the ML-GP model to be able to capture relevant structure about unseen tasks through the latent task space. That is, at any given iteration in algorithm 2, the ranking of new points $\boldsymbol{z}_*$ through the utility function eq. (2.68) encourages points far away from the training tasks in the latent space. This requires the latent space to at least partially capture structure beyond the set of observed task descriptors. In other words, as discussed in section 2.1, the active selection depends on the underlying multi-task and transfer-learning properties of the ML-GP.

2.3 Modularity and Meta-Learning (Experiments)

The results in this section are reproduced from Sæmundsson et al. [2018]. The work explores the use of the ML-GP from section 2.2.3 for learning predictive models of dy-

namical systems. It looks at predictive performance in multi-task and transfer learning settings and success at solving tasks in a model-based meta reinforcement learning (RL) setting. The learning problem consists of a distribution over related dynamical systems, where each system is governed by a set of unknown physical parameters such as lengths and masses of its components. In this problem setting, the task-specific sources of variation are the system parameters and the global sources of variation are the laws of physics that govern a particular physical system. The aim is to learn a model of the dynamics for a set of training tasks and transfer this knowledge to a set of test tasks. The model is then used to learn control policies for the systems. The experiments are designed to explore four questions about invariant task representations:

1. Are the latent task spaces meaningful compared to the true task space? The latent task space is captured by the approximate posterior distribution over task variables $q_\phi(\mathbf{Z})$ from eq. (2.38) and fig. 2.4. The true task space is the space of physical parameters that govern each dynamical system (task). Meaningful implies that transitions in the latent task space correspond to coherent transitions in the true task space.
2. Are the learned representations modular? The ML-GP from eq. (2.47) models global sources of variation through the function f (and model parameters θ) and task-specific sources through $q_\phi(\mathbf{Z})$. The model is modular by construction but the learned representations are also required to map on to the true task structure.
3. Can invariant task representations be used for transfer- and meta-learning? If the learned representations are modular then the global components should be reusable. Given a new task, transfer learning is then achievable by learning only a new task variable as described in eq. (2.44). Improved transfer should then enable improved data efficiency in the context of model-based meta RL.
4. Can invariant task representations be used for active learning of task spaces? Given the extended model from section 2.2.4, and assuming a sufficiently well behaved latent task space, the task representations should enable improved data efficiency using active meta-learning via the algorithm in algorithm 2.

The baseline models used in sections 2.3.1–2.3.2 are the standard GP from eq. (2.7) and the SVGP from definition 9. A listing of the models and a detailed description of the simulated systems, learning procedure and algorithms used in the experiments are given in appendix A.1.

2.3.1 Multi-Task and Transfer Learning

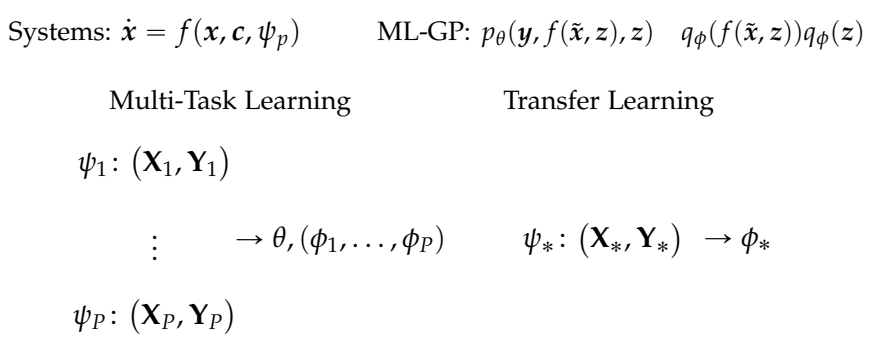


Figure 2.5: The multi-task and transfer learning setup used in the experiments. Given data from P tasks, multi-task learning consists of learning the model parameters θ and the variational parameters ϕ_p corresponding to the training tasks. Transfer learning consists of learning only the ϕ_* given observations from a test task.

Experimental setup. This section looks at multi-task and transfer learning on data from the cart-pole system as illustrated in fig. 2.5. The ML-GP is compared to the SVGP baseline, and a GP baseline to assess the effect of the sparse approximation. The models are compared in terms of predictive performance, measured as the one-step prediction quality in terms of root mean squared error (RMSE) and negative log likelihood (NLL). The experiment is repeated for 10 different seeds, and the measurements are averaged.

Simulated data. The data is generated by executing the same sequence of control signals on different configurations of the cart-pole and sampling observations at 10Hz. A detailed description of the systems and the simulation software used is given in appendix A.1. In order to produce trajectories that explore a larger portion of the state space than sampling control signal at random, the control signals are given as a sequence (given by an oracle) that solved a configuration not included in either the training or test set. Although the control signals still fail to solve the tasks, they avoid getting stuck around the initial state, and produce a more diverse dataset.

A set of 20 tasks are generated as all combinations of $m \in \{0.4, 0.6, 0.7, 0.8, 0.9\}$kg, and $l \in \{0.4, 0.5, 0.6, 0.7\}$m, where m and l denote the mass and length of the pendulum in kilograms and meters, respectively. One 100-step (10 s) trajectory is generated for each task. A set of 6 training tasks are selected as all combinations (m, l) of $m \in \{0.4, 0.6, 0.8\}$kg, and $l \in \{0.5, 0.7\}$m, amounting to a total of 600 observations, or 60 s of interaction time. The remaining 14 tasks are designated to the test set.

Evaluation. During evaluation, 10 time steps (1 s) are used for transfer learning using the ML-GP model. Specifically, the data is used to infer the variational parameters for a

new task representation z_*, while the global parameters θ are kept fixed. The ML-GP, SVGP and GP models are then used for one-step predictions of the next 90 steps.

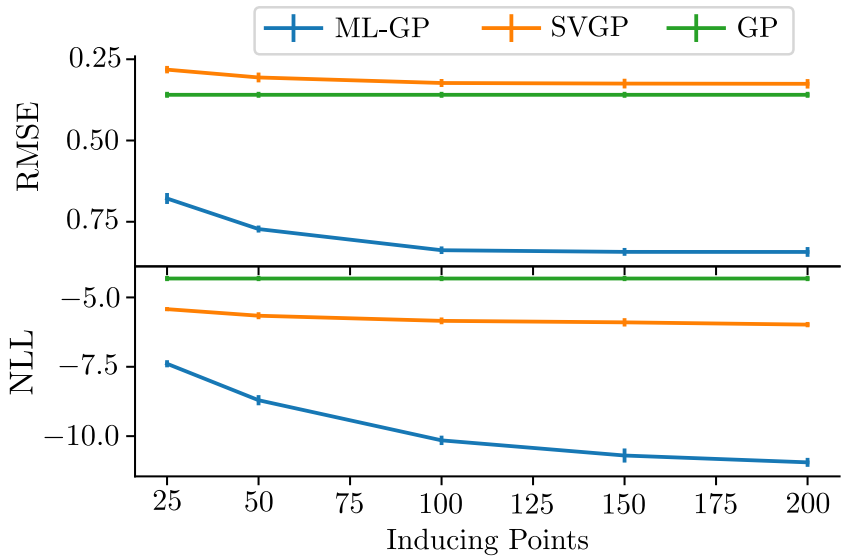


Figure 2.6: Mean and two standard deviation confidence error-bars of the RMSE and NLL for the ML-GP, SVGP and the GP as a function of the number of inducing points. The ML-GP significantly outperforms both baselines.

One-step predictive performance. Figure 2.6 shows the RMSE and NLL for the ML-GP, SVGP and the GP. The ML-GP outperforms both the SVGP and GP baselines, in terms of the accuracy of its mean predictions (as evident by the RMSE), as well as capturing the data better under its predictive distribution as seen by the NLL. The NLL accounts for both the mean prediction as well as the uncertainty of the model about the prediction. Both baselines have comparable RMSEs to each other, assuming enough inducing points for the SVGP. However, the GP slightly outperforms the SVGP in terms of RMSE. Conversely, the SVGP does better than the GP in terms of NLL. Figure 2.7 illustrates predictions from a single test task chosen at random.

The baselines fail to generalise since they have no observations from the system with this configuration. The ML-GP is able to use the task representations z_* for transfer learning. Using 10 observations from the test task, it is able to generalise well without re-learning θ. Although the experiment only considers one-step predictions, multi-step predictions are implicitly evaluated in the model-based RL experiments in section 2.3.2.

Qualitative properties of task representations. For interpretability and downstream use, such as meta-learning, it is desirable that the true task-specifications be a relatively smooth function of the task representations. In particular, that similar task

representations correspond to similar task-specifications, and that moving around in the latent space produces coherent transitions in the task-specifications. Note that the true task-specifications are not observed by the model, so it is not obvious that the learned task space need behave in this way.

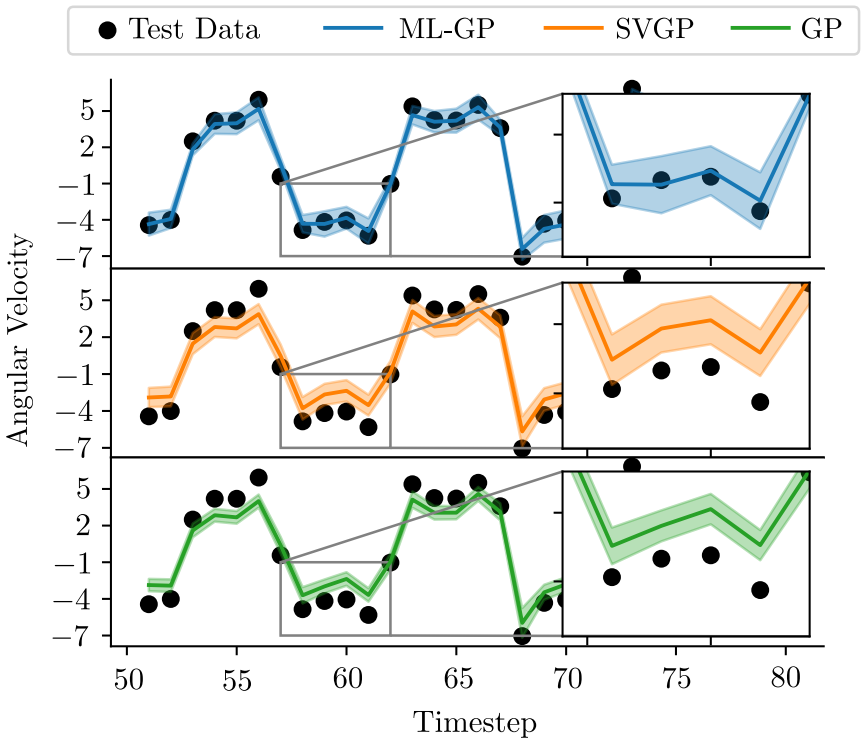


Figure 2.7: One-step predictions of the angular velocity in the cart-pole system. The figure shows the true data points (discs), and the predictive distributions with a two standard deviation confidence interval for the ML-GP, SVGP and the GP. The ML-GP generalises well to the test tasks, whereas both the SVGP and GP baselines generalise poorly in terms of RMSE. Each timestep corresponds to 0.1s in the simulation.

Figure 2.8 displays an example of learned task representations from one of the experiments in fig. 2.6. The different colours of the discs denote the four different settings of lengths, whereas the colours of the dotted lines connecting the discs denote the five different settings of mass. The figure plots the mean of each $q(z_p)$ with two standard deviation error bars in each dimension. Training tasks are represented by solid circles and test tasks are given by empty circles. The representations are arranged in an intuitive structure, where changes in length or mass are disentangled (denoted by the black arrows) into a length-mass coordinate system with the expected transitive properties, e.g. the lengths are ordered as blue ($l = 0.4$m), green ($l = 0.5$m), red ($l = 0.6$m) and orange ($l = 0.7$m). The uncertainty estimates also exhibit qualitatively the intuitive property of being less uncertain about tasks which are similar to (closer to) the training tasks, e.g. comparing the red and blue tasks in figure 2.8.

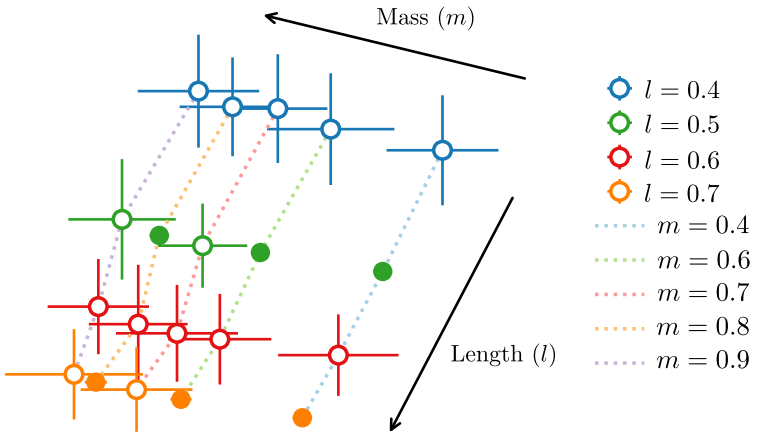


Figure 2.8: Invariant task representations learned from the cart-pole system. The figure shows the mean (discs) of the inferred task variables and two standard deviation error bars. Filled discs are training tasks and empty discs are test tasks. The colours of the discs represent the length and the colours of the dotted lines between discs represent the mass. The arrows are lines indicating directions of increasing mass (kg) and length (m)—as configured by the true system but not observed by the model.

2.3.2 Meta-Learning in Model-Based RL

Experimental Setup. This section investigates the performance of the ML-GP model in terms of data efficiency in RL settings. Specifically, we look at whether the meta learning approach is a) at least as efficient at solving a set of training tasks, b) more efficient at solving subsequent test tasks when compared to a non-meta learning baseline, and c) whether the ML-GP model improves performance when compared to the SVGP model trained with the meta learning approach. The models are compared in terms of success rate as a function of the amount of training data. We assume a fixed number of tasks divided into training and test tasks. All results are averaged over 20 independent random initialisations.

▸ Reinforcement learning tasks: Experiments are run on both the cart-pole swing-up task and the double-pendulum swing-up task, see appendix A.1 for details about the systems. In both scenarios, we use a sampling frequency of 10Hz, episodes of 30 steps (3s) and a planning horizon of 10 steps. For the cart-pole swing-up, solving the task means the pendulum is balanced closer than 8cm from the goal position for at least the last 10 steps (1s). For the double-pendulum swing-up, it means the outer pendulum is balanced closer than 22cm for at least the last 10 steps (1s). For policy learning we use MPC, minimising the cost function in (2.25) using Algorithm 1.

‣ Training: Training consists of iterating through the unsolved training tasks and attempting to solve them with MPC, and learning the model parameters θ and variational parameters ϕ. A detailed listing of the algorithm is given in appendix A.1, algorithm 4. Each attempt at solving a task constitutes a trial, and the algorithm attempts to solve all unsolved training tasks in a given trial before updating the model. The first trial for each of the training tasks consists of executing a random policy. This is repeated until all the tasks are solved, or all unsolved tasks have executed 15 trials.

‣ Evaluation: Following the training phase, a test phase is initiated where the algorithm iterates over the test tasks and attempts to solve them with MPC. The first iteration is used to measure single-shot performance as described in Algorithm 5. For the ML-GP, this involves online inference of the task variable z_p after each observed time step. Following the first iteration, the test phase mirrors the training phase, where the algorithm iterates over unsolved test tasks and re-learns the dynamics model given the additional data. These iterations are used to measure the ability of the models to transfer to new tasks given increasing amounts of data.

‣ Baselines: For comparison with the ML-GP model, we use the SVGP model trained in two different ways. To establish a lower-bound baseline, a separate SVGP model is trained for each task on both the training and test sets. After each training task we additionally attempt to solve each of the test tasks to evaluate single-shot performance where we report the mean across the training tasks as the single shot success rate. We refer to this baseline as SVGP-I, which is a sparse variant of the approach by Kamthe and Deisenroth [2018b] that achieves state-of-the-art in data efficiency. Secondly, we train a single SVGP model on all the training tasks simultaneously using the same training approach as we do for the ML-GP. We refer to this baseline as SVGP-ML.

Cart-Pole Swing-Up Experiments. We train the models on six specifications of the cart-pole dynamics, with $m \in \{0.4, 0.6, 0.8\}$kg, and $l \in \{0.6, 0.8\}$m and evaluate its performance on a set of four test tasks chosen as $m \in \{0.7, 0.9\}$kg, and $l = \{0.5, 0.7\}$m. We choose these settings to examine the performance on both interpolation and extrapolation for differing lengths and masses. We choose the squared distance between the tip of the pendulum and goal position as the cost.

Figure 2.9 shows the mean success rate (over initialisations and the four test tasks) of the ML-GP, SVGP-I and SVGP-ML against the number of trials executed on the systems. Note that horizontal axis gives the number of trials executed across all unsolved tasks.

We observe that both the ML-GP model and the SVGP-ML display generalization to new tasks as evident by the success rate in the first trial (see also Table 2.1).

However, whereas the ML-GP quickly improves with more observations in subsequent trials, the SVGP-ML model struggles to solve the remaining tasks. We attribute this failure to the inability of the SVGP-ML model to explain variation in the dynamics caused by differences in system specifications.

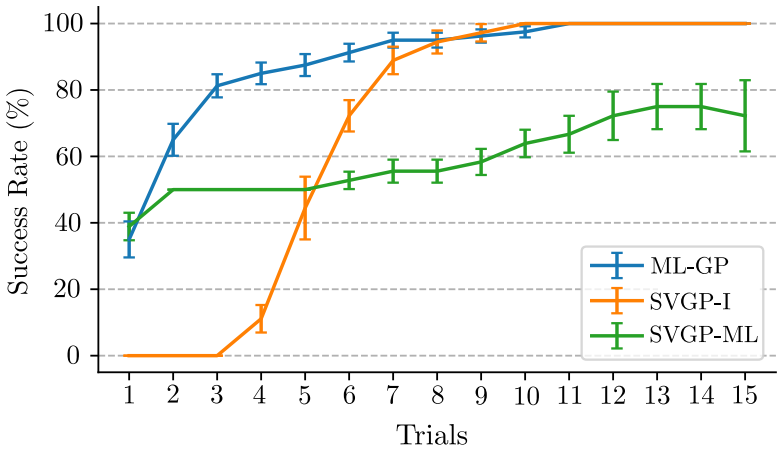


Figure 2.9: Cart-pole system: Mean success rate with standard errors over 10 random initialisations and the four test tasks after training on six tasks. The graph compares the ML-GP with the SVGP-I (trained independently) and SVGP-ML (trained on all tasks). The ML-GP and SVGP-ML models attempt to solve all unsolved tasks in a given trial before update the model. Since each episode for each task consists of 30 steps (3s), and all unsolved tasks are attempted in each trial, each tick on the horizontal axis consists of up to $4 \times 3 = 12$s.

When comparing with independent training of each system we see that the ML-GP compares favourably, reaching 80% success rate after only three trials and 90% after six trials compared to the SVGP-I, which reaches 80% after 7 trials and 90% after 8 trials. We further analyse performance of ML-GP to identify the tasks that were additionally solved between trials 3 and 6. We find that this is due to a consistently challenging system with $m = 0.9\text{kg}, l = 0.5\text{m}$, which requires the learner to extrapolate beyond the range of values seen during training. The mean number of trials required to solve this task is 4.3 ± 0.6 (12.9 ± 1.8s), compared to the task mean of 2.7 ± 0.2 trials (8.1 ± 0.6s). Table 2.1 shows the mean total time required to solve the training and test tasks. On average, ML-GP needs less than half the amount of time to solve the test tasks compared to individually training on the tasks (SVGP-I). We also see an improvement in the total training time, which

Model	Training Tasks (s)	Test Tasks (s)	Single Shot (%)
SVGP-I	16.1 ± 0.4	17.5 ± 0.4	0.08 ± 0.01
SVGP-ML	23.7 ± 1.4	20.8 ± 1.2	0.38 ± 0.04
ML-GP	15.1 ± 0.5	8.1 ± 0.6	0.35 ± 0.05

Table 2.1: Cart-pole system: Mean (with ± 1 standard errors) time spent solving the cart-pole system and the single-shot success rate.

suggests that ML-GP derives some transfer benefit during training despite training on the systems on a concurrent trial basis, i.e. we do not update the model until all systems have executed a given trial. Compared to the SVGP-ML, the ML-GP model can maintain an accurate model while learning multiple systems and quickly adapts to new dynamics, whereas the performance of SVGP-ML stagnates as reflected in the interaction time on both the training and test systems.

Double-Pendulum Swing-Up Experiments. We repeat the same experimental set-up on the double-pendulum task. We train on six systems with $m_1 \in \{0.5, 0.7\}$kg, and $l_1 \in \{0.4, 0.5, 0.7\}$m and evaluate on a set of four test tasks chosen as $m_1 \in \{0.6, 0.8\}$kg, and $l_1 = \{0.6, 0.8\}$m, where m_1, l_1 are the mass and length of the inner pendulum.

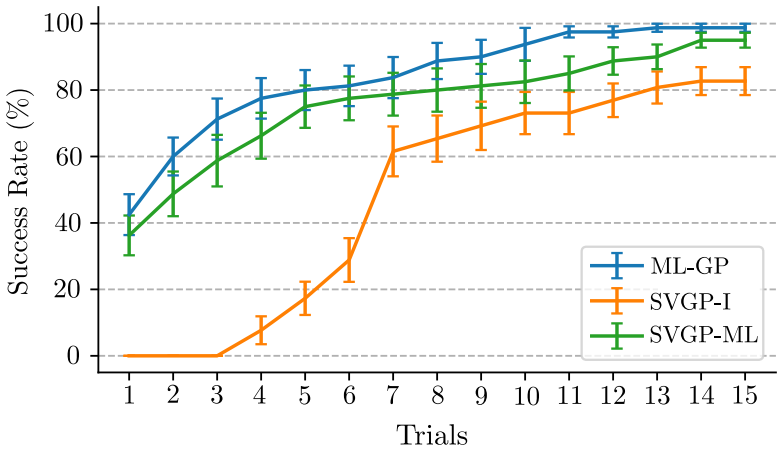


Figure 2.10: Double-pendulum system: Mean success rate with standard errors over 10 random initialisations and the four test tasks after training on six tasks. The graph compares the ML-GP against the SVGP-I (trained independently on each task) and the SVGP-ML (trained using the meta learning procedure). Since each episode for each task consists of 30 steps (3s), and all unsolved tasks are attempted in each trial, each tick on the horizontal axis consists of up to $4 \times 3 = 12$s.

The cost is the squared distance between the tip of the outer pendulum and the goal position (with both pendulums standing straight up). Figure 2.10 plots the mean success rate against the number of trials executed on the system.

Comparing the ML-GP model to the SVGP-ML we observe comparable single-shot performance and a qualitatively similar learning curve for the test tasks. However, the ML-GP reaches 90% success rate about four trials before the SVGP-ML, around trial nine on average for each task, i.e. meta learning achieves a significantly higher data efficiency. Compared to independent training of the tasks using SVGP-I, the ML-GP needs significantly less (new) training data to solve the tasks. Table 2.2 reports the mean total time required to solve the tasks. Compared to the SVGP-ML, the performance of the two is similar, although arguably the ML-GP compares favorably in terms of average time needed to solve the test tasks. Compared to the SVGP-I, we see improvement dur-

Model	Training Tasks (s)	Test Tasks (s)	Single Shot (%)
SVGP-I	18.9 ± 0.7	25.9 ± 1.5	0.07 ± 0.01
SVGP-ML	17.9 ± 1.3	13.7 ± 2.2	0.36 ± 0.06
ML-GP	16.6 ± 1.1	10.2 ± 1.6	0.43 ± 0.06

Table 2.2: Double-pendulum system: Mean (with ± 1 standard errors) time spent solving the double-pendulum system and the single-shot success rate.

ing training as well as at test time. The average time needed for ML-GP to solve the test environments is reduced to around 40% to that of the SVGP-I.

2.3.3 Active Task Learning

The results in this section are reproduced from Kaddour et al. [2020]. The work explores the use of invariant task representations for active learning of task spaces, and the resulting algorithm is termed probabilistic active meta-learning (PAML), and is given in algorithm 2. We find empirically that our approach improves data-efficiency when compared to strong baselines, in the context of learning predictive models of simulated robotic systems. Detailed descriptions of the simulated systems, learning procedure and hyperparameters is given in appendix A.2.

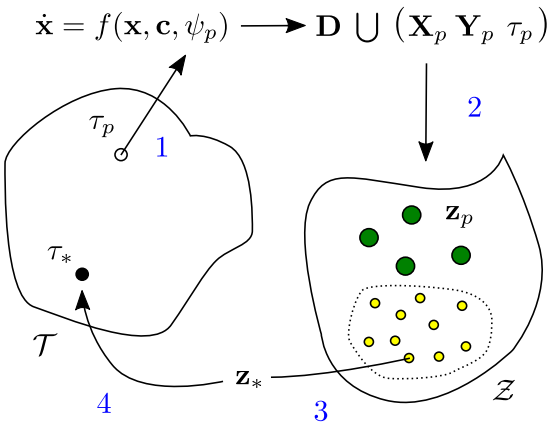


Figure 2.11: The active task learning setup in the experiments using PAML. $\mathcal{T}$ is the task descriptor space, $\mathcal{Z}$ is the latent space of the ML-GP model in eq. (2.60). Step 1 shows how when a task descriptor $\boldsymbol{\tau}_p$ is selected by the algorithm, it resolves to a physical system with system parameters ψ_p (these need not be the same). Observations are generated from the system and added to a dataset, and in step 2, the ML-GP model learns an updated latent space. In step 3, points in the latent space are ranked with the utility function from eq. (2.68) and in step 4, the selected point z_* is mapped to a new task descriptor $\boldsymbol{\tau}_*$ using the observation model $p_\theta(\boldsymbol{\tau}|z)$.

Experimental Setup. In our experiments, we assess whether PAML improves data efficiency when learning task spaces consisting of the dynamics of simulated robotic systems. The setup is illustrated in fig. 2.11. The experiments are designed to test its performance on different types of task descriptors. Specifically, we generate tasks within domains by varying configuration parameters of the simulator, such as the masses and lengths of parts of the system. We then perform experiments where the learning algo-

rithm observes: (i) fully observed task parameters, (ii) partially observed task parameters, (iii) noisy task parameters and (iv) high-dimensional image descriptors. We use control signals that alternate back and forth from one end of the range to the other to generate trajectories. This policy resulted in better coverage of the state-space, compared to a random walk. A detailed description of the systems is given in appendix A.1.

▸ Baselines: We compare PAML to uniform sampling (UNI), commonly used in meta-learning settings [Finn et al., 2017, Galashov et al., 2019] and equivalent to domain randomization [Tobin et al., 2017]; Latin hypercube sampling (LHS) [Tang, 1993]; and an oracle baseline that is trained on the test tasks. The oracle represents an upper bound on the predictive performance. Fixed, evenly spaced grids of test task parameters are chosen to reasonably cover the task domain. We start with four initial tasks and then sequentially add 15 more tasks. As performance measures, we use the negative log-likelihood (NLL) as well as the root mean squared error (RMSE) on the test tasks. The NLL considers the full posterior predictive distribution at a test input, whereas the RMSE takes only the predictive mean into account. In all plots, error bars denote ± 1 standard errors, across 10 randomly initialised trials.

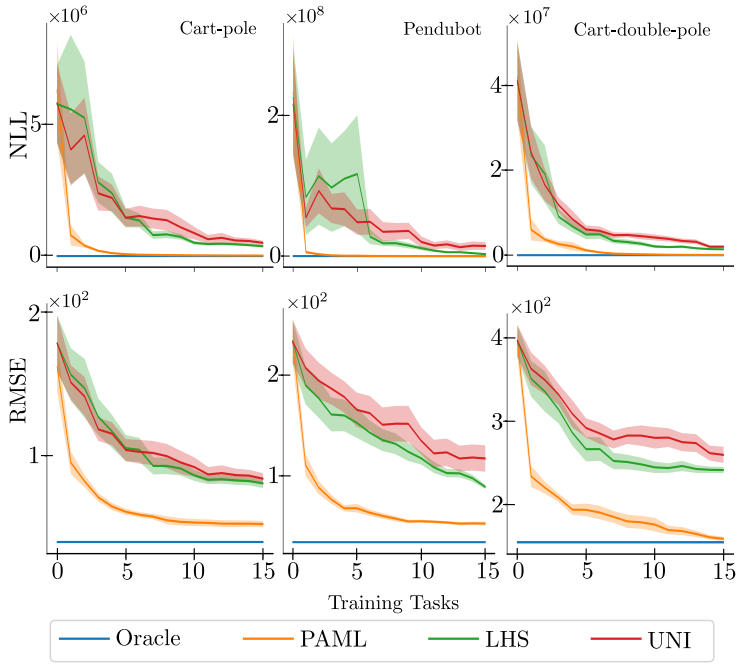


Figure 2.12: NLL/RMSE for 100 test tasks for the cart-pole, pendubot and cart-double-pole with observed task parameters as task-descriptors. Error bars denote ± 1 standard errors across 10 random initialised trials. Across all environments, PAML performs significantly better than the baselines UNI and LHS. Since PAML learns representation of the task differences, it explores the task space more efficiently than the baselines, despite the task parameters being fully observed for all models.

Observed Task Parameters. In these experiments, the observed task descriptors match the task configurations exactly. However, the nonlinear relationship between the param-

eters and the dynamics means that efficient exploration of the configuration space itself will, in general, not map directly to efficient exploration in terms of predictive performance. Here we test whether or not PAML, using the generative task model in (2.60), learns latent embeddings that are useful for active learning of the task domain.

The tasks are generated by varying masses of the attached pendulum and the cart, $p_m \in [0.5, 5.0]$ kg and $p_l \in [0.5, 2.0]$ m, respectively. Pendubot and cart-double-pole tasks have lengths of both pendulums in the ranges, $p_{l_1}, p_{l_2} \in [0.6, 3.0]$ m and $p_{l_1}, p_{l_2} \in [0.5, 3.0]$ m, respectively.

Figure 2.12 shows the results of all methods in all three environments. Comparing PAML to the baselines UNI & LHS, we see that PAML performs significantly better than UNI and LHS in terms of performance on the test tasks. For all three systems, the NLL and RMSE see a steep initial drop for PAML, whereas the performance of the baselines drops more slowly and exhibits higher variance across experimental trials. This is because PAML consistently uses prior information to select the next task whereas the baselines are more affected by chance. We note that the gap in performance obtained by our approach over the baselines remains significant across the task horizon, which is particularly noticeable in the RMSE plots (bottom row) of fig. 2.12.

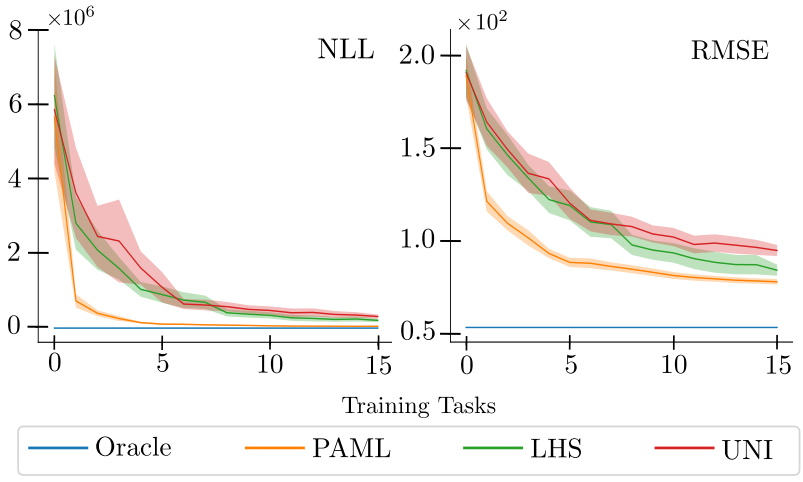


Figure 2.13: NLL/RMSE for 100 test tasks for the cart-pole system with partially observed task parameters. Error bars denote ±1 standard errors across 10 random initialised trials. PAML is able to infer useful task representations for active exploration when the task parameters are only partially observed.

Partially Observed Task Parameters. Partial observability is a typical challenge when applying learning algorithms to real-world systems [Mankowitz et al., 2019]. In these experiments, we simulate the cart-pole system where the task descriptors are chosen as the length of the pendulum, but we vary both its length and mass. The

length is varied between $p_l \in [0.4, 3.0]$ m, and the (unobserved) pendulum's mass $p_m \sim \mathcal{U}[0.4, 3.0]$ kg. That is, each time a new task-descriptor is selected (i.e., length), the mass is sampled. In contrast, the oracle observes all possible masses p_m within the test task grid.

Results are shown in fig. 2.13. PAML achieves lower prediction errors in fewer trials than the baselines. The error after one added task of our methods is approximately matched by the baselines after about five added tasks. It selects similar lengths multiple times, which has the effect of exploring different values of the stochastic mass variable. For example, in one trial, the first eight selected lengths of PAML lie in the range $[0.41, 0.58]$ m.

Intuitively, the reason for this is that the latent embedding represents the full task parameterisation, and smaller values of the length make the effects of varying the mass more apparent. We interpret these results as a demonstration of the PAML algorithm to exploit information about unobserved task configuration parameters, using the learned task representations.

Noisy Task Parameters. In this experiment, we explore the effects of adding a superfluous dimension to the task-descriptors. In particular, we simulate the cart-pole system where we add one dimension $\epsilon \in [0.5, 5.0]$ to the observations, which does not affect the dynamics.

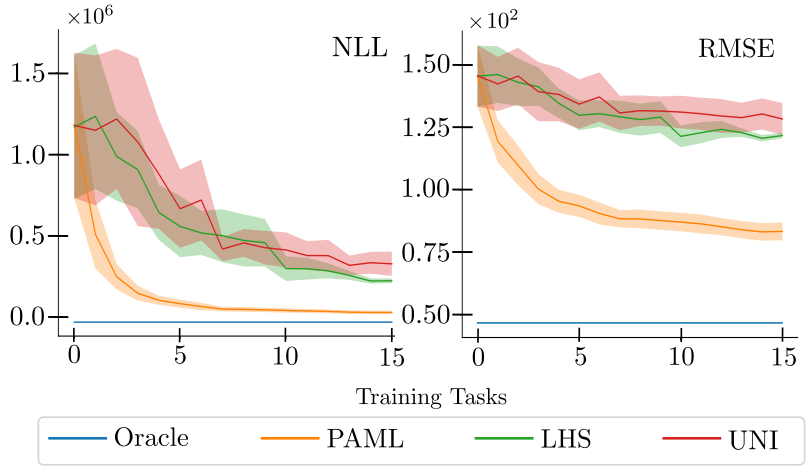


Figure 2.14: NLL/RMSE for 100 test tasks for the cart-pole system with noisy task parameters. Error bars denote ± 1 standard errors across 10 random initialised trials. PAML is able to explore the task space more efficiently since it learns to ignore the redundant dimension added to the task parameters.

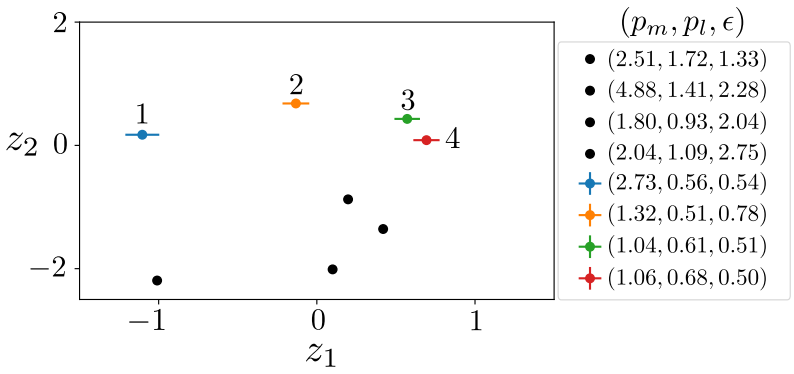


Figure 2.15: Latent embeddings from the cart-pole system with noisy task parameter ϵ. Black discs denote training tasks, and colored discs chosen by PAML (with two standard deviation error bars). The latent space dimensions are given by z_1, z_2. The numbers above each disc denotes the order in which it was selected by PAML. The legend shows the task configurations chosen by PAML after ranking points in the latent space (z_1, z_2).

To select tasks efficiently, PAML needs to learn to ignore the superfluous dimension. Results in fig. 2.15 illustrate this property. Here we show the latent embeddings corresponding to the initial training tasks (black) and the selection made by PAML. We observe that it consistently picks a value for ϵ around 0.5, corresponding to a roughly constant z_2, while exploring informative values for p_m and p_l. Figure 2.14 shows how predictive performance for PAML is better than the baselines in terms of both NLL and RMSE.

Pixel Task Parameters. In this experiment, PAML does not have access to the task parameters (e.g., length/mass), but observes indirect pixel task descriptors of a cart-pole system, shown in fig. 2.16.

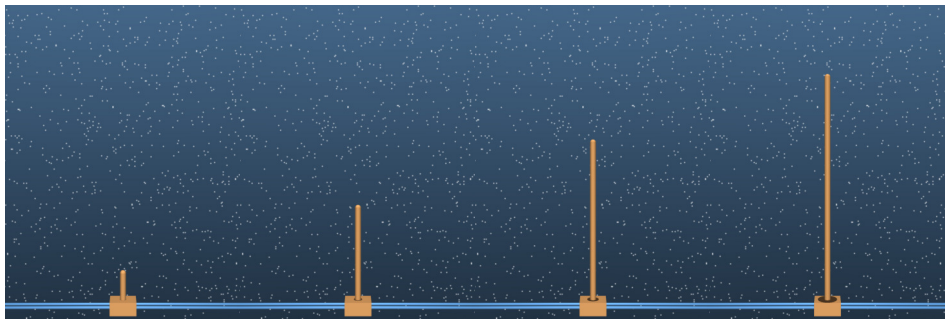


Figure 2.16: Pixel task-descriptors for the cart-pole system with different lengths. PAML can infer latent embeddings from pixel observations and exploit these for faster learning of a task domain.

We let PAML observe a single image of 100 tasks in their initial state (upright pole), where the pole length is varied between $p_l \in [0.5, 4.5]$. PAML selects the next task by choosing an image from this candidate set, after ranking the corresponding latent task variable according to the utility function eq. (2.68). The baseline selects images uniformly at random and both methods start with one randomly chosen training task. Fig-

ure 2.17 shows that PAML consistently selects more informative cart-pole images, and approaches the oracle performance significantly faster than UNI.

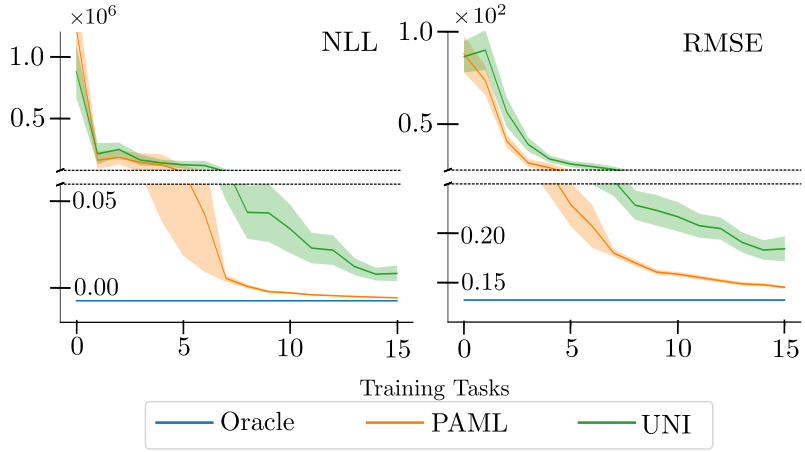


Figure 2.17: NLL/RMSE for 25 test tasks of the cart-pole system using pixel task-descriptors. Error bars denote ± 1 standard errors across 10 random initialised trials. PAML outperforms uniform sampling by exploiting a learned latent representation of the task domain. The pixel task descriptors, used for learning the task representations, are shown in fig. 2.16.

2.4 Related Work

Task Structured Learning Problems

Task structured learning problems have been explored in machine learning under a number of different related names, such as multi-task learning [Caruana, 1997, Ruder, 2017], transfer learning [Pan and Yang, 2010, Weiss et al., 2016] and meta-learning [Schmidhuber, 1987, Bengio et al., 1991, Baxter, 2000, Andrychowicz et al., 2016, Finn et al., 2017].

Multi-task learning [Caruana, 1997, Ruder, 2017] generally deals with the situation where there are a fixed number of tasks and the aim is to generalise to new examples within those tasks. The definition of a task in this case can be blurry, since an expressive model might simply model the joint distribution as a single task. The notion of a task becomes sharper in the case of transfer learning [Pan and Yang, 2010, Weiss et al., 2016], where the aim is to use data from one distribution (task) to generalise to a related task having little or no data. Perhaps the most general formulation that deals with learning from many related task distributions is meta-learning.

MAML is a relatively recent and popular meta learning approach that learns a set of model parameters that are used to efficiently learn novel tasks [Finn et al., 2017]. Another interpretation of MAML is formulated in [Grant et al., 2018], which shares

the hierarchical Bayesian formulation of the meta learning problem in this chapter. A key difference between MAML and the work in this chapter is in how global and task-specific sources of variation are represented. MAML assumes that each task is modelled by a distinct set of parameters (e.g. neural network weights), derived from a global set of parameters by gradient descent. The global parameters themselves are not used directly to make predictions and the task-specific parameters are typically of the same size as the global ones. In contrast, invariant task representations in section 2.2 use the global parameters to model a global function and the task-specific parameters are latent inputs into that function.

Meta-learning Gaussian Process

The ML-GP model resembles the GP latent variable model (GPLVM), which is typically used in unsupervised settings [Lawrence, 2004]. In the GPLVM, the GP is used to map a low-dimensional latent embedding to higher-dimensional observations. A Bayesian extension (BGPLVM) was introduced in [Titsias and Lawrence, 2010] where inference over the latent variable is performed using variational inference. To enable minibatch training, and unlike BGPLVM, the work in this chapter takes the approach of [Hensman et al., 2013] and does not marginalise out the inducing variables. The main difference between the ML-GP and the GPLVM is that the former learns a mapping from both observed and latent inputs to observations.

The combination of observed and latent inputs was investigated in [Wang and Neal, 2012] where the authors use Metropolis sampling for inference which does not scale to larger datasets. A similar setup is found in [Damianou and Lawrence, 2015] where the model is used for partially observed input data. The work also proposes uses in autoregressive settings similar to the work here. Different from this chapter, the distribution over inducing variables is analytically optimised, making minibatch training infeasible.

A related and complimentary line of research are multi-output GPs (MOGPs) [Àlvarez et al., 2012]. Recently, [Dai et al., 2017] proposed a latent variable extension to MOGPs (LVMOGP) which is similar to the ML-GP, particularly in their missing data formulation of the model. The crucial difference is that the ML-GP augments the input space by concatenating the latent variable to the input space while the LVMOGP uses the Kronecker product of two separate kernels applied on the latent and input spaces respectively. Notably, the two models are equivalent for kernels that naturally decompose as a Kronecker product (e.g. the SE kernel) but depart from there.

A similar model to the ML-GP is found in [Doshi-Velez and Konidaris, 2016], called hidden parameter Markov decision processes (HiP-MDP), which parametrises a family of related dynamics through a low dimensional latent embedding. The HiP-MDP assumes a fixed latent variable within trajectories. Different from the ML-GP, the authors use an infinite mixture of GP basis functions where the task-specific variation is obtained through the weights of the basis functions [Doshi-Velez and Konidaris, 2016]. This work was extended in [Killian et al., 2017], replacing the GP basis functions with a Bayesian neural network. This enables non-linear interactions between the latent variables and state variables and addresses scalability. In this chapter, the interactions between latent and state variables are obtained through the non-linear SE kernel, and the scalability is addressed through the variational sparse approach.

In [Deisenroth et al., 2014], an RL setting is considered that is closely related to the ML-GP. The authors use a parametric policy that depends on a known deterministic task variable and augment the policy function to include it as well. In [Deisenroth et al., 2014], the authors consider the same dynamical system but solve different tasks by augmenting the policy with a task variable. In this chapter, different settings of the dynamics are learned but the task remains the same. Additionally, the ML-GP generalise the setting to latent task variables that are inferred from interaction data.

Active Learning of Task Spaces

A similar view to the active task selection model developed in section 2.2.5 is found in Automatic curriculum learning (ACL) where, in general, a task selector is learned based on past data by optimising it with respect to some performance and/or exploration metric [Portelas et al., 2020]. For instance, the work in [Akkaya et al., 2019] uses automatic domain randomisation to algorithmically generate task distributions of increasing difficulty, enabling generalization from simulation to real-life robots. Similarly motivated work is found in [Gupta et al., 2018a], referred to as unsupervised meta-learning, and extended to ACL in [Jabri et al., 2019]. Here, unsupervised pre-training is used to improve downstream performance on related RL tasks. In comparison to ACL, the key objective in section 2.2.5 is data-efficient exploration of a task space from scratch.

More closely related to this objective is active domain randomization in [Mehta et al., 2020], which compares policy rollouts on potential reinforcement learning (RL) tasks compared to a reference environment, dedicating more time to tasks that cause the agent difficulties. PAML learns a representation of the space of tasks and makes comparisons

directly in that space. This way it does not require a) rollouts on new potential tasks, b) handpicked reference tasks and c) the task parameters to be observed directly.

Disentangled Representations

A central theme in this chapter is the separation of task-specific and global sources of variation, referred to as modularity in the text. Importantly, this separation is assumed in the form of an inductive bias, where latent task variables are used to represent unknown system parameters in physical dynamical systems. A more general approach is that of disentangled representation learning [Higgins et al., 2018, Achille and Soatto, 2018, Alemi et al., 2016, Anselmi et al., 2016]. There, the aim is to learn generative factors (sources of variation) that are well aligned with the underlying data generative process [Schmidhuber, 1992, Bengio et al., 2013]. For example, in the context of this chapter, this might correspond to learning the task grouping from state observations of the physical systems rather than assuming they are given beforehand.[17] A 'well aligned' representation would then map coherently to the grouping of systems by their parameters. Thus, extension of the work in this chapter where the assumptions about task grouping are relaxed fits squarely within this work.

[17] Importantly, some assumptions about the model and data are still required for learning disentangled representations [Locatello et al., 2019].

2.5 Summary & Discussion

This chapter introduced latent conserved quantities (invariant task representations, section 2.2) as a modelling tool for multi-task, transfer and meta-learning predictive models of physical dynamical systems. The motivation was to improve data efficiency by having a single model share information between different but related prediction tasks. In the context of this chapter, tasks were identified with system parameters such as the length and mass of a pendulum. The idea behind invariant task representations is that explicitly modelling task-specific sources of variation (system parameters) and global sources of variation (the laws of physics) as distinct components; transfer learning can be achieved by reusing the global and relearning only the task-specific for a new task.

Results. To explore this idea, the meta-learning Gaussian process (ML-GP) model was defined in section 2.2.3, and the probabilistic active meta-learning (PAML) algorithm in section 2.2.5. The ML-GP uses invariant task representations to learn a latent task space representing unknown system parameters. The PAML algorithm uses this latent task space to rank potential new tasks in terms of the expected information gain.

In section 2.3, the ML-GP model and the PAML algorithm were studied empirically in terms of data efficiency when learning from simulated robotic systems. In fig. 2.6, invariant task representations were demonstrated to enable the ML-GP to transfer knowledge to unseen tasks, showing significant improvement in comparison to an otherwise equivalent Gaussian process (GP) model. In fig. 2.8, the latent space learned by the ML-GP was found to be well aligned with the ground truth system parameters. In figures 2.9–2.10 and tables 2.1–2.2, the ML-GP model was shown to reduce the amount of trials needed to solve tasks in a model-based meta reinforcement learning setting. Finally, in section 2.3.3, the PAML algorithm was shown to enable more efficient exploration of task spaces by taking advantage of the learned structure in the latent space of the ML-GP.

Limitations. The modelling choices made for invariant task representations and the ML-GP have significant limitations. The biggest limitation of the task representations is that it requires prior knowledge about task grouping. Although this distinction is easy to make for simulated systems, this is not likely to be straightforward for practical problems. The use of GP models is well justified for data efficiency reasons, however, for larger systems (in dimension) the computational overhead may be become problematic. Additionally, only the exponential quadratic kernel was explored in the chapter, which may be unsuitable in practice.

The empirical results in section 2.3 also have limitations. The baselines that the ML-GP model was compared against are only suggestive of the benefit of modelling task-specific sources of variation, but not suggestive of how this might best be achieved. See for instance the number of related works in section 2.4. Additionally, in the model-based RL experiments, the results need to be interpreted with care due to the limited number of test tasks used for measuring the performance. Similarly, the results for the PAML algorithm only look at a single utility function, making the overall results weaker.

Future Research. Overall, the idea of using latent conserved quantities to capture task-specific sources of variation may be practically useful for data constrained learning problems where a natural task grouping is easy to define beforehand. To address problems where this is not the case, one might look at using approaches from disentangled representations [Higgins et al., 2018, Achille and Soatto, 2018, Anselmi et al., 2016] for meta-learning instead. To address the limitations of the GP in this chapter, it is possible that combining ideas in this chapter with neural processes [Garnelo et al., 2018] might produce more scalable learning algorithms.

3

Variational Integrator Networks

It is the pull of opposite poles that stretches souls. And only stretched souls make music.

—Eric Hoffer

The focus of this chapter is on geometric inductive biases for expressive function classes that encode classical mechanics. Classical mechanics has a number of important geometric properties of interest (section 3.1): phenomena are described by compact representations (generalised coordinates) evolving on manifolds; a rich source of information about the evolution is contained in conservation laws and corresponding conserved quantities; and the evolution is characterised by scalar energy functions.

Learning these structures from data is difficult and failing to do so degrades predictive performance, resulting in non-physical errors in the best case and erratic behaviour in the worst. These structures are also naturally adapted to representation learning, where the geometric structure (e.g. conserved quantities) can be related to the observations by arbitrarily complicated function approximators [Higgins et al., 2018]. The core idea of the chapter is to parametrise the action principle (section 1.2.2) with neural networks, and discretise it using variational integrators. This results in variational integrator networks (VINs).

The results from this chapter are reproduced from the work in [Saemundsson et al., 2020]. We run experiments on simulated physical systems, exploring VINs encoded with Newtonian physics in terms of data efficiency, generalisation and learning physical representations from images (section 3.3). Our results demonstrate that VINs facilitate:

1. Accurate long-term predictions;
2. Data efficient learning;
3. Interpretability;

compared to a baseline that tries to learn the physical structure from data.

3.1 Background

The learning problems in this chapter assume that the observations come from a system described by classical mechanics. From section 1.2, these systems are solutions of 2nd order ordinary differential equations (ODEs) called the Euler-Lagrange (E-L) equations of motion (EOM):

$$\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\Big) - \frac{\partial L}{\partial \boldsymbol{q}} = \mathbf{0}, \tag{3.1}$$

where $\boldsymbol{q}(t)$ is the generalised position, $\dot{\boldsymbol{q}}$ is the generalised velocity and L is the Lagrangian of the system.[1]

[1] Generalised coordinate refer to the fact that the E-L equations are independent of the choice of coordinate system section 1.2.

The Lagrangian L is a scalar function consisting of energy terms. The one considered in this chapter is the Newtonian Lagrangian from eq. (1.8),

$$L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T(\dot{\boldsymbol{q}}) - U(\boldsymbol{q}) = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}}\mathbf{M}\dot{\boldsymbol{q}} - U(\boldsymbol{q}), \tag{3.2}$$

where T is the kinetic energy, U is the potential energy, and $\mathbf{M}$ is a diagonal mass matrix.[2] Due to Noether's theorem [Noether, 1918] and the explicit time independence of the Lagrangian, the energy

$$E(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T(\dot{\boldsymbol{q}}) + U(\boldsymbol{q}), \tag{3.3}$$

is a conserved quantity of Newtonian systems described by eq. (3.2). This is a corollary of the fact that the action (functional) from section 1.2,

$$S[\boldsymbol{q}](a, b) = \int_a^b L(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))dt, \tag{3.4}$$

is invariant with respect to transformations in time. The main aim of this chapter is to build the physical structure of Newtonian systems into neural networks. This section details the two main ingredients for the developments of the chapter: the dynamical systems view of function approximation and variational integrators.

[2] For N particles in D dimensions,

$$\mathbf{M} = \mathrm{diag}(\mathbf{M}_1, \ldots, \mathbf{M}_N),$$

where

$$\mathbf{M}_n = \underbrace{m_n, \ldots, m_n}_{\text{D times}},$$

and m_n is the mass of particle n.

3.1.1 Learning Flow Maps

This subsection looks at the dynamical systems view of function approximation [E, 2017, Chen et al., 2018, Scholkopf, 2019]. The key idea is to identify the function to be learned with the solution of an ODE. One of the benefits of this framing is that global properties of the function can be manipulated to define inductive biases, by considering global

properties of the ODE. From definition 1, an ODE is described by

$$\frac{d\boldsymbol{h}}{dt} = f(\boldsymbol{h}, t), \quad \text{given } \boldsymbol{h}(t_0) = \boldsymbol{h}_0, \tag{3.5}$$

where $\boldsymbol{h}(t) \in \mathbb{R}^Q$ is the state of the system, f is the dynamics function, and $\boldsymbol{h}_0$ is the initial condition at time t_0. Given the solution $\boldsymbol{h}(t; \boldsymbol{h}_0)$, conditioned on the initial condition, the *flow map* is defined as the function that takes as argument an initial condition and evaluates the corresponding solution up to a fixed time horizon T,[3]

$$F(\boldsymbol{h}_0; T) = \boldsymbol{h}(T; \boldsymbol{h}_0). \tag{3.6}$$

[3] The semicolons in eq. (3.6) are used to differentiate between what is treated as argument (left) and what is treated as a parameter.

The key idea that connects nonlinear function approximation to dynamical systems of the form (3.5) is to treat F as the function to be learned from data. Rather than learn F directly, it is modelled implicitly through f. For example, a neural-ODE (NODE) is defined as the family of dynamical systems given by [E, 2017, Haber and Ruthotto, 2017, Chen et al., 2018]

$$\frac{d\boldsymbol{h}}{dt} = f_\theta(\boldsymbol{h}, t), \tag{3.7}$$

where f_θ is modelled by a NN. Note that:

- The inputs of the network are identified with the initial state $\boldsymbol{h}_0$;
- The outputs are identified with the state $\boldsymbol{h}(T; \boldsymbol{h}_0)$ given by the flow map (3.6);
- Evaluating the function, i.e. the flow map (3.6), requires solving the ODE.

This perspective can be used to define interesting inductive biases for expressive function classes by incorporating the desired structure in ODE form. However, the solution is intractable in general and numerical approximations are required. As it relates to defining inductive biases, there are two general perspectives to consider:

1. The approach developed in [Chen et al., 2018] allows for the use of arbitrary ODE solvers to learn continuous-time solutions. The advantages of this are memory efficiency as a function of depth, the ability to use sophisticated ODE solvers, and the ability to predict in continuous-time. The drawback of this approach is the computational cost of running the ODE solver each time the function is evaluated.
2. The second approach is to use an explicit discrete-time integrator which gives the approximation analytically in terms of update equations. The primary advantage is computational, since no optimisation is required for evaluating the function. This can be particularly beneficial during learning, where the function is evaluated frequently. Disadvantages include loss of accuracy and discrete-time (grid) solutions.

The developments in this chapter focus on explicit discrete-time solutions. Although

one of the motivations for this in the thesis is computational tractability, it is also worth considering that the accuracy of the solution may not be of primary concern [E, 2017]. For example, if global properties of the ODE are preserved in the discrete system then the inductive bias encoded by the ODE remain intact.

Euler Method. Perhaps the most well known explicit discrete-time approximation is the forward Euler method [Euler, 1768]. In its simplest form, time is discretised uniformly with a constant step size η

$$0 < \eta \leqslant 1, \tag{3.8}$$

$$t_k = t_0 + k\eta, \tag{3.9}$$

and the true path $\boldsymbol{h}(t)$ is approximated with a one-step update equation

$$\boldsymbol{h}_{k+1} = F^{\mathrm{e}}(\boldsymbol{h}_k, \eta) = \boldsymbol{h}_k + \eta f(\boldsymbol{h}_k, t) = \boldsymbol{h}_k + \eta \frac{d\boldsymbol{h}}{dt}, \tag{3.10}$$

where $\boldsymbol{h}_k \approx \boldsymbol{h}(t_k)$ is the approximation to the true path at time t_k. The one-step error of the Euler method in (3.10) is proportional to η^2, and the cumulative one-step error for an arbitrary horizon is proportional to η [Atkinson et al., 2011].

Residual Networks. It is well known that residual networks (ResNets) [He et al., 2016]

$$\boldsymbol{h}_{k+1} = \boldsymbol{h}_k + \eta f_\theta^k(\boldsymbol{h}_k), \tag{3.11}$$

can be interpreted as an Euler discretised system of a NODE [E, 2017, Haber and Ruthotto, 2017, Chen et al., 2018], which is clear when comparing (3.10) to (3.11). Note that:

- The discrete time index k is identified with the layer index of the NN;
- The state $\boldsymbol{h}_k$ is identified with the layers, as well as the input and output, which means that in eq. (3.11) they all have the same width Q. However, the dynamical system does not have to be defined in the same space as the data;
- Alternatively, eq. (3.11) is interpreted as a different dynamical system for each layer.

A common NN architecture for modelling time series are recurrent NNs [Rumelhart et al., 1986, Hochreiter and Schmidhuber, 1997]. Consider the time-invariant (autonomous) NODE given by

$$\frac{d\boldsymbol{h}}{dt} = f_\theta(\boldsymbol{h}), \tag{3.12}$$

where f_θ is not a function of t. Using the Euler method, the corresponding residual block is given by

$$\boldsymbol{h}_{k+1} = \boldsymbol{h}_k + \eta f_\theta(\boldsymbol{h}_k), \tag{3.13}$$

Note that f_θ is the same across all time steps, in contrast to eq. (3.11) which is different at each layer (corresponding to time). Equation (3.13) is used as a baseline in chapter 3.

3.1.2 Variational Integrators

This subsection introduces variational integrators, which approximately solve the E-L EOM from eq. (3.1) while preserving the underlying geometry [West, 2004]. The main idea is to discretise Hamilton's principle from (1.5)—and therefore the action in eq. (3.4)—instead of discretising the EOM directly.

The basic formulation discretises time with a uniform grid[4]

$$t_k = t_0 + k\eta, \quad 0 < \eta \leqslant 1, \quad k = 0, \dots, K, \tag{3.14}$$

[4] Adaptive and non-uniform time grids are possible but not considered in this chapter [Marsden and West, 2001, West, 2004].

such that the position in discrete-time is $\boldsymbol{q}_k = \boldsymbol{q}(t_k)$. Since the Lagrangian term in eq. (3.4) depends on the position and velocity $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, both terms need attention in the discretisation. Variational integrators address this by defining the discrete state space

$$(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}) \in \mathrm{Q} \times \mathrm{Q}, \tag{3.15}$$

where Q is the configuration manifold of the system. The intuition here is that the discrete state space contains the same amount of information as the pair $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ since each copy of $\mathcal{Q}$ can represent positions at different times on the path.[5] The configuration manifold $\mathcal{Q}$ is a reflection of the coordinate independent nature of geometric mechanics (section 1.2). Using this fact, it is possible to implement constraints by choosing appropriate local coordinates. This is explored explicitly later in the chapter.

[5] Formally, $\mathcal{Q} \times \mathcal{Q}$ is locally isomorphic to the tangent bundle $(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathcal{TQ}$ [Marsden and West, 2001].

The main approximation of variational integrators is called the discrete Lagrangian, which is an approximation to the action functional (3.4) over the time interval η,

$$L^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \eta) \approx \int_{t_k}^{t_k+\eta} L(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), t)dt, \tag{3.16}$$

where L^d is the discrete Lagrangian. The discrete Lagrangian is then used to define a discrete Hamilton's principle [West, 2004]

$$\delta S^d = 0, \tag{3.17}$$

where the discrete action sum is defined as

$$S^d(\boldsymbol{q}_0, \dots, \boldsymbol{q}_K) = \sum_{k=0}^{K-1} L^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}). \tag{3.18}$$

Analogous the continuous-time theory, the discrete variational principle leads to the

discrete Euler-Lagrange (DEL) equations [West, 2004],

$$\mathrm{D}_2 L^d(\boldsymbol{q}_{k-1}, \boldsymbol{q}_k) + \mathrm{D}_1 L^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}) = 0, \tag{3.19}$$

where D_i denotes the slot derivative, that is the partial derivative with respect to the $i-$th argument

$$\mathrm{D}_i L^d(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_i, \ldots, \boldsymbol{q}_N) = \frac{\partial L}{\partial \boldsymbol{q}_i}. \tag{3.20}$$

Directly analogous to the E-L equations, assuming regularity of the discrete Lagrangian L^d, the DEL equations imply a discrete flow

$$F^L \colon (\boldsymbol{q}_{k-1}, \boldsymbol{q}_k) \mapsto (\boldsymbol{q}_k, \boldsymbol{q}_{k+1}), \tag{3.21}$$

which is the discrete-time flow corresponding to the solution of the DEL equations.

For the Newtonian Lagrangian considered in this chapter, given in eq. (3.2), it is possible to use a change of coordinates to define an alternative form of the DEL equations.[6] Specifically, the generalised momentum coordinates $\boldsymbol{p}_k$ are given by

$$\boldsymbol{p}_k = -\mathrm{D}_1 L^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, h), \quad \boldsymbol{p}_{k+1} = \mathrm{D}_2 L^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, h), \tag{3.22}$$

which define the implicit discrete Euler-Lagrange (IDEL) equations in terms of the momentum. The IDEL equations imply a discrete-time solution flow

$$F^H \colon (\boldsymbol{q}_k, \boldsymbol{p}_k) \to (\boldsymbol{q}_{k+1}, \boldsymbol{p}_{k+1}). \tag{3.23}$$

The DEL equations eq. (3.19) and the IDEL equations eq. (3.22) are referred to as the Lagrangian/Hamiltonian viewpoints, respectively.

[6] Formally, the coordinate transformation is a discrete counterpart to the Legendre transform [Marsden and West, 2001].

Discrete Lagrangian. Both the Lagrangian viewpoint (DEL) and the Hamiltonian viewpoint (IDEL) are given in terms of the discrete Lagrangian L^d (3.16). There are many ways to approximate the integral, however, this chapter only considers two variants of one general quadrature rule.[7] The first is given by

$$L^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \eta) = \eta L(\boldsymbol{q}_k, \boldsymbol{v}_k), \tag{3.24}$$

where the velocity is approximated with a midpoint rule

$$\dot{\boldsymbol{q}}(t_k) \approx \frac{\boldsymbol{q}_{k+1} - \boldsymbol{q}_k}{\eta} = \boldsymbol{v}_k. \tag{3.25}$$

The second approximation used is the symmetric variant of (3.24),

$$L^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \eta) = \frac{\eta}{2}\left(L\Big(\boldsymbol{q}_k, \boldsymbol{v}_k\Big) + L\Big(\boldsymbol{q}_{k+1}, \boldsymbol{v}_k\Big)\right), \tag{3.26}$$

where $\boldsymbol{v}_k$ is given by (3.25).

[7] See Marsden and West [2001], section 3.6, for numerous examples.

Discrete Conservation Laws. Variational integrators are useful since they preserve global geometric structure *automatically*. In particular, for conservative systems [Lew et al., 2003]:[8]

- Variational integrators are symplectic: they conserve the phase-space volume form exactly and approximately conserve the true energy of the system.
- Variational integrators are momentum-preserving: For any symmetry in the discrete system, there is a corresponding quantity that is exactly conserved. This is a discrete counterpart to Noether's theorem [Marsden and West, 2001].

[8] The general theory works for forced and dissipative systems also. However, this changes some of the theoretical properties.

In contrast, the Euler method used to define the basic residual network architectures and given in eqs. 3.11–3.13 are not symplectic, even if the underlying ODE is, and do not conserve energy or other conserved quantities due to an accumulation of approximation errors [Cvitanović et al., 2016].

3.2 Variational Integrators as Function Classes

This section uses the theory of variational integrators to define expressive neural networks with inbuilt Newtonian physics defined by eq. (3.2), called variational integrator networks (VINs). VINs are derived for Cartesian coordinates and polar coordinates with fixed distance from the origin, corresponding to different configuration manifolds.

3.2.1 Variational Integrator Networks

A family of variational integrators is defined by parametrising the continuous-time Lagrangian appearing in the discrete Lagrangian given in eq. (3.16). Specifically,

$$L_\theta^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \eta) \approx \int_{t_k}^{t_{k+1}} L_\theta(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), t)dt, \tag{3.27}$$

where time is discretised uniformly with step size η, and $t_{k+1} = t_k + \eta$. We also assume that the Lagrangian L_θ is regular

$$\det\left(\frac{\partial^2 L}{\partial \dot{\boldsymbol{q}}}\right) \neq 0, \quad \forall(\boldsymbol{q}, \dot{\boldsymbol{q}}), \tag{3.28}$$

since the theory of variational integrators generally require additional theory to handle degenerate (non-regular) cases [Rowley and Marsden, 2002]. Equation (3.28) is made explicit here since it suggests that modelling $L_\theta(\boldsymbol{q}, \dot{\boldsymbol{q}})$ directly, for instance with a NN, might require further consideration.[9] The form of the Lagrangian used in the chapter is the Newtonian potential system given in (3.2), which is regular using both Cartesian

[9] Lutter et al. [2019] model the Hessian directly with a neural network by parametrising it as via Cholesky factors. However, the work also enforces the physics through the loss function, which is not the aim here.

coordinates and radially fixed polar coordinates.

Using eq. (3.27), the discrete form of Hamilton's principle (3.17) then implies the existence of a pair of maps

$$F_\theta^L : (\boldsymbol{q}_{k-1}, \boldsymbol{q}_k) \mapsto (\boldsymbol{q}_k, \boldsymbol{q}_{k+1}), \tag{3.29}$$

$$F_\theta^H : (\boldsymbol{q}_k, \boldsymbol{p}_k) \mapsto (\boldsymbol{q}_{k+1}, \boldsymbol{p}_{k+1}), \tag{3.30}$$

corresponding to the discrete Lagrangian and Hamiltonian flow maps from (3.21) and (3.23). In particular, the Lagrangian flow is the solution to the DEL equations from (3.19):

$$\mathrm{D}_2 L_\theta^d(\boldsymbol{q}_{k-1}, \boldsymbol{q}_k, \eta) + \mathrm{D}_1 L_\theta^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \eta) = 0; \tag{3.31}$$

and the Hamiltonian flow is the solution to the IDEL equations from (3.22):

$$\boldsymbol{p}_k = -\mathrm{D}_1 L_\theta^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \eta), \quad \boldsymbol{p}_{k+1} = \mathrm{D}_2 L_\theta^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \eta). \tag{3.32}$$

Equations (3.31)–(3.32) define the family of variational integrators, together with eq. (3.27).

In the following sections, we derive the VIN architectures used in the experiments in section 3.3. The focus is on Newtonian physics and variational integrators with explicit discrete update equations for the flow maps (3.29)–(3.30), since they result in network architectures that do not require fixed-point algorithms to evolve the dynamics.[10]

[10] This is discussed in more detail in section 3.5.

3.2.2 Newtonian VIN in Cartesian Coordinates

Consider the Newtonian Lagrangian in (3.2) for $\mathcal{Q} = \mathbb{R}^C$, and parametrise the mass matrix and potential function,

$$L_\theta(\boldsymbol{q}, \dot{\boldsymbol{q}}) = T_\theta(\dot{\boldsymbol{q}}) - U_\theta(\boldsymbol{q}) = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}} \mathbf{M}_\theta \dot{\boldsymbol{q}} - U_\theta(\boldsymbol{q}), \tag{3.33}$$

where U_θ is parametrised by a NN, and $\mathbf{M}_\theta \in \mathbb{R}^{C\times C}$ is a symmetric, positive definite mass matrix. The mass matrix is assumed to be independent of the generalised position and is parametrised as

$$\mathbf{M}_\theta = \mathbf{L}_\theta \mathbf{L}_\theta^{\mathrm{T}}, \tag{3.34}$$

where $\mathbf{L}_\theta$ is a lower-triangular matrix with $C(C+1)/2$ free parameters. The mass matrix can also be taken to be the identity matrix $\mathbf{M}_\theta = \mathbf{1}_C$. Equation (3.33) describes a separable Newtonian potential system, such as N particles in D dimensional space with no constraints on the configuration manifold. Taking the mass matrix to be independent of the position is required for explicit update equations. We now take the discrete

Lagrangian to be the one from (3.24), and apply the DEL equations (3.31),

$$\begin{aligned}
\mathrm{D}_2 L_\theta^d(\boldsymbol{q}_{k-1}, \boldsymbol{q}_k, \eta) + \mathrm{D}_1 L_\theta^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \eta) &= 0, \\
\eta\left(\frac{\partial}{\partial \boldsymbol{q}_k} L_\theta(\boldsymbol{q}_{k-1}, \boldsymbol{v}_{k-1}) + \frac{\partial}{\partial \boldsymbol{q}_k} L_\theta(\boldsymbol{q}_k, \boldsymbol{v}_k)\right) &= 0, \\
\eta\left(\frac{1}{\eta^2}\mathbf{M}_\theta(\boldsymbol{q}_k - \boldsymbol{q}_{k-1}) + \frac{1}{\eta^2}\mathbf{M}_\theta(\boldsymbol{q}_k - \boldsymbol{q}_{k+1}) - \nabla U_\theta(\boldsymbol{q}_k)\right) &= 0,
\end{aligned}$$

where $\boldsymbol{v}_k$ is the midpoint approximation to the velocity from eq. (3.25).[11] After rearranging terms, this gives the Störmer-Verlet (SV) integrator [West, 2004]

[11] Given by: $\dot{\boldsymbol{q}}(t_k) \approx \frac{\boldsymbol{q}_{k+1} - \boldsymbol{q}_k}{\eta} = \boldsymbol{v}_k$

$$\boldsymbol{q}_{k+1} = 2\boldsymbol{q}_k - \boldsymbol{q}_{k-1} - \eta^2 \mathbf{M}_\theta^{-1} \nabla U_\theta(\boldsymbol{q}_k). \tag{3.35}$$

Alternatively, in flow map form, the VIN-SV is defined as:

$$\boldsymbol{h}_k^q = F_\theta^L(\boldsymbol{h}_{k-1}^q, \eta) = \begin{bmatrix} \boldsymbol{q}_k \\ 2\boldsymbol{q}_k - \boldsymbol{q}_{k-1} - \eta^2 \mathbf{M}_\theta^{-1} \nabla U_\theta(\boldsymbol{q}_k) \end{bmatrix}. \tag{3.36}$$

If instead the discrete Lagrangian from (3.26) is used in (3.22), the first momentum term is given by

$$\begin{aligned}
\boldsymbol{p}_k = -\mathrm{D}_1 L_\theta^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, h) &= -\frac{\eta}{2}\left(\frac{\partial}{\partial \boldsymbol{q}_k} L_\theta\Big(\boldsymbol{q}_k, \boldsymbol{v}_k\Big) + \frac{\partial}{\partial \boldsymbol{q}_k} L_\theta\Big(\boldsymbol{q}_{k+1}, \boldsymbol{v}_k\Big)\right) \\
&= \frac{1}{\eta}\mathbf{M}_\theta(\boldsymbol{q}_{k+1} - \boldsymbol{q}_k) + \frac{1}{2}\eta \nabla U_\theta(\boldsymbol{q}_k),
\end{aligned}$$

and the second momentum term is given by

$$\begin{aligned}
\boldsymbol{p}_{k+1} = \mathrm{D}_2 L_\theta^d(\boldsymbol{q}_k, \boldsymbol{q}_{k+1}, h) = \boldsymbol{p}_{k+1} &= \frac{\eta}{2}\left(\frac{\partial}{\partial \boldsymbol{q}_{k+1}} L_\theta\Big(\boldsymbol{q}_k, \boldsymbol{v}_k\Big) + \frac{\partial}{\partial \boldsymbol{q}_{k+1}} L_\theta\Big(\boldsymbol{q}_{k+1}, \boldsymbol{v}_k\Big)\right) \\
&= \boldsymbol{p}_{k+1} = \frac{1}{\eta}\mathbf{M}_\theta(\boldsymbol{q}_{k+1} - \boldsymbol{q}_k) - \frac{1}{2}\eta \nabla U_\theta(\boldsymbol{q}_{k+1}).
\end{aligned}$$

After subtracting $\boldsymbol{p}_{k+1} - \boldsymbol{p}_k$ and solving for $\boldsymbol{q}_{k+1}$, the Velocity-Verlet (VV) integrator is given as [West, 2004]

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \mathbf{M}_\theta^{-1}\Big(\eta \boldsymbol{p}_k - \frac{1}{2}\eta^2 \nabla U_\theta(\boldsymbol{q}_k)\Big), \tag{3.37}$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k - \frac{1}{2}\eta\Big(\nabla U_\theta(\boldsymbol{q}_k) + \nabla U_\theta(\boldsymbol{q}_{k+1})\Big). \tag{3.38}$$

Alternatively, in flow map form, the VIN-VV is defined as:

$$\boldsymbol{h}_{k+1}^p = F_\theta^H(\boldsymbol{h}_k^p, \eta) = \begin{bmatrix} \boldsymbol{q}_k + \mathbf{M}_\theta^{-1}\Big(\eta \boldsymbol{p}_k - \frac{1}{2}\eta^2 \nabla U_\theta(\boldsymbol{q}_k)\Big) \\ \boldsymbol{p}_k - \frac{1}{2}\eta\Big(\nabla U_\theta(\boldsymbol{q}_k) + \nabla U_\theta(\boldsymbol{q}_{k+1})\Big) \end{bmatrix}. \tag{3.39}$$

3.2.3 Newtonian VIN for the Pendulum

Meyers [2011] describes a variational integrator for the pendulum system that automatically evolves on a circle manifold. Formally, the underlying configuration manifold is the Lie group manifold $SO(2)$ and the integrator is an instance of a Lie group variational integrator [Meyers, 2011, Lee et al., 2015].

From eq. (1.12), the Lagrangian of a single pendulum system is given by

$$L_{\mathrm{p}}(\omega, \dot{\omega}) = T_{\mathrm{p}}(\dot{\omega}) - U_{\mathrm{p}}(\omega) = \frac{1}{2} m l^2 \dot{\omega}^2 + mgl(1 - \cos\omega), \quad \omega \in (0, 2\pi), \tag{3.40}$$

where ω is an angle measuring the displacement from vertical and $\dot{\omega}$ is the angular velocity; m is the point-mass and l is the length of the rod; and g is the magnitude of the gravitational field.

The form of the integrator in Meyers [2011] is[12]

[12] Page 148, equations (23)-(24) in [Meyers, 2011]

$$\sin(\omega_{k+1} - \omega_k) = \sin(\omega_k - \omega_{k-1}) - \eta^2 \frac{1}{ml^2} \nabla U_{\mathrm{P}}(\omega_k) \tag{3.41}$$

$$= \sin(\omega_k - \omega_{k-1}) - \eta^2 \frac{g}{l} \sin(\omega_k). \tag{3.42}$$

To turn eq. (3.41) into a VIN, we parametrise the potential function

$$\omega_{k+1} = \omega_k + \sin^{-1}\Big(\sin(\omega_k - \omega_{k-1}) - \eta^2 I_\theta^{-1} \nabla U_\theta(\omega_k)\Big), \tag{3.43}$$

where I_θ^{-1} is an optional model parameter, and we call the network VIN-$SO(2)$. Alternatively, in flow map form:

$$\boldsymbol{h}^{\omega}_{k+1} = F^L_\theta(\boldsymbol{h}^{\omega}_k, \eta) = \begin{bmatrix} \omega_k \\ \omega_k + \sin^{-1}\Big(\sin(\omega_k - \omega_{k-1}) - \eta^2 I_\theta^{-1} \nabla U_\theta(\omega_k)\Big) \end{bmatrix}, \tag{3.44}$$

where $\boldsymbol{h}^{\omega}$ is used to distinguish the $SO(2)$ integrator from the Euclidean integrators above. Note that if the changes in angles $\omega_{k+1} - \omega_k$ and $\omega_k - \omega_{k-1}$ are assumed to be small, then the flow in (3.44) reduces to the one in (3.36).[13] However, for larger displacements, only the VIN-$SO(2)$ accurately captures the underlying constraint on the circle.

[13] Under the small angle approximation $\sin\Delta\omega \approx \Delta\omega$.

3.2.4 Learning Variational Integrator Networks

Since section 3.3 models dynamic systems, it will be useful to frame the learning problem in terms of time-series. The observations will be assumed to be of the form

$$\boldsymbol{y} = (y_d), \quad d = 1, \dots, D, \tag{3.45}$$

$$\mathbf{Y} = (\boldsymbol{y}_k), \quad k = 1, \dots, K, \tag{3.46}$$

$$\mathbf{D} = (\mathbf{Y}_n), \quad n = 1, \dots, N, \tag{3.47}$$

where there are N time-series consisting of K steps, and each step is a D dimensional observation. Implicit in this structure is the time discretisation defined for (3.27).

Generative Model. The physical system, i.e. the variational integrator, will be assumed to live in a latent space corresponding to either the discrete velocity phase-space

$$\boldsymbol{h}_k^q = (\boldsymbol{q}_k, \boldsymbol{q}_{k+1}), \quad \boldsymbol{h}_0^q = (\boldsymbol{q}_0, \boldsymbol{q}_1), \tag{3.48}$$

or the discrete momentum phase-space

$$\boldsymbol{h}_k^p = (\boldsymbol{q}_k, \boldsymbol{p}_k), \quad \boldsymbol{h}_0^p = (\boldsymbol{q}_0, \boldsymbol{p}_0), \tag{3.49}$$

where (3.48) corresponds to the Lagrangian flow map in (3.29), and (3.49) corresponds to the Hamiltonian flow map in (3.30). In this case, the flow maps are denoted as

$$\boldsymbol{h}_{k+1}^q = F_\theta^L(\boldsymbol{h}_k^q, \eta), \tag{3.50}$$

$$\boldsymbol{h}_{k+1}^p = F_\theta^H(\boldsymbol{h}_k^p, \eta). \tag{3.51}$$

To ease the notation, specific references to which phase-space or flow map is being used are dropped when both are applicable. In this case, $\boldsymbol{h}_k$ is used to denote the state space and F_θ the flow map. The dimension of the configuration manifold is assumed to be C, so that the phase-spaces are $2C$-dimensional.

Since the phase-space is latent, a prior distribution over the states is required. Collecting the states in a matrix, they are denoted by

$$\mathbf{H} = (\boldsymbol{h}_k), \quad k = 0, \dots, K. \tag{3.52}$$

The approach taken in the thesis is to model only the distribution over the initial state $\boldsymbol{h}_0$, and treat the dynamics as being part of the likelihood function. This is justified by noting that geometric mechanics, and the relevant geometric structure is valid for deterministic ODEs.[14] Denoting the repeated composition of the flow map by

[14] An intuitive example is conservation of energy, which clearly does not hold if noise is added into the system.

$$\boldsymbol{h}_k = F_\theta^k(\boldsymbol{h}_0, \eta) = (\underbrace{F_\theta \circ \ldots \circ F_\theta}_{k \text{ times}})(\boldsymbol{h}_0, \eta), \quad k \geqslant 1, \tag{3.53}$$

then the generative model over observations and the latent initial state is given by

$$p_\theta(\mathbf{Y}, \boldsymbol{h}_0) = p(\boldsymbol{h}_0) \prod_{k=1}^{K} p_\theta(\boldsymbol{y}_k | f(\boldsymbol{h}_k)), \tag{3.54}$$

where f is an observation function that relates the state $\boldsymbol{h}_k$ to the observation $\boldsymbol{y}_k$, and $p(\boldsymbol{h}_0)$ is the isotropic Gaussian given in eq. (1.60),

$$p(\boldsymbol{h}_0) = \mathcal{N}(\mathbf{0}, \mathbf{1}_{2C}). \tag{3.55}$$

For the problems considered in section 3.3 this prior is not found to be overly restrictive, however, performance may be degraded if the posterior is highly correlated, as discussed in section 1.4.[15]

[15] Such a situation may arise, for example, when the observation function is nonlinear, since $\boldsymbol{h}_0$ represents position and velocity/ momentum coordinates which may not be easily inferred from the observations.

The observation function f plays the same role as the observation function in definition 2. However, due to the inherent structure of the phase-spaces, it is worth exploring in more detail:

‣ For noisy phase-space observations, the observation function will be the identity function and the observation model is given by the Gaussian in (1.21),

$$p_\theta(\boldsymbol{y}_k | f(\boldsymbol{h}_k)) = \mathcal{N}(\boldsymbol{y}_k | \boldsymbol{h}_k, \sigma^2 \mathbf{1}_D). \tag{3.56}$$

‣ For pixel observations, the observation function will be neural network f_θ and the observation model is given by the Bernoulli in (1.22). The Bernoulli is chosen since the pixel observations are greyscale. Specifically,

$$p_\theta(\boldsymbol{y}_k | f(\boldsymbol{q}_k)) = \mathcal{B}(\boldsymbol{y}_k | f_\theta(\boldsymbol{q}_k)), \tag{3.57}$$

and only the position coordinate is used as input. This is important for two related reasons. First, the observations are assumed to be single frames, and f should not use future states (3.48) or momenta (3.49) to generate them. Second, the dimension of the configuration space is typically much smaller than the observation space ($C \ll D$), and since f is agnostic to the physics it can use the extra dimensions to store non-physical information about the frames that improves the log-likelihood (reconstruction error of images).

The generative model in (3.54) is similar to the one presented in section 1.4. The main difference is that the data is in the form of time-series. However, since only the initial state is latent and the dynamics is taken to be part of the likelihood, the different steps of the series could be thought of as extra output dimensions in eq. (1.56).

Variational Inference. To approximate the posterior distribution over the initial state, a variational distribution parametrised by an inference network is used. Specifically, a Gaussian family of the form given in (1.73) is specified where the input to the network is a sequence of observations,

$$p_\theta(\boldsymbol{h}_0|\mathbf{Y}) \approx q(\boldsymbol{h}_0|h_\phi(\mathbf{Y})) = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)), \tag{3.58}$$

where $\mathbf{Y}$ denotes the full sequence for a single trajectory. In section 3.3, the sequence is limited to the first K steps, with K being a hyperparameter. Since the distribution is only over the initial state, it is reasonable for many problems to assume that only a truncated sequence of steps are needed. However, as the observation function gets more complex this gets harder to reason about.

The ELBO corresponding to the generative model in (3.54) and the variational posterior in (3.58) follows directly from eq. (1.65). In particular, given a time-series from the dataset in (3.47),

$$\mathcal{O}_{ELBO}(\theta, \phi; \mathbf{Y}) = \mathbb{E}_{q_\phi^{\boldsymbol{h}_0|\mathbf{Y}}}\Big[\log p_\theta(\mathbf{Y}|\boldsymbol{h}_0)\Big] - \mathbb{KL}\big[q_\phi^{\boldsymbol{h}_0|\mathbf{Y}}||p^{\boldsymbol{h}_0}\big] \tag{3.59}$$

$$= \mathbb{E}_{q_\phi^{\boldsymbol{h}_0|\mathbf{Y}}}\left[\sum_{k=1}^{K} \log p_\theta(\boldsymbol{y}_k|f \circ F_\theta^k(\boldsymbol{h}_0, \eta))\right] - \mathbb{KL}\big[q_\phi^{\boldsymbol{h}_0|\mathbf{Y}}||p^{\boldsymbol{h}_0}\big], \tag{3.60}$$

which is optimised with respect to the model parameters θ and variational parameters ϕ using SVI (1.81). Specifically, the expected log-likelihood is computed by sampling (eq. (1.78)) and the KL term is computed using eq. (1.77), from a batch of sequences.

3.3 Learning from Physical Systems (Experiments)

The results in this section extend the experiments from Saemundsson et al. [2020]. The work explores learning predictive models of physical dynamical systems using VINs, and compares them to Neural-ODEs (N-ODEs) [E, 2017, Haber and Ruthotto, 2017, Chen et al., 2018], Hamiltonian neural networks (HNNs) [Greydanus et al., 2019] and (recurrent) residual networks (ResNets) [He et al., 2016]. The learning problem consists of time-series observations from eq. (3.47), originating from simulated Lagrangian/Hamiltonian ODEs. The aim is to learn the generative model in eq. (3.54), including the dynamics F_θ and potentially nonlinear observation function f_θ.

The first subsection (section 3.3.1) describes the problem setting in detail, including the simulated systems and data generation. The next subsection (section 3.3.2) contains experiments using noisy phase-space observations from physical systems. The final subsection (section 3.3.3) contains experiments using pixel (image) observations of physical

systems. The experiments study the following questions:

1. Do VINs preserve physical geometry? Variational integrators preserve physical structure such as configuration manifolds, symplecticity and conservation laws (section 3.1). The experiments evaluate the energy behaviour and the path of the learned dynamics to assess these properties for VINs and the baselines.
2. Does the inductive bias improve data efficiency? To evaluate the quality of the inductive bias imposed by VINs, we compare them to the baselines in terms of data-efficiency.
3. Can VINs be used to learn physical representations from images? When the observation function in eq. (3.54) is nonlinear, the physical geometry can be used as a general-purpose inductive bias in the latent space. As a proof-of-concept, section 3.3.3 looks at learning VINs from pixel observations of physical systems.

Additional details about model architectures, hyperparameters, and data generation is given in appendix B.

3.3.1 Problem Setting

Experimental Setup. To study the performance of VINs, data is generated from two physical systems: (a) an ideal pendulum, and (b) an ideal mass-spring system. The Lagrangian/Hamiltonian for the pendulum system are given by

$$L_{\mathrm{p}}(q, \dot{q}) = T_{\mathrm{p}}(\dot{q}) - U_{\mathrm{p}}(q) = \frac{1}{2} m r^2 \dot{q}^2 + mg(1 - \cos q), \tag{3.61}$$

$$H_{\mathrm{p}}(q, p) = T_{\mathrm{p}}(p) + U_{\mathrm{p}}(q) = \frac{1}{2mr^2} p^2 + mg(\cos q - 1), \quad q \in (-\pi, \pi), \tag{3.62}$$

where $q, \dot{q}, p$ are the angle (with respect to the vertical axis), angular velocity and angular momentum respectively; m is the mass of the pendulum bob, r is the length of the pendulum and g is the acceleration due to gravity. The Lagrangian/Hamiltonian for the mass-spring system are given by

$$L_{\mathrm{ms}}(q, \dot{q}) = T_{\mathrm{ms}}(\dot{q}) - U_{\mathrm{ms}}(q) = \frac{1}{2} m \dot{q}^2 - \frac{1}{2} k q^2, \tag{3.63}$$

$$H_{\mathrm{ms}}(q, p) = T_{\mathrm{ms}}(p) + U_{\mathrm{ms}}(q) = \frac{1}{2m} p^2 + \frac{1}{2} k q^2, \quad q \in \mathbb{R}, \tag{3.64}$$

where $q, \dot{q}, p$ are the displacement, velocity and linear momentum respectively; m is the mass, and k is the spring constant (stiffness of the spring). Since there is no friction and no external forces, and neither Lagrangian (3.61)–(3.63) depends on time explicitly, the path due to the equations of motion (EOM) from eq. (1.7) conserves the total energy of

the system from eq. (1.11), given by their Hamiltonians, eqs. (3.62), (3.64).

The systems are simulated using a 5th order Runge-Kutta method (RK45) [Dormand and Prince, 1980], implemented in the SciPy package [Virtanen et al., 2020], to generate trajectories of fixed length T,[16]

$$\mathbf{X} = (\boldsymbol{x}_t), \quad t = 1, \dots, T, \tag{3.65}$$

where $\boldsymbol{x}_t = (q_t, p_t)$ and the error tolerance is set at 10^{-9}. The initial states were sampled uniformly at random, bounded from above and below by energy to avoid pathological behaviour. Details are given in appendix B. The sampling rate for the observations is set at 0.1s/10Hz, which is typically larger than the step size of RK45 (which is adaptive). Noise was added to the generated states to produce the observations used for learning,

$$\boldsymbol{y}_t = \boldsymbol{x}_t + \boldsymbol{\epsilon} \odot \sigma_{\boldsymbol{x}}, \tag{3.66}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}_2)$, $\sigma_{\boldsymbol{x}}$ is an experimental parameter used to study noise-robustness and $\odot$ denotes element-wise multiplication.

[16] Note that T does not have to equal K from section 3.2.4. The former denotes the length of the data trajectory whereas the latter is a model hyperparameter.

Baselines. The baselines in section 3.3.2 are the N-ODE and HNN as described in [Greydanus et al., 2019], and a recurrent ResNet as described in section 3.1.1. HNNs are closely related to VINs since both function classes are restricted to Hamiltonian/ Lagrangian mechanics.[17] The key difference between the two is that HNNs operate in continuous-time like N-ODEs, eq. (3.7), whereas VINs operate in discrete-time. This is similar to the relationship between N-ODEs and ResNets.

[17] Technically, the HNN presented in [Greydanus et al., 2019] is formulated only in terms of Hamiltonian physics. The modification of the same approach to Lagrangians is explored in [Cranmer et al., 2020b] but is not included in the experiments.

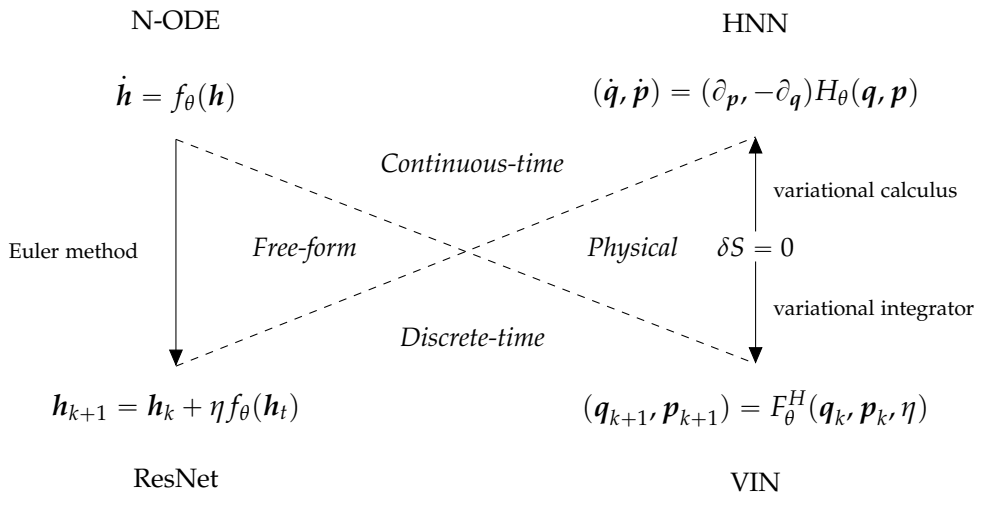


Figure 3.1: Relationship between baselines and VINs. N-ODE is a ODE parametrised by a NN, eq. (3.7). HNN adds physical structure in continuous-time. ResNet is an Euler discretisation of a N-ODE, eq. (3.13). VIN is a discrete-time counterpart to the HNN, eq. (3.51).

Figure 3.1 illustrates the high-level relationships between the baselines and VINs. Note

that VINs are *not* a direct discretisation of HNNs but rather use variational integrators which discretise Hamilton's principle eq. (1.5), as described in section 3.2. However, VINs are usefully thought of as the discrete-time counterpart to HNNs.[18]

[18] Specifically, if one assumes that the Hamiltonian in the HNN is separable and given by a Newtonian Hamiltonian, then a VIN is obtained by applying a suitable discretisation. For example the one given in eq. (3.35).

3.3.2 Noisy Observations

This subsection compares VINs to N-ODEs, HNNs and ResNets in terms of data-efficiency and qualitative forecasting behaviour. Data from the mass-spring system and pendulum are generated according to eqs. (B.3)–(3.66). The VIN architecture used in this subsection is the VIN-VV architecture from eq. (3.39). All the models use a single fully-connected neural network FCNN eq. (1.30) with one hidden layer, consisting of 500 units, and softplus activations [Zheng et al., 2015]. Softplus is chosen over the ReLU activation since the HNN and VIN models require the gradients with respect to the inputs.[19]

[19] The softplus can be seen as a smooth approximation to the ReLU.

Objective. The training objective is chosen as the squared error ($L2$) norm between predictions and observations to match the setup from [Greydanus et al., 2019]. However, the N-ODE and HNN (continuous-time) differ from the ResNet and VIN (discrete-time) models in terms of targets. Specifically, the continuous-time models are trained to predict the derivative observations,

$$L_{\text{N-ODE}} = \left|[\dot{q}, \dot{p}] - f_\theta(q, p)\right|_2 , \tag{3.67}$$

$$L_{\text{HNN}} = \left|[\dot{q}, \dot{p}] - [\partial_p, \partial_q] H_\theta(q, p)\right|_2 , \tag{3.68}$$

where $|\cdot|_2$ is the $L2$ norm. In contrast, the discrete-time models predict a sequence of states

$$L_{\text{ResNet}} = \sum_{k=1}^{K} \left|[q_{k+1}, p_{k+1}] - \left([q_k, p_k] + \eta f_\theta(q_k, p_k)\right)\right|_2 , \tag{3.69}$$

$$L_{\text{VIN}} = \sum_{k=1}^{K} \left|[q_{k+1}, p_{k+1}] - F_\theta^H(q_k, p_k, \eta)\right|_2 . \tag{3.70}$$

Note that in contrast to section 3.2.4, eq. (3.70) corresponds to assuming that the initial condition is observed and that the noise variance in the likelihood from eq. (3.56) is fixed. This is done to more closely match the setup for the continuous-time models. In the experiments, $K = 9$ is used for the VIN and ResNet models and the step size is chosen as $\eta = 0.1$, matching the sampling rate of the observations.[20] The simulated sequences were generated in 10 step (1.0s) intervals to match the prediction horizon.

[20] Preliminary experiments were also run for different values of K. For $K \geqslant 20$, the optimisation during training was found to get stuck in bad local optima. Values of K between $5 - -10$ were found to work well in these initial experiments.

Training. The amount of training data is varied between $T = \{40, 80, 120, 160, 200\}$ steps

sampled at 10Hz. The amount of noise is varied between $\sigma_x = \{0.1, 0.2, 0.3, 0.4\}$. For each value of T and σ_x, the experiment is repeated for 10 different seeds producing different training datasets. Equations (3.67)–(3.70) are then optimised using Adam [Kingma and Ba, 2017], with a learning rate of 3×10^{-4} and a batch size of 100. For each value of T, a validation dataset of equal size to the training dataset is used for early stopping. Specifically, training is stopped if no improvement is made on the validation dataset for 3 epochs or the maximum number of epochs 2×10^4 is reached.

Evaluation. The models are compared quantitatively in terms of root-mean-squared-error (RMSE) of predicted phase-space trajectories. Since the continuous- and discrete-time models are trained on different observations and objectives, their comparisons have confounding variables. However, the relative relationship between N-ODEs and HNNs in contrast to ResNets and VINs is still informative, which is the main reason the continuous-time baselines are included. The training, validation, and test datasets were generated for each seed and kept fixed across the different models. No effort beyond the random sampling of initial states was made to ensure that the test set did not have significant overlap with the observed data. See appendix B.1 for more details about the data.

For the continuous-time models, predictions are made by using the RK45 solver on the learned models. For the discrete-time models, predictions are made by iteratively using their respective update equations, (3.13) and eq. (3.39) respectively. All the models are evaluated on the same 25 trajectories, which are generated independently of the training data and kept fixed across the random seeds also. The initial conditions for the evaluation trajectories are given noise-free and the RMSE is computed for a 50−step/5s prediction horizon. The qualitative evaluation is presented for 200−step/20s prediction horizons. The difference is due to the erratic behaviour of the N-ODE and ResNet models for longer prediction horizons which makes the RMSE diverge.

Data Efficiency. Figures 3.2 (a)–(b) summarise the quantitative results for the mass-spring and pendulum system respectively. In the left column of the figures, each disc corresponds to one experiment, across all noise levels, and the box-plot denotes the minimum/maximum by the whiskers, and the 25−50−75th percentiles respectively. The median (50th percentile) is the horizontal line in the middle of each box. In the right column of the figures, each disc corresponds to the median of the highest and lowest noise levels $0.1, 0.4$ respectively. The models are differentiated by colour (and order on the left), N-ODE (red), HNN (purple), ResNet (green) and VIN-VV (blue). Figure 3.2

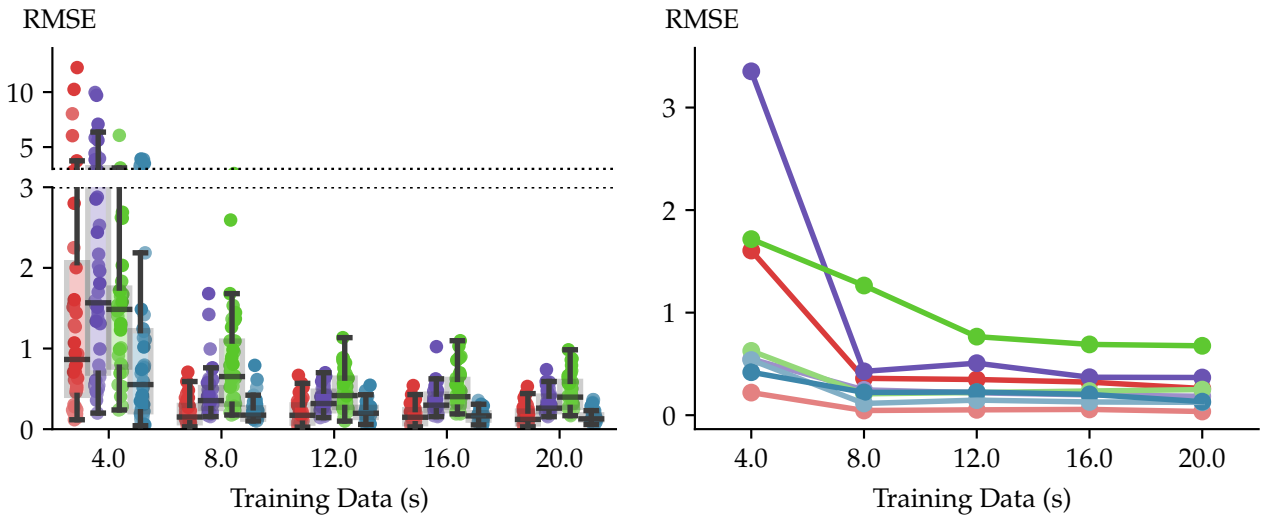
shows that:

- (a) Mass-spring system: The N-ODE performs best in terms of RMSE for low-noise $\sigma_x = 0.1$ and the VIN performs best for high-noise $\sigma_x = 0.4$. This is seen by the overall lowest median value for the N-ODE on the left and more explicitly by the median plots on the right. Surprisingly, the N-ODE outperforms the HNN despite its built-in physical structure (discussed below under *Predictions and Energy*). For the lowest amount of training data (4s), the VIN model performs best overall, however this performance is not robust to high noise levels $\sigma_x = 0.4$. The ResNet performs worst overall in terms of RMSE and also exhibits the highest variance.
- (b) Pendulum system: As illustrated by the median plot on the right, the HNN performs best in terms of RMSE for both low- and high-noise ($\sigma_x = 0.1, 0.4$) experiments given 12s of training data or more. For $4, 8$s of training data, the HNN performs better than the VIN in the low-noise experiments but worse in high-noise experiments. In contrast to the mass-spring system, the N-ODE results are highly bi-modal, as seen in the left column. At low-noise levels, the N-ODE performs on par with the VIN but at high-noise levels it performs the worst as seen on the right. Excluding the high-noise N-ODE results, the ResNet performs the worst and surprisingly exhibits an upwards trend in RMSE with increasing data (discussed below under *Predictions and Energy*).

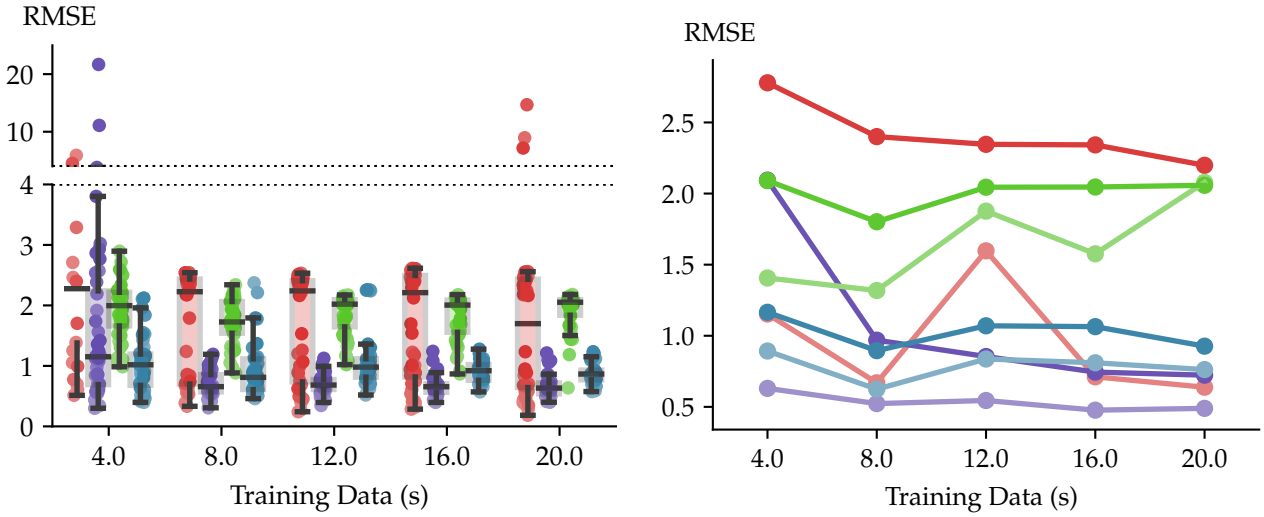
The key takeaway from fig. 3.2 comes from comparing the ResNet to the VIN. Clearly, the introduction of physical structure improves data-efficiency in the discrete-time case. This is evident at all data and noise amounts in fig. 3.2. Figure 3.2 also contains two surprising results that are explored next: the N-ODE outperforms the HNN on the mass-spring system and the ResNet performs worse with increasing amounts of data on the pendulum system.

Predictions And Energy. To shed further light on the observations above, Figure 3.3 shows the predicted trajectories and associated energies for (a) the mass-spring system, and (b) the pendulum system; for the experiments with 20s of training data and high noise levels $\sigma_x = 0.4$, and plotted in fig. 3.2. Each line for the model corresponds to one of the 10 random seeds. The upper row shows the phase-space with the ground truth in black and the model predictions in colour.

The initial state is given by a black disc and the final state by a cross (white for the ground truth and in colour for the model). The lower row shows the energy, given by the Hamiltonian in eq. (3.64) for the model (colour) and the ground truth (black). Comparing the N-ODE (left-most) to the HNN (second-to-left) partly explains why the former



(a) Mass-spring system.



(b) Pendulum system.

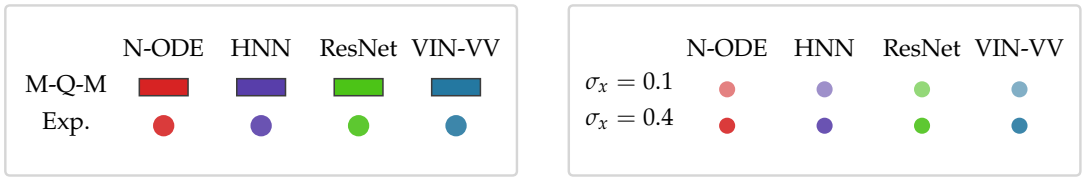


Figure 3.2: RMSE across varying noise levels $\sigma_x = [0.1, 0.2, 0.3, 0.4]$, amount of training data and 10 random seeds; on (a) the mass-spring system and (b) the pendulum system. The *left* column shows each experiment as a coloured disc for the different models: N-ODE (red), HNN (purple), ResNet (green), VIN (blue). The box-plots denotes the minimum/maximum by the whiskers, and the 25−50−75th percentiles respectively. The median (50th percentile) is the horizontal line in the middle of each box. The RMSE axis is broken to account for outliers in the scaling. The *right* column plots the median (trend) of the highest and lowest noise levels (0.1, 0.4). Adding physical structure in the discrete-time case is clearly advantageous on both systems, as seen by comparing ResNets (green) to VINs (blue) on both the left and right.

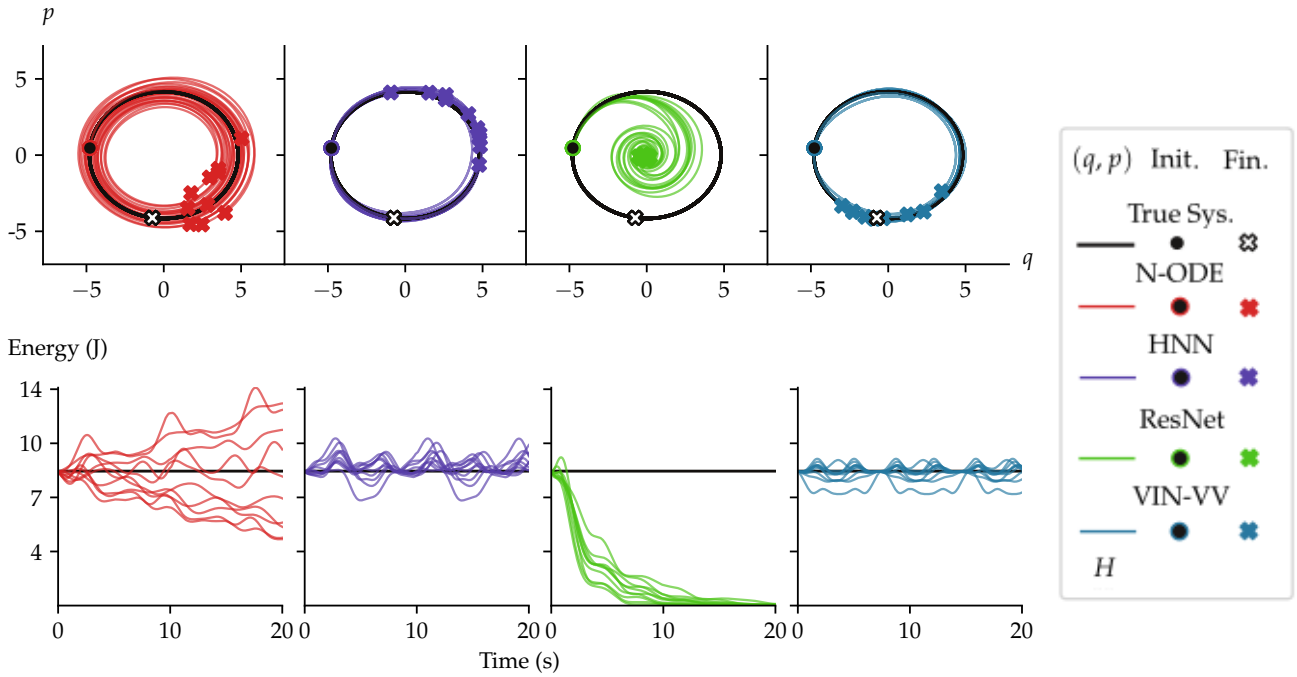
outperforms the latter in terms of RMSE. The N-ODE model evolves on paths (q, p) that dissipate or add energy as seen in the lower row. In contrast, the HNN evolves on paths that oscillate around the true energy.

Looking at the final state only, i.e. the crosses in the upper row, it is qualitatively clear that the N-ODE can be closer in RMSE to the ground truth final state (white cross) despite evolving on non-physical (energy diverging) trajectories.[21] This is because the the RMSE is computed in terms of Euclidean distance so that points off the manifold can achieve lower error. The VIN model also oscillates around the true energy but performs better in terms of RMSE in this setting. One of the key differences between the two is that the VIN uses a sequence of steps for learning. Therefore it is reasonable to think that the HNN would perform better if it were trained on sequences. Figure 3.3 (b) shows the predicted trajectories and associated energies for the pendulum system. Since the pendulum has more complicated dynamics than the mass-spring system (non-linear vs. linear forces), the diverging energy behaviour of the N-ODE can lead to highly erratic predictions. This is clear from both the phase-space and the energy plots. In a few experiments, it learns near-stable trajectories similar to the ones in the mass-spring system. This is the reason for the bi-modal RMSE results for the N-ODE in fig. 3.2.

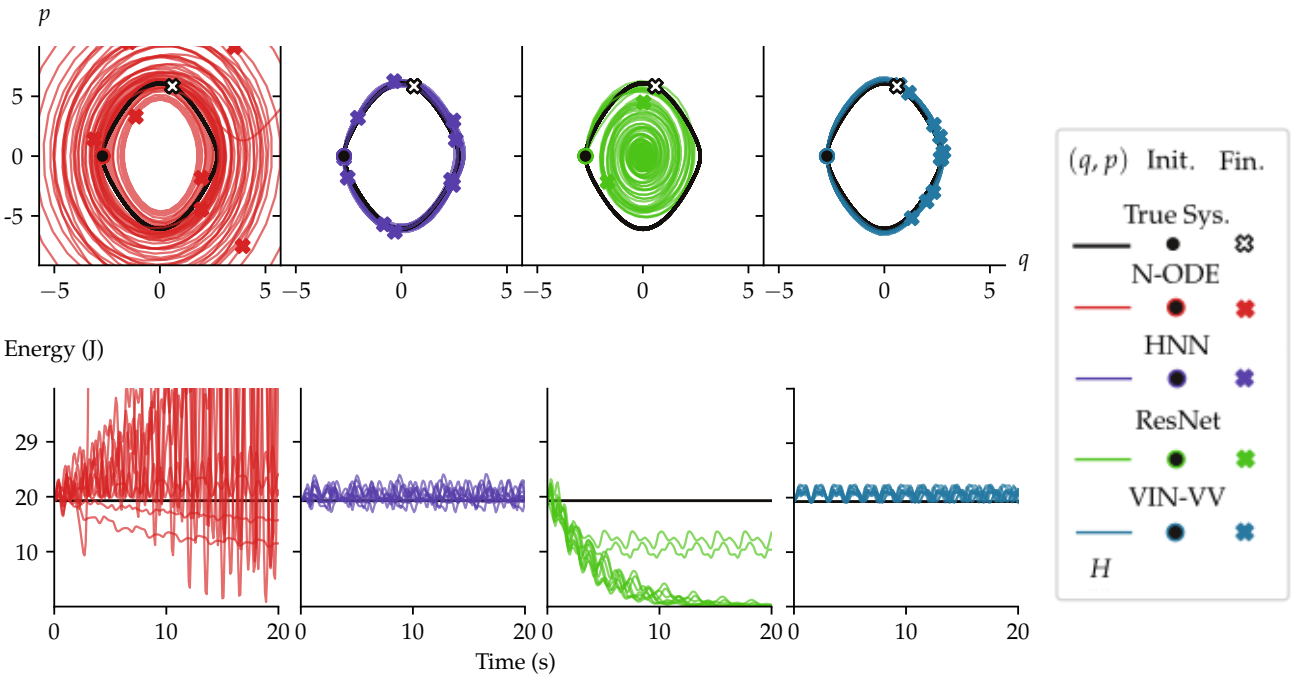
[21] The RMSE is averaged across all states, however, the final state is easier to compare visually.

Looking at the ResNet plots in fig. 3.3 (b) it is evident that it exhibits the same dissipating behaviour as in the mass-spring system. Since fig. 3.2 is given for a prediction horizon of 5s, the dissipating behaviour explains the degraded performance with increased data. Figure 3.4 makes this explicit. The columns show the energy behaviour on the pendulum system for increasing amounts of training data (left-to-right), for the ResNet and the VIN models (rows). As the training data increases, more experiments for the ResNet converge to a model that quickly dissipates energy to zero.

One hypothesis for the dissipation behaviour of the ResNet is that multiple Euler steps favour vector fields that dissipate energy on average over the training horizon. However, as the prediction horizon goes beyond the training horizon, the dissipation errors then accumulate.



(a) Mass-spring system.



(b) Pendulum system.

Figure 3.3: (Top) Phase space plots of predictions for a single test trajectory and 10 random seeds, given 20s of training data corrupted with high levels of noise $\sigma_x = 0.4$; for (a) the mass-spring and (b) pendulum systems. The true system is plotted in black and the models are plotted in colour. The initial condition is given by a disc and the final state by a cross (white for the true system and coloured for the models). (Bottom) The energy associated with the evolution given by the top row. The HNN (purple) and VIN (blue) evolve on a symplectic trajectory like the true system. The N-ODE (red) and ResNet (green) artificially add or remove energy. This can lead to significant failures, e.g. the N-ODE model on the pendulum system or the ResNet on both systems.

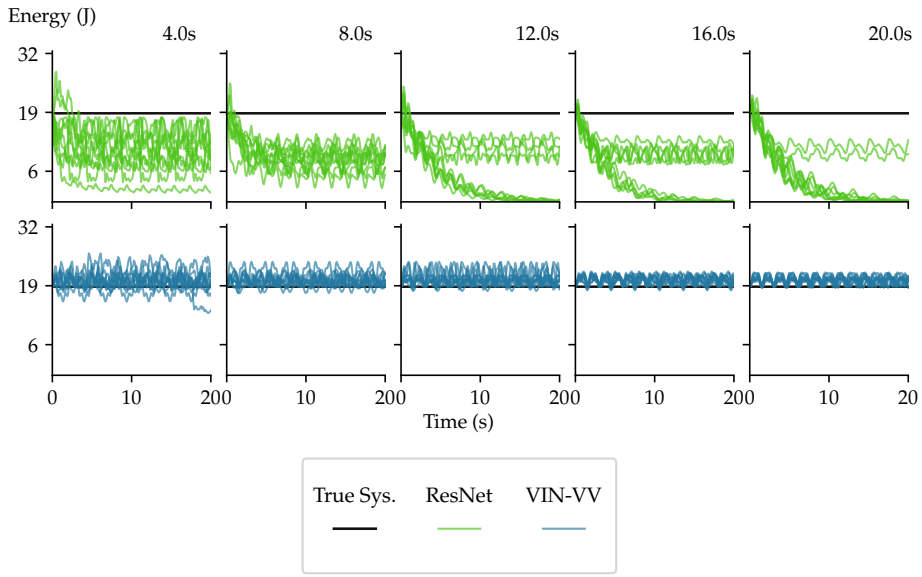


Figure 3.4: Energy behaviour of the ResNet (top, green) and the VIN (bottom, blue) models with increasing amounts of training data, from left-to-right, for each of the 10 random seeds. With increasing data, the ResNet is more likely to converge to a model that quickly dissipates energy to 0. The VIN model oscillates around the true energy with decreasing variance.

The key takeaway from figures 3.3–3.4 is that VINs approximately conserve energy and avoid the diverging energy problem exhibited by the ResNet. As shown in fig. 3.2, the dissipation of energy for the ResNet model means that predictions for a 5s horizon get worse with increasing data when the model is trained on a 0.9s horizon.

3.3.3 Pixel Observations

This subsection studies VINs in a variational auto-encoder (VAE) setting, which adds an auxiliary image processing task to prediction. The motivation is an interest in exploring the use of VINs for more general-purpose representation learning. The observations consist of $28 \times 28 \times 1$ greyscale pixel images depicting the mass-spring and pendulum systems (fig. 3.5). A detailed description of the data generation process is given in appendix B.2.

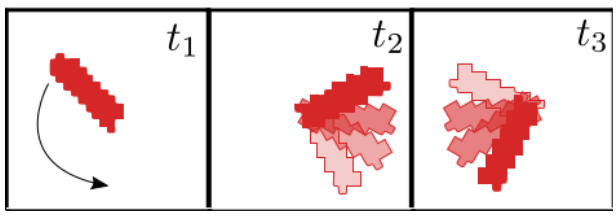


Figure 3.5: Illustration of pendulum pixel observations at different times t_1, t_2, t_3. The faded pendulums are shown only to indicate the dynamic nature of the system. The actual observations are also greyscale.

Learning physically structured representations from images is a challenging task [Greydanus et al., 2019, Toth et al., 2019]. In the experiments, we mainly compare VINs to

ResNets and do not implement the Pixel-HNN from Greydanus et al. [2019]. The reason for this is that the authors rely on an Euler discretisation and an auxiliary loss term to enforce the phase-space structure in latent space, which does not easily extend to the multi-step training setup considered here. This is one of the benefits of working with discrete-time formulations that preserve the physical structure automatically.

Another potential baseline would be Hamiltonian generative networks (HGNs) from Toth et al. [2019], since the authors also use explicit (symplectic) integrators for Hamiltonian systems in a VAE setup. However, for the hyperregular systems considered in the thesis, the latent dynamics defined by VINs and HGNs are practically equivalent. Since HGNs are more readily applied to non-separable systems (in this case the dynamics are not explicit), it can also be seen as a more general approach than the one taken here.[22]

[22] One key difference between the two is that VINs preserve momentum maps in addition to being symplectic. However, momentum maps are not explored in the experiments.

To study VINs, we implement all three network architectures from section 3.2. For the mass-spring system, the VIN-SV from eq. (3.36) is used for the dynamics. For the pendulum, both the VIN-VV from eq. (3.39) and the VIN-$SO(2)$ are used. The main baseline is a recurrent residual network (3.13) (ResNet), having the same number of layers as the VINs. For qualitative results, the results also include a standard VAE [Kingma and Welling, 2014, Jimenez Rezende et al., 2014], and a VAE with the $SO(2)$ constraint given in eq. (B.11) (VAE-$SO(2)$). The latent configuration space for the VIN models is chosen as 1D, so that the discrete phase spaces are 2D. The latent space for the ResNet is chosen as 2D to match the size of the discrete phase spaces. The training (and inference) horizon K is chosen as 9 in all the experiments.

A detailed description of the baseline models is given in appendix B.2.

▸ Training: The VIN and ResNet are trained using the ELBO from eq. (3.60). The VAE and VAE-$SO(2)$ are trained using the ELBO from eq. (1.70). All models are optimised using Adam [Kingma and Ba, 2017] with a learning rate of 3×10^{-4} and a batch size of 200. For quantitative results, the experiments are repeated for 10 random seeds and the results are averaged. No validation data is used for early stopping. Training is stopped if no improvement is made on the training data for 10 epochs or the maximum number of epochs $5{\times}10^4$ is reached.

Qualitative Intuition. The structure of the latent space learned by the VIN models is compared qualitatively to a standard VAE, a VAE with the $SO(2)$ constraint given in eq. (B.11) (VAE-$SO(2)$) and the ResNet. For the VAE and VAE-$SO(2)$, the models consist only of the first inference network h^1_ϕ and the observation model f_θ from fig. B.1.

Figure 3.6 visualises the latent spaces after training on 25s (250 observations), consist-

ing of 5 simulated trajectories of the pendulum system with 5s of data (50 observations) each. After learning the models, the observations from 5 test trajectories with 10s (100 observations) each are mapped into latent space using the inference network h_ϕ. For the ResNet and VIN models, 1s (10 observations) from the test trajectories are used for approximate inference of the initial condition and the sequence is predicted by the dynamics model. For the VAE and VAE-$SO(2)$, the inference network is used for prediction for observations. The black discs denote mean values of the training data and the coloured triangles denote mean values of the test data. The grey lines show the dynamics as they evolve in pixel space. The main points from fig. 3.6 are:

- The ResNet (top-left) exhibits diverging energy/phase space behaviour, analogous to the experiments from section 3.3.2. This is evident from the scale of the axes.
- The VAE (top-middle) captures local structure: observations close together in image space are mapped to points close together in latent space. However, it fails to capture the global structure of the state space and has discontinuities with respect to the sequential nature of the dataset.
- The VAE-$SO(2)$ (top-right) captures the correct global structure for the pendulum by restricting the manifold, but still exhibits discontinuities with respect to the time dimension, since it does not model the dynamics.
- The VIN models (bottom row) all learn latent representations consisting of elliptical or circular trajectories without discontinuities or diverging energy behaviour in the dynamics. The fact that the circles/ellipses are of different size reflects the fact that the true model is not uniquely identified.

Forecasting. To assess the models quantitatively, experiments are run on the mass-spring and pendulum systems and the predictive performance is measured in pixel-space, in terms of RMSE and log-likelihood. For both systems, 6s of training data (60 observations) from a single simulated trajectory is used to learn the models. Evaluation is then performed by measuring extrapolation performance (in time) on the same trajectory. Specifically, the models are trained using a 0.9s (9-step) prediction horizon—i.e. the 6s is broken up into 0.9s batches—but the quantitative evaluation is performed on 5s (50-step) predictions on the same trajectory. This is done to isolate properties of the dynamics model, rather than evaluate the generalisation performance of the full model, i.e. including the observation and inference models f_θ, h_ϕ. A qualitative analysis of 20s predictions is also included to explore the long-term behaviour of the latent dynamics.

Table 3.1 shows the quantitative results with standard errors from 10 random seeds.

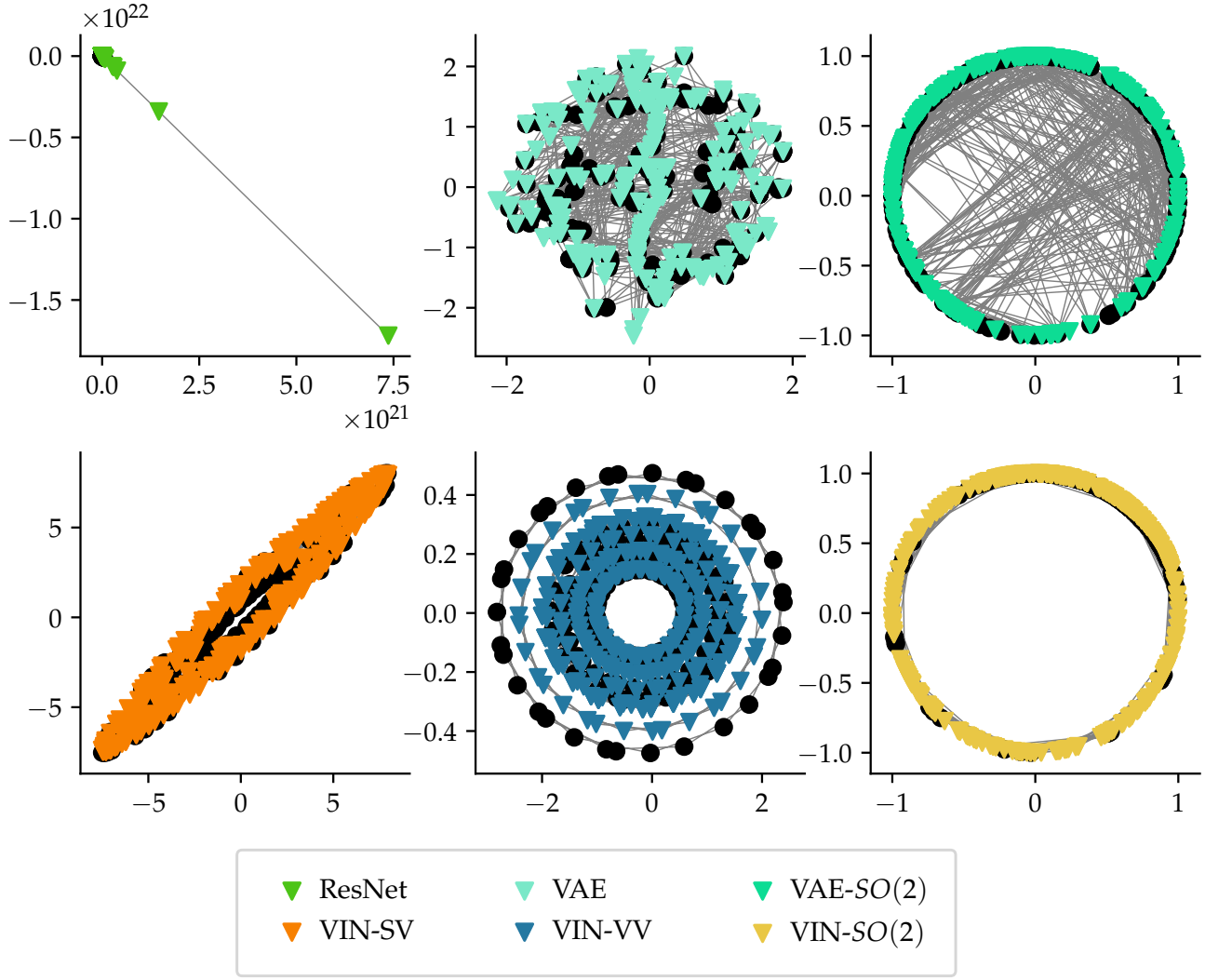


Figure 3.6: The structure of the learned latent space (2D), given 28×28 images of a pendulum in motion; for the ResNet (top-left), VAE (top-middle), VAE-$SO(2)$ (top-right), VIN-SV (bottom-left), VIN-VV (bottom-middle), VIN-$SO(2)$ (bottom-right). The black discs denote the mean of training observations encoded into latent space, and the coloured triangles denote the mean of observations from test trajectories. The grey lines plot the dynamics in pixel space, i.e. they connect points in latent space corresponding to neighbouring points in time. In the top row, the two axes represent the 2D Euclidean latent spaces. In the bottom row, the VIN-SV (left) plots the position coordinate on the horizontal axis, and the change in position on the vertical axis. The VIN-VV (middle) plots the position on the horizontal axis and the momentum on the vertical axis. The VIN-$SO(2)$ (right) plots the $\sin, \cos$ of the position on the horizontal/vertical axis respectively, representing eq. (B.11). (Top) The ResNet trajectory diverges due to the form of the Euler method. The VAE does not capture the underlying circle manifold, whereas the VAE-$SO(2)$ captures the circle by construction. However, neither models the underlying dynamics. (Bottom) The VIN models all capture physically structured manifolds due to their inductive bias. The VIN-SV traverses elliptical paths in latent space, whereas the VIN-VV and VIN-$SO(2)$ are both circular. The VIN-VV maps to circles with a different radius for each trajectory but the VIN-$SO(2)$ maps all trajectories to the unit length circle by construction.

The VIN models perform significantly better than the ResNet in terms of both RMSE and log-likelihood. The VIN-$SO(2)$ shows a meaningful improvement in terms of log-likelihood when compared to the VIN-VV on the pendulum system, whereas the RMSE is inconclusive.

To explore the results in more detail we look at examples of the predictions in pixel space and the corresponding evolution in latent space. Figure 3.7 plots example predictions from the ResNet (green), VIN-VV (blue), VIN-$SO(2)$ (yellow) and VIN-SV (orange) on the pendulum system (left), and the corresponding latent space evolution (right). The

System	**Model**	RMSE	$\log p(\boldsymbol{y}\|\boldsymbol{x}) \times 10^2$
Pendulum	ResRNN	6.1 ± 0.2	-246.7 ± 79.2
	VIN-VV	4.3 ± 0.6	-13.4 ± 5.8
	VIN-$SO(2)$	$\mathbf{3.4 \pm 0.6}$	$\mathbf{-3.2 \pm 1.9}$
Mass-spring	ResRNN	6.1 ± 0.1	-4.7 ± 2.4
	VIN-SV	$\mathbf{3.2 \pm 0.2}$	$\mathbf{-0.2 \pm 0.0}$

Table 3.1: RMSE and log-likelihood (with ± 1 standard errors) for the pendulum and mass-spring systems over $5s$ forecasts on pixel observations.

ResNet does not learn dynamics that match the geometric properties of the true system (i.e. symplectic) but instead spirals away from the initial condition (denoted by the large circle). This is because the Euler discretisation scheme used by residual networks ignores the underlying geometry. On the other hand, both the VIN-VV and the VIN-$SO(2)$ models automatically preserve symplectic structure and evolve strictly on a sub-manifold in their respective latent phase-spaces. Importantly, while the flexibility afforded by the decoder allows the ResNet setup to generate plausible observations up to some fixed horizon, the unbounded behaviour of the evolution eventually causes significant failures.

Conversely, the VINs do not exhibit such non-physical behaviour, since the latent path remains bounded on the data manifold despite forecasting for effectively arbitrary long horizons. The VIN-VV does display signs of going out of phase with the ground truth around $15s$ in fig. 3.7, becoming more pronounced around the $20s$ mark. One explanation is that we only consider the path traversed by the mean of the variational posterior, and ignore the build-up in uncertainty as the prediction horizon increases. However, looking at the same reconstructions from the VIN-$SO(2)$ model, we see that it does not suffer from this problem within the $20s$ prediction horizon. Therefore, another potential explanation for the error comes from assuming an Euclidean manifold.

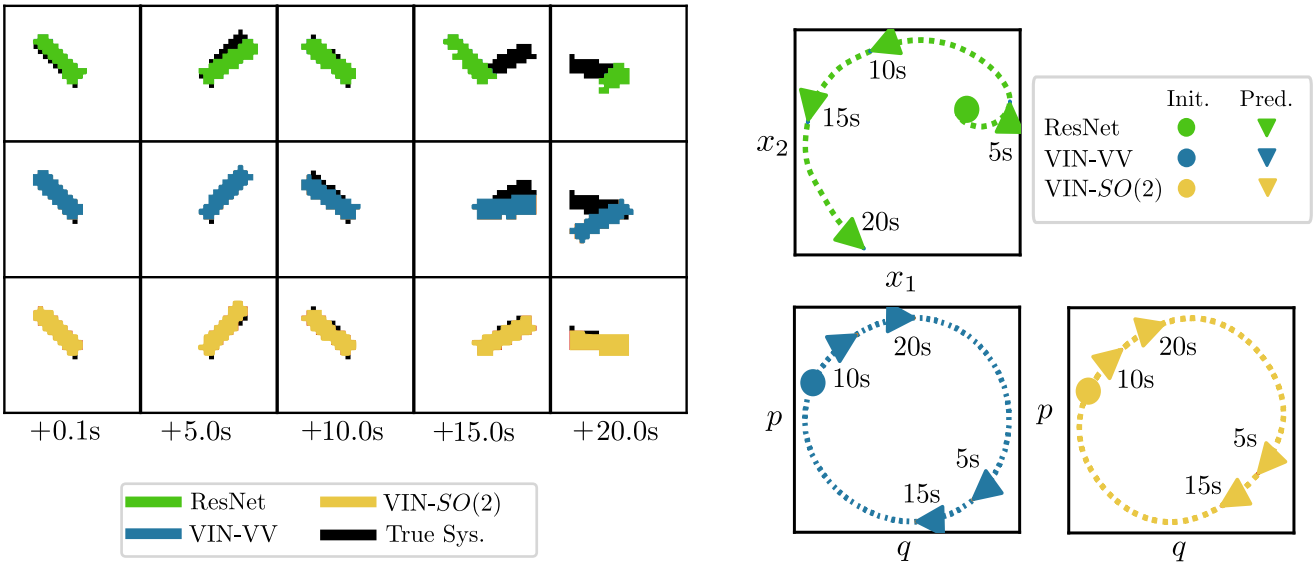


Figure 3.7: Example predictions in pixel space for the pendulum (left) and the associated paths in latent space for each model (right). Due to the diverging behaviour of the ResNet (green), the predictions in pixel space exhibit discontinuous and otherwise erratic behaviour such as disappearing. The VIN-VV (blue) evolves on a physically structured manifold in latent space, producing coherent predictions in pixel space. However, after 15s, it is out of phase with the true system. The VIN-$SO(2)$ remains faithful to the true system after 20s due to the additional constraint to the $SO(2)$ manifold.

3.4 Related Work

The work in this chapter can be seen as part of a growing body of research on structural or scientific machine learning [Battaglia et al., 2016, 2018, Rahaman et al., 2020, Willard et al., 2020, Rackauckas et al., 2020]. In particular, it combines the perspective of neural ODEs [Haber and Ruthotto, 2017, E, 2017, Chen et al., 2018, Chang et al., 2018, Ruthotto and Haber, 2018], learning physical ODEs [Solin et al., 2018, Bapst et al., 2019, Lutter et al., 2019, Raissi et al., 2019, Greydanus et al., 2019, Ehrhardt et al., 2018, Champion et al., 2019], and learning geometric embeddings [Chamberlain et al., 2017, Nickel and Kiela, 2017, Ganea et al., 2018, Davidson et al., 2018].

Closely related to VINs are Hamiltonian neural networks (HNNs) by Greydanus et al. [2019], and later Lagrangian neural networks (LNNs) [Cranmer et al., 2020b], which can be seen as the continuous-time counterparts to VINs, see fig. 3.1. In Greydanus et al. [2019], the authors learn a Hamiltonian from noise-free derivative observations and noisy phase-space observations of physical systems. In contrast, VINs are explicitly discrete-time systems. However, both approaches preserve physical structure as demonstrated empirically in section 3.3.2.

A benefit of using the continuous-time formulation is that sophisticated ODE solvers can be used to evolve the dynamics once they are learned. Conversely, requiring that derivative observations are available is a limitation. For example, to learn HNNs from images, the authors introduce an auxiliary loss term to enforce that the latent space approximately take the shape of a phase-space. Practically, this involves evolving the dynamics by one-step in the latent space in discrete-time.[23]

[23] In the current (August, 2021) implementation found online at: *github.com/greydanus/hamiltonian-nn*, this is achieved by an Euler step.

Discretisation is considered more explicitly by Toth et al. [2019] in developing Hamiltonian generative networks (HGNs). In this work Hamilton's EOM are used to construct flexible generative models, similar to the image setup in [Greydanus et al., 2019] and the VIN models in section 3.3.3. Crucially, the authors use a symplectic integrator to evolve the dynamics to ensure that the latent space corresponds to a phase-space. By replacing the symplectic integrator used in [Toth et al., 2019] with a variational integrator, the models are very similar for separable Lagrangian/Hamiltonian systems.[24] However, Toth et al. [2019] also consider generative flows, and the authors propose methods for dealing non-separable systems. Thus, the work is relevant to much more general systems. Other interesting works using symplectic integrators are found in [Chen et al., 2019, Mattheakis et al., 2019].

[24] This refers to the model in section 3.2 in Toth et al. [2019] The work also considers generative flows which are not considered in this chapter.

The work in [Zhong and Leonard, 2020] also considers learning Lagrangian systems from images in an unsupervised manner, where the systems are both non-separable and include control forces. Similarly, the work in [Allen-Blanchette et al., 2020] develops a video prediction model based on a non-separable Lagrangian system. Clearly, both approaches are applicable to more general systems than the developments in this chapter. An overview of related approaches is also given in a recent paper by Botev et al. [2021].

Another related approach is found in Lutter et al. [2019], which proposes an architecture that imposes Lagrangian mechanics, and is optimised to minimize the deviation from the of the Euler-Lagrange EOM. A similar idea is also used in [Raissi et al., 2019] to learn general non-linear differential equations from physics. A potential drawback of encoding physical plausibility through the loss function is the need for training data that reasonably covers the configuration space. In other words, physical structure is only preserved in areas of phase space where the loss is equal to zero. In contrast, VINs preserve physical structure automatically. Conversely, both approaches [Lutter et al., 2019, Raissi et al., 2019] apply to a more general class of physical systems.

3.5 Summary & Discussion

The work in this chapter connects the theory of variational integrators to the design of physically structured NNs. The resulting network architectures are called variational integrator networks (VINs). Analogous to the view of ResNets as Euler discretised N-ODEs, VINs are derived by parametrising Hamilton's principle through the Lagrangian and discretising the system with a variational integrator (section 3.2). Variational integrators, and therefore VINs, preserve important physical structures:

- Symplecticity, a measure of signed area which also implies conserved phase-space volume. Hamiltonian/Lagrangian flows are symplectic;
- Momentum maps, that correspond to conservation laws and conserved quantities;
- Lie group structure (when defined on Lie groups) of the configuration manifold.

Results. To explore this idea, section 3.2.1 derives VINs for Newtonian physics and section 3.2.4 proposes a framework for learning them from observations. Three variants of Newtonian networks are derived: one from a Lagrangian perspective (VIN-SV), one from a Hamiltonian perspective (VIN-VV) and one that is constrained to the $SO(2)$ manifold (VIN-$SO(2)$).

Figure 3.2 demonstrates clearly that VIN-VV improves data efficiency in comparison to the unstructured ResNet baseline; in terms of RMSE generalisation performance across varying levels of noise for the mass-spring and pendulum systems. Figure 3.3 shows how the VIN-VV is symplectic and approximately conserves energy. Finally, figures 3.6–3.7 and table 3.1 give indicative results that VINs can be used to learn physical latent spaces from pixel observations.

Limitations. The main limitation of VINs, as presented in section 3.2, is that the underlying integrator is required to be explicit. For example, if the Lagrangian/Hamiltonian is not separable, then it is not possible to derive an explicit update equation. In contrast, several related works [Toth et al., 2019, Zhong and Leonard, 2020, Allen-Blanchette et al., 2020] are more readily adapted to more general settings. However, it should be noted that this is not a limitation of variational integrators in general.

The experiments in section 3.3 are also limited. In particular, only the ResNet model is evaluated as a discrete-time baseline, when many interesting physically structured neural networks have been proposed in the literature [Botev et al., 2021]. From this point of view, the results only validate that having physical structure is more data efficient

than not, assuming it is present in the true system. Similarly, the experiments on pixel observations are only performed with the correct latent dimension given to the model, so no indication of whether or not VINs can discover the true dimension is uncovered. Further, the experiments do not measure generalisation of the full model, but only the properties of the dynamics in data-scarce regimes.

Future Research. VINs give a theoretically principled approached to deriving physically structured network architectures from continuous-time systems. However, the theory in this chapter is limited to simple systems. One way to extend VINs would be to use implicit integrators. When the integrator is not explicit, evolving the dynamics requires fixed-point algorithms. However, commonly, only a few iterations of Newton's algorithm are required to solve for the next step [West, 2004] and highly efficient solutions have been developed in terms of the number of bodies/particles [Lee et al., 2020].[25]

This chapter only considered learning from physical systems. However, the structure of conservation laws and configuration manifolds might also be useful for more general-purpose representation learning [Higgins et al., 2018]. This is explored by Toth et al. [2019], for example, the symplectic nature of the flow is used to define efficient[26] flow-based generative models with added interpretability.

[25] This means that a naive but perhaps practically viable approach to extending the work in this chapter is simply to implement a few steps of Newton's algorithm as part of the model.

[26] Since phase-space volume is conserved, the distribution does not have to be normalised despite being transformed by potentially nonlinear dynamics. This requires that the discrete flow be reversible, e.g. the VIN-SV model which can trivially be rearranged for the reverse flow.

4

Conclusion

Life can only be understood backwards; but it must be lived forwards.

—Søren Kierkegaard.

This thesis explored conservation laws as inductive biases in the context of learning predictive models of physical dynamical systems. The main aim was to improve data efficiency by encoding relevant assumptions into the learning algorithm.

Chapter 2 identified conserved quantities that explicitly parametrise physical systems (e.g. length, mass) with a latent task variable. By combining these with a global model shared across many tasks, the derived models can be used for multi-task, transfer and meta-learning of physical dynamical systems–thereby improving data efficiency.

Chapter 3 considered conservation laws and conserved quantities more generally in the form encoding geometric mechanics into neural networks. The resulting variational integrator networks (VINs) are shown to improve data efficiency when learning noisy data, and limited results are presented that show that they can be used to learn physical latent spaces from image observations.

Is the theory developed in the thesis practically relevant? Without further work—no.

However, learning accurate predictive models of physical dynamical systems has a large number of application areas in the general case, a prime example being model-based reinforcement learning (RL) for robotics [Polydoros and Nalpantidis, 2017]. From this perspective, an interesting idea might be to combine the ideas from the two chapters. Generally speaking, extending the ideas to practically relevant settings would require addressing some fundamental limitations.

Task Representations and Physical Geometry

One aspiration for combining the ideas from chapters 2–3 would be to achieve data efficiency gains from both. The general aim being to use multi-task and transfer learning to share information between predictive models with inbuilt physical structure. Since VINs explicitly contain system parameters such as mass and length, these can naturally be treated as the dimensions of a latent task variable. Similarly, the global model can be identified with the laws of physics, as specified by the VIN.[1]

The methods developed in chapter 2 for multi-task and transfer learning then apply directly to the models developed in chapter 3—with the exception that the global model is not treated probabilistically. Theoretically, however, using the task variable for multi-task and transfer learning does not require a probabilistic global model.[2] This combination would then allow for transfer learning by inferring only the physical parameters of a new system, and reusing the rest of the model.

In order to apply the combination of ideas to RL problems, it is necessary to consider how control forces affect the form of the VINs.[3] Since the VINs considered in chapter 3 do not account for dissipative forces (e.g. control forces), it is not possible to directly apply them to the RL setting considered in chapter 2. However, follow up work by Hochlehnert et al. [2021] addresses the question of how to introduce dissipative forces.[4]

Overall, the work in chapters 2–3 is only applicable to relatively simple physical systems. In chapter 2, this is exemplified by the use of low dimensional latent spaces, representing only a few system parameters. Additionally, the use of Gaussian processes poses potential scalability issues. In chapter 3, a key limitation is the requirement for explicit variational integrators, representing separable Lagrangian/Hamiltonian systems. In light of these, some suggestions for potential future work are outlined below.

[1] For example, the mass parameters in eq. (3.36) could be treated as the latent task variable and the global parameters would model the potential function.

[2] One could also consider using variational integrators with Gaussian processes.

[3] Since the models in chapter 2 are not explicitly physical, the control forces could simply be added as additional input dimensions, see eq. (2.2).

[4] As well as contact forces/collisions.

Limitations

Scalability. The Gaussian process (GP) model used in chapter 2 makes strong independence assumptions about the output dimensions for reasons of computational tractability. Scaling GPs to large dimensional systems is non-trivial, however, many potential approaches are available [Eriksson et al., 2018, Zhe et al., 2019, Pandita et al., 2019, Liu et al., 2020, Padidar et al., 2021]. Alternatively, neural processes [Garnelo et al., 2018] give a more scalable alternative that retains the probabilistic framework.

Dimension of Latent Task Space. The latent task spaces considered in chapter 2

were only 2-dimensional. Since these represent the system parameters, it is reasonable to assume that higher dimensional latent spaces would be required. At the same time, the approach requires that the latent task space be meaningfully aligned with the true task space. From this perspective, approaches from disentangled representations [Higgins et al., 2018, Achille and Soatto, 2018, Anselmi et al., 2016] offer a potential way forward.

Simple Physics. The physics modelled by the VINs in chapter 3, is not general enough to be practically relevant. However, VINs have already been extended to physics beyond the work in this chapter. Desai and Roberts [2020] propose variational integrator graph networks (VIGNs), i.e. a combination of VINs and graph NNs. Hochlehnert et al. [2021] add dissipation and contact forces to VINs. Havens and Chowdhary [2021] propose forced VINs (FVINs) to learn predictive modelling and control of mechanical systems. Many options have not been explored at the time of writing.[5] For example, variational integrators exist for highly complicated rigid-body robotic systems defined on Lie group product manifolds [Lee et al., 2020], for Maxwell's equations [Stern et al., 2008], for partial differential equations [Kraus and Maj, 2015] and fluids [Liu et al., 2015] and for stochastic physical systems [Bou-Rabee and Owhadi, 2009], to name a few.

[5] August, 2021

Bibliography

Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018.

Amina Adadi. A survey on data-efficient algorithms in big data era. *Journal of Big Data*, 8 (1):24, January 2021. ISSN 2196-1115. DOI: 10.1186/s40537-021-00419-9.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

Christine Allen-Blanchette, Sushant Veer, Anirudha Majumdar, and Naomi Ehrich Leonard. Lagnetvip: A lagrangian neural network for video prediction. *arXiv preprint arXiv:2010.12932*, 2020.

Mauricio A. Àlvarez, Lorenzo Rosasco, and Neil D. Lawrence. Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning (FTML)*, 4(3):195, 2012.

James R Anderson and Carsten Peterson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

Fabio Anselmi, Lorenzo Rosasco, and Tomaso Poggio. On invariance and selectivity in representation learning. *Information and Inference: A Journal of the IMA*, 5(2):134–158, 2016.

Kendall Atkinson, Weimin Han, and David E. Stewart. *Numerical Solution of Ordinary Differential Equations*. John Wiley & Sons, October 2011. ISBN 978-1-118-16452-5.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018.

Victor Bapst, Alvaro Sanchez-Gonzalez, Carl Doersch, Kimberly Stachenfeld, Pushmeet Kohli, Peter Battaglia, and Jessica Hamrick. Structured agents for physical construction. In *International Conference on Machine Learning*, pages 464–474. PMLR, 2019.

Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

J. Baxter. A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research*, 12: 149–198, March 2000. ISSN 1076-9757. DOI: 10.1613/jair.731.

FRS Bayes. An essay towards solving a problem in the doctrine of chances. *Biometrika*, 45 (3-4):296–315, 1958.

Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 969 vol.2–, July 1991. DOI: 10.1109/IJCNN.1991.155621.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35 (8):1798–1828, 2013.

CM Berners-Lee. Cybernetics and forecasting. *Nature*, 219(5150):202–203, 1968.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

Aleksandar Botev, Andrew Jaegle, Peter Wirnsberger, Daniel Hennes, and Irina Higgins. Which priors matter? benchmarking models for learning latent dynamics. 2021.

Nawaf Bou-Rabee and Houman Owhadi. Stochastic variational integrators. *IMA Journal of Numerical Analysis*, 29(2):421–443, 2009.

George EP Box. Science and statistics. *Journal of the American Statistical Association*, 71 (356):791–799, 1976.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Joaquin Quinonero Candela, Agathe Girard, Jan Larsen, and Carl Edward Rasmussen. Propagation of uncertainty in bayesian kernel models-application to multiple-step ahead forecasting. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 2, pages II–701. IEEE, 2003.

Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, July 1997. ISSN 1573-0565. DOI: 10.1023/A:1007379606734.

Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. Neural embeddings of graphs in hyperbolic space. *arXiv preprint arXiv:1705.10359*, 2017.

Kathleen Champion, Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, November 2019. ISSN 0027-8424, 1091-6490. DOI: 10.1073/pnas.1906995116.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *AAAI Conference on Artificial Intelligence*, 2018.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. *arXiv preprint arXiv:1909.13334*, 2019.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. DOI: 10.1145/1390156.1390177.

National Research Council, Physics Survey Overview Committee, et al. Physics in a new era: An overview. 2001.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, December 2020a.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020b.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning*, pages 1078–1086. PMLR, 2018.

Imre Csiszár. I-divergence geometry of probability distributions and minimization problems. *The annals of probability*, pages 146–158, 1975.

P. Cvitanović, R. Artuso, R. Mainieri, G. Tanner, and G. Vattay. *Chaos: Classical and Quantum*. Niels Bohr Inst., Copenhagen, 2016.

Zhenwen Dai, Mauricio A. Álvarez, and Neil D. Lawrence. Efficient modeling of latent information in supervised learning using Gaussian processes. In *Neural Information Processing Systems (NIPS)*. 2017.

Andreas Damianou and Neil D. Lawrence. Semi-described and semi-supervised learning with Gaussian processes. *Uncertainty in Artificial Intelligence (UAI)*, 2015.

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. In *Uncertainty in Artificial Intelligence*, 2018.

M. Deisenroth, D. Fox, and C. Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015. DOI: 10.1109/TPAMI.2013.218.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

Marc P. Deisenroth, Peter Englert, Jan Peters, and Dieter Fox. Multi-task policy search for robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

Marc Peter Deisenroth. *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing, 2010.

Shaan Desai and Stephen Roberts. Vign: Variational integrator graph networks. *arXiv preprint arXiv:2004.13688*, 2020.

M. D. Donsker and S. R. S. Varadhan. Asymptotic evaluation of certain markov process expectations for large time. IV. *Communications on Pure and Applied Mathematics*, 36(2): 183–212, 1983. ISSN 1097-0312. DOI: 10.1002/cpa.3160360204.

John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Finale Doshi-Velez and George Konidaris. Hidden parameter Markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

Weinan E. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, March 2017. ISSN 2194-671X. DOI: 10.1007/s40304-017-0103-z.

Sebastien Ehrhardt, Aron Monszpart, Niloy Mitra, and Andrea Vedaldi. Unsupervised intuitive physics from visual observations. In *Asian Conference on Computer Vision*, pages 700–716. Springer, 2018.

David Eriksson, Kun Dong, Eric Hans Lee, David Bindel, and Andrew Gordon Wilson. Scaling gaussian process regression with derivatives. *arXiv preprint arXiv:1810.12283*, 2018.

Andre Esteva, Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean, and Richard Socher. Deep learning-enabled medical computer vision. *npj Digital Medicine*, 4(1):1–9, January 2021. ISSN 2398-6352. DOI: 10.1038/s41746-020-00376-2.

Leonhard Euler. *Institutiones calculi integralis*. Acad. Imper. scientiarum, 1768.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning*, 2017.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980. ISSN 1432-0770. DOI: 10.1007/BF00344251.

Alexandre Galashov, Jonathan Schwarz, Hyunjik Kim, Marta Garnelo, David Saxton, Pushmeet Kohli, S. M. Ali Eslami, and Yee Whye Teh. Meta-learning surrogate models for sequential decision making. *arXiv:1903.11907*, 2019.

Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. *Advances in Neural Information Processing Systems*, 2018.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

Izrail Moiseevitch Gelfand, Richard A Silverman, et al. *Calculus of variations*. Courier Corporation, 2000.

Andrew Gelman and Yuling Yao. Holes in Bayesian Statistics. *Journal of Physics G: Nuclear and Particle Physics*, 48(1):014002, January 2021. ISSN 0954-3899, 1361-6471. DOI: 10.1088/1361-6471/abc3a5.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, May 2015. ISSN 1476-4687. DOI: 10.1038/nature14541.

Ryan Giordano, Tamara Broderick, and Michael I Jordan. Covariances, Robustness, and Variational Bayes. page 49, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical Bayes. In *International Conference on Learning Representations (ICLR)*, 2018.

Samuel J Greydanus, Misko Dzumba, and Jason Yosinski. Hamiltonian neural networks. 2019.

Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv:1806.04640*, 2018a.

Anvita Gupta, Alex T Müller, Berend JH Huisman, Jens A Fuchs, Petra Schneider, and Gisbert Schneider. Generative recurrent networks for de novo drug design. *Molecular informatics*, 37(1-2):1700111, 2018b.

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

William Rowan Hamilton. On a general method in dynamics, by which the study of the motions of all free systems of attracting or repelling points is reduced to the search and differenciation of one central relation, or characteristic function. In *Abstracts of the Papers Printed in the Philosophical Transactions of the Royal Society of London*, number 3, pages 275–276. The Royal Society London, 1837.

Aaron Havens and Girish Chowdhary. Forced variational integrator networks for prediction and control of mechanical systems. In *Learning for Dynamics and Control*, pages 1142–1153. PMLR, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.

James Hensman, Alexander Matthews, and Zoubin Ghahramani. Scalable variational gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR, 2015.

Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

Andreas Hochlehnert, Alexander Terenin, Steindór Sæmundsson, and Marc Deisenroth. Learning contact dynamics using physically structured neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 2152–2160. PMLR, 2021.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, November 1997. ISSN 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

Matthew D Hoffman and Matthew J Johnson. ELBO surgery: Yet another way to carve up the variational evidence lower bound. page 4, 2018.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

Darryl D Holm, Tanya Schmah, and Cristina Stoica. *Geometric mechanics and symmetry: from finite to infinite dimensions*, volume 12. Oxford University Press, 2009.

Antti Honkela and Harri Valpola. On-line variational bayesian learning. In *4th International Symposium on Independent Component Analysis and Blind Signal Separation*, pages 803–808, 2003.

Eyke Hullermeier, Thomas Fober, and Marco Mernberger. *Inductive Bias*. 2013.

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

Aleksei Grigorevich Ivakhnenko and Valentin Grigorevich Lapa. Cybernetic predicting devices. Technical report, PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGINEERING, 1966.

Tommi S Jaakkola and Michael I Jordan. Variational probabilistic inference and the qmr-dt network. *Journal of artificial intelligence research*, 10:291–322, 1999.

Allan Jabri, Kyle Hsu, Abhishek Gupta, Ben Eysenbach, Sergey Levine, and Chelsea Finn. Unsupervised curricula for visual meta-reinforcement learning. In *Advances in Neural Information Processing Systems*, 2019.

Johan Ludwig William Valdemar Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta mathematica*, 30(1):175–193, 1906.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the International Conference on Machine Learning*, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the International Conference on Machine Learning*, 2014.

Douglas S. Jones. *Elementary Information Theory*. Clarendon Press, 1979.

Jean Kaddour, Steindor Saemundsson, and Marc Deisenroth (he/him). Probabilistic active meta-learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20813–20822. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/ef0d17b3bdb4ee2aa741ba28c7255c53-Paper.pdf.

Sanket Kamthe and Marc Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. In *International conference on artificial intelligence and statistics*, pages 1701–1710. PMLR, 2018a.

Sanket Kamthe and Marc P. Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018b.

Taylor Killian, Samuel Daulton, George Konidaris, and Finale Doshi-Velez. Robust and efficient transfer learning with hidden parameter Markov decision processes. In *Neural Information Processing Systems (NIPS)*, Long Beach, CA, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations*, 2014.

Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. Generalized Variational Inference: Three arguments for deriving new Posteriors. *arXiv:1904.02063 [cs, stat]*, December 2019.

Michael Kraus and Omar Maj. Variational integrators for nonvariational partial differential equations. *Physica D: Nonlinear Phenomena*, 310:37–71, 2015.

Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate Uncertainties for Deep Learning Using Calibrated Regression. *arXiv:1807.00263 [cs, stat]*, June 2018.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Frantzeska Lavda, Magda Gregorová, and Alexandros Kalousis. Data-Dependent Conditional Priors for Unsupervised Learning of Multimodal Data. *Entropy*, 22(8):888, August 2020. ISSN 1099-4300. DOI: 10.3390/e22080888.

Neil D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Neural Information Processing Systems (NIPS)*. 2004.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object Recognition with Gradient-Based Learning. In *Shape, Contour and Grouping in Computer Vision*, page 319, Berlin, Heidelberg, January 1999. Springer-Verlag. ISBN 978-3-540-66722-3.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3(22):500, 2018.

Jeongseok Lee, C Karen Liu, Frank C Park, and Siddhartha S Srinivasa. A linear-time variational integrator for multibody systems. *Algorithmic Foundations of Robotics XII. Springer*, pages 352–367, 2020.

Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Global formulations of lagrangian and hamiltonian mechanics on two-spheres. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 6010–6015. IEEE, 2015.

Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: A survey of trends and technologies. *Artificial Intelligence Review*, 44(1):117–130, 2015. ISSN 0269-2821. DOI: 10.1007/s10462-013-9406-y.

Adrian Lew, Jerrold E Marsden, Michael Ortiz, and Matthew West. AN OVERVIEW OF VARIATIONAL INTEGRATORS. *Finite Element Methods*, page 18, 2003.

Henry W. Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, September 2017. ISSN 0022-4715, 1572-9613. DOI: 10.1007/s10955-017-1836-5.

Beibei Liu, Gemma Mason, Julian Hodgson, Yiying Tong, and Mathieu Desbrun. Model-reduced variational fluid simulation. *ACM Transactions on Graphics (TOG)*, 34(6):1–12, 2015.

Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11):4405–4423, 2020.

Xuhan Liu, Adriaan P IJzerman, and Gerard JP van Westen. Computational approaches for de novo drug design: Past, present, and future. In *Artificial Neural Networks*, pages 139–165. Springer, 2021.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pages 4114–4124. PMLR, 2019.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs Created Equal? A Large-Scale Study. *arXiv:1711.10337 [cs, stat]*, October 2018.

Michael Lutter, Christian Ritter, and Jan Peters. Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning. *arXiv:1907.04490 [cs, eess, stat]*, July 2019.

David JC MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 354(1):73–80, 1995.

Daniel J. Mankowitz, Gabriel Dulac-Arnold, and Todd Hester. Challenges of real-world reinforcement learning. In *ICML Workshop on Real-Life Reinforcement Learning*, 2019.

J. E. Marsden and M. West. Discrete mechanics and variational integrators. *Acta Numerica*, 10:357–514, May 2001. ISSN 0962-4929, 1474-0508. DOI: 10.1017/S096249290100006X.

Marios Mattheakis, Pavlos Protopapas, David Sondak, Marco Di Giovanni, and Efthimios Kaxiras. Physical symmetries embedded in neural networks. *arXiv preprint arXiv:1904.08991*, 2019.

Alexander G de G Matthews, James Hensman, Richard Turner, and Zoubin Ghahramani. On sparse variational methods and the kullback-leibler divergence between stochastic processes. In *Artificial Intelligence and Statistics*, pages 231–239. PMLR, 2016.

Alexander Graeme de Garis Matthews. *Scalable Gaussian process inference using variational methods*. PhD thesis, University of Cambridge, 2017.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, 2020.

Robert A Meyers. *Mathematics of complexity and dynamical systems*. Springer Science & Business Media, 2011.

M.L. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1972.

Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2021. URL `probml.ai`.

Devang K. Naik and R. Mammone. Meta-neural networks that learn by learning. *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, 1992. DOI: 10.1109/IJCNN.1992.287172.

Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *ICML*, page 8, 2010.

Radford M Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*, 2017.

E. Noether. Invariante variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1918:235–257, 1918. URL `http://eudml.org/doc/59024`.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Manfred Opper and David Saad. *Advanced mean field methods: Theory and practice*. MIT press, 2001.

Misha Padidar, Xinran Zhu, Leo Huang, Jacob Gardner, and David Bindel. Scaling gaussian processes with derivative information using variational inference. *Advances in Neural Information Processing Systems*, 34, 2021.

Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010. ISSN 1558-2191. DOI: 10.1109/TKDE.2009.191.

Piyush Pandita, Jesper Kristensen, and Liping Wang. Towards scalable gaussian process modeling. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 59193, page V02BT03A032. American Society of Mechanical Engineers, 2019.

Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep RL: A short survey. *arXiv:2003.04664*, 2020.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Alan Edelman. Universal Differential Equations for Scientific Machine Learning. page 45, 2020.

Nasim Rahaman, Anirudh Goyal, Muhammad Waleed Gondal, Manuel Wuthrich, Stefan Bauer, Yash Sharma, Yoshua Bengio, and Bernhard Schölkopf. S2RMs: Spatially Structured Recurrent Modules. *arXiv:2007.06533 [cs, stat]*, July 2020.

Tom Rainforth, Adam R Kosiorek, Tuan Anh Le, Chris J Maddison, Maximilian Igl, Frank Wood, and Yee Whye Teh. Tighter Variational Bounds are Not Necessarily Better. page 9.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019. ISSN 0021-9991. DOI: 10.1016/j.jcp.2018.10.045.

Bharath Ramsundar, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande. Massively multitask networks for drug discovery. *arXiv:1502.02072*, 2015.

Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

Clarence W Rowley and Jerrold E Marsden. Variational integrators for degenerate lagrangians, with application to point vortices. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 2, pages 1521–1527. IEEE, 2002.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. *arXiv:1706.05098 [cs, stat]*, June 2017.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. ISSN 1476-4687. DOI: 10.1038/323533a0.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *arXiv:1804.04272*, 2018.

Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. Meta Reinforcement Learning with Latent Variable Gaussian Processes. *arXiv:1803.07551 [cs, stat]*, July 2018.

Steindor Saemundsson, Alexander Terenin, Katja Hofmann, and Marc Peter Deisenroth. Variational Integrator Networks for Physically Structured Embeddings. *arXiv:1910.09349 [cs, stat]*, March 2020.

Hugh Salimbeni. Deep Gaussian Processes: Advances in Models and Inference. July 2019. DOI: 10.25560/81669.

Masa-Aki Sato. Online model selection based on the variational bayes. *Neural computation*, 13(7):1649–1681, 2001.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, January 2009. ISSN 1941-0093. DOI: 10.1109/TNN.2008.2005605.

Jürgen Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook. *Diploma thesis, Institut für Informatik, Technische Universität München*, 1987.

Jürgen Schmidhuber. Learning factorial codes by predictability minimization. *Neural computation*, 4(6):863–879, 1992.

Jurgen Schmidhuber. Deep Learning in Neural Networks: An Overview. page 88, 2014.

Jürgen Schmidhuber. Metalearning Machines Learn to Learn (1987-), 2020.

Urgen Schmidhuber. On Learning How to Learn Learning Strategies Technical Report Fki-198-94 (revised). /paper/On-Learning-How-to-Learn-Learning-Strategies-Report-Schmidhuber/81dcd9f8b2e553dfe6754ddca09a2f09af26fb27, 1995.

Bernhard Scholkopf. Causality for machine learning. *arXiv preprint arXiv:1911.10500*, 2019.

Burr Settles. Active learning literature survey. 2009.

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257, 2006.

Hamdy S Soliman and Mohammed Omari. A neural networks approach to image data compression. *Applied Soft Computing*, 6(3):258–271, 2006.

Arno Solin, Manon Kok, Niklas Wahlstrom, Thomas B Schon, and Simo Sarkka. Modeling and Interpolation of the Ambient Magnetic Field by Gaussian Processes. page 17, 2018.

Ari Stern, Yiying Tong, Mathieu Desbrun, and Jerrold E Marsden. Variational integrators for maxwell's equations with sources. *arXiv preprint arXiv:0803.2070*, 2008.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

Boxin Tang. Orthogonal array-based latin hypercubes. *Journal of the American Statistical Association*, 88(424):1392–1397, 1993.

Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.

Michalis Titsias and Neil D. Lawrence. Bayesian Gaussian process latent variable model. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. *arXiv preprint arXiv:1909.13789*, 2019.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

C. Wang and R. M. Neal. Gaussian Process Regression with Heteroscedastic or Non-Gaussian Residuals. *ArXiv e-prints*, December 2012.

Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, May 2016. ISSN 2196-1115. DOI: 10.1186/s40537-016-0043-6.

Matthew West. *Variational Integrators*. PhD thesis, California Institute of Technology, 2004.

Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating Physics-Based Modeling with Machine Learning: A Survey. *arXiv:2003.04919 [physics, stat]*, July 2020.

David Wolpert and William Macready. No Free Lunch Theorems for Search. March 1996.

Yuling Yao, Aki Vehtari, Daniel Simpson, and Andrew Gelman. Yes, but Did It Work?: Evaluating Variational Inference. *arXiv:1802.02538 [stat]*, July 2018.

Cheng Zhang, Judith Butepage, Hedvig Kjellstrom, and Stephan Mandt. Advances in Variational Inference. *arXiv:1711.05597 [cs, stat]*, October 2018.

Shandian Zhe, Wei Xing, and Robert M Kirby. Scalable high-order gaussian process regression. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2611–2620. PMLR, 2019.

Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, and Yanpeng Li. Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4. IEEE, 2015.

Yaofeng Desmond Zhong and Naomi Leonard. Unsupervised learning of lagrangian dynamics from images for prediction and control. *Advances in Neural Information Processing Systems*, 33, 2020.

A

Representing Tasks as Conserved Quantities

A.1 Multi-Task and Transfer Learning, Meta-Learning in Model-Based RL

Model and Baselines

The core model being studied in the experiments is the ML-GP from eq. (2.47),

$$p_\theta(\boldsymbol{y}, f(\tilde{\boldsymbol{x}}, \boldsymbol{z}), \boldsymbol{z}) = p_\theta(\boldsymbol{y}|f(\tilde{\boldsymbol{x}}, \boldsymbol{z}))p_\theta(f(\tilde{\boldsymbol{x}}, \boldsymbol{z}))p(\boldsymbol{z}) \tag{A.1}$$

where $\tilde{\boldsymbol{x}}$ denotes the state-control pair from eq. (2.2). The corresponding variational approximation is given in eq. (2.51),

$$q_\phi(f(\tilde{\boldsymbol{x}}, \boldsymbol{z}), \boldsymbol{z}) = q_\phi(f(\tilde{\boldsymbol{x}}, \boldsymbol{z}))q_\phi(\boldsymbol{z}). \tag{A.2}$$

The baseline models are the standard GP model from eq. (2.7),

$$p_\theta(\boldsymbol{y}, f(\tilde{\boldsymbol{x}})) = p_\theta(\boldsymbol{y}|f(\tilde{\boldsymbol{x}}))p_\theta(f(\tilde{\boldsymbol{x}})), \tag{A.3}$$

and the SVGP model from definition 9, which approximates the posterior of the standard GP model in eq. (A.3) with a variational family $q_\phi(f(\tilde{\boldsymbol{x}}))$. All models use the EQ kernel from eq. (2.8). For the baselines:

$$\Sigma_\theta(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}') = \sigma_k^2 \exp\left(-\frac{1}{2}(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}')^\mathrm{T}\mathbf{L}^{-1}(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}')\right). \tag{A.4}$$

For the ML-GP:

$$\Sigma_\theta([\tilde{\boldsymbol{x}}, \boldsymbol{z}], [\tilde{\boldsymbol{x}}', \boldsymbol{z}']) = \sigma_k^2 \exp\left(-\frac{1}{2}([\tilde{\boldsymbol{x}}, \boldsymbol{z}] - [\tilde{\boldsymbol{x}}', \boldsymbol{z}'])^\mathrm{T}\mathbf{L}^{-1}([\tilde{\boldsymbol{x}}, \boldsymbol{z}] - [\tilde{\boldsymbol{x}}', \boldsymbol{z}'])\right) \tag{A.5}$$

$$= \sigma_k^2 \exp\left(-\frac{1}{2}(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}')^\mathrm{T}\mathbf{L}_{\tilde{\boldsymbol{x}}}^{-1}(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}')\right) \exp\left(-\frac{1}{2}(\boldsymbol{z} - \boldsymbol{z}')^\mathrm{T}\mathbf{L}_{\boldsymbol{z}}^{-1}(\boldsymbol{z} - \boldsymbol{z}')\right), \tag{A.6}$$

where $\mathbf{L}_{\tilde{x}}, \mathbf{L}_z$ denote the characteristic lengthscales corresponding the state-control dimensions and latent task dimensions, respectively. The outputs are assumed to be the one step difference model from eq. (2.9),

$$\boldsymbol{y}_{[p,t]} = (\boldsymbol{x}_{[p,t+1]} - \boldsymbol{x}_{[p,t]}) + \boldsymbol{\epsilon} = \boldsymbol{f}_{[p,t]} + \boldsymbol{\epsilon}, \tag{A.7}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{1}_D)$ is independent Gaussian noise.

In order to approximate the expected long term cost for MPC given in eq. (2.30), using the ML-GP model, the integral over the task variable needs to be additionally considered. Since it is required that the input distribution is Gaussian in eq. (2.28), we assume that the distribution over the concatenated state $[\tilde{\boldsymbol{x}}, \boldsymbol{z}]$ is given by

$$q(\tilde{\boldsymbol{x}}_t, \boldsymbol{z} | \boldsymbol{c}_t, \theta, \phi) = p_\theta(\boldsymbol{x}_t | \boldsymbol{c}_t) q_\phi(\boldsymbol{z}) = \mathcal{N}([\boldsymbol{\mu}_t^{\tilde{x}}, \boldsymbol{\mu}_t^z], \texttt{blockdiag}(\boldsymbol{\Sigma}_t^{\tilde{x}}, \boldsymbol{\Sigma}_t^z)), \tag{A.8}$$

where $\boldsymbol{\mu}_t^{\tilde{x}}, \boldsymbol{\Sigma}_t^{\tilde{x}}$ are obtained according to eq. (2.29) and $\boldsymbol{\mu}_t^z, \boldsymbol{\Sigma}_t^z$ are given by the variational parameters over the latent task variables ϕ. The rest of the steps follow the SVGP results from section 2.1.5.

Learning and Inference

To train the model parameters θ and variational parameters ϕ we jointly optimise them with respect to the ELBO. For the ML-GP this is given in eq. (2.54), for the SVGP this is given in eq. (2.21). The standard GP has no variational parameters but the model parameters are optimised with respect to the log-marginal likelihood in eq. (1.53).

The Adam algorithm [Kingma and Ba, 2017] is used to update the parameters, with default hyperparameters: $\alpha = 1 \times 10^{-2}, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$, and a single sample of the latent task variable $\boldsymbol{z}_p$ is drawn at each iteration to approximate expectation of the log likelihood in eq. (2.54). In particular, stochastic variational inference with minibatches is used to estimate the gradient [Hensman et al., 2013, Jimenez Rezende et al., 2014, Kingma and Welling, 2014]. The size of the minibatches was chosen as 5 episodes, each containing 30 steps for a total size of 150. The number of inducing points used for the ML-GP and SVGP models in section 2.3.2 was 180. Learning is continued for 10000 epochs of the data or until the ELBO stops increasing.

We use L-BFGS-B to minimise the cost in the MPC algorithm, as implemented in the SciPy package [Virtanen et al., 2020].

Systems

In sections 2.3.1–2.3.2, two systems of the form (2.1) are studied in the experiments. The systems are simulated using numerical methods implemented in the physics simulator DART [Lee et al., 2018].

- **Cart-pole swing-up:** The cart-pole system consists of a cart that moves horizontally on a track with a freely swinging pendulum attached to it. The state of this nonlinear system is the position x and velocity $\dot{x}$ of the cart, and the angle ω and angular velocity $\dot{\omega}$ of the pendulum, that is,

  $$\boldsymbol{x} = [x, \omega, \dot{x}, \dot{\omega}].$$

  The control signals act as a horizontal force on the cart limited to the range

  $$c \in [-15, 15]\,\mathrm{N}, \quad \boldsymbol{c} = [0, 0, c, 0].$$

  The mean of the initial state distribution is the state where the pendulum is hanging downward $\omega = 0$. The control task is to learn to swing up and balance the pendulum in the inverted position in the middle of the track.
- **Double-pendulum swing-up:** The double-pendulum system is a two-link robotic arm with two motors, one in the shoulder and one in the elbow. The state of the system comprises the angles ω^1, ω^2, and angular velocities $\dot{\omega}^1, \dot{\omega}^2$ of the inner and outer pendulums,

  $$\boldsymbol{x} = [\omega^1, \omega^2, \dot{\omega}^1, \dot{\omega}^2].$$

  The control signals are the torques

  $$c^{1,2} \in [-4, 4]\,\mathrm{Nm}, \quad \boldsymbol{c} = [0, 0, c^1, c^2].$$

  applied to the two motors. The mean of the initial state distribution is the position where both pendulums are hanging downward. The RL task is to find a strategy that swings the double pendulum up and balances it in the inverted position.

Meta-Learning Algorithms

Algorithm 3 gives a top-level view of the two main steps, multi-task learning on the training tasks in line 2 and transfer learning to test tasks in line 4. They key variables are the model parameters θ, variational parameters ϕ, the policy parameters π and the tasks (systems) $\mathcal{S}$. The first two (θ, ϕ) are defined by the latent variable model from eq. (2.32) and variational posterior from eq. (2.38). The specific model used in the experiments

is the ML-GP from eq. (2.47) with variational posterior given in eq. (2.50). The policy parameters π represent control parameters in MPC and are detailed after a description of the systems.

Algorithm 4 outlines the multi-task learning function from Algorithm 3. It loops over the training tasks in S and tries to solve them with model predictive control algorithm 1. Each attempt produces observations from the systems which are added to the full dataset. The full dataset is then used to update the model parameters θ and variational parameters ϕ. The transfer learning procedure in the experiments is built on eq. (2.44), accounting for the model-based RL setting. Specifically, since each step of the MPC algorithm produces an observation, the algorithm can do online inference of the latent task variable after each step. The function is outlined in Algorithm 5 for one system (for ease of notation).

Require: p_θ, q_ϕ ▷ Section 2.2

Require: $\mathcal{S}, \Pi$ ▷ This section

1: Initialise θ, ϕ, π

2: $\widehat{\theta}, \widehat{\phi}, \widehat{\pi}, \mathbf{D} = \textsc{MultiTaskLearning}(p_\theta, q_\phi, \pi, S)$

3: Initialise $\phi_\star, \pi_\star$

4: $\widehat{\phi}_\star, \widehat{\pi}_\star, \mathbf{D}_\star = \textsc{TransferLearning}(p_\theta, q_\phi, \pi_\star, S_\star)$

Algorithm 3: A birds eye view of a meta-learning algorithm using invariant task representations. It starts with a model p_θ and a variational distribution q_ϕ from section 2.2. $\mathcal{S}$ contains training and test systems $S, S_\star$ which are subjected to controls according to policies in Π, producing observations that are collected in $\mathbf{D}$. These observations are used to learn new parameters, denoted by the hat symbol $\widehat{\cdot}$. The subscript $\star$ denotes parameters and data corresponding to test systems. These are systems that are used in the transfer learning function [line 4].

1: **function** $\textsc{MultiTaskLearning}(p_\theta, q_\phi, \pi, S, \mathbf{D}{=}\varnothing)$

2: **while** tasks are not solved **do**

3: $\mathbf{D}_i, \pi = \text{MPC}(p_\theta, q_\phi, S, \pi)$ ▷ Algorithm 1

4: $\mathbf{D} = \mathbf{D} \cup \mathbf{D}_i$

5: $\theta, \phi = \text{SVI}(\theta, \phi, \mathbf{D})$ ▷ Learn θ, ϕ using ELBO from eq. (2.41).

6: **end while**

7: **return** $\theta, \phi, \pi, \mathbf{D}$

8: **end function**

Algorithm 4: Multi-task learning with model predictive control (MPC). The function loops over the tasks in S, applying control signals based on MPC and producing observations $\mathbf{D}_i$ [line 3]. In each iteration, the observations are added to the full dataset which is used to update the model and variational parameters θ, ϕ [line 5] using SVI from eq. (1.81).

1: **function** TRANSFERLEARNING(p_θ, q_ϕ, π, S)
2: $\mathbf{D} = \varnothing$
3: **for** $t = 1, \ldots, T-1$ **do**
4: $\mathbf{D}_{t+1}, \pi = \text{MPC}(p_\theta, q_\phi, \pi, S; T{=}1, \boldsymbol{x}_t, \ldots)$ ▷ MPC for one-step, alg. 1.
5: $\mathbf{D} = \mathbf{D} \cup \mathbf{D}_{t+1}$
6: $\phi = \text{SVI}(\phi, \mathbf{D})$ ▷ Learn ϕ using ELBO from eq. (2.41).
7: **end for**
8: **return** $\phi, \pi, \mathbf{D}$
9: **end function**

Algorithm 5: Transfer learning with MPC. After each step of MPC [line 4], the new observation is added to the dataset and the variational parameters for the ongoing task is updated using SVI [line 6]. Since the model parameters are global across all tasks, these are not updated. The algorithm can be thought of as doing online inference of the task variables.

A.2 Active Task Learning

Model and Baselines

The core model being studied in the experiments is the ML-GP with additional descriptors given in eq. (2.60),

$$p_\theta(\boldsymbol{\tau}, \boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{z}) = p_\theta(\boldsymbol{\tau}|\boldsymbol{z}) p_\theta(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{z}), \boldsymbol{z}), \tag{A.9}$$

using either a GP observation model for the task descriptors, as given in eq. (2.61),

$$p_\theta(\boldsymbol{\tau}, \boldsymbol{z}) = p_\theta(\boldsymbol{\tau}|g(\boldsymbol{z})) p_\theta(g(\boldsymbol{z})), \tag{A.10}$$

or a neural network observations model as given in eq. (2.62),

$$p_\theta(\boldsymbol{\tau}|\boldsymbol{z}) = p_\theta(\boldsymbol{\tau}|g(\boldsymbol{z})). \tag{A.11}$$

The variational posterior is for the task variables is given by eq. (2.63)

$$q_\phi^{z|\tau} = q(\boldsymbol{z}|h_\phi(\boldsymbol{\tau})) \approx p_\theta(\boldsymbol{z}|\boldsymbol{\tau}, \mathbf{Y}), \tag{A.12}$$

when an inference network is used. Otherwise it is given by a Gaussian q_ϕ^z whose parameters are optimised directly. Both the ML-GP and the GP for the task descriptors in eq. (A.10) use the EQ kernel from eq. (1.40), and repeated in eq. (A.4) and eq. (A.5). The neural network observation model in eq. (A.11) is given by a fully-connected NN (FCNN) eq. (1.30) with two hidden layers consisting of 200 units each and ReLU activations. The inference network in eq. (A.12) mirrors the observation network, i.e. a FCNN with two hidden layers consisting of 200 units each and ReLU activations.

The ML-GP model in eq. (A.9), using the PAML algorithm algorithm 2 for task selection, is compared to uniform sampling (UNI), commonly used in meta-learning settings

[Finn et al., 2017, Galashov et al., 2019] and equivalent to domain randomization [Tobin et al., 2017]; Latin hypercube sampling (LHS) [Tang, 1993]; and an oracle baseline that is trained on the test tasks. The oracle represents an upper bound on the predictive performance. The first two baselines use the ML-GP model from eq. (2.47)—i.e. without task descriptor observations in the model—but select tasks in the true task configuration space. Conversely, PAML ranks tasks in latent space and proposes the next task using the observation model over task descriptors.

Learning and Inference

To train the model parameters θ and variational parameters ϕ we jointly optimise them with respect to the ELBO. For the baseline ML-GP this is given in eq. (2.54). For the ML-GP with task descriptors this is given in eq. (2.67).

The Adam algorithm [Kingma and Ba, 2017] is used to update the parameters, with an initial learning rate of 10^{-2}, and a single sample of the latent task variable z_p is drawn at each iteration to approximate expectation of the log likelihood in the ELBO objectives. In particular, stochastic variational inference with minibatches is used to estimate the gradient [Hensman et al., 2013, Jimenez Rezende et al., 2014, Kingma and Welling, 2014]. The size of the minibatches was chosen as 1000 and the number of inducing points for all models was 300. Learning was continued for 5000 steps at each iteration or until the ELBO stopped increasing.

Systems

In section 2.3.3, three robotic systems of the form (2.1) were simulated in the experiments. The systems are simulated using numerical methods implemented in the Mujoco physics simulator [Todorov et al., 2012].

- **Cart-pole:** The cart-pole system consists of a cart that moves horizontally on a track with a freely swinging pendulum attached to it. The state of this nonlinear system comprises the position and velocity of the cart as well as the angle and angular velocity of the pendulum. The control signals $c \in [-25, 25]\,\mathrm{N}$ act as a horizontal force on the cart.
- **Pendubot:** The pendubot system is an underactuated two-link robotic arm. The inner link exerts a torque $c \in [-10, 10]\,\mathrm{Nm}$, but the outer joint cannot. The uncontrolled system is chaotic, so that modelling the dynamics is challenging. The system has four

continuous state variables that consist of two joint angles and their corresponding joint velocities.

- **Cart-double-pole:** The cart-double-pole consists of a cart running on a horizontal track with a freely swinging double-pendulum attached to it. As in the cart-pole system, a horizontal force $c \in [-25, 25]\,\mathrm{N}$ can be applied to the cart. The state of the system is the position and velocity of the cart as well as the angles and angular velocities of both attached pendulums.

B

Variational Integrator Networks

B.1 Noisy Observations

The pendulum and mass-spring systems from eq. (3.62) and eq. (3.64) respectively, given by the Hamiltonians

$$H_{\mathrm{p}}(q,p) = T_{\mathrm{p}}(p) + U_{\mathrm{p}}(q) = \frac{1}{2mr^2}p^2 + mg(\cos q - 1), \quad q \in (-\pi, \pi), \tag{B.1}$$

$$H_{\mathrm{ms}}(q,p) = T_{\mathrm{ms}}(p) + U_{\mathrm{ms}}(q) = \frac{1}{2m}p^2 + \frac{1}{2}kq^2, \quad q \in \mathbb{R}, \tag{B.2}$$

were simulated using a 5th order Runge-Kutta method (RK45) [Dormand and Prince, 1980], implemented in the SciPy package [Virtanen et al., 2020], to generate trajectories of fixed length T,

$$\mathbf{X} = (\boldsymbol{x}_t), \quad t = 1, \ldots, T, \tag{B.3}$$

where $\boldsymbol{x}_t = (q_t, p_t)$ and the error tolerance is set at 10^{-9}. The initial states were sampled uniformly at random, bounded from above and below by energy to avoid pathological behaviour. For the pendulum systems, the mass m was fixed at 1kg, the length r at 1m and the gravitational acceleration at $9.81\mathrm{m/s}^2$. The energy of the initial state was bounded such that the pendulum did not swing over the top. To do so, we restrict the initial angle to

$$q_0 \in \big[-\frac{7}{8}\pi, \frac{7}{8}\pi\big] = [-q_{\mathrm{limit}}, q_{\mathrm{limit}}], \tag{B.4}$$

which has energy (assuming no momentum $p_0 = 0$)

$$U_{\mathrm{p}}(-q_{\mathrm{limit}}) = U_{\mathrm{p}}(q_{\mathrm{limit}}) = g(\cos q_{\mathrm{limit}} - 1) \approx -18.9\mathrm{J}. \tag{B.5}$$

To sample initial states, we sample the initial angle uniformly from eq. (B.4) and sample the initial momentum uniformly with rejection such that the total energy does not go

under -18.9J. Since the total energy is conserved, this ensures that the pendulum does not start with enough initial momentum to swing over the top.

For the mass-spring system in eq. (B.2), the mass was fixed at 1kg and the spring constant at 0.75Nm. We sample the initial state uniformly from

$$q_0 \sim [-2, 2]\text{m}, \quad p_0 \sim [-1, 1]\text{m/s}. \tag{B.6}$$

B.2 Pixel Observations

Model and Baselines

The core models studied in the experiments are given by eq. (3.54)

$$p_\theta(\mathbf{Y}, \boldsymbol{h}_0) = p(\boldsymbol{h}_0) \prod_{k=1}^{K} p_\theta(\boldsymbol{y}_k | f(\boldsymbol{h}_k)), \tag{B.7}$$

where f is an observation function that relates the state $\boldsymbol{h}_k$ to the observation $\boldsymbol{y}_k$, and $p(\boldsymbol{h}_0)$ is the isotropic Gaussian given in eq. (3.55). The dynamic states $\boldsymbol{h}_k$ are given by a repeated composition of a discrete flow map

$$\boldsymbol{h}_k = F_\theta^k(\boldsymbol{h}_0, \eta) = (\underbrace{F_\theta \circ \ldots \circ F_\theta}_{k \text{ times}})(\boldsymbol{h}_0, \eta), \quad k \geqslant 1, \tag{B.8}$$

where F_θ^k is given by one of VIN-SV eq. (3.36), VIN-VV eq. (3.39), VIN-$SO(2)$ eq. (3.44) or the recurrent ResNet from eq. (3.13). All models are trained using the ELBO from eq. (3.60).

The Bernoulli observation model is given by eq. (3.57),

$$p_\theta(\boldsymbol{y}_k | f(\boldsymbol{q}_k)) = \mathcal{B}(\boldsymbol{y}_k | f_\theta(\boldsymbol{q}_k)), \tag{B.9}$$

and only the position coordinate is used as input. The variational posterior over the initial state is given by eq. (3.58),

$$p_\theta(\boldsymbol{h}_0 | \mathbf{Y}) \approx q(\boldsymbol{h}_0 | h_\phi(\mathbf{Y})) = \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)). \tag{B.10}$$

The individual networks in the VIN and ResNet VAE models have identical architectures, with the exception of the input to the observation function f_θ which maps the latent representations to images. The VIN models use only the position coordinate q as input, as described in eq. (3.57), whereas the ResNet uses both dimensions of the latent space. Figure B.1 illustrates the full model.

‣ The observation function f_θ consists of a FCNN, eq. (1.30) with two hidden layers,

consisting of 1000 hidden units each, and ReLU activations eq. (1.31). The output layer is a linear layer with no activation function.

‣ The first part of the inference network h^1_ϕ, from eq. (B.10), consists of a FCNN with two hidden layers, with 1000 hidden units each and ReLU activations, followed by a linear layer with 4 hidden units (twice the dimension of the latent space) and no activation function. Following the FCNN, the output (denoted by z_k in fig. B.1) is processed (in reverse) by a single layer recurrent long short-term memory NN [Hochreiter and Schmidhuber, 1997] (LSTM) with 50 hidden units and ReLU activations, denoted by h^2_ϕ. The LSTM is included to enable the model to infer velocity/momentum information about the initial condition from sequences.[1]

[1] Alternatively, one could stack observations to avoid using an LSTM.

‣ The base neural network for the dynamics is a FCNN with a single hidden layer with 1000 hidden units and softplus activations.

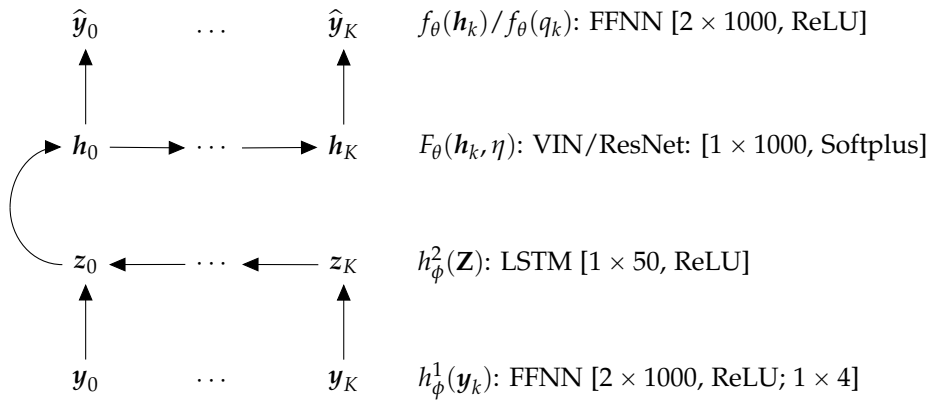


Figure B.1: Model architecture for pixel experiments. The images $\boldsymbol{y}_k$ are processed by a FFNN to give a sequence z_k. This sequence is given in reverse to a LSTM, producing the distribution $q_\phi(\boldsymbol{h}_0)$. The sequence $\boldsymbol{h}_k$ is generated by a ResNet or VIN, which is processed by a FFNN to generate reconstructions $\widehat{\boldsymbol{y}}_k$.

Additional baselines in section 3.3.3 are the standard variational auto-encoder (VAE) as described in definition 8, and a VAE constrained to the $SO(2)$ manifold (VAE-$SO(2)$). These are used only for qualitative evaluation of the latent embeddings since they do not model the dynamics. The $SO(2)$ constraint for the VAE-$SO(2)$ and the VIN-$SO(2)$ from eq. (3.44) is achieved by a mapping

$$f_{SO(2)}(x) = \text{Concat}([\sin x, \cos x]), \tag{B.11}$$

which is used as input to the observation function f_θ.

Generating Pixel Observations

To generate greyscale pixel images of the pendulum and the mass-spring systems, trajectories were first generated as described in appendix B.1. The position states were then turned into images using the Python plotting package Matplotlib [Hunter, 2007]. For the pendulum, a line from the origin $(0, 0)$ and up to the (x, y) coordinates specified by the angle q[2] was plotted on a $28 \times 28 \times 3$ RGB pixel canvas. For the mass-spring system, the mass was plotted as a single point,with position given by the x coordinate directly from the system, on a $28 \times 28 \times 3$ RGB pixel canvas. The canvas was then converted into a binary $28 \times 28 \times 1$ array by taking the R channel and setting any pixel with R-value over 0 to 1; and conversely the other pixels to 0.

[2] $x = r \sin q,$
$y = -rcosq.$