Imperial College London

MSc Individual Project

Imperial College London

Department of Computing

PATA: Probabilistic Active Task Acquisition

Supervisor:
Dr. Marc Peter Deisenroth

Author:
Jean Kaddour

Second Marker:
Dr. Stefan Leutenegger

Submitted in partial fulfillment of the requirements for the MSc degree in Advanced Computing of Imperial College London

September 2019

Acknowledgements

First of all, I am very grateful to Steindór Sæmundsson for his extensive guidance on this work, even after he left London and worked full-time. Steindór's first-principles-approach to solving problems and his eye for details were deeply inspiring to me. He was always ready to help, and he elevated my excitement for this project to a whole new level. Besides, it is always fun and fascinating to talk about life with him.

Secondly, I thank Dr. Marc Peter Deisenroth for his support as well as for making this interesting project for MAC students accessible on request. Marc's 493 course inspired me to look beyond deep learning in ML research and his hands-on approach in meetings makes working with him very pleasant.

Furthermore, I thank Dr. Stefan Leutennegger for his feedback on my background report and his "crazy uncle" ideas that motivated me to push the project's boundaries even further.

Additionally, I thank all the researchers that answered my questions on their papers through emails.

Moreover, I thank my family and my friends from home for their support as well as all the lovely people I met at Imperial for such an amazing year together.

Abstract

Humans are remarkably well equipped to learn new concepts from only a few examples and to adapt quickly to unforeseen situations by leveraging experience. Meta-learning, a recent research field in machine learning, aims to build systems with the same human-inspired ability: Discovering similarities between different tasks and leveraging this knowledge when solving new ones. However, current meta-learning algorithms lack a systematic approach to select informative tasks during training. They randomly sample training tasks from a given task distribution. In doing so, they ignore their experience with previously learned tasks when deciding which task to acquire next. This algorithmic design renders meta-learning to an inherently inefficient and data-intensive approach.
This work aims to complement meta-learning with a sample-efficient active task acquisition scheme. The three main contributions are: First, we propose PATA, a generic framework for the sequential acquisition of the most informative training tasks. Second, we aim to leverage the inferred task-specific properties of observed tasks when acquiring new ones. Concretely, we propose an exploration-by-optimisation mechanism that seeks to find the optimal latent region to exploit. Lastly, we provide seven different task acquisition tactics, broadly categorised into model-based and model-free strategies, based on ideas from active learning research and information theory. Here, we empirically demonstrate strong performances of all tactics across multiple reinforcement learning benchmark environments outperforming a random baseline.

Contents

List of Figures

List of Tables

List of Algorithms

Notation

Coloured boxes

A grey-coloured box like this will denote a corollary, definition, lemma, theorem or other general results.

A blue-coloured box like this will denote an example.

Mathematical notation

The mathematical notation used in this thesis is mostly adopted from (Deisenroth 2010 and Deisenroth, Faisal, and Ong 2020).

Symbol	Meaning
$a, b, c, \alpha, \beta, \gamma$	Scalars are lowercase
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	Vectors are bold lowercase
$\mathbf{A}, \mathbf{B}, \mathbf{C}$	Matrices are bold uppercase
$\mathbf{x}^\top, \mathbf{A}^\top$	Transpose of a vector or matrix
$\mathbf{A}^{-1}$	Invese of a matrix
$\mathbf{x}^\top \mathbf{y}$	Dot product of **x** and **y**
$\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$	Matrix of column vectors stacked horizontally
$\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$	Set of vectors (unordered)
$\mathbb{N}^+$	Integers
$\mathbb{R}$	Real numbers
$\mathbb{R}^n$	n-dimensional vector space of real numbers

Table of Abbreviations and Acronyms

Abbr.	Meaning
CS	Configuration space
e.g.	Exempli gratia (Latin: for example)
EMC	Expected model change
EMOC	Expected model output change
GMM	Gaussian mixture model
GSCS	Greedy sampling of configuration space
GSLS	Greedy sampling of latent space
i.e.	Id est (Latin: this means)
LC	Latent to configuration (space)
LLH	Log-likelihood
MB	Model-based
MF	Model-free
MI	Mutual information
ML	Meta-learning
MBRL	Model-based reinforcement learning
MRL	Meta reinforcement learning
PATA	Probabilistic Active Task Acquisition
RDN	Random
RL	Reinforcement learning
RMSE	Root mean squared error
SECS	Safe exploration of candidate space
VAR	Variance

Chapter 1

Introduction

Imagine you are attending a driving school to get your license. For practical sessions, the school provides you with minivans with manual transmission. In the beginning, this makes the first sessions hard for you. Parking the comparatively large vehicle feels tedious and shifting gears smoothly is not as easy as you imagined. At some point, you ask your teacher why you can't practise with a more convenient car. She tells you that after having learned how to handle these more difficult cars, you can quickly drive any car.

Figure 1.1: A generic minivan, resembling typical driving school cars. Royalty-free graphic taken from Shutterstock.com.

Now, imagine you were the driving school teacher, and you could choose among several different cars for your students' practical sessions. Assume you would let each student practise with each car. Your goal is to equip each student with universal driving skills so that a graduate can drive as many cars as possible in the future. Your decision of which set of vehicles to pick is constrained by a specific budget, for example, the number of vehicles you can park in the school's garage. One strategy could be to choose cars varying lengths, such as a compact sports car, a medium-length SUV, and finally a minitruck. Alternatively, you could select different vehicle heights, weights or motorisations. The number of different parameters that potentially change a car's driving dynamics is large and it is not always obvious which parameters make a difference for a student's learning progress. Presumably, the ideal choice takes into account a weighted mix of the above factors.

Intelligent systems have not reached the same level of human's learning efficiency and versatility yet. Comparatively, humans are efficient learners and adapt quickly to new tasks by building upon prior related experience they have learnt over time. In general, a *task* refers to a broad range of problems to be solved. *Learning* refers to attaining the ability to perform the task. For example, human motorists can learn to drive a new car (the task) they have never driven before in minutes. Within the machine learning community, end-to-end learning from scratch is highly popular, as it is an approach that does not require any previous knowledge to solve a task (C. B. Finn 2018). From the standpoint of how humans learn, that is like asking a human newborn to learn to win the *Monaco Grand Prix* [1]. To address such inherent learning inefficiencies, *meta-learning* algorithms have arisen and received much attention in the last few years.

[1]The Monaco Grand Prix is one of the most renowned automobile races in the world.

Meta-learning is an umbrella term for methods that aim to learn new tasks quickly by using knowledge from previously seen tasks (Sæmundsson, Hofmann, and Deisenroth 2018). One technique that falls into this category is multi-task learning (MTL), i.e., the idea of learning multiple tasks simultaneously and adapting to new tasks quickly by disentangling global and task-specific properties. Referring back to our above example, imagine the system aims to learn the dynamics of different cars. An example of a global property among cars could be that all of them are exposed to the earth's gravity, impacting their motion. A task-specific property might be the car's motorisation or its number of passengers.

Current meta-learning algorithms lack a systematic approach to aquire tasks and randomly sample them. To make meta-learning more efficient, this work aims to leverage inferred task-specific properties to sequentially acquire the most informative tasks.

1.1 Motivation

We briefly touch upon the research fields that lay the foundation for this paper and motivate the need for it. In autonomous systems, there are usually three key challenges: *modelling, predicting and decision making* (Deisenroth 2019a). Starting with decision making, *reinforcement learning* (RL) is inherently concerned with sequential decision making - deciding which action to take in a situation (*state*) - to maximise a numerical reward signal. It is a goal-directed computational approach to learning from interactions with an environment. In general, an RL task describes the problem of finding an optimal *policy* that captures the actions to take in order to maximise the total reward an *agent* that interacts with the environment. Typically, differences between tasks within the same environment (sometimes referred to as *world*) are either *reward*- or *dynamics*-driven. The former means that the same actions applied to the same environment can lead to different rewards. Differences in dynamics result in the agent arriving at a different new state when applying the same control in the same previous state among all tasks. As an example from the real world, researchers formalised autonomous car driving as a reinforcement learning problem (Kendall et al. 2018; Levine 2019), where the reward signal could be the distance travelled by the vehicle without a human driver intervening.

In general, current RL algorithms usually suffer from *data-inefficiency*, meaning that the required number of interactions with the environment is impractically high (Kamthe and Deisenroth 2017). *Model-based reinforcement learning* (MBRL) is one way to improve data-efficiency of RL by modelling the transition dynamics of a system. This model can then be used as a surrogate for the real environment, removing the need for interactions with the real system. Assuming to have trained a model well, it can then make predictions about the long-term consequences caused by the actions of an agent. Note that the predictive performance of the model is crucial for *planning processes* that aim to produce optimal policies. The faster the model's predictive performance on the transition dynamics improves, the quicker a planning algorithm finds optimal policies. When the model is poorly trained, the planning component is likely to compute a suboptimal policy (Barto 2018). One suitable class of models for MBRL is the class of *probabilistic models* that allow for incorporating *uncertainty* within their predictions explicitly. These models are equipped to propagate the uncertainty over time and deal with reasonably limited experience in a principled way (Deisenroth 2010). Research in cognitive sciences has shown that this resembles how humans make decisions (Körding and Wolpert 2006).

Meta-reinforcement-learning (MRL) models extend the single-tasked MBRL approach to multiple tasks. The goal is to learn how to adapt a model to new reinforcement learning tasks quickly. In essence, these models capture two distinct properties; properties that are shared across all tasks and task-specific properties. Probabilistic MRL approaches infer so-called *latent embeddings* for each task in form of random variables. These embeddings lie in a *latent space* that represents how the tasks relate to each other. In other words, the model infers which combination of task configuration variables is different among observed tasks and by how much they differ.

In the context of the previous car example: All things equal, assume a race track with two same cars except for different motorisations leading to a different brake horsepower specification. Now, assume a model observing two rides per car, where each ride starts from standing still and the driver then simply presses full throttle over the whole observation time. In one ride, the driver is alone in the car, and in the other ride she has a passenger. The *configuration space* of the tasks would include the number of passengers, brake horsepower, and many other variables, some of which have identical values in both cars. Ideally, the model would infer latent embeddings that look similar to what is shown in Figure (1.2), i.e., they express the above-mentioned configuration relations between the tasks.

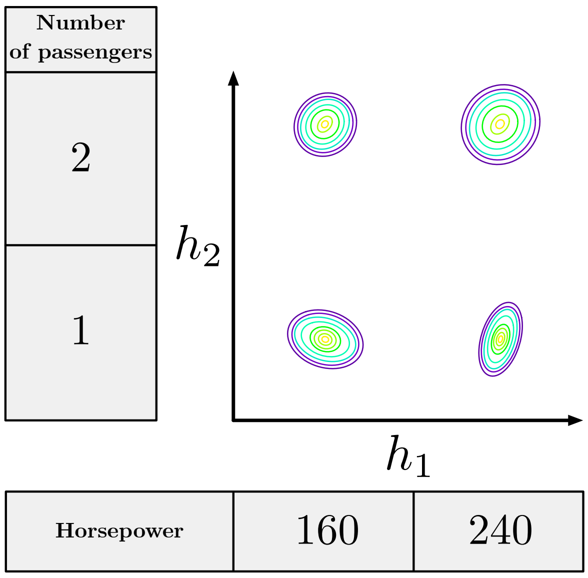


Figure 1.2: An illustration of a two-dimensional latent space with four inferred task embeddings. The actual differences of the observed tasks' configurations are exemplified in the grey-shaded boxes. The goal of the model is to infer embeddings that capture these differences in the task configurations and express them sensibly.

Having yielded latent embeddings of some initial training tasks, how can we exploit this latent space? We will propose a technique for exploring that space fully automated without any domain knowledge about the task difficulties at hand. Knowing the configurations of some training tasks, we can map the latent embeddings to the configuration space. For example, consier the means of embeddings as inputs and the numerical configuration parameters as outputs. We can fit a *latent-to-configuration space* (LC) regression model which we refer to as f_{LC}. The overall procedure is illustrated in Figure (1.3). With that regression model, we can map any other point in the latent space to the configuration space. We refer to these points as *candidate* points. How do we decide which point to select next? We will suggest a variety of *utility functions* to score these candidates and iteratively select the point that has the highest utility. Another problem arising here is the question of how aggressively one can explore the latent space. Points that are too far away from the known embeddings might map to invalid configurations, e.g., a negative number of passengers. However, only considering points that are close to the known embeddings prevents a proper exploration of the latent space, leading to the acquisition of a set of homogeneous tasks and ultimately a bad model. How can we tackle the problem of deciding where to draw the line between points that are should and should not be considered? We will turn this issue of safe exploration into an optimisation problem to find the optimal latent space region from which tasks can then be drawn.

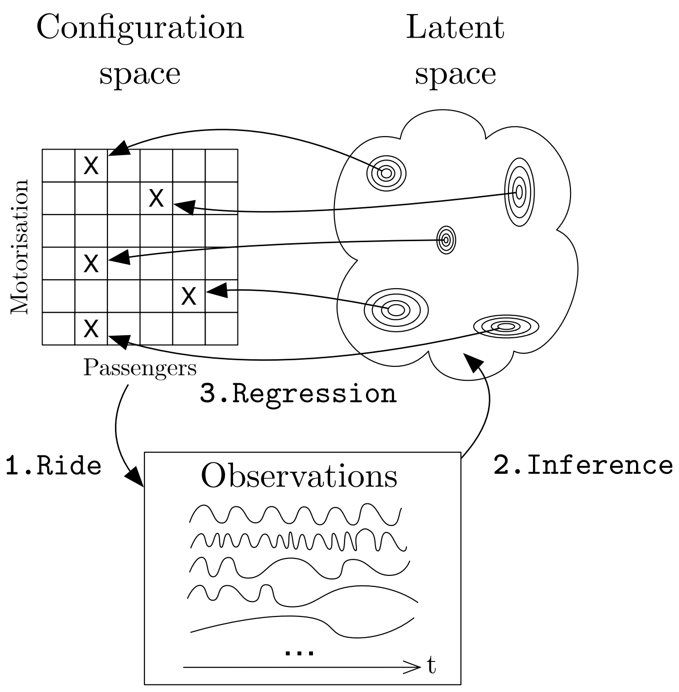


Figure 1.3: An demonstration Pre-PATA's starting point with initial training tasks before any new tasks are actively acquired. Firstly, `1.Ride` refers to a car with a certain configuration from the configuration space going for a ride. Secondly, `2.Inference` highlights the model inferring task-specific latent embeddings based on observations. Lastly, `3.Regression` points out that one can map those embeddings, e.g., their means or samples drawn from their distribution, back to the configuration space with a regression model.

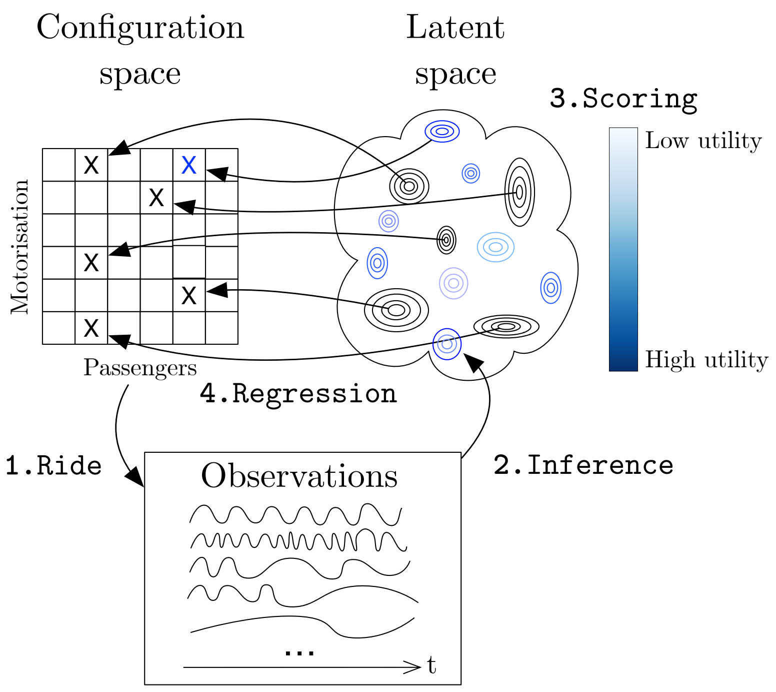


Figure 1.4: Complementig the previously shown starting point with a *3. Scoring* module. Now, after having inferred some initial training tasks, potential candidates in the latent space are scored, the corresponding configuration of the most informative task gets acquired and then the training repeats.

Finally, we will validate PATA's effectiveness empirically by competing against a random baseline on challenging continuous control benchmarks.

1.2 Contributions

The major novel contributions of this work are summarised below:

1. **Probabilistic Active Task Acquisition (PATA)**: A task-agnostic framework for efficient training of probabilistic model-based meta-reinforcement-learning through active task acqusition
2. **Model-based PATA strategies (MB-PATA)** which acquire tasks with small computational overhead but still outperform the random baseline
3. **Model-free PATA strategies (MF-PATA)** which acquire tasks exploiting the meta-learning model's uncertainty and beats the random baseline
4. **Safe exploration of candidate space (SECS)** by trading off exploration/exploitation for generation of candidate sets which can be applied with any acquisition method
5. **Seven different task acquisition tactics** incorporated into the PATA framework allowing to follow complementary objectives

1.3 Outline

The structure of this thesis is summarised below:

- **Chapter 2** will inform the reader about the key ideas behind the overall problem field in which we are: Reinforcement learning. Also, we will briefly elaborate on the prinicples of *Model-based reinforcement learning* (MBRL), the type of RL approach we use later for MRL.
- In **Chapter 3**, we will introduce probabilistic modelling (PM), one class of modelling techniques that are convenient to use for MBRL. This chapter lays the foundation for many of the upcoming algorithms: Bayesian inference illustrates how we can fit a probabilistic model to data, variational inference shows one method to make PM more scalable. Furthermore, we will introduce the concept of latent variable models that allow us to infer a low-dimensional latent space we can then explore. Lastly, we touch upon Bayesian optimisation, a concept we will later use to better explore the latent space.
- **Chapter 4** presents concrete probabilistic modelling methods for MBRL. We start with Gaussian Process (GP) regression, touch briefly upon one method that makes GPs more scalable (variational sparse GP regression) and finally, we will introduce a MRL model as basis for our framework.
- Before we start to propose our framework, **Chapter 5** conducts a brief literature review, concluding with more motivation for PATA.
- Having guided the reader through the background and related work, we can now convey our overall framework PATS in **Chapter 6**. We also describe particular tactics a practitioner can use right-off-the-bat
- To evaluate our proposed methods, we empirically observe their performance in the context of RL benchmark tasks in **Chapter 7**.
- Lastly, we summarise the findings of this project in **Chapter 8** and discuss future work.

Chapter 2

Reinforcement learning

Famous achievements of machine learning in the context of board games (Silver et al. 2018), e-sports (Vinyals, Vezhnevets, and Silver 2017) as well as real world application areas such as robotics (Kober, Bagnell, and Peters 2009) or economics (Perolat, Beattie, and Tuyls 2017) originate from reinforcement learning. To better understand its fundamentals, we start with *control theory*, a decades-old field of engineering that aims to find optimal controllers for well-understood systems. To find an equivalent of controllers in situations where we do not know the underlying system we interact with (or not completely), we bridge the gap by introducing RL. Lastly, we move on to model-based RL, one class of RL algorithms that aims to make RL more data-efficient.

2.1 Control theory

In engineering and mathematics, *control theory* deals with the behaviour of *dynamic systems*, i.e., systems that evolve over time (Dorf and R. H. Bishop 2000). The evolution of the system as a function of time is measured in *states* which fully describe the system at a particular point in time. *Control signals* are external signals that can modify the current state. Together with the control signals and equations describing the dynamics, the future state can be determined (deterministically or stochastically). *Closed-loop feedback control systems* are systems with an additional measure of the actual output, capable of comparing it with the desired output response.

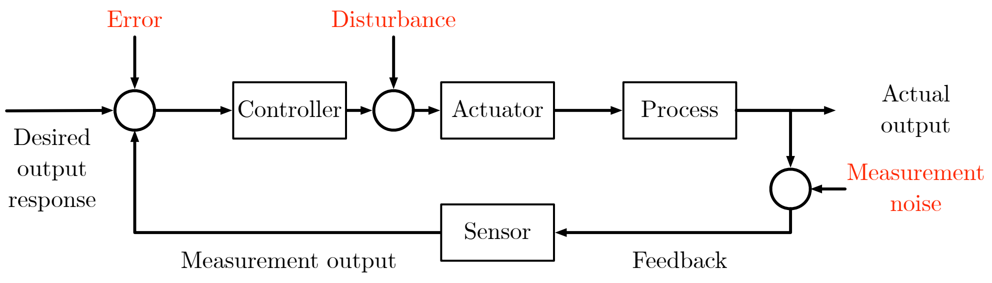


Figure 2.1: Closed-loop feedback system with external disturbances and measurement noise described in the book by (Dorf and R. H. Bishop 2000).

Real world example
A car with adaptive cruise control is a modern control system: The system's sensors measure the distance between the car's position and other cars to determine its state, typically with external disturbances and measurement noise. The software computes control signals to adjust the car's acceleration and wheel steering.

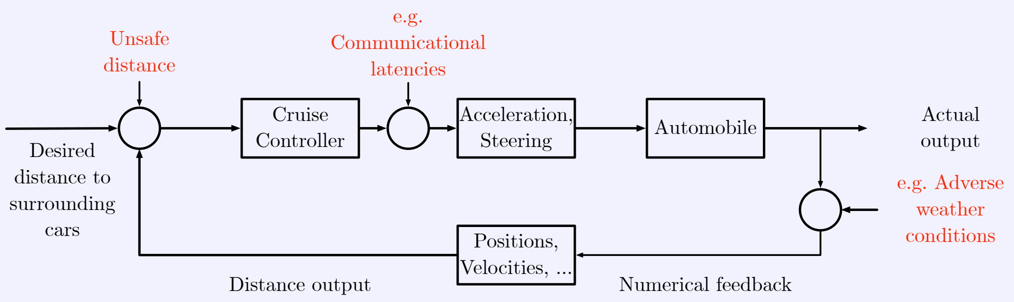

Unsafe distance
e.g. Communicational latencies
Desired distance to surrounding cars
Cruise Controller
Acceleration, Steering
Automobile
Actual output
e.g. Adverse weather conditions
Positions, Velocities, ...
Distance output
Numerical feedback

Figure 2.2: An illustrative closed-loop feedback system of an adaptive cruise control system as installed in various modern cars.

Control theory example
The cart-pole system (or *inverted pendulum*) is a classic control theory system which was was adapted from the earlier work of (Widrow and Smith 1964). It allows to apply horizontal forces (left and right) on the moving cart such that its attached pendulum, the *pole*, freely swings around. Depending on its start state, classic tasks are swinging the pole up from hanging straight down or balancing it around its unstable equilibrium to keeping it from falling over (Tedrake 2009).

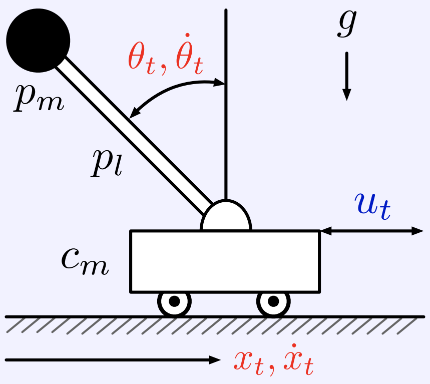


Figure 2.3: The cart-pole setup as described in (Tedrake 2009). State variables are marked in red, the control signals in blue and the constants in black. $x_t, \dot{x}_t$ refer to the horizontal position and velocity of the cart and θ_t is the anti-clockwise angle of the pendulum with $\dot{\theta}_t$ being the corresponding velocity. p_m, p_l describe the mass and length of the pole, respectively, c_m refers to the mass of the cart and g to gravity.

As described in (Deisenroth 2010), one possible state representation $\mathbf{x}_t$ of the system as a function of time t consists of the position x_1 of the cart, the velocity $\dot{x}_1$ of the cart of the system, the angle θ (measured anti-clockwise from hanging down) and the angular velocity $\dot{\theta}$, both of the pendulum. In practise, the angle is often represented as a complex number on the unit circle, such that it is mapped to its sine and cosine. We explain the reasons for that later. In total, for the simulation the state is internally represented as

$$\mathbf{x} = \begin{bmatrix} x_1 & \dot{x}_1 & \dot{\theta} & \sin\theta & \cos\theta \end{bmatrix}^\top \in \mathbb{R}^5. \tag{2.1}$$

Intuitively, when speaking about the *dynamics* of a system, we refer to the transition function f mapping a given state and a control signal to the next state. Here, we typically assume that the system has the *Markov property*: The present state captures all relevant information of the history, such that the sequence of events preceding is irrelevant, i.e., f only depends on the present state. Furthermore, we consider stochastic dynamical systems which are dynamical systems in which the equations of motion have an element of randomness to them (Bhattacharya and Majumdar 2003).

Definition 2.1.1 (Stochastic dynamical system)**.** *A stochastic dynamical system consists of state variables $\boldsymbol{x}_t \in \mathbb{R}^D$ and control signals $\mathbf{c}_t \in \mathbb{R}^K$ where the state follows the Markovian transition dynamics*

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \mathbf{c}_t) + \epsilon, \tag{2.2}$$

with an unknown transition function f, and i.i.d. system noise ϵ, where $\boldsymbol{E} = diag(\sigma_1^2, \dots, \sigma_D^2)$.

2.2 Reinforcement learning

Classic control theory often assumes that exact mathematical formulations for the system to-be-controlled are available (Deisenroth 2010). To come up with these formulations, experts have to obtain a thorough understanding of the system dynamics. Since complex systems can have many latent parameters leading to severe difficulties for experts aiming to find formal descriptions or intractable equations (Bertsekas 2019), machine learning methods can provide remedy for automatic control: A learning algorithm can be used to model the dynamics of a system by extracting structures in observations. Methods that rely on learning algorithms, i.e, approximations of the a priori unknown environment, to produce sub-optimal controllers are commonly referred to as *reinforcement learning* (RL).

RL borrows ideas from control theory, in particular from optimal control of incompletely-known Markov decision processes (Barto 2018). However, according to (Kaelbling, Littman, and Moore 1996), its focus lies more on finding a balance between exploration of "uncharted territory" and exploitation of over time acquired experience rather than finding guaranteed-optimal controllers of a well-understood system for which analytical functions exist that can be hard-coded.

In this work, we do not touch upon all central ideas of RL, especially not the methods for learning optimal controllers (or *policies* in RL terminology). This is because we are only interested in modelling the dynamics of a system in an efficient way. However, to motivate the need of modelling the dynamics, we briefly cover some of the fundamental principles in RL and link them the objective of this thesis.

In a RL setting, the key objective is to find a policy consisting of control signals $\pi_* := \mathbf{c}_0^*, \dots, \mathbf{c}_{T-1}^*$ that minimises the expected long-term cost J of starting in a state $\mathbf{x}$ and following π^*. We can determine the cost J by computing an approximative *value function* for a given state.

Definition 2.2.1 (Value function)**.** *The value function of a state $\mathbf{x}$ under a policy π is the expected cost J (or return $R := -J$) when starting in that state and following π thereafter for a finite horizon of T time steps denoted by*

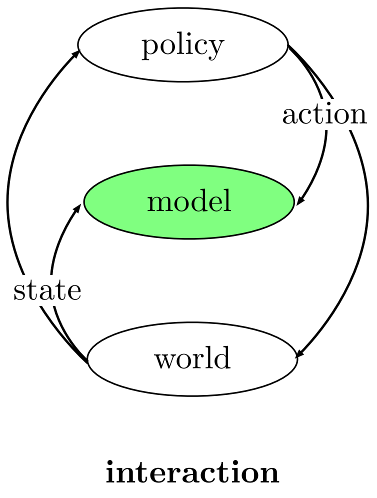
$$J := V^\pi(\boldsymbol{x}_0) = \mathbb{E}_{\boldsymbol{\tau}}\Big[\sum_{t=0}^{T} l(\boldsymbol{x}_t)\Big] = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}\Big[l(\boldsymbol{x}_t)\Big], \tag{2.3}$$

with $\boldsymbol{\tau} := (\boldsymbol{x}_0, \dots, \boldsymbol{x}_T)$ as the trajectory of states visited and l being a known cost function that encodes the task objective (Deisenroth 2010).

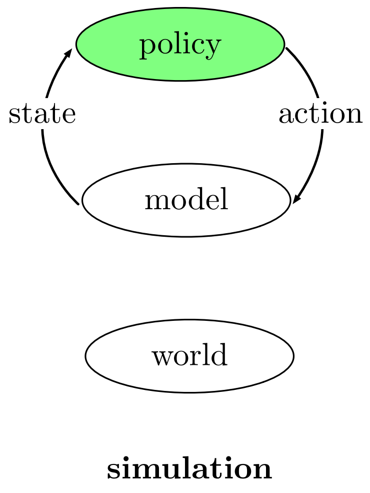
2.3 Model-based reinforcement learning

To gain knowledge about an environment, a so called *agent*, an autonomous entity, interacts with the environment in a trial-and-error fashion. In practise, many trials are required to learn to solve a real-world task, making many RL approaches time-consuming or expensive.

One way to improve the data-efficiency is *model-based RL*, in which models learn the transition dynamics of a system. These models can then be used as a surrogate for the real environment, such that policies can be obtained from it without the need for interactions with the real system (Kamthe and Deisenroth 2017) but by speculating about the system's long-term behaviour.



(a) The interaction phase aims to refine the model of the world. The agents interact with the real world: it applies actions directly to the actual environment. Then, we can observe how the world changes and transitions to a new state. This procedure repeats so that the model can learn over time: it takes the applied actions and the states of the real world to refines itself.



(b) The simulation phase aims to refine the policy given a model of the world. The agent applies actions to the model and receives states of which the model thinks the world might be in. Then, the policy determines the next action according to the state returned by the model and applies it again. Using this simulated experience, the policy improves over time.

Figure 2.4: Model-based reinforcement learning in a nutshell. Two alternating phases, interaction and simulation, improve the model and policy over time, respectively. Green color indicates that the corresponding component is being refined. Both figures are adopted from (Deisenroth 2010).

One weakness of this approach to obtain optimal policies is *model bias*. Optimising a policy based on a biased model of the world leads to sub-optimal policies. The more erroneous the model is, the worse the policy learned. Probabilistic dynamics model explicitly incorporate uncertainty. Thereby, long-term planning with probabilistic models benefits from "knowing what they do not know" (for sure). Details on such approaches are beyond the scope of this work, but we refer the reader to (Deisenroth 2010) in particular, which is the basis for the meta-learning model presented in the upcoming chapter which in turn forms the foundation for this work.

To conclude, MBRL is relevant for this thesis due to two reasons: First, PATA aims to reduce model bias fast by selecting informative tasks. Second, later in this thesis, we will use a model of the real world to predict short trajectories of candidate tasks .

Chapter 3

Probabilistic Modelling

3.1 Bayesian inference

In this section, we briefly cover the basics of *Bayesian inference* that allows us to go from what is known (what we like to refer to as *data* or *observations*) to extrapolate backwards and make probabilistic statements about its generative processes that generated the observations (Lambert 2018). Think of these parameters as levers that one can pull to change the behaviour of the *latent* processes we do not observe but we like to model. In Bayesian inference, probability distributions are used to express uncertain beliefs (*prior knowledge*) about the distribution of parameters to be modelled. These beliefs can be updated in light of new observations to derive more informed, *posterior* beliefs. To do so, we use the *Bayes' theorem* (also *Bayes' rule* or *Bayes' law*), defined as follows.

Definition 3.1.1 (Bayes' theorem)**.** *Consider the observable random variable* **y** *and the unobserved random variable* $\boldsymbol{\theta}$*. We assume some prior knowledge* $p(\boldsymbol{\theta})$ *about* $\boldsymbol{\theta}$ *and a relationship* $p(\mathbf{y}|\boldsymbol{\theta})$ *between* $\boldsymbol{\theta}$ *and* **y***. Having observed* **y***, the Bayes' theorem allows us to estimate a probability distribution over* $\boldsymbol{\theta}$ *given the observations such that*

$$\underbrace{p(\boldsymbol{\theta}|\,\mathbf{y})}_{\textit{posterior}} = \frac{\overbrace{p(\mathbf{y}|\boldsymbol{\theta})}^{likelihood}\ \overbrace{p(\boldsymbol{\theta})}^{prior}}{\underbrace{p(\mathbf{y})}_{\textit{evidence}}}. \tag{3.1}$$

From a statistics standpoint, the *likelihood* $p(\mathbf{y}|\boldsymbol{\theta})$ tells us the probability of generating a particular observation if the parameters of our model were equal to $\boldsymbol{\theta}$. The *prior* $p(\boldsymbol{\theta})$ encapsulates our subjective knowledge of the unobserved variable before we observe any data, so it needs to be (carefully) designed by the practitioner. The denominator term $p(\mathbf{y})$ is called the *evidence* and represents the probability of yielding **y** if we assume a particular model and prior; hence, it is defined by

$$p(\mathbf{y}) := \int p(\mathbf{y}|\boldsymbol{\theta})p(\mathbf{y})\,\mathrm{d}\boldsymbol{\theta} = \mathbb{E}_{\theta}[p(\mathbf{y}|\boldsymbol{\theta})]. \tag{3.2}$$

Finally, our quantity of interest is the posterior probability distribution $p(\boldsymbol{\theta}|\,\mathbf{y})$. It describes our uncertainty over the values of a parameter vector $\boldsymbol{\theta}$. Intuitively speaking, if the posterior is narrower, then this indicates greater confidence in our estimates of the parameter's value (Lambert 2018). Making the posterior more narrow can be achieved by collecting more data.

From a pure mathematical standpoint, the Bayes' theorem is a direct consequence of the product rule

$$p(\boldsymbol{\theta}, \mathbf{y}) = p(\boldsymbol{\theta}|\mathbf{y})p(\mathbf{y}) = p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{3.3}$$

$$p(\boldsymbol{\theta}|\mathbf{y})p(\mathbf{y}) = p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \iff p(\boldsymbol{\theta}|\mathbf{y}) = \frac{p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{y})}. \tag{3.4}$$

3.2 Bayesian optimisation

In mathematics, global optimisation is a branch that deals with finding global minima or maxima of functions. For many practical applications, we do not always know an easy functional form of the function we like to optimise. We call these functions *black-box* functions, since we might not know (exactly) its internal workings but can give it an input and observe an output. This prevents us from calculating its derivatives and applying other common optimisation techniques like *gradient descent*. Also, we might be able to compute the function pointwise but we assume its evaluation is costly. Additionally, another issue might be that the function lacks special structures like concavity or linearity that would improve efficiency during optimisation (Frazier 2018). If the reader is familiar with deep learning, she can think of such an "expensive to evaluate" black-box function as a deep neural network with many layers, resulting in week-long evaluation times for a single run, she likes to tune hyper-parameters of such as the number of layers and their sizes.

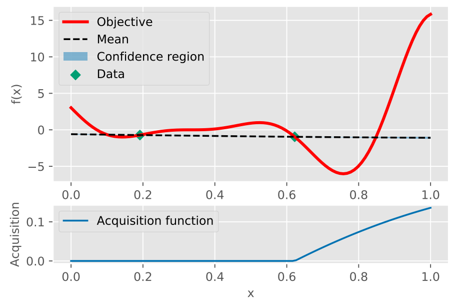
One solution to find the global optimum of such a function is *Bayesian optimisation* which builds a probabilistic surrogate model approximating the objective function f using observed function evaluations as training data and then optimise that cheap proxy model $\tilde{f}$ to determine where to evaluate the true objective next (Deisenroth 2019b). More formally, it is a sequential analysis strategy that aims to find a global optimum of a black-box function f:

$$\mathbf{x}_* = \operatorname*{argmin}_{\mathbf{x}\in\mathcal{A}} f(\mathbf{x}), \tag{3.5}$$

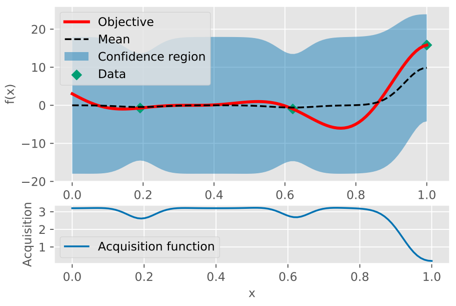
where f denotes the expensive-to-evaluate objective function, $\mathcal{A}$ is the feasible set of all possible points that are eligible, e.g. $\mathcal{A} \subset \mathbb{R}^D$ and D denotes the dimension with typically $D \leq 20$, i.e., not too many dimensions which would cause difficulties.

An often used probabilistic model to approximate the objective function is a Gaussian process (GP) which we will cover in the next chapter. A probabilistic model, such as the GP, returns a probability distribution for each prediction, informing the practitioner about the uncertainty it has over its prediction. This uncertainty is then used in Bayesian optimisation to trade off exploration and exploitation during the acquisition of a next point.

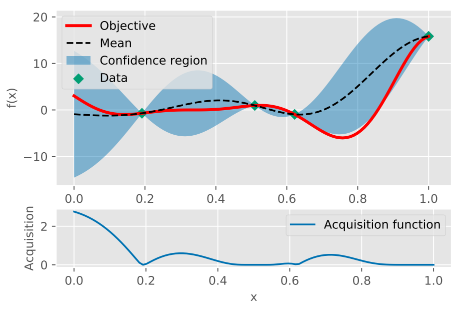
Exploration in that context means, that the BO algorithm tries to explore the function at very uncertain inputs, usually near places it has not evaluated the function yet. Exploration during optimisation is especially important when the function has many local optima and there is a high likelihood of having the optimiser being stuck in a non-global optima. On the other hand, exploitation within BO refers to seeking the evaluation of inputs near already known points which have high/low function values (depending on whether it is an maximisation or minimisation problem, respectively). The function that decides which point to evaluate next is called *acquisition function*. In literature, there exist a couple of closed-form acquisition functions and the research in that field is active (e.g. Wilson, Hutter, and Deisenroth 2018; Wilson, Moriconi, et al. 2017). Covering these is beyond the scope of this work but we refer the reader to (Frazier 2018) to learn more about them.



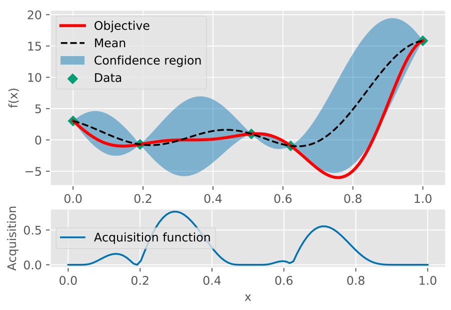
(a) Iteration 0 with only two initial points.



(b) Iteration 1.



(a) Iteration 2.



(b) Iteration 3.

Figure 3.2: An illustration of the principle behind Bayesian optimisation. The acquisition function decides where to acquire the next point; the higher its value, the more it is interested in the corresponding candidate.

3.3 Variational inference

One of the challenges of applying probabilistic models to the real-world is the computation of the posterior distribution as in Equation (3.1.1). There exist multiple hurdles that can make the exact evaluation range from difficult to not analytically tractable at all. In our case of model-based Meta-RL, the probabilistic model is trained on trajectory data of multiple tasks which can quickly becomes infeasible (which we show later when detailing GPs and their computational complexity for exact inference). For this reason, we turn to approximate inference schemes which derive approximate versions of the posterior. These schemes can be broadly categorised into two classes, stochastic and deterministic approximations (C. M. Bishop 2006). We will now introduce one family of deterministic approximations on which our probabilistic meta-learning model is based on to cope with an increasing number of data points when selecting and then observing new tasks: *Variational inference* (VI).

The key idea behind VI is to find an approximation by optimisation: it first posits a parameterised family of approximating distributions and then aims to find the member of that family that is as close as possible to the true posterior. The complexity of the family determines the complexity of the optimisation, i.e., the more complex the distribution, the more complex the optimisation (Blei, Kucukelbir, and McAuliffe 2017). An example of such a family is the *mean-field variational family* where all latent variables are independent of each other (a very simple but naive assumption). In order to compare the approximating distribution with the true posterior, we need a distance measure. We make a brief excursion into the world of information theory that equips us with such.

Excursion: Information theory
One fundamental question in information theory is how to quantify the amount of information of a probability distribution. A key measure for that is *entropy* which quantifies the amount of uncertainty in the values a random variable can have (Cover and J. A. Thomas 2006). Consider the following example: Imperial College London (ICL) is a quite international university [a] and assume you would describe the nationality of a randomly selected student as a random variable. Now, consider a random variable describing the nationality of a member of the Austrian Student Society (ASS) at ICL. There might be some non-Austrian members of that society, for example people that just love Wiener schnitzel, but empirically speaking, we assume there is less 'uncertainty' in the nationality of a randomly selected Austrian-Society-member than of a typical student. We could describe this difference in uncertainty with entropy and would see that the entropy of a RV modelling all student's nationality is higher than the entropy of only considering members of the Austrian student society.

Definition 3.3.1 (Entropy). *The entropy $\mathbb{H}(X)$ of a random variable X is a measure of the amount of information required on the average to describe it (Cover and J. A. Thomas 2006;*

Murphy 2012). Hence, it can be interpreted as a measure of uncertainty. For a discrete random variable with states $x \in \mathcal{X}$, it is defined by

$$\mathbb{H}(X) \triangleq -\sum_{x \in \mathcal{X}} p(x) \log p(x), \tag{3.6}$$

or for the continuous case by

$$\mathbb{H}(X) \triangleq -\int_{\mathcal{X}} p(x) \log p(x). \tag{3.7}$$

Coming back to the above example, let's consider the two distributions $p(\text{nationality}|\text{ ICL})$ and $p(\text{nationality}|\text{ASS})$. Suppose a first-year PhD student, coming from another university ABC, likes to hang around with a group of people of very different nationalities. This is because at her former university, the number of students from one country was approximately equal for all countries and the probability of a randomly selected student's nationality was uniformly-distributed as $q(\text{nationality}|\text{ ABC})$. She considers the options of joining the Austrian-society which meets at Austrian restaurants or spending more time with a broader range of ICL students, e.g., at the Union Bar. In order to determine where she should spend her leisure time, she decides to select the group of people with a lower relative entropy to her former university's nationality distribution, i.e., the group which resembles her previous environment better. She computes the relative entropy between both $q(\text{nationality}|\ ABC)$ and $p(\text{nationality}|\text{ASS})$ as well as between $q(\text{nationality}|\text{ ABC})$ and $p(\text{nationality}|\text{ ICL})$. Assuming adequate distributions for the other two RVs, she would certainly see that the KL-divergence of the former is higher then the latter and decide to not join the ASS.

Definition 3.3.2 (Relative entropy). *The relative entropy or **Kullback-Leibler distance** is a measure of dissimilarity between two probability mass functions $p(x), q(x)$ defined as:*

$$\mathbb{KL}(p \parallel q) \triangleq \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}, \tag{3.8}$$

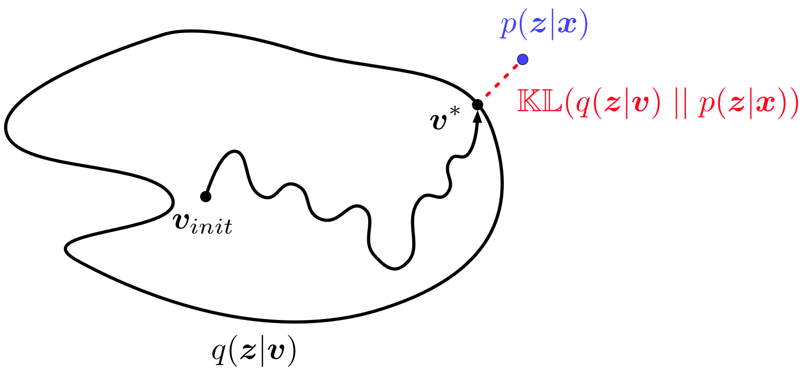
and when considering probability density functions of continuous random variables, it is given by

$$\mathbb{KL}(p \parallel q) \triangleq \int_{\mathcal{X}} p(x) \log \frac{p(x)}{q(x)}. \tag{3.9}$$

The relative entropy is always non-negative and zero if and only if $p = q$. Furthermore, it is not a true distance between distributions since it is not symmetric and does not satisfy the triangle inequality.

[a]According to the ICL website, 64% of its in 2017/2018 enrolled students were not from the UK.

Now, being aware of the KL-divergence, we want to compare the variational approximation q and the true posterior p with $\mathbb{KL}(q(\mathbf{z}|\ \mathbf{v}) \parallel p(\mathbf{z}|\mathbf{x}))$ and $q(\mathbf{z}|\ \mathbf{v}) = p(\mathbf{z}|\mathbf{x}) \iff \mathbb{KL}(q(\mathbf{z}|\ \mathbf{v}) \parallel p(\mathbf{z}|\mathbf{x})) = 0$ would be the optimal solution as illustrated in Figure (3.3).



$p(\boldsymbol{z}|\boldsymbol{x})$
$\mathbb{KL}(q(\boldsymbol{z}|\boldsymbol{v}) \,||\, p(\boldsymbol{z}|\boldsymbol{x}))$
$\boldsymbol{v}^*$
$\boldsymbol{v}_{init}$
$q(\boldsymbol{z}|\boldsymbol{v})$

Figure 3.3: Illustration of the principle behind variational inference: approximation by optimisation. The black shape demonstrates the family of approximating distributions parameterised by variational parameters $\mathbf{v}$. We like to find the optimal parameters $\mathbf{v}^*$ in the sense that $q(\mathbf{z}|\mathbf{v}^*)$ has the smallest KL divergence (shown in the red term) to the true posterior $p(\mathbf{z}|\mathbf{x})$ (blue term) of all possible parameterisations (all points inside the black-lined surface). The figure is adopted from (Blei, Ranganath, and Mohamed 2016).

However, the KL divergence is not computable because it requires computing the evidence $p(\mathbf{x})$, which is also one of the reasons why we approximate inference in the first place. This can be seen by

$$\mathbb{KL}(q(\mathbf{z}|\mathbf{v}) \parallel p(\mathbf{z}|\mathbf{x})) = \int_{\mathcal{Z}} q(\mathbf{z}|\mathbf{v}) \log \frac{q(\mathbf{z}|\mathbf{v})}{p(\mathbf{z}|\mathbf{x})} \tag{3.10}$$

$$= \int_{\mathcal{Z}} q(\mathbf{z}|\mathbf{v}) \log q(\mathbf{z}|\mathbf{v}) - \int_{\mathcal{Z}} q(\mathbf{z}|\mathbf{v}) \log p(\mathbf{z}|\mathbf{x}) \tag{3.11}$$

$$= \int_{\mathcal{Z}} q(\mathbf{z}|\mathbf{v}) \log q(\mathbf{z}|\mathbf{v}) - \int_{\mathcal{Z}} q(\mathbf{z}|\mathbf{v}) \log \frac{p(\mathbf{z},\mathbf{x})}{p(\mathbf{x})} \tag{3.12}$$

$$= \int_{\mathcal{Z}} q(\mathbf{z}|\mathbf{v}) \log q(\mathbf{z}|\mathbf{v}) - \int_{\mathcal{Z}} q(\mathbf{z}|\mathbf{v}) \log p(\mathbf{z},\mathbf{x}) + \int_{\mathcal{Z}} q(\mathbf{z}|\mathbf{v}) \log p(\mathbf{x}) \tag{3.13}$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}[\log q(\mathbf{z}|\mathbf{v})] - \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}[p(\mathbf{z},\mathbf{x})] + \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}[\log p(\mathbf{x})]. \tag{3.14}$$

However, instead of minimising $\mathbb{KL}(q(\mathbf{z}|\mathbf{v}) \parallel p(\mathbf{z}|\mathbf{x}))$, equivalently, we can remove the summand involved with $p(\mathbf{x})$ from the equation, which is a constant with respect to $q(\mathbf{z}|\mathbf{v})$, multiply it with (-1), and maximise that new objective which is called the *evidence lower bound* (ELBO). Precisely, the ELBO is the sum of the expected log likelihood of the data and the KL divergence between the priors $p(\mathbf{z})$ and $q(\mathbf{z})$, given as:

$$\text{ELBO}(q) := \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}[p(\mathbf{z},\mathbf{x})] - \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}[\log q(\mathbf{z}|\mathbf{v})]. \tag{3.15}$$

Its name stems from the fact that it lower bounds the (log of the) evidence:

$$\log p(\mathbf{x}) = \log \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\, \mathrm{d}\mathbf{z} \tag{3.16}$$

$$= \log \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z}) \frac{q(\mathbf{z})}{q(\mathbf{z})}\, \mathrm{d}\mathbf{z} \tag{3.17}$$

$$= \log \int p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z})} q(\mathbf{z})\, \mathrm{d}\mathbf{z} \tag{3.18}$$

$$= \log \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}\left[p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}\right] \tag{3.19}$$

$$\geq \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}\left[\log\left(p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}\right)\right] \qquad \text{(Jensen's inequality)} \tag{3.20}$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}[\log p(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}\left[\log\left(\frac{q(\mathbf{z}|\mathbf{v})}{p(\mathbf{z})}\right)\right] \tag{3.21}$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{v})}[\log p(\mathbf{x}|\mathbf{z})] - \mathbb{KL}(q(\mathbf{z}|\mathbf{v}) \parallel p(\mathbf{z})), \tag{3.22}$$

where the Jensen's inequality in Equation (3.20) is an result from convex analysis that states that for concave functions f it holds that: $f(\mathbb{E}[\mathbf{z}]) \geq \mathbb{E}[f(\mathbf{z})]$ and since logarithms are concave, it holds that $\log \mathbb{E}[f(\mathbf{z})] \geq \mathbb{E}[\log f(\mathbf{z})]$.

In conclusion, the key takeaway of this subsection is that there exist approximations of posterior distributions that make inference more scalable and applicable to large datasets. We will come back to this in Section (4.1.7) when we introduce the variational sparse GP we use in this work.

3.4 Latent variable models

This part of the work explains the concept of latent embeddings and how to infer them. After introducing the idea of dimensionality reduction through a practical example to the reader, we briefly touch upon the principles behind *Gaussian process latent variable models* which form the basis of our probabilistic meta-learning model introduced later.
Data with many dimensions can make its analysis complicated. Dimensionality reduction (DR) methods aim to exploit the structure of data and output a more compact representation of it, ideally without loosing (too much) information (Deisenroth, Faisal, and Ong 2020). This works especially well for data sets who have the property that its points all lie close in some topological space of much lower dimensionality than the observed one (C. M. Bishop 2006) which possesses redundant information to some degree. A simple approach for DR is to obtain a linearly uncorrelated data representation by *principal component analysis* (PCA) (Hotelling 1936; Pearson 1901).

Example: Principal Component Analysis (PCA)
The following example is adopted from (Ng and Soo 2017) and aims to introduce the reader to the concept of dimensionality reduction by the example of PCA. Imagine that you are a nutritionist and you would like to differentiate between food items. You think of different variables such as macronutrients (e.g. fat) or micronutrients (e.g. vitamin C) to describe the differences between foods. PCA aims to find underlying variables (i.e. the *principal components*) that best differentiate your data points. For this, it finds the dimensions along which your data-points are most spread out. Mathematically speaking, it projects the a set of observations onto a lower dimensional space while maximising the variance of that projected data. That means, the first principal component accounts for as much of the variability of the data as possible, the second component for the second most data variance, and so on.

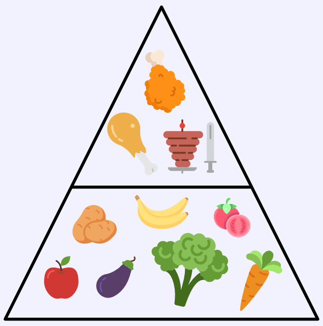


Figure 3.4: A simple food pyramid illustrating the problem of finding reasonable variables to differentiate food. Figure adopted from (Ng and Soo 2017). Icons taken from flaticon.com.

To find such a principal component, PCA computes a linear combination of the known variables. For example, we could only use the level of vitamin C in the food as the single variable to list our food items which can be seen in the leftmost column in Figure (3.5). This would result in many items indistinguishably put together because although vitamin C is present in many vegetables it is absent in meat. In the second attempt, we could try to spread the meat items out by using fat content as a secondary variable since it is present in meat but mostly absent in vegetables. To do so, we need to standardise the vitamin C and meat levels because they are measured in different units. That means, we compute the empirical mean and standard deviation for each variable, subtract the mean from its values and divide it by the standard deviation. We can then combine them by subtracting the standardised fat variable from the standardised vitamin C variable, as shown in the second column. To spreading the foods even further, we could add fiber which vegetables have in varying levels. As a consequence, we would not loose too much information if we would just use a linear combination of fat and protein as well as fiber and vitamin C to come up with two principal components and halve the number of variables.

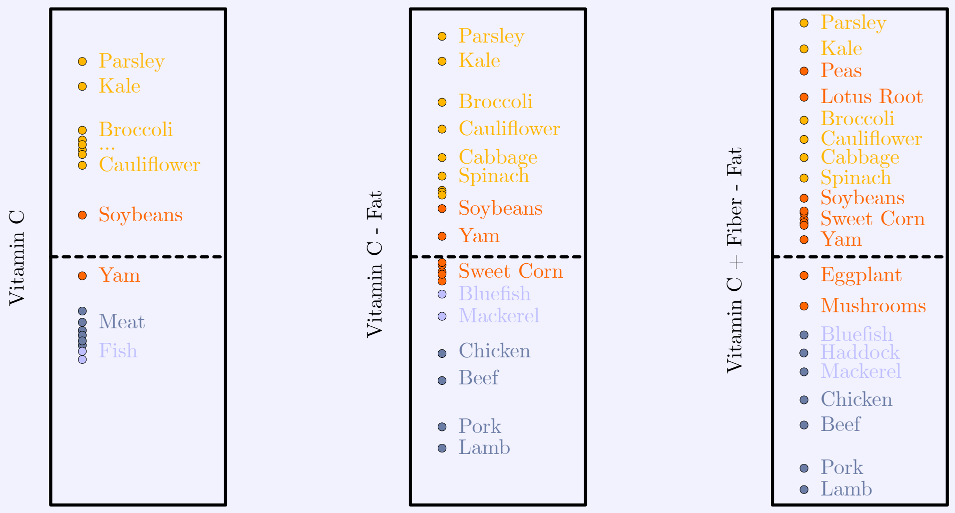


Figure 3.5: Illustration of how combining the variables changes the sorting of the food items. Figure adopted from (Ng and Soo 2017).

At this point, a nutritionist might have the feeling, that some variables are correlated, i.e., they move in the same direction for most items. When we plot the levels of fat, protein, fiber (macronutrients) and vitamin C (micronutrient), as shown in Figure (3.6), we can see that there are some high correlations between fat and protein as well as between fiber and vitamin C.

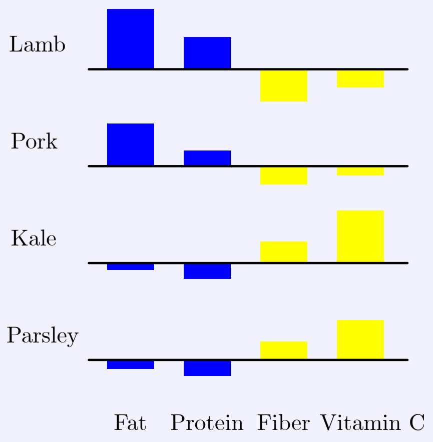


Figure 3.6: Comparison of nutrition levels in different food items. Figure adopted from (Ng and Soo 2017).

Instead of defining the linear combination manually, we can use PCA to yield precise weights with which variables can be combined to best differentiate our items. We call the principal components in this example h_1 and h_2 because we refer to the latent random variables used in this work later with the letter h and aim to make the transition smooth.

Exemplary, it can be seen that h_1 summarises our previous findings by pairing and fat with protein and fiber with vitamin C and then inversely correlating these two pairs. Thereby, meat is clearly differentiated form vegetables. h_2 further differentiates sub-categories within meat and vegetables by fat and vitamin C, respectively. For example, seafood items have lower fat content than some meat items as well as lower vitamin C and fiber levels, hence, they are located in the bottom left. Here, the space spanned by the two principal components can be interpreted as *latent space*.

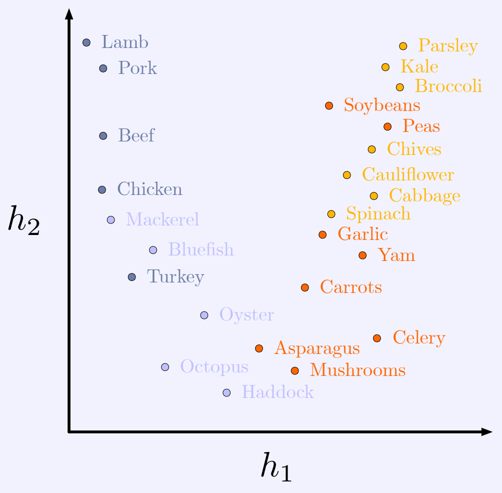


Figure 3.7: Plot of food items in a two-dimensional latent space using the top two principal components. Figure adopted from (Ng and Soo 2017).

A probabilistic framework for dimensionality reduction was introduced by (Neil D Lawrence 2003) named *Gaussian process latent variable model* (GPLVM), extending the idea of PCA to new properties such as coming with a likelihood function that can explicitly deal with noisy observations or being treated as a generative model, which allows to simulate new data (Deisenroth, Faisal, and Ong 2020). This can be done by introducing a continuous-valued latent variable $\mathbf{z} \in \mathbb{R}^M$, turning PCA into *probabilistic PCA* (PPCA), a probabilistic latent variable model (Deisenroth, Faisal, and Ong 2020). In fact, the classical PCA can be shown to arise as the maximum likelihood solution to a particular form of a linear-Gaussian latent variable model (C. M. Bishop 2006).

The GPLVM can be seen as unsupervised regression problem (Gal and Wilk 2014). That means that the inputs are now latent and the mapping $f : \mathbf{X} \mapsto \mathbf{Y}, \mathbf{X} \in \mathbb{R}^{N\times Q}, \mathbf{Y} \in \mathbb{R}^{N\times D}, Q \ll D$ from a latent space to the observation space is inferred at the same time as the inputs. More formally, for N data points and all its dimensions D, we model D independent GPs with the following likelihood functions

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{d=1}^{D} p(\mathbf{y}_d|\mathbf{X}), \tag{3.23}$$

$$p(\mathbf{y}_d|\mathbf{x}) = \mathcal{N}(\mathbf{y}_d|\mathbf{0}, \mathbf{K}_{NN} + \sigma_n^2 \mathbf{I}_N), \tag{3.24}$$

and a prior over the latent inputs $\mathbf{X}$:

$$p(\mathbf{X}) = \prod_{n=1}^{N} \mathcal{N}(\mathbf{x}_n|\mathbf{0}, \mathbf{I}_Q), \tag{3.25}$$

such that the final joint probability model for the GPLVM is

$$p(\mathbf{Y}, \mathbf{X}|\boldsymbol{\theta}, \sigma_n^2) = p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}, \sigma_n^2)p(\mathbf{X}), \tag{3.26}$$

where the hyper-parameters of the model are the kernel parameters and the signal variance σ_n^2.

Chapter 4

Modelling transition dynamics

In the following, we will apply probabilistic modelling to (meta) reinforcement learning through Gaussian Processes. We will start with a simple *regression* setting, where we aim to find a function f that maps inputs $\mathbf{x} \in \mathbb{R}^D$ to corresponding function values $f(\mathbf{x}) \in \mathbb{R}$. We assume we observed a set of trianing inputs $\mathbf{X} := [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and corresponding noisy observations $y_n = f(\mathbf{x}_n) + \epsilon$ where ϵ is an i.i.d. random variable that describes observation noise. We then extend this setting to multi-dimensional targets. Moreover, GPs suffer from high computational complexity and can become infeasible as the dataset size grows. Therefore, we introduce *variational sparse Gaussian Processes*, a variational-inference-based technqiue to make GP regression more scalable. Lastly, putting everything together, we introduce the MRL model this work incorporates.

4.1 Gaussian Process regression

4.1.1 Definition

Gaussian processe are useful for non-parametric regression which means that we do not assume a parameter set ϕ that imposes a fixed structure upon the latent function we like to model (Deisenroth 2010). We assume that the "parameters" of interest are the values of the latent function itself. This might sound counterintuitive, since we like to model transition dynamics which are typically defined by a finite number of parameters ϕ that follow physical laws such as Newton's laws of motion. However, aiming to find these exact parameters is very difficult, especially with real-world dynamical systems and noisy observations, i.e., observations collected by real sensors and not under idealised circumstances as in computer simulations.

Definition 4.1.1 (Stochastic Process). *A Stochastic process (SP) is a collection of random variables $\{X(t), t \in T\}$, where for each $t \in T, X(t)$ is a random variable taking values in a common measurable space Ω endowed with an appropriate σ-algebra and T is called the index set of the process.*

A common SP is a *temporal stochastic process* where the index t refers to the time such that $X(t)$ can be interpreted as the *state* of the process at time t. T can also be a set of spatial coordinates (*spatial process*) or a set of both (*spatio-temporal process*).

Definition 4.1.2 (Gaussian Process). *A Gaussian process is a stochastic process with a finite number of random variables which are jointly Gaussian distributed (Carl Edward; Rasmussen and Williams 2004).*

We write a function f that is GP distributed as $f \sim \mathcal{GP}$, or $f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$ when specifying the mean and/or covariance function, respectively. One can think of a Gaussian process as a generalisation of the Gaussian distribution. A Gaussian probability distribution describes random variables which are scalars (univariate distribution) or vectors (multivariate distribution). The Gaussian process framework describes a distribution over functions rather than function values. Intuitively speaking, one can interpret each of these functions as an infinitely long vector of function values $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ at corresponding input locations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (for $n \in \mathbb{N}$). In comparison, the Gaussian distribution is only defined for finite-dimensional vectors. However, since we are usually only interested in finite training and test sets, we can partition the set of all function values into training set $\mathbf{f}$, test set $\mathbf{f}_*$ and all 'other' infinitely many function values $\mathbf{f}_{\text{other}}$. The marginalisation property of the Gaussian distribution (that every probability distribution has) allows integrating out $\mathbf{f}_{\text{other}}$

$$p(\mathbf{f}, \mathbf{f}_*) = \int p(\mathbf{f}, \mathbf{f}_*, \mathbf{f}_{\text{other}}) d\mathbf{f}_{\text{other}} = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_f \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{ff} & \boldsymbol{\Sigma}_{f*} \\ \boldsymbol{\Sigma}_{*f} & \boldsymbol{\Sigma}_{**} \end{bmatrix} \right), \tag{4.1}$$

such that we obtain a finite joint Gaussian distribution. Similar to a Gaussian distribution, which is completely specified by a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$, a GP is completely specified analogously by a *mean function* $m(\mathbf{x})$ and a *covariance function* $k(\mathbf{x}_i, \mathbf{x}_j)$ of a real-valued process $f(\mathbf{x})$ as

$$\begin{aligned} m(\mathbf{x}) &:= \mathbb{E}[f(\mathbf{x})] \\ k(\mathbf{x}_i, \mathbf{x}_j) &:= \mathbb{E}[\big(f(\mathbf{x}_i) - m(\mathbf{x}_i)\big)\big(f(\mathbf{x}_j) - m(\mathbf{x}_j)\big)] \\ &= cov[f(\mathbf{x}_i), f(\mathbf{x}_j)], \end{aligned} \tag{4.2}$$

with $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^D$. This means, that we place a prior $p(f)$ directly on the space of functions (Deisenroth 2010). The mean function specifies a prior mean on the functions (describing how the 'average' function is expected to look and the prior covariance function $k(\cdot, \cdot)$ (or *kernel*) describes the covariance between any two function values.

Covariance functions

Intuitively speaking, the covariance function characterises the notion of similarity between two input points. For instance, these differences can describe the spatial or temporal covariance of a stochastic process Wackernagel 1998.

This definition is adopted from (Deisenroth, Faisal, and Ong 2020).

Definition 4.1.3. *Covariance functions output a Gram matrix $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ with respect to $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, where $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ (often referred to as covariance matrix or kernel matrix). They are functions $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ that characterise the similarity between $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$, for which there exists a Hilbert space $\mathcal{H}$ (or feature space) and a feature map $\phi : \mathcal{X} \mapsto \mathcal{H}$ such that*

$$\boldsymbol{K} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle_{\mathcal{H}}, \tag{4.3}$$

and $\boldsymbol{K}$ is symmetric and positive semi-definite matrices which means that

$$\forall \boldsymbol{z} \in \mathbb{R}^N : \boldsymbol{z}^\top \boldsymbol{K} \boldsymbol{z} \geq 0. \tag{4.4}$$

This generalisation from an inner product to a kernel function is known as the *kernel trick* (Scholkopf and Smola 2001). This trick gives freedom to apply a possibly non-linear map $\boldsymbol{\Phi}$ to change the representation of $\mathbf{x}$ into one that is more suitable for a given problem. Consequently, it allows to use many different similarity measures.

In this work, we consider a prior mean function $\mathbf{m}_f \equiv 0$ and the squared exponential kernel function with automatic relevance determination

$$k_{SE}(\mathbf{x}_i, \mathbf{x}_j) := \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^\top \boldsymbol{\Lambda}^{-1} (\mathbf{x}_i - \mathbf{x}_j)) + \sigma_n^2 \delta_{ij}, \tag{4.5}$$

where $\boldsymbol{\theta} = (\{\boldsymbol{\Lambda}\}, \sigma_f^2, \sigma_n^2)^\top$ is a vector collecting all *hyper-parameters* of the latent function f whose values allow for adjusting the covariance function to the given problem. The relevant hyper-parameters when using a $k_{SE}(\cdot, \cdot)$ kernel, as in our work, are:

- $\sigma_f^2 = \sigma^2(f(\mathbf{x}))$, the scaling of the *amplitude* (or *signal variance*) of the latent function f.

- σ_n^2, not a direct parameter of the GP but a parameter of the likelihood function used for predictions (Deisenroth, Luo, and Van der Wilk n.d.). It indicates how much measurement noise the observations contain; i.e., by how much the function values $f(\mathbf{x})$ are corrupted by independent and identically distributed Gaussian noise with variance σ_n^2.

- $\boldsymbol{\Lambda} = \text{diag}([\ell_1^2, \dots, \ell_D^2])$, a diagonal matrix of squared length-scales $\ell_i, i = 1, \dots, D$. These hyper-parameters determine how relevant a particular input dimension is. For example, if the length-scale ℓ_2 for dimension $i = 2$ is very large, the output of $k_{SE}(\cdot, \cdot)$ will be almost independent of that input. This can be helpful, if some input features are not relevant for the problem at hand and need to be 'removed' for good inference results. On the other side, longer lengthscales cause long-range correlations (Deisenroth, Luo, and Van der Wilk n.d.).

In the next subsection, we discuss how we can find good values for these hyper-parameters.

The SE kernel assumes that the latent function f is *stationary* and *smooth*. Stationary covariance functions are of the form $\mathbf{x}_i - \mathbf{x}_j, \forall \mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^D$, meaning that their value only depends on the difference between two input values. This property makes the function invariant to translations in the input space (Carl Edward; Rasmussen and Williams 2004). Smooth functions are infinitely differentiable, i.e., $k_{SE}(\mathbf{x}_i, \mathbf{x}_j)$ is of differentiability class C^∞. In literature, there exist a variety of kernels of which some do not share these two properties but allow for more flexible structures of the latent function. A good starting point is (Duvenaud 2014; Carl Edward; Rasmussen and Williams 2004). Kernels can also be combined, e.g., by multiplication or addition.

4.1.2 Training

A GP possesses a set of hyper-parameters, usually denoted by the hyper-parameter vector $\boldsymbol{\theta}$. These are parameters of the mean and covariance function Ψ (e.g. length-scales $\boldsymbol{\ell}$ and signal variance σ_f^2) and the likelihood parameter noise variance σ_n^2 (in the case of a Gaussian likelihood). Since we do not know good values for them a priori, we treat them as latent variables and place a prior $p(\boldsymbol{\theta})$ on them. To illustrate the motivation of the upcoming estimation of good hyper-parameters $\hat{\boldsymbol{\theta}}$, we firstly show why we can not simply follow a fully Bayesian approach by integrating them out (Deisenroth 2010), such that

$$p(f) = \int p(f|\boldsymbol{\theta})p(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}, \tag{4.6}$$

$$p(\mathbf{y}|\mathbf{X}) = \int\int p(\boldsymbol{y}|\mathbf{X}, f, \boldsymbol{\theta})p(f|\boldsymbol{\theta})p(\boldsymbol{\theta})\,\mathrm{d}f\,\mathrm{d}\boldsymbol{\theta} \tag{4.7}$$

$$= \int p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}. \tag{4.8}$$

The above integral required for the normalising constant $p(\mathbf{y}|\mathbf{X})$ is intractable and can only be approximated, e.g. by Markov chain Monte Carlo (MCMC) methods. This can be seen by taking the log of the *marginal likelihood* (or *evidence*) with

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{X}, f, \boldsymbol{\theta})p(f|\boldsymbol{\theta})\,\mathrm{d}f, \tag{4.9}$$

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top\left(\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y} - \frac{1}{2}\log\left|\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}\right| - \frac{D}{2}\log\left(2\pi\right), \tag{4.10}$$

where D is the dimension of the input space, $\mathbf{I}$ is the identity matrix and the kernel matrix $\mathbf{K}$ depends on the hyper-parameters $\boldsymbol{\theta}$. Therefore, for finding good hyper-parameters $\hat{\boldsymbol{\theta}}$ approximately, we maximise the marginal likelihood with respect to $\boldsymbol{\theta}$, i.e.,

$$\hat{\boldsymbol{\theta}} \in \underset{\boldsymbol{\theta}}{\text{argmax}} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}), \tag{4.11}$$

which is also called a *type II maximum likelihood estimate* (ML-II) of the hyper-parameters. Following (Carl Edward; Rasmussen and Williams 2004), we use the partial derivatives of the marginal likelihood w.r.t. $\boldsymbol{\theta}$, such that:

$$\begin{aligned}\frac{\partial \log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})}{\partial \theta_i} &= \frac{1}{2}\mathbf{y}^\top \mathbf{K}_{\boldsymbol{\theta}}^{-1}\frac{\partial \mathbf{K}_\theta}{\partial \theta_i}\mathbf{K}_{\boldsymbol{\theta}}^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}\left(\mathbf{K}_{\boldsymbol{\theta}}^{-1}\frac{\partial \mathbf{K}_{\boldsymbol{\theta}}}{\partial \theta_i}\right)\\ &= \frac{1}{2}\mathrm{tr}\left(\left(\boldsymbol{\alpha}\boldsymbol{\alpha}^\top - \mathbf{K}_{\boldsymbol{\theta}}^{-1}\right)\frac{\partial \mathbf{K}_{\boldsymbol{\theta}}}{\partial \theta_i}\right) \quad \text{where} \quad \boldsymbol{\alpha} = \mathbf{K}_{\boldsymbol{\theta}}^{-1}\mathbf{y}. \end{aligned} \tag{4.12}$$

Important to note here is that computing the log-marginal likelihood in 4.10 requires matrix inversion of the kernel matrix.

4.1.3 Predictions

After introducing GPs briefly, we now cover how they make predictions; the main reason why we use them in the first place. In this thesis, our GP model will deal with trajectory data of dynamical systems, i.e., it aims to predict the next state, given the current state and a control signal. There are two types of predictions we will make in the upcoming sections. First, the multi-step-ahead prediction of a trajectory (a discrete-time series) requires iterative one-step-ahead predictions where the given inputs are probability distributions (*uncertain inputs*) themselves. In other words, we feed the predictive posterior distribution for the state of an individual time step (except for the last one) back into the model for predicting the next one. Second, later throughout this thesis, we make one time-step ahead predictions for evaluating the model's performance by comparing these predictions with the actual test data (*deterministic inputs*). In this section, we start with the theoretical foundations of deterministic inputs first and will then cover uncertain inputs.

Deterministic inputs

Following Definition (4.1), we can write the joint distribution of observed target values at locations $\mathbf{X}$ and function values at test locations $\mathbf{X}_*$ as

$$p(\mathbf{f},\mathbf{f}_*|\mathbf{X},\mathbf{X}_*) = \mathcal{N}\left(\begin{bmatrix} m(\mathbf{X}) \\ m(\mathbf{X}_*)\end{bmatrix}, \begin{bmatrix}\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I} & \mathbf{K}(\mathbf{X},\mathbf{X}_*)\\ \mathbf{K}(\mathbf{X}_*,\mathbf{X}) & \mathbf{K}(\mathbf{X}_*,\mathbf{X}_*)\end{bmatrix}\right) \tag{4.13}$$

Univariate predictions Inputs $\mathbf{x}_* \in \mathbb{R}^D$, Outputs $y_* \in \mathbb{R}$
We need to restrict this joint prior distribution to contain only those functions which agree with the observed data points. This is done by *conditioning* the joint Gaussian prior distribution on the observations, i.e., $p(\mathbf{f}_*|\mathbf{X},\mathbf{X}_*,\mathbf{f}) = \mathcal{N}(m_f(\mathbf{x}_*), \sigma_f^2(\mathbf{x}_*))$.

Theorem 4.1.1 (Univariate predictive posterior distribution with deterministic inputs)**.** *According to (Carl Edward; Rasmussen and Williams 2004), we obtain the GP posterior predictive distribution by*

$$\mu_* := m_f(\boldsymbol{x}_*) = \mathbb{E}[\boldsymbol{f}_*|\boldsymbol{X},\boldsymbol{y},\boldsymbol{X}_*] = \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})[\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}]^{-1}\boldsymbol{y} \tag{4.14}$$

$$= K(\boldsymbol{X}_*,\boldsymbol{X})\boldsymbol{\beta} = \sum_{i=1}^{n}\beta_i k(\boldsymbol{x}_i,\boldsymbol{x}_*), \tag{4.15}$$

$$\sigma_*^2 := \sigma_f^2(\boldsymbol{x}_*) = cov(\boldsymbol{f}_*) = \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})[\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}]^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*), \tag{4.16}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{\beta} := (\boldsymbol{K} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y}$.

Multivariate predictions Inputs $\mathbf{x}_* \in \mathbb{R}^D$, Outputs $\mathbf{y}_* \in \mathbb{R}^E$
In this work, we wish to predict target values with multiple dimensions simultaneously. One approach for this is to train E independent GP models using the same training inputs $X = [\mathbf{x}_1,\dots,\mathbf{x}_n], \mathbf{x}_i \in \mathbb{R}^D$ but different training targets $\mathbf{y}_a = [y_1^a,\dots,y_n^a]^\top, a = 1,\dots,E$. Consequently, we assume that the function values $f_1(\mathbf{x}),\dots,f_E(\mathbf{x})$ are conditionally independent given an input

$\mathbf{x}$. However, within the same dimension, the function values are still fully jointly Gaussian (Deisenroth 2010). Hence, we can write the predictive posterior distribution of $f(\mathbf{x}_*)$ at test input $\mathbf{x}_*$ as a Gaussian with mean and covariance:

$$\boldsymbol{\mu}_* := \mathbb{E}[\mathbf{f}_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*] = \begin{bmatrix} m_{f_1}(\mathbf{x}_*) & \dots & m_{f_E}(\mathbf{x}_*) \end{bmatrix}^\top, \tag{4.17}$$

$$\boldsymbol{\Sigma}_* := \operatorname{cov}(\mathbf{f}_*) = \operatorname{diag}\Big(\begin{bmatrix} \sigma^2_{f_1} & \dots & \sigma^2_{f_E} \end{bmatrix}\Big), \tag{4.18}$$

respectively.

In literature, there also exist other GP methods for modelling vector-valued functions. A good overview can be found in Álvarez, Rosasco, and Neil D. Lawrence 2012. A simple approach is the sum of separable kernels, i.e., the sum of the products between a kernel function for the input space alone and a kernel function that encodes the correlations among the outputs. Linear models of coregionalisation models follow the form of a sum of separable kernels and can be considered as generative approaches for valid covariance functions (Goulard and Voltz 1992). Non-separable kernels can also be constructed from a generative point of view through convolving a base process with a smooth kernel (Hoef and Barry 1998).

Uncertain inputs

Uncertain inputs arise when we like to predict a function value $f(\mathbf{x}_*)$, $f : \mathbb{R}^D \mapsto \mathbb{R}$ with $\mathbf{x}_* \sim p(\mathbf{x}_*)$, i.e., the input $\mathbf{x}_*$ has a probability distribution. One example for this is the multi-step-ahead prediction of a discrete time series (Carl Edward; Rasmussen and Williams 2004) such as later on in this thesis when predicting a trajectory. For iterative one-step-ahead predictions, the uncertainty (i.e. the predictive posterior variance) needs to be propagated forward. In other words, we feed the predictive posterior distribution for a data point $\mathbf{x}_t$ for time step $t = 0, \dots, T-1, T \in \mathbb{N}$ back into the model for predicting the next data point $\mathbf{x}_{t+1}$. For both univariate and multivariate predictions we assume that $f \sim \mathcal{GP}(\mathbf{0}, k_{SE}(\cdot, \cdot))$. The predictive posterior distribution is obtained by integrating over $\mathbf{x}_*$ such that

$$p(f(\mathbf{x}_*)|\boldsymbol{\mu}_{\mathbf{x}_*}, \boldsymbol{\Sigma}_{\mathbf{x}_*}) = \int p(f(\mathbf{x}_*)|\, \mathbf{x}_*, \mathbf{X}, \mathbf{y}) p(\mathbf{x}_*)\, \mathrm{d}\mathbf{x}_*, \tag{4.19}$$

where the on the training data conditioned distribution $p(f(\mathbf{x}_*)|\, \mathbf{x}_*, \mathbf{X}, \mathbf{y})$ is Gaussian (Girard et al. 2002) but the predictive posterior distribution $p(f(\mathbf{x}_*)|\boldsymbol{\mu}_{\mathbf{x}_*}, \boldsymbol{\Sigma}_{\mathbf{x}_*})$ is not Gaussian and therefore not computable. The reason for that is that the mapping of the Gaussian distribution $\mathbf{x}_*$ through a non-linear function (here, the GP) leads to a "complicated" non-Gaussian predictive distribution Deisenroth 2010; Girard et al. 2002. One solution for this intractability is to approximate the posterior by an analytical Gaussian approximation that possesses the same mean and variance (Deisenroth 2010; Girard et al. 2002; Kuss 2006; Quinonero-Candela, Girard, and C. Rasmussen 2003).

Univariate predictions Inputs $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \in \mathbb{R}^D$, Outputs $y_* \in \mathbb{R}$
The following results are adopted from (Deisenroth 2010). Since we use the SE kernel, we can compute the approximation of mean μ_* and variance σ^2_* in closed-form by using the law of iterated expectations (Fubini's theorem). We start with computing the first moment $m_1$of Equation (4.19) by taking the expectation over the posterior mean (Kuss 2006)

$$\begin{aligned} m_1 := \mu_* = \int\int f(\mathbf{x}_*) p(f, \mathbf{x}_*)\, \mathrm{d}(f, \mathbf{x}_*) &= \mathbb{E}_{\mathbf{x}_*, f}[f(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\ &= \mathbb{E}_{\mathbf{x}_*}[\mathbb{E}_{f(\mathbf{x}_*)}[f(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{\mathbf{x}_*}[m_f(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}], \end{aligned} \tag{4.20}$$

such that we rewrite the predictive mean as a linear combination of the covariance between the new $\mathbf{x}_*$ and the training inputs,

$$\mathbb{E}_{\mathbf{x}_*}[m_f(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int m_f(\mathbf{x}_*) \mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma})\, \mathrm{d}\mathbf{x}_* \tag{4.21}$$

$$= \sum_j^n \beta_j \int k(\mathbf{x}_*, \mathbf{x}_j) \mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma})\, \mathrm{d}\mathbf{x}_* = \boldsymbol{\beta}^\top \mathbf{q}, \tag{4.22}$$

where $\mathbf{q} = [q_1, \ldots, q_n]^\top \in \mathbb{R}^n$ with

$$q_i := \int k(\mathbf{x}_i, \mathbf{x}_*)\mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,\mathrm{d}\mathbf{x}_* \tag{4.23}$$

$$= \sigma_f^2 |\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu})^\top(\boldsymbol{\Sigma} + \boldsymbol{\Lambda})^{-1}(\mathbf{x}_i - \boldsymbol{\mu})\Big), \tag{4.24}$$

corresponding to the standard SE kernel $k(\mathbf{x}_i, \boldsymbol{\mu})$ which has been ”inflated” by $\boldsymbol{\Sigma}$ (Deisenroth 2010) and using the integral identity of multiplying two Gaussians. Intuitively speaking, each q_i is an expectation of $k(\mathbf{x}_i, \mathbf{x}_*)$, the covariance between $f(\mathbf{x}_i)$ and $f(\mathbf{x}_*)$, over the probability distribution $\mathbf{x}_*$. The variance of the predictive distribution σ_*^2, the second central moment, can by decomposed into three expectations:

$$\sigma_*^2 = \mathbb{V}_{\mathbf{x}_*, f}[f(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{\mathbf{x}_*}[\mathbb{V}_f[f(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] + \mathbb{V}_{\mathbf{x}_*}[\mathbb{E}_f[f(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \tag{4.25}$$

$$= \mathbb{E}_{\mathbf{x}_*}[\sigma_f^2(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] + \mathbb{E}_{\mathbf{x}_*}[m_f(\mathbf{x}_*)^2|\boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mathbb{E}_{\mathbf{x}_*}[m_f(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}]^2, \tag{4.26}$$

where we used $m_f(\mathbf{x}_*) = \mathbb{E}_f[f(\mathbf{x}_*)|\mathbf{x}_*]$ and $\sigma_f^2(\boldsymbol{x}_*) = \mathbb{V}_f[f(\mathbf{x}_*)|\mathbf{x}_*]$. Starting with the first expectation in Equation (4.26), we get

$$\mathbb{E}_{\mathbf{x}_*}[\sigma_f^2(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int \sigma_f^2(\mathbf{x}_*)\mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,\mathrm{d}\mathbf{x}_* \tag{4.27}$$

$$= \sigma_f^2 - \int k(\mathbf{x}_*, \mathbf{x}_*)^\top(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}k_f(\mathbf{x}_*, \mathbf{x}_*)\mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,\mathrm{d}\mathbf{x}_* \tag{4.28}$$

$$= \sigma_f^2 - \mathrm{tr}\big((\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{L}\big), \tag{4.29}$$

where the elements of the matrix $\mathbf{L} \in \mathbb{R}^{M \times M}$ are given by

$$\mathrm{L}_{ij} = \frac{k(\mathbf{x}_i, \mu)k(\mathbf{x}_j, \mu)}{\sqrt{|2\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1} + \mathbf{I}|}} \exp\Big((\mathbf{z}_{ij} - \boldsymbol{\mu})^\top(\frac{1}{2}\boldsymbol{\Lambda} + \boldsymbol{\Sigma})^{-1}\boldsymbol{\Sigma}\boldsymbol{\Lambda}^{-1}(\mathbf{z}_{ij} - \boldsymbol{\mu})\Big), \tag{4.30}$$

where we again used the expression for the product of two Gaussians (Quinonero-Candela, Girard, and C. Rasmussen 2003) and $\mathbf{z}_{ij} := \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j)$ is an arithmetic average of the i-th and j-th input. The second expectation in Equation (4.26) is obtained by

$$\mathbb{E}_{\mathbf{x}_*}[m_f(\mathbf{x}_*)^2|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int (\boldsymbol{\beta}^\top k(\mathbf{x}_*, \mathbf{x}_*))^2\mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma})\,\mathrm{d}\mathbf{x}_* = \boldsymbol{\beta}^\top\mathbf{L}\boldsymbol{\beta}, \tag{4.31}$$

which depends on matrix $\mathbf{L}$ as defined in Equation. 4.30. The third term in Equation (4.26) comes simply in the form of the squared value of the first moment m_1 Equation (4.22). Thus, putting all together, the analytical expression for the variance (the second moment) is

$$m_2 = \underbrace{\sigma_f^2 - \mathrm{tr}\big((\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{L}\big)}_{\mathbb{E}_{\mathbf{x}_*}[\mathbb{V}_f[f(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}]} + \underbrace{\boldsymbol{\beta}^\top\mathbf{L}\boldsymbol{\beta} - m_1^2}_{\mathbb{V}_{\mathbf{x}_*}[\mathbb{E}_f[f(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}]}. \tag{4.32}$$

Multivariate predictions Inputs $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \in \mathbb{R}^D$, Outputs $y_* \in \mathbb{R}^E$
The following results are adopted from (Deisenroth 2010). Since we model all output dimensions independently, the predictive mean vector $\boldsymbol{\mu}_*$ of $p(f(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the collection of all E independently predicted means computed according to Equation (4.22):

$$\boldsymbol{\mu}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma} = \Big[\boldsymbol{\beta}_1^\top \boldsymbol{q}_1 \ \ldots \ \boldsymbol{\beta}_E^\top \boldsymbol{q}_E\Big]^\top, \tag{4.33}$$

where the vectors $\boldsymbol{q}_i$ for all target dimensions $i = 1, \ldots, E$ are given by Equation (4.24). In comparison to the predictions at deterministic inputs, the target dimensions of uncertain input predictions covary. That means that the predictive covariance matrix

$$\boldsymbol{\Sigma}_*|\ \boldsymbol{\mu}, \boldsymbol{\Sigma} = \begin{bmatrix} \mathbb{V}_{f,\mathbf{x}_*}[f_1^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] & \ldots & \mathrm{cov}_{f,\mathbf{x}_*}[f_1^*, f_E^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\ \vdots & \ddots & \vdots \\ \mathrm{cov}_{f,\mathbf{x}_*}[f_E^*, f_1^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] & \ldots & \mathbb{V}_{f,\mathbf{x}_*}[f_E^*|\ \boldsymbol{\mu}, \boldsymbol{\Sigma}] \end{bmatrix} \tag{4.34}$$

is no longer diagonal, where $f_a(\mathbf{x}_*)$ abbreviates $f_a^*, a \in \{1, \dots, E\}$. The diagonal variances are the predictive variances of the individual target dimensions given in Equation (4.32). The cross-covariances are given by

$$\text{cov}_{f,\mathbf{x}_*}[f_a^*, f_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{f,\mathbf{x}_*}[f_a^* f_b^*|\ \boldsymbol{\mu}, \boldsymbol{\Sigma}] - \boldsymbol{\mu}_a^* \boldsymbol{\mu}_b^*. \tag{4.35}$$

Plugging in $p(\mathbf{x}_*) = \mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we obtain

$$\mathbb{E}_{f,\mathbf{x}_*}[f_a^* f_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{\mathbf{x}_*}[\mathbb{E}_{f_a}[f_a^*|\ \boldsymbol{\mu}_x]\mathbb{E}_{f_b}[f_b^*|\mathbf{x}_*]|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int m_f^a(\mathbf{x}_*) m_f^b(\mathbf{x}_*) p(\mathbf{x}_*)\, \mathrm{d}\mathbf{x}_*, \tag{4.36}$$

due to the conditional independence of f_a and f_b given $\mathbf{x}_*$. Since we can write the mean function m_f^a as

$$m_f^a(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{X}) \underbrace{(\mathbf{K}_a + \sigma_{f_a}^2 \mathbf{I})^{-1} \mathbf{y}_a}_{=:\boldsymbol{\beta}_a}, \tag{4.37}$$

according to Equation (4.15), we can then rewrite Equation (4.36) as

$$\begin{aligned}
\mathbb{E}_{f,\mathbf{x}_*}[f_a^* f_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] &\overset{(4.36)}{=} \int m_h^a(\mathbf{x}_*) m_f^b(\mathbf{x}_*) p(\mathbf{x}_*)\, \mathrm{d}\mathbf{x}_* && (4.38)\\
&\overset{(4.37)}{=} \int \underbrace{k^a(\mathbf{x}_*, \mathbf{X})\boldsymbol{\beta}_a}_{\in\mathbb{R}} \underbrace{k^b(\mathbf{x}, \mathbf{X})\boldsymbol{\beta}_b}_{\in\mathbb{R}} p(\mathbf{x}_*)\, \mathrm{d}\mathbf{x}_* && (4.39)\\
&= \boldsymbol{\beta}_a^\top \underbrace{\int k_f^a(\mathbf{x}_*, \mathbf{X})^\top k_f^b(\mathbf{x}_*, \mathbf{X}) p(\mathbf{x}_*)\, \mathrm{d}\mathbf{x}_*\, \boldsymbol{\beta}_b}_{=:\boldsymbol{Q}}, && (4.40)
\end{aligned}$$

by arranging the inner products to pull terms out of the integral that are independent of the test input $\mathbf{x}_*$. The elements of the matrix $\mathbf{Q}$ are given by

$$\begin{aligned}
Q_{ij} &= \sigma_{f,a}^2 \sigma_{f,b}^2 |(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1})\boldsymbol{\Sigma} + \mathbf{I}|^{-\frac{1}{2}}\\
&\quad \times \exp\Big(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^\top (\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}(\mathbf{x}_i - \mathbf{x}_j)\Big)\\
&\quad \times \exp\Big(-\frac{1}{2}(\hat{\mathbf{z}}_{ij} - \boldsymbol{\mu})^\top ((\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1})^{-1} + \boldsymbol{\Sigma})^{-1}(\hat{\mathbf{z}}_{ij} - \boldsymbol{\mu})\Big), && (4.41)\\
\hat{\mathbf{z}_{ij}} &:= \boldsymbol{\Lambda}_b(\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}\mathbf{x}_i + \boldsymbol{\Lambda}_a(\boldsymbol{\Lambda}_a + \boldsymbol{\Lambda}_b)^{-1}\mathbf{x}_j. && (4.42)
\end{aligned}$$

We can obtain an equivalent but numerically relatively stable expression by defining $\mathbf{R} := \boldsymbol{\Sigma}(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1}) + \mathbf{I}, \boldsymbol{\zeta}_i := (\mathbf{x}_i - \boldsymbol{\mu})$ and $\mathbf{z}_{ij} := \boldsymbol{\Lambda}_a^{-1}\boldsymbol{\zeta}_i + \boldsymbol{\Lambda}_b^{-1}\boldsymbol{\zeta}_j$ such that we can rewrite

$$\begin{aligned}
Q_{ij} &= \frac{k_a(\mathbf{x}_i, \boldsymbol{\mu}) k_b(\mathbf{x}_j, \boldsymbol{\mu})}{\sqrt{|\mathbf{R}|}} \exp\big(\frac{1}{2}\mathbf{z}_{ij}^\top \mathbf{R}^{-1}\boldsymbol{\Sigma}\mathbf{z}_{ij}\big) = \frac{\exp(n_{ij}^2)}{\sqrt{|\mathbf{R}|}}, && (4.43)\\
n_{ij}^2 &= 2(\log(\sigma_{f,a}) + \log(\sigma_{f,b})) - \frac{\boldsymbol{\zeta}_i^\top \boldsymbol{\Lambda}_a^{-1}\boldsymbol{\zeta}_i + \boldsymbol{\zeta}_j^\top \boldsymbol{\Lambda}_b^{-1}\boldsymbol{\zeta}_j - \mathbf{z}_{ij}^\top \mathbf{R}^{-1}\boldsymbol{\Sigma}\mathbf{z}_{ij}}{2}, && (4.44)
\end{aligned}$$

where $k_a = \exp(\log(k_a))$ and $k_b = \exp(\log(k_b))$. We emphasise that $\mathbf{Q}$ in Equation (4.43) equals $\mathbf{L}$ in Equation (4.30) for identical target dimensions $a = b$.

In conclusion, following (Deisenroth 2010), the entries of the covariance matrix of the predictive posterior distribution are

$$\text{cov}[f_a^*, f_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}] = \begin{cases} \boldsymbol{\beta}_a^\top \mathbf{Q}\boldsymbol{\beta}_b - \mathbb{E}_{f,\mathbf{x}_*}[f_a^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}]\mathbb{E}_{f,\mathbf{x}_*}[f_b^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}], & a \neq b\\ \boldsymbol{\beta}_a^\top \mathbf{Q}\boldsymbol{\beta}_a - \mathbb{E}_{f,\mathbf{x}_*}[f_a^*|\boldsymbol{\mu}, \boldsymbol{\Sigma}]^2 + \sigma_f^2 - \text{tr}\big((\mathbf{K}_a + \sigma_n^2\mathbf{I})^{-1}\mathbf{Q}\big), & a = b. \end{cases} \tag{4.45}$$

The term $\sigma_f^2 - \text{tr}\big((\mathbf{K}_a + \sigma_n^2\mathbf{I})^{-1}\mathbf{Q}\big)$ equals zero for $a \neq b$ due to the assumption that the target dimensions a and b are conditionally independent given the input.

4.1.4 Input-Output Covariance

As seen later in section, we sometimes need to compute the *cross-covariance* $\mathbf{\Sigma}_{\mathbf{x}_*,f}$ between a test input $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$ and the corresponding predicted function value $f(\mathbf{x}_*) \sim \mathcal{N}(\boldsymbol{\mu}_*, \mathbf{\Sigma}_*)$. To illustrate this, according to (Deisenroth 2010), we assume we like to compute the joint distribution between two given marginal distributions of $\mathbf{x}_*$ and $f(\mathbf{x}_*)$ as

$$p(\mathbf{x}_*, f(\mathbf{x}_*)|\boldsymbol{\mu}, \mathbf{\Sigma}) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_*\end{bmatrix}, \begin{bmatrix}\mathbf{\Sigma} & \mathbf{\Sigma}_{x_*,f_*}\\ \mathbf{\Sigma}_{x_*,f_*}^\top & \mathbf{\Sigma}_*\end{bmatrix}\right), \tag{4.46}$$

where $\mathbf{\Sigma}$ and $\mathbf{\Sigma}_*$ is known. The missing piece is the cross-covariance matrix

$$\mathbf{\Sigma}_{x_*,f_*} := \text{cov}_{\mathbf{x}_*,f_*}[\mathbf{x}_*, f_a(\mathbf{x}_*)|\boldsymbol{\mu}, \mathbf{\Sigma}] \tag{4.47}$$

$$= \mathbb{E}_{\mathbf{x}_*,f}[\mathbf{x}_* f(\mathbf{x}_*)^\top] - \mathbb{E}_{\mathbf{x}_*}[\mathbf{x}_*]\mathbb{E}_{\mathbf{x}_*,f}[f(\mathbf{x}_*)]^\top = \mathbb{E}_{\mathbf{x}_*,f}[\mathbf{x}_* f(\mathbf{x}_*)^\top] - \boldsymbol{\mu}\boldsymbol{\mu}_*^\top. \tag{4.48}$$

In the following, we will break down how to compute this matrix. We start with $\mathbb{E}_{\mathbf{x}_*,f_a}[\mathbf{x}_* f_a(\mathbf{x}_*)|\boldsymbol{\mu}, \mathbf{\Sigma}]$ for each target dimension $a = 1, \dots, E$ as

$$\mathbb{E}_{\mathbf{x}_*,f_a}[\mathbf{x}_* f_a(\mathbf{x}_*)|\boldsymbol{\mu}, \mathbf{\Sigma}] = \mathbb{E}_{\mathbf{x}_*}[\mathbf{x}_* \mathbb{E}_{f_a}[f_a(\mathbf{x}_*)|\mathbf{x}_*]|\boldsymbol{\mu}, \mathbf{\Sigma}] = \int \mathbf{x}_* m_f^a(\mathbf{x}_*) p(\mathbf{x}_*)\,\mathrm{d}\mathbf{x}_* \tag{4.49}$$

$$= \int \mathbf{x}_* \left(\sum_{i=1}^{n} \beta_{a_i} k_f^a(\mathbf{x}_*, \mathbf{x}_i)\right) p(\mathbf{x}_*)\,\mathrm{d}\mathbf{x}_*, \tag{4.50}$$

where the representation of $m_f(\mathbf{x})$ is by means of a finite kernel expansion. Eq. () can be rewritten by pulling all constants (in $\mathbf{x}_*$) out of the integral and swapping summation and integration, such that

$$\mathbb{E}_{\mathbf{x}_*,f_a}[\mathbf{x}_* f_a(\mathbf{x}_*)|\boldsymbol{\mu}, \mathbf{\Sigma}] = \sum_{i=1}^{n} \beta_{a_i} \int \mathbf{x}_* k_f^a(\mathbf{x}_*, \mathbf{x}_i) p(\mathbf{x}_*)\,\mathrm{d}\mathbf{x}_* \tag{4.51}$$

$$= \sum_{i=1}^{n} \beta_{a_i} \int \mathbf{x}_* \underbrace{c_1 \mathcal{N}(\mathbf{x}_*|\mathbf{x}_i, \mathbf{\Lambda}_a)}_{=k_f^a(\mathbf{x}_*,\mathbf{x}_i)} \underbrace{\mathcal{N}(\mathbf{x}_*|\boldsymbol{\mu}, \mathbf{\Lambda})}_{p(\mathbf{x}_*)}\,\mathrm{d}\mathbf{x}_*, \tag{4.52}$$

$$\text{with} \quad c_1^{-1} = \sigma_f^{-2}(2\pi)^{-\frac{D}{2}}|\mathbf{\Lambda}_a|^{-\frac{1}{2}}, \tag{4.53}$$

such that $k_f^a(\mathbf{x}_*, \mathbf{x}_i) = c_1 \mathcal{N}(\mathbf{x}_*|\ \mathbf{x}_i, \mathbf{\Lambda}_a)$ is a normalised Gaussian distribution in the test input $\mathbf{x}_*$, where $x_i, i = 1, \dots, n$ are the training inputs. The product of the two Gaussians in Equation (4.53) results in a unnormalised Gaussian $c_2^{-1}\mathcal{N}(\mathbf{x}_*|\ \boldsymbol{\psi}_i, \mathbf{\Psi})$ with

$$c_2^{-1} = (2\pi)^{-\frac{D}{2}}|\mathbf{\Lambda}_a + \mathbf{\Sigma}|^{-\frac{1}{2}} \exp\big(-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu})^\top(\mathbf{\Lambda}_a + \mathbf{\Sigma})^{-1}(\mathbf{x}_i - \boldsymbol{\mu})\big), \tag{4.54}$$

$$\mathbf{\Psi} = (\mathbf{\Lambda}_a^{-1} + \mathbf{\Sigma}^{-1})^{-1}, \tag{4.55}$$

$$\boldsymbol{\psi}_i = \mathbf{\Psi}(\mathbf{\Lambda}_a^{-1}\mathbf{x}_i + \mathbf{\Sigma}^{-1}\boldsymbol{\mu}). \tag{4.56}$$

To obtain the expected value of the product of the two Gaussians, we obtain

$$\mathbb{E}_{\mathbf{x}_*,f_a}[\mathbf{x}_* f_a(\mathbf{x}_*)|\boldsymbol{\mu}, \mathbf{\Sigma}] = \sum_{i=1}^{n} c_1 c_2^{-1} \beta_{a_i} \boldsymbol{\psi}_i, \qquad a = 1, \dots, E \tag{4.57}$$

$$= \sum_{i=1}^{n} c_1 c_2^{-1} \beta_{a_i} \boldsymbol{\psi}_i - \boldsymbol{\mu}(\mu_*)_a, \qquad a = 1, \dots, E \tag{4.58}$$

We can simplify the the covariance in Equation (4.58) by recognising that $c_1 c_2^{-1} = q_{a_i}$ (see Equation (4.24)) and with $\boldsymbol{\psi}_i = \mathbf{\Sigma}(\mathbf{\Sigma} + \mathbf{\Lambda}_a)^{-1}\mathbf{x}_i + \mathbf{\Lambda}(\mathbf{\Sigma} + \mathbf{\Lambda}_a)^{-1}\boldsymbol{\mu}$. Moving the term $\boldsymbol{\mu}(\mu_*)_a$ into the sum and using the first moment from Equation (4.22) for $(\mu_*)_a$, we obtain:

$$\mathbf{\Sigma}_{x_*,f_*} = \sum_{i=1}^{n} \beta_{a_i} q_{a_i} (\mathbf{\Sigma}(\mathbf{\Sigma} + \mathbf{\Lambda}_a)^{-1}\mathbf{x}_i + (\mathbf{\Lambda}_a(\mathbf{\Sigma} + \mathbf{\Lambda}_a)^{-1} - \mathbf{I})\boldsymbol{\mu}) \tag{4.59}$$

$$= \sum_{i=1}^{n} \beta_{a_i} q_{a_i} (\mathbf{\Sigma}(\mathbf{\Sigma} + \mathbf{\Lambda}_a)^{-1}(\mathbf{x}_i - \boldsymbol{\mu}) + (\mathbf{\Lambda}_a(\mathbf{\Sigma} + \mathbf{\Lambda}_a)^{-1} + \mathbf{\Sigma}(\mathbf{\Sigma} + \mathbf{\Lambda}_a)^{-1} - \mathbf{I})\boldsymbol{\mu}) \tag{4.60}$$

$$= \sum_{i=1}^{n} \beta_{a_i} q_{a_i} \mathbf{\Sigma}(\mathbf{\Sigma} + \mathbf{\Lambda}_a)^{-1}(\mathbf{x}_i - \boldsymbol{\mu}), \tag{4.61}$$

which together with Equation (4.57) determines the joint distribution $p(\mathbf{x}_*, f(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ in Equation (4.46).

4.1.5 Computational complexity

At the time of writing this thesis, the standard methods for inversion of a matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$ (e.g. Gauss-Jordan elimination) require $\mathcal{O}(n^3)$. This assumes that arithmetic with individual elements has complexity $\mathcal{O}(1)$, as is the case with fixed-precision floating-point arithmetic (Knuth 1997).

- Training
  - Scalar targets: The kernel matrix $K \in \mathbb{R}^{n\times n}$ in Equation (4.12) has to be inverted in each gradient step. Thus, the complexity is simply $\mathcal{O}(n^3)$.
  - Multidimensional targets: Since we model each target dimension as an independent GP, for E target dimensions this sums up to $\mathcal{O}(En^3)$ (Deisenroth 2010).
- Predictions
  - Deterministic inputs
    * Univariate targets: Common implementations start with using the Cholesky factorisation of the kernel matrix to obtain $\mathbf{L} :=$ cholesky$(\mathbf{K} + \sigma_n^2\mathbf{I})$ and solve

$$\mu_* = k(\mathbf{x}_*, \mathbf{x}_*)^\top \mathbf{L}^\top \setminus (\mathbf{L} \setminus \mathbf{y}) \tag{4.62}$$

$$\sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) - (\mathbf{L} \setminus k(\mathbf{x}_*, \mathbf{x})^\top k(\mathbf{x}_*, \mathbf{x}), \tag{4.63}$$

      where solving the triangular systems for each test case prediction takes $\mathcal{O}(n^2)$ (Carl Edward; Rasmussen and Williams 2004).
    * Multivariate targets: $\mathcal{O}(En^2)$
  - Uncertain inputs
    * Univariate targets: Computing $\mathbf{L} \in \mathbb{R}^{m\times m}$ in eq (see Equation 4.30) requires a D-dimensional scalar product per entry, summing up to $\mathcal{O}(Dn^2)$ computations, where D is the dimensionality of the training inputs.
    * Multivariate targets: $\mathbf{Q} \in \mathbb{R}^{n\times n}$ in Equation (4.43) has to be computed for each entry of the predictive covariance matrix $\boldsymbol{\Sigma}_* \in \mathbb{R}^{E\times E}$, where E is the dimension of the training targets. $\mathcal{O}(E^2n^2D)$

4.1.6 Sparse GP regression

Large data sets prohibit the usage of standard GPs due to their computational burden as described in the last Section (4.1.5) (Deisenroth 2010; Carl Edward; Rasmussen and Williams 2004). To make GPs more scalable, sparse approximations aim to reduce the computational complexity of hyper-parameter training. Whereas standard GPs compute the inversion of the full $n \times n$ kernel matrix $\mathbf{K}$, sparse GPs aim to find a reduced-rank approximation of the true covariance matrix $\mathbf{K}$, often denoted by $\mathbf{Q}$. That means that they the entries in the approximate covariance matrix $\mathbf{Q}$ refer to covariances of inducing inputs $\bar{\mathbf{X}}_m$ which are either a subset of m training data inputs or a pseudo-training set of fictitious training data. To obtain a sparse GP method, we need to learn $\bar{\mathbf{X}}_m$ and the hyper-parameters $\boldsymbol{\theta}$. $\bar{\mathbf{X}}_m$ can be seen as extra kernel hyper-parameter that parameterises the covariance matrix $\mathbf{Q}$. In the GP community, there exist different approximations of $\bar{\mathbf{X}}_m$, e.g. (Bauer, Wilk, and Carl Edward Rasmussen 2016; Lázaro-Gredilla et al. 2010; Quiñonero-Candela and Carl Edward Rasmussen 2005; Seeger, Williams, and Neil D Lawrence 2003; Titsias 2009).

4.1.7 Variational learning of inducing variables

In this work, we use the variational sparse GP approximation of (Titsias 2009). With this approach, a distance between the exact GP posterior and a variational approximation is minimised. That implies that the inducing inputs $\bar{\mathbf{X}}_m$ become now variational parameters and are learned alongside the kernel parameters by maximising the evidence lower bound. We write the predictive distribution as conditional prior over any finite set of function points $\mathbf{f}_*$ as:

$$p(\mathbf{f}_*|\mathbf{y}) = \int p(\mathbf{f}_*|\mathbf{f})p(\mathbf{f}|\mathbf{y})\,\mathrm{d}\mathbf{f}. \tag{4.64}$$

and by using the joint distribution $p(\mathbf{y}|\mathbf{f})p(\mathbf{f}_*, \mathbf{f}_m, \mathbf{f})$ we write equivalently

$$p(\mathbf{f}_*|\mathbf{y}) = \int p(\mathbf{f}_*|\mathbf{f}_m, \mathbf{f})p(\mathbf{f}|\mathbf{f}_m, \mathbf{y})p(\mathbf{f}_m|\mathbf{y})\,\mathrm{d}\mathbf{f}\,\mathrm{d}\mathbf{f}_m. \tag{4.65}$$

We assume that $\mathbf{f}_m$ is a sufficient statistic for $\mathbf{f}$ such that $\mathbf{f}_*$ and $\mathbf{f}$ are independent given $\mathbf{f}_m$, i.e., $p(\mathbf{f}_*|\mathbf{f}_m, \mathbf{f}) = p(\mathbf{f}_*|\mathbf{f}_m)$. We can then rewrite the above as

$$q(\mathbf{f}_*) := p(\mathbf{f}_*|\mathbf{y}) = \int p(\mathbf{f}_*|\mathbf{f}_m)p(\mathbf{f}|\mathbf{f}_m)\phi(\mathbf{f}_m)\,\mathrm{d}\mathbf{f}\,\mathrm{d}\mathbf{f}_m \tag{4.66}$$

$$= \int p(\mathbf{f}_*|\mathbf{f}_m)\phi(\mathbf{f}_m)\,\mathrm{d}\mathbf{f}_m = \int q(\mathbf{f}_*, \mathbf{f}_m)\,\mathrm{d}\mathbf{f}_m, \tag{4.67}$$

where $q(\mathbf{f}_*) = p(\mathbf{f}_*|\mathbf{y})$, $\phi(\mathbf{f}_m) = p(\mathbf{f}_m|\mathbf{y})$. Since $\mathbf{y}$ is a noisy observation of $\mathbf{f}$ and applying the product and marginalisation rule to

$$p(\mathbf{f}_*|\mathbf{f}_m, \mathbf{y}) = \frac{\int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}_*, \mathbf{f}_m, \mathbf{f})\,\mathrm{d}\mathbf{f}}{\int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}_*, \mathbf{f}_m, \mathbf{f})\,\mathrm{d}\mathbf{f}\,\mathrm{d}\mathbf{f}_*} \tag{4.68}$$

shows that $p(\mathbf{f}|\mathbf{f}_m) = p(\mathbf{f}|\mathbf{f}_m, \mathbf{y})$. By acknowledging the difficulty of finding $\mathbf{f}_m$ that is sufficient close to $\mathbf{f}$, we expect $q(/vect f_*)$ to be only an approximation to $p(\mathbf{f}_*|\mathbf{y})$. Hence, we can choose $\phi(\mathbf{f}_m)$ to be a variational Gaussian distribution, where we accept that $\phi(\mathbf{f}_m) \neq p(\mathbf{f}_m|\mathbf{y})$. This freedom allows us to choose a mean vector $\boldsymbol{\mu}$ and covariance matrix $\mathbf{A}$ such that the general form of the approximate posterior GP distribution is

$$m_f^q(\mathbf{x}_*) = k(\mathbf{x}_*, \bar{\mathbf{X}}_m)k^{-1}(\bar{\mathbf{X}}_m, \bar{\mathbf{X}}_m)\boldsymbol{\mu} \tag{4.69}$$

$$k_f^q(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j) - k(\mathbf{x}_i, \bar{\mathbf{X}}_m)k(\bar{\mathbf{X}}_m, \bar{\mathbf{X}}_m)^{-1}k(\bar{\mathbf{X}}_m, \mathbf{x}_j) + k(\mathbf{x}_i, \bar{\mathbf{X}}_m)\mathbf{B}k(\bar{\mathbf{X}}_m, \mathbf{x}_j), \tag{4.70}$$

with $\mathbf{B} := k(\bar{\mathbf{X}}_m, \bar{\mathbf{X}}_m)^{-1}\mathbf{A}k(\bar{\mathbf{X}}_m, \bar{\mathbf{X}}_m)^{-1}$. This sparse posterior GP is computed in $\mathcal{O}(nm^2)$. Then, (Titsias 2009) introduces a variational method to find the optimal $\phi^*(\mathbf{f}_m) = \mathcal{N}(\mathbf{f}_m|\boldsymbol{\mu}, \mathbf{A})$ by minimising the KL divergence $\mathbb{KL}(q(\mathbf{f}, \mathbf{f}_m) \parallel p(\mathbf{f}, \mathbf{f}_m, \mathbf{y}))$ and treating $\bar{\mathbf{X}}_m, \phi$ as variational parameters. Here, this can be expressed as the maximisation of the following variational lower bound of the true log marginal likelihood:

$$\log p(\mathbf{y}|\mathbf{X}, \bar{\mathbf{X}}_m) = \int p(\mathbf{f}|\mathbf{f}_m)\phi(\mathbf{f}_m)\log\frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}_m)}{\phi(\mathbf{f}_m)}\,\mathrm{d}\mathbf{f}\,\mathrm{d}\mathbf{f}_m, \tag{4.71}$$

where the term $p(\mathbf{f}|\mathbf{f}_m)$ inside the $\log$ cancels out. Analytically solving for the optimal choice of the variational distribution ϕ gives us

$$\log p(\mathbf{y}|\mathbf{X}, \bar{\mathbf{X}}_m) = \log[\mathcal{N}(\mathbf{y}|\mathbf{0}, \sigma_n^2\mathbf{I} + \mathbf{Q}_{nn})] - \frac{1}{2\sigma_n^2}\mathrm{tr}(\mathbf{K}_{nn} - \mathbf{Q}_{nn}), \tag{4.72}$$

$$= -\frac{1}{2}\log\left|\mathbf{Q}_{nn} + \sigma_n^2\mathbf{I}\right| - \frac{1}{2}\mathbf{y}^\top(\mathbf{Q}_{nn} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y} - \frac{n}{2}\log(2\pi) - \frac{1}{2\sigma_n^2}\mathrm{tr}(\mathbf{K}_{nn} - \mathbf{Q}_{nn}), \tag{4.73}$$

where $\mathbf{Q}_{nn} = \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{mn}$ and $\frac{1}{2\sigma_n^2}\mathrm{tr}(\mathbf{K}_{nn} - \mathbf{Q}_{nn})$ can be interpreted as a regularisation term. Further maximisation of the ELBO can be achieved by optimising over $\bar{\mathbf{X}}_m$ by differentiating $\log p(\mathbf{y}|\mathbf{X}, \bar{\mathbf{X}}_m$ with respect to $\phi(\mathbf{f}_m)$ without imposing any constraints, such that

$$\phi^*(\mathbf{f}_m) = \mathcal{N}(\mathbf{f}_m|\, \sigma^{-2}\mathbf{K}_{mm}\boldsymbol{\Sigma}\mathbf{K}_{mn}\mathbf{y}, \mathbf{K}_{mm}\boldsymbol{\Sigma}\mathbf{K}_{mm}), \tag{4.74}$$

with $\boldsymbol{\Sigma} := (\mathbf{K}_{mm} + \sigma^{-2}\mathbf{K}_{mn}\mathbf{K}_{nm})^{-1}$. Now, the variational GP is fully specified and predictions can be made by Equation (4.70).

4.2 Meta reinforcement learning with latent variable models

In this crucial subsection, we introduce the reader to the meta-learning model f_{ML} that lays the foundation for this work. In the previous section, we have learned that in model-based RL, we can approximate the dynamics of a system with a model. In our setting of model-based meta reinforcement learning, we want to expand the model to learn not just the dynamics of one but several related systems. Sæmundsson, Hofmann, and Deisenroth 2018 proposed a Meta RL framework based on a hierarchical latent variable model which infers the relationship between tasks automatically from data. We will now introduce their approach in more detail.

Assume a setting with a potentially infinite number of dynamical systems that are of the same type but with different configurations, i.e., a distribution over dynamical systems with samples $f_p \sim p(f)$ indexed by $p = 1, \ldots, P$ is assumed. We model the distribution over systems using latent embeddings $\mathcal{H} := \{\mathbf{h}_1, \ldots, \mathbf{h}_P\}$ and model the global properties of all systems through a shared function conditioned on the latent embedding

$$\boldsymbol{x}_{t+1} = f(\mathbf{x}_t, \mathbf{c}_t, \mathbf{h}_p) + \boldsymbol{\epsilon}, \tag{4.75}$$

such that the successor state depends on the task-specific latent variable $p(\mathbf{h}_p)$. In the upcoming sections, f_{ML} denotes the meta-learning GP model on the unknown transition function as defined in Equation (4.75). The input to this model is a concatenated state $\tilde{\mathbf{x}} = (\mathbf{x}_t, \mathbf{c}_t, \mathbf{h}_p) \in \mathbb{R}^{D+K+Q}$. The targets are defined by $\mathbf{y}_t = \mathbf{x}_{t+1} - \mathbf{x}_t$. The GP's mean function is specified by $m(\tilde{\mathbf{x}}_t) = \mathbf{0}$, which encodes that a priori the state does not change (Deisenroth, Fox, and Carl Edward Rasmussen 2015). Each dimension of the targets $\mathbf{y}$ is modelled by an independent GP. A Gaussian likelihood is used and defined by

$$p(\mathbf{y}_t \mid \tilde{\mathbf{x}}_t, \boldsymbol{f}(\cdot), \boldsymbol{\theta}) = \mathcal{N}\big(\mathbf{y}_t \mid \boldsymbol{f}(\tilde{\mathbf{x}}_t, \mathbf{E})\big), \tag{4.76}$$

where $\boldsymbol{\theta} = \{\mathbf{E}, \mathbf{L}, \sigma_f^2, Q\}$ are the model hyper-parameters and $\boldsymbol{f}(\cdot) = \big(f^1(\cdot), \ldots, f^D(\cdot)\big)$ denotes a multi-dimensional function. The authors place a standard-normal prior $\mathbf{h}_p \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ on the latent variables $\mathbf{h}_p$. Hence, the full specification of the model is

$$p(\mathbf{Y}, \mathbf{H}, \mathbf{f}(\cdot) | \mathbf{X}, \mathbf{C}) = \prod_{p=1}^{P} p(\mathbf{h}_p) \prod_{t=1}^{T_p} p\big(\mathbf{y}_t | \mathbf{x}_t, \mathbf{c}_t, \mathbf{h}_p, \mathbf{f}(\cdot)\big) p\big(\mathbf{f}(\cdot)\big), \tag{4.77}$$

where $\mathbf{H}$ are the latent variables $\mathcal{H}$ concatenated to a matrix. The corresponding graphical model is given in Figure (4.1).

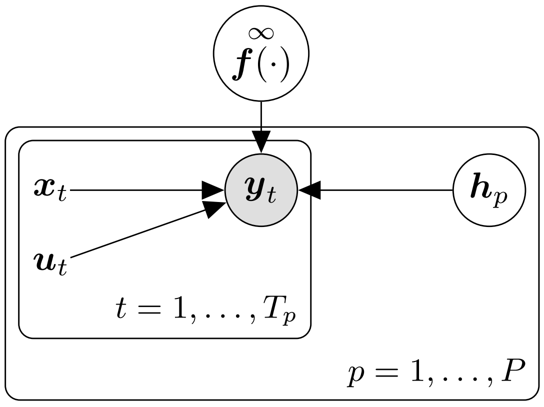


Figure 4.1: The graphical model of the meta-learning model f_{ML}, taken from (Sæmundsson, Hofmann, and Deisenroth 2018).

4.2.1 Variational Inference

For training f_{ML} and inference of the latent variables $\mathcal{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_P)$ for P tasks, we use variational inference. We will briefly detail the model details in this subsection and refer the reader to Section (3.3) for an introduction to VI and to Section (4.1.6) for details on how the variational sparse GP regression works. Both the global function f_{ML} and the latent embeddings $\mathbf{h}_p$ are learned by given trajectory observations $\mathcal{D} = \{(\widetilde{\mathbf{x}}, \mathbf{y})_{1:T}^{P}\}$ from a set of training systems, with inputs and outputs as defined previously. During training of the GP f_{ML} and inferring the train task variables $\mathbf{H}$, the GP hyper-parameters $\boldsymbol{\theta}$ and the variational parameters $\phi = \{\mathbf{Z}, M\{\mathbf{m}^d, \mathbf{S}^d\}_{d=1}^{D}, \{\mathbf{n}_p, \mathbf{T}_p\}_{p=1}^{P}\}$ are jointly optimised w.r.t the ELBO. For inferring the task variable for a test task, Sæmundsson et al. only optimise the variational parameters for the latent variables $\mathbf{h}$, i.e., $\phi_{\mathbf{h}} = \{\mathbf{n}_p, \mathbf{T}_p\}_{p=1}^{P}$. In practice, we use a single sample $\mathbf{h}_p \sim q(\mathbf{h}_p)$ drawn from the variational distribution for each system.

We posit a variational distribution that assumes independence between the latent functions of the GP and the latent task variables

$$Q(\mathbf{f}(\cdot), \mathbf{H}) = q(\mathbf{f}(\cdot))q(\mathbf{H}). \tag{4.78}$$

We can now minimise the Kullback-Leibler divergence between the approximate and true posterior distributions by maximising the ELBO which we denote here with $\mathcal{L}$, i.e.,

$$\mathcal{L} = \mathbb{E}_{Q(\mathbf{f}(\cdot),\mathbf{H})}\Big[\log \frac{p(\mathbf{Y}, \mathbf{H}, \mathbf{f}(\cdot) \mid \mathbf{X}, \mathbf{C})}{Q(\mathbf{f}(\cdot), \mathbf{H})}\Big]. \tag{4.79}$$

More over, we turn to a variational sparse GP approximation (see subsection 4.1.6 or Titsias 2009 for more details) and approximate the posterior GP with a variational distribution $q(\mathbf{f}(\cdot))$ that depends on a small set of $M \ll T$ *inducing points*, where T is the total number of observed time steps across all tasks. There is a set of M inducing inputs $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_M) \in \mathbb{R}^{M \times (D+K+Q)}$, which live in the same space as $\widetilde{\mathbf{x}}$ with corresponding GP function values $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_M) \in \mathbb{R}^{M \times D}$. Following (Hensman, Fusi, and Neil D Lawrence 2013), we specify the variational approximation as a combination of the conditional GP prior and a variational distribution over the inducing function values, independent across output dimensions

$$q(f^d(\cdot)) = \int p(f^d(\cdot) \mid \mathbf{u}^d) q(\mathbf{u}^d)\,\mathrm{d}\mathbf{u}^d, \tag{4.80}$$

where $q(\mathbf{u}^d) = \mathcal{N}(\mathbf{u}^d \mid \mathbf{m}^d, \mathbf{S}^d)$ is a full rank Gaussian distribution with covariance matrix $\mathbf{S}^d$. The above integral can be computed in closed form since both terms are Gaussian and standard results apply here (e.g., one can find them in Deisenroth, Faisal, and Ong 2020), resulting in a GP with mean and covariance by

$$m_q(\cdot) = \mathbf{k}_Z^\top(\cdot)\mathbf{K}_{ZZ}^{-1}\mathbf{m}^d, \tag{4.81}$$

$$k_q(\cdot, \cdot) = k(\cdot, \cdot) - \mathbf{k}_Z^\top(\cdot)\mathbf{K}_{ZZ}^{-1}(\mathbf{K}_{ZZ} - \mathbf{S}^d)\mathbf{K}_{ZZ}^{-1}\mathbf{k}_Z(\cdot), \tag{4.82}$$

with $\mathbf{k}_Z(\cdot)_i = k(\cdot, \boldsymbol{z}_i)$ and $\boldsymbol{K}_{ZZ} = k(\mathbf{z}_i, \mathbf{z}_j)$.

For the latent variables $\mathbf{H}$, we assume a Gaussian variational posterior

$$q(\mathbf{H}) = \prod_{p=1}^{P} \mathcal{N}(\mathbf{h}_p \mid \mathbf{n}_p, \mathbf{T}_p), \tag{4.83}$$

where $\mathbf{T}_p$ is a full rank covariance matrix. In practice, a diagonal covariance matrix is more efficient for computation of the ELBO, which is why we concatenate the diagonal covariance matrix entries to the vectors $\mathbf{h}_1, \ldots, \mathbf{h}_P$. In this work, that means that when we refer to an embedding vector $\mathbf{h}$, that vector fully specifies the embedding's Gaussian random variable. The ELBO can be shown to decompose into

$$\mathcal{L} = \sum_{p=1}^{P}\sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{f}_t \mid \mathbf{x}_t, \mathbf{c}_t)}\big[\log p(\mathbf{y}_t \mid \mathbf{f}_t)\big] - \mathbb{KL}(q(\mathbf{H}) \parallel p(\mathbf{H})) - \mathbb{KL}(q(\mathbf{U}) \parallel p(\mathbf{U})), \tag{4.84}$$

where the expectation is taken with respect to

$$q(\mathbf{f}_t \mid \mathbf{x}_t, \mathbf{c}_t) = \int q(\mathbf{f}_t \mid \mathbf{x}_t, \mathbf{c}_t, \mathbf{h}_p) q(\mathbf{h}_p) d\mathbf{h}_p. \tag{4.85}$$

This integral is intractable due to the nonlinear dependence on $\mathbf{h}_p$. Instead of computing the first and second moments in closed form, the authors decide to approximately integrate out the latent variable using Monte Carlo sampling due to smaller computational complexity.

Chapter 5

Related work

In this chapter, we briefly review related concepts to our approach. Being at the intersection between active learning and meta reinforcement learning, we will present recent ideas in those two communities and discuss how they relate to PATA.

5.1 Active learning

An active learning (AL) setting is one in which the learner has the ability to influence or select its own training data (D. Cohn, Ghahrami, and Jordan 1996). Traditional "passive" learning systems lead to a hypothesis that explains the given training data. In contrast, an active learning system is characterised by eagerly developing and testing new hypotheses as part of a continuous learning process (Settles 2009). Hence, it actively selects which data points to learn. Formally, a mapping $X \mapsto Y$ based on a set of training examples $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{m}$, where $\mathbf{x}_i \in \mathcal{X}$ and $\mathbf{y}_i \in \mathcal{Y}$, is an active learning problem when the learner is allowed to iteratively select new inputs $\mathbf{x}_*$, observe the resulting output $\mathbf{y}_*$ and incorporate the new examples $(\mathbf{x}_*, \mathbf{y}_*)$ into its training set.

Generally speaking, the primary question of active learning is which $\mathbf{x}_*$ to query next. The active learning community differentiates between multiple scenarios in which the learner may be able to query data. The to this work similar scenario is *pool-based sampling* for which large collections of unlabelled data $\mathcal{U}$ can be gathered at once and there is only a small set of labelled data $\mathcal{L}$. The algorithm then requests labels for points in the pool $\mathcal{U}$.

For this work, we consider the AL literature as reference for utility functions we want to incorporate in our framework. Technically speaking, one could frame our generated set of candidate tasks as a pool (we cover how we generate that set later) and categorise our problem as a pool-based learning setting. However, the fundamental difference in our setting is that our pool is dynamic rather than static. After each task acquisition, the inferred relations between the known tasks can change significantly. This is illustrated by an example later on the thesis. For now, we ask the reader to keep in mind that the positions of embeddings in the latent space might change over added tasks. For this work, we borrowed ideas from active learning research; in particular, closed-form heuristics that compute fast and do not require much experience. We present all the concepts from the AL community we have included in this work later in the Chapter (6.1). Apart from these approaches, there have arisen new ideas to tackle typical AL settings. We will quickly review two of them and elaborate on why we have decided to not include them in our framework.

Learning Active Learning from Data Konyushkova, Sznitman, and Fua 2017 as well as (Bachman, Sordoni, and Trischler 2016) suggest a "data-driven approach to active learning". The idea is to train a regression model on how previously acquired candidates reduced the objective model's error in a particular learning state. Thereby, they frame the query selection procedure as a regression problem. One benefit of this approach is to not rely on AL heuristics but learning a tailored selection strategy for the particular underlying data distribution. The authors demonstrate that this approach works well on real data from several different domains.

Apart from the fact that the paper caters to classification problems rather than regression problems (this issue might have been solved by some model changes), additional reasons why this approach is not applicable here are two-fold: Firstly, this approach assumes a static distribution of the candidate pool it aims to select candidates from. This assumption does not hold in our setting, as explained above. Secondly, another implication of the dynamic latent space, candidates belonging to configurations similar to those already learned do not improve the model as significantly as they were previously when they were not solved. That means a model that aims to determine which structure in candidates is essential for improving another model's error and which are not, can not be applied here.

Framing AL as RL problem Following (Fang, Li, and T. Cohn 2017; Konyushkova, Sznitman, Fua, and Al 2019; Pang et al. 2018), the authors formalise AL with a Markov decision process with universal state and action spaces as well as a reward function that motivates an agent to minimise the annotation cost until a certain performance level has been reached. Both papers use deep neural networks to derive a query criterion that is parameterised by embeddings of the respective data set. In comparison to our approach, this procedure is very data-hungry. For instance, in (Konyushkova, Sznitman, Fua, and Al 2019) the RL procedure starts with 100 episodes with random actions. Later on, they perform 1000 RL iterations, each of which comprises 10 AL episodes. In each episode, they actively select 100 data points. In our setting, we consider a new candidate to be a T steps long observed trajectory of a corresponding task configuration. Hence, we argue that applying such an RL approach within this work's scope is infeasible.

5.2 Meta reinforcement learning

Meta reinforcement learning (MRL) has received much attention in recent years. We do not attempt to review MRL techniques in general but to focus on papers in which considerations have been done regarding the active acquisition of tasks.

Unsupervised Meta-Learning for Reinforcement Learning In (Gupta et al. 2018), the authors propose a framework for unsupervised meta-learning for reinforcement learning with the promise of removing the need for manual task design. That means that the paper's approach is constructing a distribution over reward functions, i.e., a mapping from a latent variable $z \sim p(z)$ to a reward function $r_z(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, with $\mathcal{S}$ denoting to states and $\mathcal{A}$ denoting to actions. The authors present a "general recipe" and finally, show an experimental evaluation with an RL algorithm (*not* MRL) that learns from scratch and a meta-RL method that used a hand-designed task distribution. To the best of our understanding, the concrete suggestions in this paper do not solve our problem. Firstly, learning a task distribution is not the same as exploring a task distribution efficiently. We see the need for automatic task distribution construction justified by other reasons such as avoiding a misspecified one.

Secondly, the authors claimed that during experimental evaluation, automating the design burden of the task design through their method exceeded the performance of a RL model learning to solve tasks sampled from a manually-designed distribution from scratch. We argue that a more appropriate baseline would have been a MRL-model that samples tasks from the manually-designed distribution randomly, like in our paper. In fact, such a model had also been evaluated in their experiments but it was claimed that to compare that model with their approach was a "unfair comparison" and it was framed as "upper bound".

Lastly, the paper does not cover task distributions over transition functions. In comparison, we mostly speak about distributions over transition functions in this work, but in principle, the here introduced framework is agnostic to the actual type of task distribution as long as reasonable task embeddings can be inferred. In fact, later on in the experiment section, we demonstrate the attempt to acquire tasks with different reward functions actively.

A Model-based Approach for Sample-efficient Multi-task Reinforcement Learning To the best of our knowledge, the paper published by Landolfi, G. Thomas, and Ma 2017 offers insights into an "active task selection" scheme that is closest to this thesis. The paper's framework assumes a distribution over tasks with different reward functions and attempts to learn a dynamics model during the meta-training allowing for quick adaptation to new tasks at test time with only few samples. Interestingly, the paper then mentions task sampling as one of the critical steps within its sequential multi-task training procedure and suggests some heuristics to choose tasks actively.

Concretely, they suggest two ideas: Firstly, they propose **skipping tasks**, a concept that assumes the setting provides a function $\mu : \mathbf{\Psi} \mapsto \mathbb{R}$ that rates the difficulty of a task configuration. Ideally, this function could be used to focus on tasks which are *hard* and skip tasks that are similar to already learned ones in terms of their difficulty. However, the authors do neither suggest how such a function would be defined in practise nor what *hard* tasks would be. Secondly, **rating tasks** aims to compute the difference between a model's predicted policy reward and the real reward. Later on, they empirically evaluate this method on one RL environment with "modest" results including "vanishing gains" when reaching a higher sampling budget. We found these attempts encouraging to see during our background research but could not adapt any of their methodology to approach our problem.

Chapter 6

Active task acquisition

6.1 Introduction

The goal of PATA is to actively acquire tasks through exploring a latent space that describes task relations. We want to find an efficient way of selecting tasks such that our model generalises fast to all possible tasks within a given range. In the previous sections, we have become acquainted with an introduction of the problem space we are in, probabilistic modelling techniques that allow us to speculate about task dynamics and related work that has not tackled informed task selection methods yet. Based on these preliminaries, we can now formalise the problem more rigorously.

6.2 Problem formulation

We consider a distribution over dynamical systems with samples $f_p \sim p(f)$ indexed by $p = 1, \dots, P$, where each sample f_p has system-specific configurations $\mathbf{\Psi} \subset \mathbb{R}^E$. Let $\mathbf{\Psi}^+$ parameterise a family of admissible configurations that specify the family of admissible transition dynamics $f_{\mathbf{\Psi}^+} : \mathcal{X} \times \mathcal{C} \mapsto \Delta\mathcal{X}$. Starting from an initial state $\mathbf{x}_0 \in \mathbb{R}^D$ and applying control signals $\mathbf{C} \in \mathcal{C} \subset \mathbb{R}^{T \times K}$, $\boldsymbol{\tau} := (\mathbf{x}_0, \dots, \mathbf{x}_T)$ denotes the trajectory of states visited. Based on N pairs of control signals $\mathcal{C} := \{\mathbf{C}_1, \dots, \mathbf{C}_N\}$ and observed trajectories $\mathbf{T} := \{\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_N\}$, let f_{ML} simultaneously learn the transition dynamics for predictions and infer task-specific latent embeddings $p(\mathbf{h}_p)$ for each task, with $\mathcal{H} = \{\mathbf{h}_1, \dots, \mathbf{h}_P\}$. Furthermore, let $\mathcal{L}(\boldsymbol{\tau}_n, \hat{\boldsymbol{\tau}}_n) := \frac{1}{N} \sum_{n=1}^{N} \ell(\boldsymbol{\tau}_n, \hat{\boldsymbol{\tau}}_n)$ refer to a loss function of f_{ML} expressing its predictive performance on a test trajectory $\boldsymbol{\tau}$. Using this notation, we define our objective:

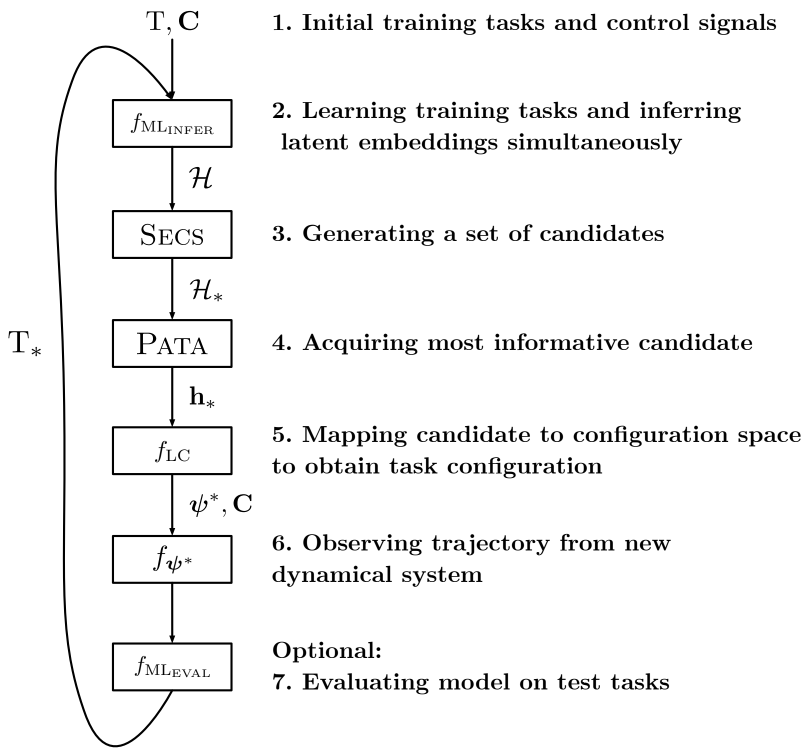
> Without domain knowledge available, we aim to actively select $\mathbf{\Psi}^* \subset \mathbf{\Psi}$ that lead to low losses $\mathcal{L}(\boldsymbol{\tau}_n, \hat{\boldsymbol{\tau}}_n)$ of predicting trajectories $\mathbf{T}_*$ from all admissible task configurations $\mathbf{\Psi}^+$ while minimising $|\mathbf{\Psi}^*|$. Instead of exploring $\mathbf{\Psi}^+$ directly, we want to leverage explicit knowledge about the relations between the previously learned tasks in form of $\mathcal{H}$.

Table 6.1: Overview of the notation used in the subsequent sections.

$\psi \in \Psi$	Task configuration
f_{ψ}	Transition function of dyn. system configured with ψ
$\mathbf{C}$	Set of control signals in matrix form
$\mathbf{x}$	State of a dynamical system
$\boldsymbol{\tau} \in \mathrm{T}$	Observed trajectories
$\mathbf{h} \in \mathcal{H}$	Inferred latent embeddings
$\mathbf{h}_* \in \mathcal{H}_*$	Set of candidates in latent space
$\mathcal{L}(\boldsymbol{\tau}_n, \hat{\boldsymbol{\tau}}_n)$	Loss function of f_{ML}

6.3 High-level perspective

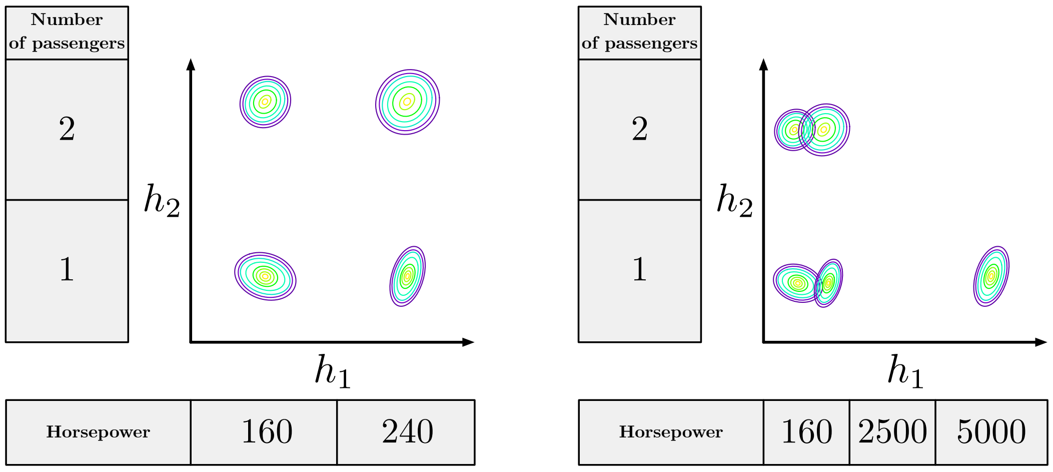
Now that we have formulated the problem, we are ready to present our approach PATA. The upcoming subsections cover the details of the two main mechanisms within PATA: The generation of a candidate set (SECS) as well as the latent candidate selection strategies. Let us quickly consider the steps before and after PATA within the MRL context. Before it starts, step 1 and 2, as noted in Figure (6.1) provide the starting point as detailed in the next subsection. Post-PATA, i.e., after a latent candidate has been selected, the corresponding configuration is obtained by mapping the candidate to configuration space through f_{LC}. In step 6, we observe the trajectory from the actively acquired task configuration. This step is the "expensive" piece of the procedure. Here, we apply control signals on the newly configured transition function from the real world. Step 7 denotes the ability to evaluate the model's performance before repeating. For the sake of brevity, we left out the stopping criterion in the Figure. Furthermore, we want to inform the reader that



T, C
1. Initial training tasks and control signals
$f_{\mathrm{ML}_{\mathrm{INFER}}}$
2. Learning training tasks and inferring latent embeddings simultaneously
$\mathcal{H}$
SECS
3. Generating a set of candidates
$\mathcal{H}_*$
T_*
PATA
4. Acquiring most informative candidate
$\mathbf{h}_*$
f_{LC}
5. Mapping candidate to configuration space to obtain task configuration
$\psi^*, \mathbf{C}$
f_{ψ^*}
6. Observing trajectory from new dynamical system
$f_{\mathrm{ML}_{\mathrm{EVAL}}}$
Optional:
7. Evaluating model on test tasks

Figure 6.1: The procedure of PATA in a nutshell.

already inferred embeddings are dynamic. This brings the benefit of always selecting a task based on information on the current set of learned tasks and their relations. Illustrated in (6.2), the dynamic embeddings motivate the reason why we do not select multiple tasks at once or use non-myopic selection strategies. Instead, we greedily select the most informative task at each iteration and re-train the model and re-infer the embeddings after adding a task.



(a) The latent embeddings of four different car configurations (as stated in the grey-shaded boxes) inferred by observations.

(b) The latent space, i.e., the arrangement of latent embeddings, after observations of a jet with 5000hp were added.

Figure 6.2: The inferred latent embeddings are dynamic and can (dramatically) change after observations of a new task are added. Remember the example from the introduction (1.2), where the model infers embeddings for different cars. Imagine it observes a small private jet taking off and adds those observations to the data set. Note the difference in the distances between tasks assuming that the scale of the fictional graph has not changed. The takeaway here is that the embedding positions changed significantly after the jet observations were added.

$$\begin{tikzcd} \mathrm{T} \arrow[d] & \tilde{\mathbf{x}}_t \arrow[d] & \mathbf{h}_* \arrow[d] \\ \boxed{f_{\mathrm{ML}_{\mathrm{INFER}}}} \arrow[d] & \boxed{f_{\mathrm{ML}_{\mathrm{EVAL}}}} \arrow[d] & \boxed{f_{\mathrm{LC}}} \arrow[d] \\ \mathcal{H} & \Delta\mathbf{x}_{t+1} & \boldsymbol{\psi}_* \end{tikzcd}$$

Figure 6.3: A black-box overview of the both models used in this work with its inputs and outputs. Since f_{ML} can be used in two modes, we show the inputs and outputs for both modes separately.

Algorithm 1 Pata

```
1:  input: $\psi \in \Psi^+, \boldsymbol{x}_0$
2:  sample random control signals $\mathbf{C}$
3:  obtain $\mathbf{T} = \mathbf{T}_{\text{init}}$ with $\mathbf{C}, \boldsymbol{x}_0$ and $f_\psi$    ▷ Generate initial trajectories
4:  train $f_{\text{ML}}$ and infer initial latent embeddings $\mathcal{H} = \mathcal{H}_{\text{init}}$
5:  while the stopping criterion has not been met do
6:      obtain candidates $\mathcal{H}_*$ by optimising latent space region    ▷ initialise model
7:      select most informative candidate in latent space    ▷ depends on selection policy
8:      map selected candidate to configuration space to obtain $\psi_*$
9:      apply C to $f_{\psi_*}$ and yield trajectory $\boldsymbol{\tau}^*$    ▷ expanding dataset
10:     retrain model $f_{\text{ML}}$ with $\mathbf{T} = \mathbf{T} \cup \boldsymbol{\tau}^*$    ▷ update embeddings
11:     optional: evaluate on test tasks    ▷ depends on stopping criterion
12: end while
```

6.4 Warm-up

To actively select new tasks, a regressor f_{LC} learns the mapping between latent and configuration space. Unfortunately, our approach suffers from the *cold-start problem*: The regression function f_{LC} requires initial data to output predictions, i.e., points in the latent space and annotations of the corresponding system configurations.

To obtain these points in latent space, we observe initial trajectories $\mathbf{T}_{\text{training}}$ from either manually chosen or randomly sampled configurations from the domain of admissible configurations. Then, we infer corresponding latent embeddings. Apart from the *quantity* of required initial trajectories, we emphasise that the corresponding task configurations need to incorporate a certain degree of *diversity*. If they are too similar, the meta-learning model f_{ML} struggles to infer distinct embeddings.

6.5 Safe exploration of candidate space (Secs)

To generate task candidates, we use the latent embeddings of the previous step and make assumptions about the infinite latent space. Note that in theory, we can map all possible task configurations (the whole *task distribution*) to that latent space by inference. The other way around is possible too: given some observed task configurations, we can map then inferred embeddings back to the configuration space.

In this work, we explore that space by selecting points out of a grid discretisation of a H-dimensional region $I = I_1 \times \cdots \times I_H$ resulting in a finite set of candidates $\mathcal{U} \in \mathbb{R}^H$. Given that the embeddings come in the form of random variables rather than point estimates, one way to yield $\mathcal{U}$ is to span a grid over the means of the latent embeddings $\mathcal{M} := \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_P\}$ where P is the number of learned embeddings so far. Alternatively, one could draw samples from these distributions and span a grid over these.

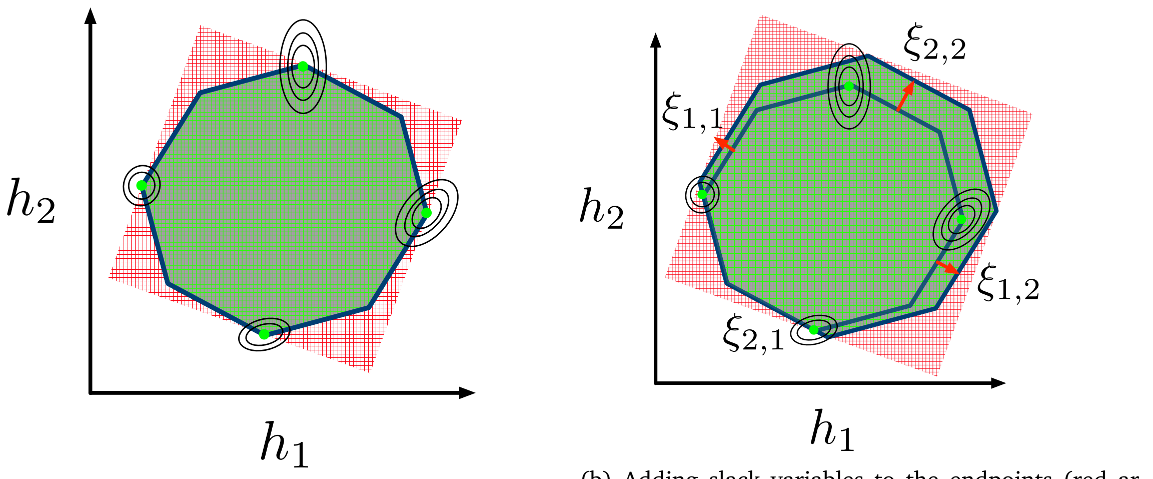
To determine reasonable endpoints of the grid, one could take the minimum and maximum of the set of known embedding means $\mathcal{M}$ for each dimension. However, given the nature of the embeddings, this might limit the task exploration to only considering tasks in-between the known task configurations. A naive solution could be to define the endpoints after gaining some empirical knowledge manually. A more principled approach is to borrow the idea of *slack variables* from support vector machines and add constant values $\boldsymbol{\xi} \in \mathbb{R}^{H \times 2}$ to the endpoints of each axis to allow to increase the distance from known embeddings. One can understand these parameters as a heuristic the practitioner can use to specify the degree of aggressiveness to explore.

Points that lie too distant from the known ones lead to mappings in the configuration space that refer to inadmissible task configurations. Intuitively, the problem of finding good endpoints for the region to be discretised can be interpreted as a classic *exploration vs. exploitation* problem. The region should be large enough to allow for exploration of new admissible configurations but only select safe ones, hence, exploiting the gained knowledge about known embeddings. Here, we relax the condition of having only points that map to admissible points and allow for *invalid* points, i.e., points mapping to infeasible task configurations, within the multi-dimensional region spanned by the grid intervals. However, for a set of endpoints, we compute the mapped configurations for all points in the grid with f_{LC} and then filter points out that have invalid configurations. That said, we still should aim to find a region for which we do not have to reject too many points from membership of the candidate set. Otherwise, we waste many points and lose much of the grid's resolution.

Furthermore, not all mapped, non-rejected task configurations are of 'high quality'. Especially when the regression model f_{LC} is supposed to output configurations for candidate points in latent regions, it has not gained much experience with (i.e., high distance to known task embeddings) yet. It might output points that lie within the admissible configuration boundaries but are somehow degenerate. For example, it outputs the same values plus some noise for every input in a certain region.

For the mapping f_{LC}, we can use a standard GP with an squared exponential kernel, modelling each target dimension independently. We use its predictive variance $\mathbf{\Sigma}_*$ to speculate about the quality of its predicted configurations. One heuristic is to compute the mean of the predictive variance for all predicted configurations and reject the ones for which its predictive variance is higher than on average.

To find a good solution for the endpoints, we frame this trade-off as an optimisation problem and turn to *Bayesian optimisation* as defined in (3.2). This optimisation technique is convenient because we treat the rejection mechanism as a black-box function. An alternative approach to avoid infeasible mappings could be to warp the learned mapping around the admissible configuration region, e.g. by the usage of a warped GP (Lázaro-Gredilla 2012; Rios and Tobar 2019; Snelson, Ghahramani, and Carl E Rasmussen 2004).



(a) A naive approach: The means of the latent embeddings specify the endpoints of the region to be discretised.

(b) Adding slack variables to the endpoints (red arrows) to expand the region of the discretised region, leading to a more diverse candidate pool by accepting points outside of the known embeddings.

Figure 6.4: Different approaches for discretising the latent space. The green dots denote the latent embedding means.

6.5.1 Bayesian optimisation of slack parameters

Let $\mathcal{M} := \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_P\}$ be the set of means of random variables $\mathbf{h} \in \mathcal{H} \subset \mathbb{R}^H$ and $\mathcal{I} := \{\boldsymbol{\mu}_{\min}, \boldsymbol{\mu}_{\max}\}$ with $\mathcal{I} \subset \mathbb{R}^H$ two vectors whose entries are the minimum and maximum value of $\mathcal{M}$ in each of its elements dimensions. Furthermore, consider the set of two slack variables $\Xi := \{\boldsymbol{\xi}_{\min}, \boldsymbol{\xi}_{\max}\}$ that are added to $\mathcal{I}$ in order to expand the region in latent space to be discretised. Let $\mathcal{X} \subset \mathbb{R}^H$ be the set discretised region whose endpoints are determined by $\mathcal{I}$. Analogously, let $\mathcal{X}' \subset \mathbb{R}^H$ be the by the slack variables expanded region whose endpoints are determined by $\mathcal{I}' := \{\boldsymbol{\mu}_{\min} + \boldsymbol{\xi}_{\min}, \boldsymbol{\mu}_{\max} + \boldsymbol{\xi}_{\max}\}$.

For the sake of brevity and with some abuse of mathematical formalism, we define the function $r : \mathbb{R}^{H\times 2} \mapsto \mathbb{N}$ that takes an region interval as input, specified by endpoints such as $\mathcal{I}, \mathcal{I}'$, and returns the number of feasible points within that region, where a feasible point is a point that maps to an admissible task configuration. In other words, this black-box function incorporates the rejection mechanism. Now, we formalise the optimisation problem of finding slack parameters that have maximum absolute-value (L1) norms

$$\underset{\Xi}{\operatorname{argmax}} \ \mathbb{1} \sum_{n=1}^{N} \sum_{d=1}^{D} |\xi|_{n,d} \tag{6.1}$$

$$\mathbb{1} := \begin{cases} 1 & \text{if } \frac{r(\mathcal{X}')}{r(\mathcal{X})} > \alpha, \\ 0 & \text{if } \frac{r(\mathcal{X}')}{r(\mathcal{X})} <= \alpha, \end{cases} \tag{6.2}$$

with an indicator function $\mathbb{1}$ and a threshold hyper-parameter $\alpha \in [0, 1]$. The threshold hyper-parameter allows to influence how wide the boundaries of the space are. Choosing a low value leads to more rejected points but more aggressive exploration. A high value makes the region small.

6.6 Selection of task candidate

There are two classes of selection strategies we introduce in this work: *Model-based* (MB) and *model-free* (MF) tactics. For each of them, we cover concrete *tactics* that are active-learning-inspired ideas to derive utility functions. An utility function is a function $u(\mathbf{h}_*) : \mathbb{R}^D \mapsto \mathbb{R}$ that scores a candidate. In general, given an objective, the better the candidate, the higher the score. We start with the model-free strategy because it is conceptually simpler to understand. After the reader became acquainted with the fundamental principles of acquiring tasks, we move on to the more complex model-based strategy.

6.6.1 Strategy 1: Model-free tactics

They key idea of model-free policies is to select the most informative latent points based on their feature characteristics, independent of the model f_{ML}. That means, we do not touch the model at all when deciding which task to select next. This approach breaks down to ranking all points within the candidate set by either comparing them to each other or using the inferred random variables of known tasks. The fundamental idea of these policies has also been applied to typical active learning settings, see (Yu and Kim 2010).

The main advantage of not requiring any predictions or other interactions with f_{ML} while scoring the candidates is lower computational costs than with a model-based policy. Also, due to its conceptual simplicity, its theoretical background is fairly easy to understand. Note that in principle, when applied in typical pool-based AL settings, model-free selection policies would allow us to spend our whole selection budget $b \in \mathbb{N}^+$ at once. One could formalise the selection of the best set of candidates as a combinatorial optimisation problem since we do not need any information from the model. However, since the latent embeddings of known tasks can change after adding task trajectories from new configurations, we need to sequentially generate new candidates after adding a new task trajectory to our dataset (as described in (6.5)). Therefore, the typical benefit of the unnecessity of updating the model when passively selecting data points does not apply here.

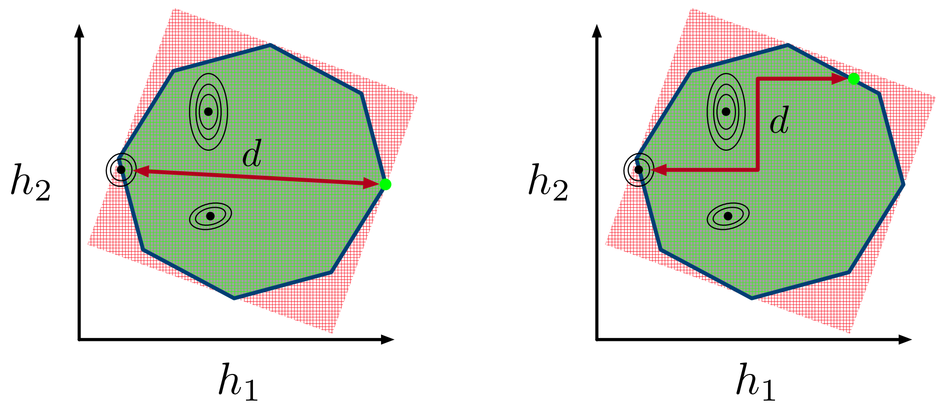
Algorithm 2 PATA with model-free tactics

```
1:  input: $\psi \in \Psi^{+}, \boldsymbol{x}_0, \alpha,$                              ▷ this is a comment
2:  sample random control signals $\mathbf{C}$
3:  obtain $\mathbf{T} = \mathbf{T}_{\text{init}}$ with $\mathbf{C}, \boldsymbol{x}_0$ and $f_\psi$         ▷ Generate initial trajectories
4:  train $f_{\text{ML}}$ and infer initial latent embeddings $\mathcal{H} = \mathcal{H}_{\text{init}}$
5:  $\mathcal{S} = \varnothing$
6:  while the stopping criterion has not been met do
7:      find candidates $\mathcal{H}_*$ by optimising latent space region
8:      for all $\mathbf{h}_* \in \mathcal{H}_*$ do
9:          compute utility $u(\mathbf{h}_*)$ and add score to $\mathcal{S}$
10:     end for
11:     select best candidate $\mathbf{h}_* = \text{argmax}_{\mathbf{h}_*} \mathcal{S}$
12:     apply $\mathbf{C}$ to $f_{\psi_*}$ and yield trajectory $\boldsymbol{\tau}^*$                ▷ expanding dataset
13:     add $\boldsymbol{\tau}^*$ to dataset $\mathbf{T} = \mathbf{T} \cup \boldsymbol{\tau}^*$
14:     retrain model $f_{\text{ML}}$ with $\mathbf{T} = \mathbf{T} \cup \boldsymbol{\tau}^*$                  ▷ update embeddings
15:     optional: evaluate on test tasks                         ▷ depends on stopping criterion
16: end while
```

In the following, we touch upon two different types of utility functions within the model-free strategy. We distinguish them by the nature of how they compute the utility score for a candidate.

Tactic 1.1: Geometric utility functions

We denote *geometric utility functions* as functions that compute the pairwise distance between candidates and select the point with the highest distance to all other candidates in either latent or configuration space. In geometry, there exist various distance functions, such as the *Chebyshev*, *Euclidean*, *Manhattan*, or *Minkowski* distance (M. M. Deza and E. Deza 2009). We derive two



(a) The selected point with highest Euclidean distance to any other candidate.

(b) The selected point with highest Manhattan distance to any other candidate.

Figure 6.5: Two exemplary distance functions illustrating how the selection of the point with the highest distance works.

methods from this idea: (a) greedy sampling the latent space (GSLS) and (b) greedy sampling the configuration space (GSCS) (for typical AL-situations, this idea has been used in the past, e.g., see Wu, Lin, and Huang 2019).

GSLS: Assume we have K embeddings and N candidate points in each iteration, where we select one candidate in an iteration, as described in algorithm (3). Intuitively, the process of GSLS is threefold: (1) We start with computing the distances between each candidate and each known embedding's mean, (2) for each candidate, we compute the shortest distance to any embedding's mean, (3) we select the candidate with the maximum shortest distance. Hence, the utility function a candidate $\mathbf{h}_*$ is defined as

$$u_{\texttt{GSLS}}(\mathbf{h}_*) := \min_{\mathbf{h}_k} \|\mathbf{h}_* - \boldsymbol{\mu}(\mathbf{h}_k)\| \,, \tag{6.3}$$

with $k = 1, \ldots, K$ indexing the known task embeddings.

The idea behind GSCS is very similar with the only difference that we map the candidates into the configuration space and compute the distances between them and the known configurations.

$$u_{\texttt{GSCS}}(\mathbf{h}_*) := \min_{\boldsymbol{\psi}_k} \|f_{\mathrm{LC}}(\mathbf{h}_*) - \boldsymbol{\psi}_k\| \,, \tag{6.4}$$

with $k = 1, \ldots, K$ indexing the known task configurations.

Arguably, one criticism about this tactic might be that it does not need the task embeddings in the first place. If we were to maximise the distances between configurations, why do we not simply discretise the configuration space and greedily sample the most distant points from that set? However, for the sake of good order, we include this tactic as well and let the reader decide in which scenario GSCS could be appropriate to use. Note, that this argument does not hold for GSLS: In practice, we might not know which configuration parameters are essential to consider when learning the relations between tasks. By the usage of f_{ML}, we infer these latent parameters, and by greedily sampling the latent space, we exclusively focus on them, i.e., the most critical variables.

Tactic 1.2: Probabilistic utility functions

Given that the task random variables $\mathcal{H}$ have a probability distribution, we refer to an *probabilistic utility function* one that considers the full distribution to score the candidates. In comparison, in the previous tactic, we only considered the mean of the embeddings.

One approach is to treat the collection of Gaussian distributed latent embeddings as a Gaussian mixture model and choose the candidate that is most unlikely to be sampled from that model [1]. Consider N latent embeddings $\mathcal{H} := \{\mathbf{h}_1, \ldots, \mathbf{h}_N\}$, where each $\mathbf{h}$ has a mean and variance (in practise, we would concatenate the mean and the diagonal entries of the variance, see ()). Each of these task random variables constitutes a mixture component and we equally weight all components. Then, the likelihood of the GMM for a candidate point $\mathbf{h}_*$ is defined by

$$u_{\mathrm{GMM}} := p(\mathbf{h}_*|\boldsymbol{\theta}) = \sum_{n}^{N} \pi_n \mathcal{N}(\mathbf{h}_*|\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n), \tag{6.5}$$

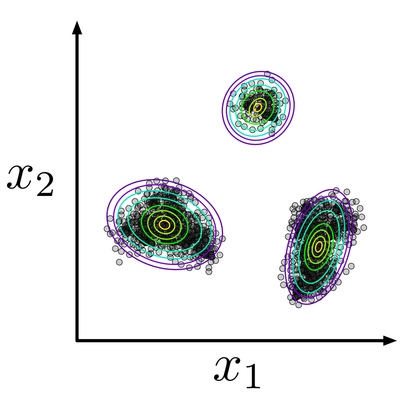
$$\pi_n = \frac{1}{N}, \forall n \in \{1, \ldots, N\}, \tag{6.6}$$

where we denote the collection of all model parameters as $\boldsymbol{\theta} := \{\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n, \pi_n : n = 1, \ldots, N\}$.

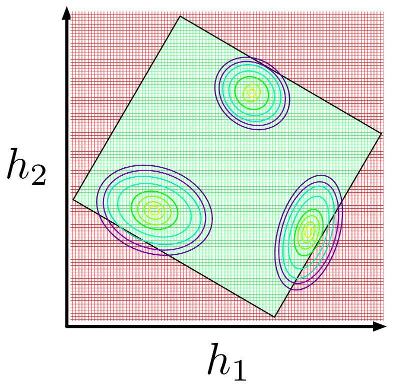
6.6.2 Strategy 2: Model-based policies

Model-based policies use the predictive posterior distribution of the model f_{ML} to score candidates. They use the model to evaluate the quality of a candidate instead of only using the candidates' characteristics as a proxy. In theory, this allows for better-informed decision making than in the model-free strategy: we directly "ask" the to-be-improved model which task trajectories it prefers,

[1] I thank Dr. Marc Peter Deisenroth for pointing this idea out.



(a) A Gaussian mixture model with $K = 3$ components fitted to a two-dimensional dataset. Figure is adapted from (Deisenroth, Faisal, and Ong 2020).



(b) Three latent embeddings treated as GMM with equal-weighted components. The green grid refers to all feasible latent points, the red grid denotes the infeasible candidates.

Figure 6.6: Illustration of Gaussian mixture models for density estimation in Figure (6.6a) and candidate selection in fig (6.6b).

where the preference criterion depends on the tactic. To yield information from the model about a candidate, it needs to be in the same shape as the inputs of the training set.

$$\tilde{\mathbf{x}}_t := \begin{bmatrix}\mathbf{x}_t & \mathbf{c}_t & \mathbf{h}_*\end{bmatrix}^\top \in \mathbb{R}^{D+K+Q}, \tag{6.7}$$

where $\mathbf{h}_*$ is the candidate point in latent space. We assume the same start state $\mathbf{x}_0$ for all trajectories regardless of the task configuration and we assume that we can apply the same control signals from the initial training tasks to generate a trajectory. Naively, we could just put together one input vector for each candidate point with

$$\tilde{\mathbf{x}}_{*,t} := \begin{bmatrix}\mathbf{x}_0 & \mathbf{c}_0 & \mathbf{h}_*\end{bmatrix}^\top, \text{ with } t = 0, \tag{6.8}$$

but this would ignore the how the states evolve over time and how uncertain the model is about the transitions. We hypothesise that letting the model select $\mathbf{h}_*$ simply by $\tilde{\mathbf{x}}_{*,0}$ would lead to a selection mechanism that would prefer candidates most distant to the already inferred ones. Note that we have not thoroughly investigated this further. Instead of the most distant candidate, an approach that would resemble a model-free policy, we want to take into account the configurations' state evolution over time. Thereby, ranking a candidate by taking a trajectory with a small number of steps $T > 10$ into account seems more appropriate. Nonetheless, we have not actually realised any state observations over time, i.e., we miss the states $\mathbf{x}_t$ for $t > 0$.

In theory, we could obtain this short trajectory from the actual dynamical system and still justify this procedure's data efficiency by the relatively small interaction time we would need. However, especially in real world applications, we want to minimise the number of any interactions with a specifically-configured environment since we would assume that this would cause overhead costs like the initial setup or post-interaction dismantling. Hence, we refrain from generating trajectories by interacting with the real system.

Predicting candidate trajectories [2] Remember from (4.1.2) that a GP dynamics model gives us a distribution over all plausible dynamics models that could have generated observations (Deisenroth 2010). Our meta-learning model f_{ML} is such a dynamics model with the extension of task-specific conditioning. Thus, the model is capable of propagating uncertainty about unobserved states forward through time. We can take advantage of that by predicting a short trajectory coherently with the model's uncertainty instead of simply interpolating previous observations. For

[2] I thank Steindor Sæmundsson for pointing this idea out.

a D-dimension state space, we use D independent GPs and denote f_d as the function that maps the input $\tilde{\mathbf{x}}_t$ to the dth dimension of the successor state. For predicting a multi-step trajectory, we cascade one-step predictions yielding predictive state distributions $p(\mathbf{x}_1), \ldots, p(\mathbf{x}_T)$ given by (Deisenroth 2010):

$$p(\mathbf{x}_t) = \int \int p(\mathbf{x}_t | \, \mathbf{x}_{t-1}, \mathbf{c}_{t-1}) p(\mathbf{c}_{t-1} | \mathbf{x}_{t-1}) \, \mathrm{d}\mathbf{x}_{t-1} \, \mathrm{d}\mathbf{c}_{t-1}, \quad t = 1, \ldots, T, \tag{6.9}$$

where the mean and the variance of the Gaussian successor state distribution $p(f_d(\mathbf{x}_*, \mathbf{c}_*, \mathbf{h}_*))$ for a deterministically given triplet of state, action and task variable $\tilde{\mathbf{x}}_* := (\mathbf{x}_*, \mathbf{c}_*, \mathbf{h}_*)$ are given by

$$\mathbb{E}_f[f_d(\tilde{\mathbf{x}}_*) | \tilde{\mathbf{x}}_*] = x_{*d} + \mathbb{E}_f[\Delta x_{*d} | \tilde{\mathbf{x}}_*], \tag{6.10}$$

$$\mathrm{var}_f[f_d(\tilde{\mathbf{x}}_*) | \tilde{\mathbf{x}}_*] = \mathrm{var}_f[\Delta x_{*d} | \tilde{\mathbf{x}}_*], \tag{6.11}$$

$$\Delta x_{id} := f_d(\mathbf{x}_i, \mathbf{c}_i, \mathbf{h}_*) - x_{id}, \quad d = 1, \ldots, D, i = 1, \ldots, n, \tag{6.12}$$

respectively, with $d = 1, \ldots, D$ being the dth dimension of the successor state and Δx_{id} refers to differences between the input state and the next time step, i.e., the targets, for the dth target dimension. We can write the full predictive distribution $p(f(\tilde{\mathbf{x}}_*) | \tilde{\mathbf{x}}_*)$ for all D dimensions

$$\mathcal{N}\left(\begin{bmatrix} \mathbb{E}_f[f_1(\tilde{\mathbf{x}}_*) | \tilde{\mathbf{x}}_*] \\ \vdots \\ \mathbb{E}_f[f_D(\tilde{\mathbf{x}}_*) | \tilde{\mathbf{x}}_*] \end{bmatrix}, \begin{bmatrix} \mathrm{var}_f[f_1(\tilde{\mathbf{x}}_*) | \tilde{\mathbf{x}}_*] & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \mathrm{var}_f[f_D(\tilde{\mathbf{x}}_*) | \tilde{\mathbf{x}}_*] \end{bmatrix} \right), \tag{6.13}$$

with diagonal covariance. Here the deterministically given triplet of state, action and task variable consists of the start state $\mathbf{x}_0$, the first control signal $\mathbf{c}_0$ and the candidate point $\mathbf{h}_*$.

Algorithm 3 PATA with model-based policies

```
 1: input: $\psi \in \Psi^+, x_0, \alpha$,                                  ▷ this is a comment
 2: sample random control signals $\mathbf{C}$
 3: obtain $\mathbf{T} = \mathbf{T}_{\text{init}}$ with $\mathbf{C}, x_0$ and $f_\psi$          ▷ Generate initial trajectories
 4: train $f_{\text{ML}}$ and infer initial latent embeddings $\mathcal{H} = \mathcal{H}_{\text{init}}$
 5: $\mathcal{S} = \varnothing$
 6: while the stopping criterion has not been met do
 7:     find candidates $\mathcal{H}_*$ by optimising latent space region
 8:     for all $\mathbf{h}_* \in \mathcal{H}_*$ do
 9:         cascade one-step predictions $p(\mathbf{x}_t), t = 1, \ldots, T$
10:         concatenate candidate points $\tilde{\mathbf{x}}_{*,t} := \begin{bmatrix} \boldsymbol{\mu}(p(\mathbf{x}_t)) & \mathbf{c}_t & \mathbf{h}_* \end{bmatrix}^\top$
11:         compute utility $u(\tilde{\mathbf{x}}_t)$ for each $t = 1, \ldots, T$
12:         add sum of utilities over all time steps $\mathcal{S} = \mathcal{S} \cup \sum_{t=1}^{T} u(\tilde{\mathbf{x}}_t)$
13:     end for
14:     select best candidate $\mathbf{h}_* = \mathrm{argmax}_{\mathbf{h}_*} \mathcal{S}$
15:     apply $\mathbf{C}$ to $f_{\psi_*}$ and yield trajectory $\boldsymbol{\tau}^*$              ▷ expanding dataset
16:     add $\boldsymbol{\tau}^*$ to dataset $\mathbf{T} = \mathbf{T} \cup \boldsymbol{\tau}^*$
17:     retrain model $f_{\text{ML}}$ with $\mathbf{T} = \mathbf{T} \cup \boldsymbol{\tau}^*$               ▷ update embeddings
18:     optional: evaluate on test tasks                          ▷ depends on stopping criterion
19: end while
```

After predicting the distribution of the state for a time step $p(\mathbf{x}_t)$, we concatenate the mean of it $\boldsymbol{\mu}(p(\mathbf{x}_t))$, the respective control signal $\mathbf{c}_t$ for transitioning into the next state and the candidate point in latent space $\mathbf{h}_*$ to obtain a candidate point $\tilde{\mathbf{x}}_*$ we can now score. For each candidate in latent space $\mathbf{h}_*$, we will compute this score for each time step of the predicted trajectory and then sum up all scores. In the end, we will select the candidate $\mathbf{h}_*$ with the highest sum of scores.

In the following, we discuss different tactics to score candidates and define their respective utility functions $u : \mathbb{R}^{D+K+H} \mapsto \mathbb{R}$ that take an input $\tilde{\mathbf{x}}_{*,t}$ and return a score, where $t = 1, \ldots, T$. When we will speak about a *candidate* or an *input point*, we will refer to $\tilde{\mathbf{x}}_{*,t}$ rather than the whole trajectory or only the candidate in latent space $\mathbf{h}_*$.

Tactic 2.1: Uncertainty sampling

The idea behind uncertainty sampling is to select the input the model is currently most uncertain about. Since f_{ML} is a GP which gives us a predictive distribution for each prediction, the integration of this tactic is straight-forward: The utility function is simply the predictive variance $\sigma_*^2(\mathbf{x}_*)$, as it has been defined in section (4.1.3).

$$u_{\text{VAR}} := \sigma^2_{f_{\text{ML}}}(\mathbf{h}_*) \tag{6.14}$$

An inherent weakness of this approach is that it will always select the boundary values, although, it may be better to do something different, e.g. choosing equally spaced points (Krause 2010). This is illustrated in Figure (6.7).

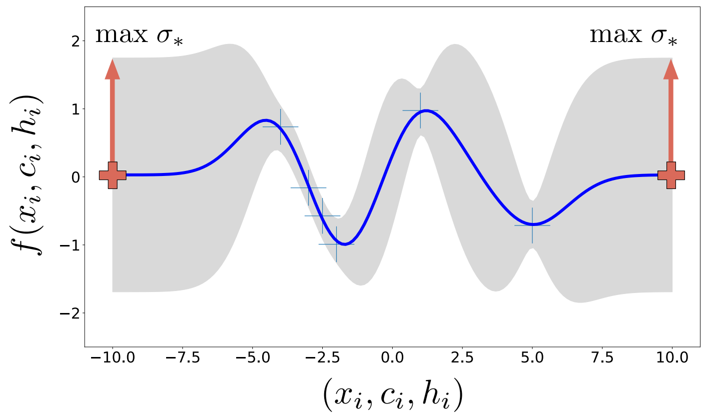


Figure 6.7: Uncertainty sampling with a Gaussian Process. The blue function denotes the mean of the function samples drawn from the posterior distribution. The shaded area represents the model uncertainty, captured by the standard deviation ($\sqrt{\sigma_*^2}$). The blue crosses denote training points, the red plus-symbols the selected candidates.

Tactic 2.2: Expected model change maximisation

Expected model change (EMC) is a framework that aims to select the input that will result in the maximum change of the model parameters (Cai, Zhang, and Zhou 2013). This approach is inspired by the stochastic gradient descent (SGD) update rule which estimates the gradient of the loss function with respect to a *mini-batch* of randomly sampled training examples. Here, we approximate the gradient of the loss function at the candidate inputs.

According to the authors (Cai, Zhang, and Zhou 2013), the theoretical justification of EMC is three-fold:

1. During the active learning phase, the model's capacity can be changed if and only if its parameters are changed.
2. The candidates that change the model parameter's fastest are expected to result in faster convergence to the latent function.
3. Negative effects of selecting outlier inputs are limited. In our context, an outlier input might consist of an abnormal state or an unallowed control signal. Since outliers are rare by definition, if an outlier was selected it is likely that the subsequently chosen input is a representative one, compensating the confusion caused by the outlier candidate quickly.

To derive the framework, we start by introducing SGD as described in (Deisenroth, Faisal, and Ong 2020; Goodfellow, Bengio, and Courville 2016). Given a loss function $\mathcal{L}(\tilde{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the model's parameters, an estimate of the gradient of the loss function with respect to a mini-batch of examples $\mathcal{B} := \{\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_m\}$ of size M is given by

$$\frac{1}{m} \sum_{m=1}^{M} \Big(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\tilde{\mathbf{x}}_i, \mathbf{y}_i, \boldsymbol{\theta}) \Big)^{\top}. \tag{6.15}$$

In order to search for optimal model parameters $\boldsymbol{\theta}^*$, SGD updates $\boldsymbol{\theta}$ iteratively according to that estimated gradient

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} - \gamma \frac{1}{m} \sum_{m=1}^{M} \Big(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\tilde{\mathbf{x}}_i, \mathbf{y}_i, \boldsymbol{\theta}) \Big)^{\top}, \tag{6.16}$$

where γ is the step-size (or *learning rate*) parameter. With that in mind, we can formulate the model change $\mathbf{C}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_*)$ as an approximation of the gradient of the loss function at a candidate instance as:

$$\mathbf{C}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_*) = \Delta \boldsymbol{\theta} \approx \gamma \big(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\tilde{\mathbf{x}}_*, \mathbf{y}, \boldsymbol{\theta}) \big)^{\top}. \tag{6.17}$$

Having demonstrated the general case, we can apply this candidate selection criterion to our model as follows: After converting the already observed trajectories to a N-sized data set $\mathcal{D} = \{(\tilde{\mathbf{x}}_1, \mathbf{y}_1), \dots, (\tilde{\mathbf{x}}_N, \mathbf{y}_N)\}$ with targets $\mathbf{y}$ as the change of the input state after applying the control signal, we can define the overall empirical generalisation error of our model f_{ML} as

$$\hat{\epsilon}_{\mathcal{D}} = \sum_{i=1}^{N} \sum_{e=1}^{E} \mathcal{L}\big(f_{\text{ML}}(\tilde{\mathbf{x}}_i), y_{i,e}\big), \tag{6.18}$$

where we sum up the losses for each of the E target dimensions we model independently and $\mathcal{L}$ can be any loss function suitable for regression problems. Adding a candidate instance consisting of input-output pair $(\tilde{\mathbf{x}}_*, \mathbf{y}_*)$ to the data set $\mathcal{D}' = \mathcal{D} \cup (\tilde{\mathbf{x}}_*, \mathbf{y}_*)$ would update the empirical error to

$$\hat{\epsilon}_{\mathcal{D}'} = \sum_{i=1}^{N} \sum_{e=1}^{E} \mathcal{L}\big(f(\tilde{\mathbf{x}}_i), \mathbf{y}_i\big) + \sum_{e=1}^{E} \mathcal{L}\big(f_{\text{ML}}(\tilde{\mathbf{x}}_*), y_{*,e}\big). \tag{6.19}$$

Since f_{ML} is a GP, we can use its predictive mean and formulate the model's parameters of the training set $\boldsymbol{\beta}$ as well as an updated version $\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*}$ after adding the candidate $\tilde{\mathbf{x}}_*$.

Lemma 6.6.1 (Closed-form model parameters update of GP)**.** *Given GP model parameters $\boldsymbol{\beta}$, we can formulate the model parameters $\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*}$ after adding a new input $\tilde{\boldsymbol{x}}_*$:*

$$\boldsymbol{\beta} := (\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1} \mathbf{y}, \tag{6.20}$$

$$\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*} := \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2 \boldsymbol{I} & \boldsymbol{K}(\boldsymbol{X}, \tilde{\boldsymbol{x}}_*) \\ \boldsymbol{K}(\tilde{\boldsymbol{x}}_*, \boldsymbol{X}) & \boldsymbol{K}(\tilde{\boldsymbol{x}}_*, \tilde{\boldsymbol{x}}_*) \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{y} \\ y_{*,e} \end{bmatrix} \tag{6.21}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \frac{1}{\sigma_*^2 + \sigma_n^2} \begin{bmatrix} (\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{K}(\boldsymbol{X}, \tilde{\boldsymbol{x}}_*) \\ -1 \end{bmatrix} \left(\boldsymbol{K}(\tilde{\boldsymbol{x}}_*, \boldsymbol{X})^{\top} \boldsymbol{\beta} - y_{*,e} \right), \tag{6.22}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \frac{\mu_* - y_*}{\sigma_*^2 + \sigma_n^2} \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})^{-1} \boldsymbol{K}(\tilde{\boldsymbol{x}}_*, \tilde{\boldsymbol{x}}_*) \\ -1 \end{bmatrix}. \tag{6.23}$$

for all target dimensions $e = 1, \dots, E$, respectively.

Proof. The following proof is adapted from (Freytag et al. 2013a) [a]. We define $\mathbf{K} :=$

$k(\mathbf{X}, \mathbf{X}), \mathbf{k}_* := k(\mathbf{X}, \tilde{\mathbf{x}}_*), k_{**} := k(\tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}_*)$.

$$\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*} = \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^\top & k_{**} + \sigma_n^2 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \tag{6.24}$$

$$= \begin{bmatrix} \left(\mathbf{K} - \tfrac{1}{k_{**}+\sigma_n^2}\mathbf{k}_*\mathbf{k}_*^\top\right)^{-1} & -\left(\mathbf{K} - \tfrac{1}{k_{**}+\sigma_n^2}\mathbf{k}_*\mathbf{k}_*^\top\right)^{-1} \tfrac{1}{k_{**}+\sigma_n^2}\mathbf{k}_* \\ -\left(k_{**} + \Sigma_n^2 - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*\right)^{-1} \mathbf{k}_*^\top \mathbf{K}^{-1} & \left(k_{**} + \Sigma_n^2 - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*\right)^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \tag{6.25}$$

$$= \begin{bmatrix} \mathbf{K}^{-1} + \mathbf{K}^{-1}\mathbf{k}_*\left(k_{**} + \sigma_n^2 - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*\right)^{-1} \mathbf{k}_*^\top \mathbf{K}^{-1} \\ -\left(k_{**} + \sigma_n^2 - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*\right)^{-1} \mathbf{k}_*^\top \mathbf{K}^{-1} \end{bmatrix}_{\text{COLUMN 1}} \tag{6.26}$$

$$\begin{bmatrix} -\left(\mathbf{K}^{-1} + \mathbf{K}^{-1}\mathbf{k}_*(k_{**} + \sigma_n^2 - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*)^{-1} \mathbf{k}_*^\top \mathbf{K}^{-1}\right) \tfrac{1}{k_{**}+\sigma_n^2}\mathbf{k}_* \\ \left(k_{**} + \sigma_n^2 - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*\right)^{-1} \end{bmatrix}_{\text{COLUMN 2}} \begin{bmatrix} \mathbf{y} \\ y \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{\beta} + \mathbf{K}^{-1}\mathbf{k}_* \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} - \left(\mathbf{K}^{-1} + \mathbf{K}^{-1}\mathbf{k}_* \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \mathbf{K}^{-1}\right) \tfrac{y_*}{k_{**}+\sigma_n^2}\mathbf{k}_* \\ -\tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} + \tfrac{y_*}{\sigma_*^2+\sigma_n^2} \end{bmatrix} \tag{6.27}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_* \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} - \tfrac{y_*}{k_{**}+\sigma_n^2}\left(\mathbf{K}^{-1} + \mathbf{K}^{-1}\mathbf{k}_* \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \mathbf{K}^{-1}\right)\mathbf{k}_* \\ -\tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} + \tfrac{y_*}{\sigma_*^2+\sigma_n^2} \end{bmatrix} \tag{6.28}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_* \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} - \tfrac{y_*}{k_{**}+\sigma_n^2}\left(\mathbf{K}^{-1} + \mathbf{K}^{-1}\mathbf{k}_* \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \mathbf{K}^{-1}\right)\mathbf{k}_* \\ \tfrac{1}{\sigma_*^2+\sigma_n^2}\left(y_* - \mathbf{k}_*^\top \boldsymbol{\beta}\right) \end{bmatrix} \tag{6.29}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_* \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} - \tfrac{y_*}{k_{**}+\sigma_n^2}\left(\mathbf{K}^{-1}\mathbf{k}_* + \mathbf{K}^{-1}\mathbf{k}_* \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*\right) \\ \tfrac{1}{\sigma_*^2+\sigma_n^2}\left(y_* - \mathbf{k}_*^\top \boldsymbol{\beta}\right) \end{bmatrix} \tag{6.30}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_*\left(\tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} - \tfrac{y_*}{k_{**}+\sigma_n^2}\left(1 + \tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*\right)\right. \\ \tfrac{1}{\sigma_*^2+\sigma_n^2}\left(y_* - \mathbf{k}_*^\top \boldsymbol{\beta}\right) \end{bmatrix} \tag{6.31}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_*\left(\tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*\boldsymbol{\beta} - \tfrac{y_*}{k_{**}+\sigma_n^2}\left(1 + \tfrac{k_{**}-\sigma_*^2}{\sigma_*^2+\sigma_n^2}\right)\right) \\ \tfrac{1}{\sigma_*^2+\sigma_n^2}\left(y_* - \mathbf{k}_*^\top \boldsymbol{\beta}\right) \end{bmatrix} \tag{6.32}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_*\left(\tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} - \tfrac{y_*}{k_{**}+\sigma_n^2}\left(\tfrac{\sigma_*^2+\sigma_n^2}{\sigma_*^2+\sigma_n^2} + \tfrac{k_{**}-\sigma_*^2}{\sigma_*^2+\sigma_n^2}\right)\right) \\ \tfrac{1}{\sigma_*^2+\sigma_n^2}\left(y_* - \mathbf{k}_*^\top \boldsymbol{\beta}\right) \end{bmatrix} \tag{6.33}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_*\left(\tfrac{1}{\sigma_*^2+\sigma_n^2}\mathbf{k}_*^\top \boldsymbol{\beta} - \tfrac{y_*}{\sigma_*^2+\sigma_n^2}\right) \\ \tfrac{1}{\sigma_*^2+\sigma_n^2}\left(y_* - \mathbf{k}_*^\top \boldsymbol{\beta}\right) \end{bmatrix} \tag{6.34}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \frac{1}{\sigma_*^2 + \sigma_n^2} \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_*\left(\mathbf{k}_*^\top \boldsymbol{\beta} - y_*\right) \\ \left(y_* - \mathbf{k}_*^\top \boldsymbol{\beta}\right) \end{bmatrix} \tag{6.35}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \frac{\mathbf{k}_*^\top - y_*}{\sigma_*^2 + \sigma_n^2} \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_* \\ -1 \end{bmatrix} \tag{6.36}$$

$$= \begin{bmatrix} \boldsymbol{\beta} \\ 0 \end{bmatrix} + \frac{\mu_* - y_*}{\sigma_*^2} \begin{bmatrix} \mathbf{K}^{-1}\mathbf{k}_* \\ -1 \end{bmatrix} \tag{6.37}$$

□

[a] I thank Dr. Alexander Freytag for answering my questions, in particular, regarding this proof.

We can now formulate the model change $\Delta\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*}$ as difference between the two above:

$$\Delta\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*} := \begin{bmatrix}\boldsymbol{\beta}\\0\end{bmatrix} - \boldsymbol{\beta}_* = \cancel{\begin{bmatrix}\boldsymbol{\beta}\\0\end{bmatrix}} - \cancel{\begin{bmatrix}\boldsymbol{\beta}\\0\end{bmatrix}} + \underbrace{\frac{1}{\sigma_*^2+\sigma_n^2}}_{\langle 1\rangle}\underbrace{\begin{bmatrix}(\mathbf{K}(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{X},\tilde{\mathbf{x}}_*)\\-1\end{bmatrix}}_{\langle 2\rangle}\underbrace{\left(\mathbf{K}(\tilde{\mathbf{x}}_*,\mathbf{X})^\top\boldsymbol{\beta}-y_{*,e}\right)}_{\langle 3\rangle}, \tag{6.38}$$

$$= \frac{\mathbf{K}(\tilde{\mathbf{x}}_*,\mathbf{X})^\top\boldsymbol{\beta}-y_{*,e}}{\sigma_*^2+\sigma_n^2}\begin{bmatrix}(\mathbf{K}(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{X},\tilde{\mathbf{x}}_*)\\-1\end{bmatrix}. \tag{6.39}$$

According to (Freytag et al. 2013b), the three factors in (6.38) can be interpreted as follows: $\langle 1\rangle$ penalises outliers by enforcing small values of $\Delta\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*}$ for large predictive variances. $\langle 2\rangle$, similarly, can be understood as a weighted Parzen estimate. $\langle 3\rangle$ avoids redundant information to be added, i.e., when the error of the prediction of the target is small, the model change will be small too.

The missing piece to compute $\Delta\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*}$ is the target y_* which we will estimate in the following. We can decompose the former into a factor depending on y_* and a vector $g(\tilde{\mathbf{x}}_*)$ independent of the target:

$$\Delta\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*} = g(\tilde{\mathbf{x}}_*)(\mathbf{K}(\tilde{\mathbf{x}}_*,\mathbf{X})^\top\boldsymbol{\beta}-y_*), \tag{6.40}$$

$$\text{with } g(\tilde{\mathbf{x}}_*) := \frac{1}{\sigma_*^2+\sigma_n^2}\begin{bmatrix}(\mathbf{K}(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{X},\tilde{\mathbf{x}}_*)\\-1\end{bmatrix}. \tag{6.41}$$

Rewriting the definition of the predictive mean of f_{ML} as,

$$\boldsymbol{\mu}_* = [k(\mathbf{X},\cdot), k(\tilde{\mathbf{x}}_*,\cdot)]^\top\boldsymbol{\beta}, \tag{6.42}$$

we can formulate the expectation of change of the predictive mean (not the model parameters) for a new candidate $\tilde{\mathbf{x}}_*$ as $\Delta f_{\mathrm{ML}}(\tilde{\mathbf{x}}_*) := \mathbb{E}_{\tilde{\mathbf{x}}}\mathbb{E}_{y_*|\tilde{\mathbf{x}}_*}\mathcal{L}_P\big(f_{\mathrm{ML}}(\tilde{\mathbf{x}}), f_{\mathrm{ML}}(\tilde{\mathbf{x}}_*)\big)$, where we consider any general P-normed loss function $\mathcal{L}_P\big(f_{\mathrm{ML}}(\tilde{\mathbf{x}}), f_{\mathrm{ML}}(\tilde{\mathbf{x}}_*)\big) = \big\|\big(f_{\mathrm{ML}}(\tilde{\mathbf{x}}), f_{\mathrm{ML}}(\tilde{\mathbf{x}}_*)\big)\big\|_P$. Instead of taking the expectation over all training inputs, we can compute the expectation of change given a single training example $\tilde{\mathbf{x}}$ with respect to the unknown target y_* as:

$$\Delta f_{\mathrm{ML}}(\tilde{\mathbf{x}}_*,\tilde{\mathbf{x}}) = \mathbb{E}_{y_*|\tilde{\mathbf{x}}_*}\left\|[\mathrm{k}(\mathbf{X},\cdot),\mathrm{k}(\tilde{\mathbf{x}}_*,\cdot)]^\top g(\tilde{\mathbf{x}}_*)\big([\mathrm{k}(\mathbf{X},\cdot),\mathrm{k}(\tilde{\mathbf{x}}_*,\cdot)]^\top\boldsymbol{\beta}-y_*\big)\right\|_P \tag{6.43}$$

$$= \left\|\underbrace{[\mathrm{k}(\mathbf{X},\cdot),\mathrm{k}(\tilde{\mathbf{x}}_*,\cdot)]^\top g(\tilde{\mathbf{x}}_*)}_{v}\right\|_P \cdot \mathbb{E}_{y_*|\tilde{\mathbf{x}}_*}\left\|\underbrace{[\mathrm{k}(\mathbf{X},\cdot),\mathrm{k}(\tilde{\mathbf{x}}_*,\cdot)]^\top\boldsymbol{\beta}}_{c}-y_*\right\|_P, \tag{6.44}$$

where the terms v and c are independent of y_* and $\cdot$ denotes a placeholder for an argument of the kernel function. For the expected model change, we are not interested in the change of the model output and we can ignore the constant term $[k(\mathbf{X},\cdot), k(\tilde{\mathbf{x}}_*,\cdot)]^\top$ within v. Leaving that out, summarising the second factor by $z := y_* - c$ and writing the expectation as the general case in form of the Lebesgue integral over all $y_* \in \mathcal{Y}$ brings:

$$\Delta\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*} = \|g(\tilde{\mathbf{x}}_*)\|_P \int_{\mathcal{Y}} \|z\|_P\, p(z+c|\tilde{\mathbf{x}}_*)\,\mathrm{d}z, \tag{6.45}$$

where $p(z+c|\mathbf{x}_*)$ is the GP posterior distribution of $y_* = z+c$ given $\tilde{\mathbf{x}}_*$. Rewriting the posterior as a Gaussian distribution with (predictive) mean $\mu_*(\tilde{\mathbf{x}}_*)$ and variance $\sigma_*^2(\tilde{\mathbf{x}}_*)$ at the candidate input leads to

$$\Delta\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*} = \|g(\tilde{\mathbf{x}}_*)\|_P \int_{\mathcal{Y}} \|z\|_P\, \mathcal{N}\big(z+c|\ \mu(\tilde{\mathbf{x}}_*), \sigma_*^2(\tilde{\mathbf{x}}_*)\big)\,\mathrm{d}z \tag{6.46}$$

$$= \|g(\tilde{\mathbf{x}}_*)\|_P \int_{\mathcal{Y}} \|z\|_P\, \mathcal{N}\big(z|\ \tilde{\mu}(\tilde{\mathbf{x}}_*), \sigma_*^2(\tilde{\mathbf{x}}_*)\big)\,\mathrm{d}z \tag{6.47}$$

$$= \|g(\tilde{\mathbf{x}}_*)\|_P\, \mathbb{E}[\|z\|]_P, \tag{6.48}$$

where we introduced $\tilde{\mu}(\tilde{\mathbf{x}}_*) := \mu(\tilde{\mathbf{x}}_*) - c$ for convenience. The Pth-moment for a Gaussian distribution can be expressed by involving the Kummer's confluent hypergeometric function of the first kind ${}_1F_1(\alpha;\gamma;z)$:

$$\mathbb{E}\big[\,\|\boldsymbol{z}\|\,\big]_P = \sigma_*^2(\tilde{\mathbf{x}}_*) \cdot 2^{\frac{P}{2}} \cdot \frac{\Gamma(\frac{1+P}{2})}{\sqrt[2]{\pi}} \cdot {}_1F_1\bigg(-\frac{P}{2}, \frac{1}{2}, -\frac{1}{2}\bigg(\frac{\widetilde{\mu}(\tilde{\mathbf{x}}_*)}{\sigma_*^2(\tilde{\mathbf{x}}_*)}\bigg)^2\bigg), \tag{6.49}$$

with P being the norm to apply, $\Gamma(\cdot)$ a gamma function and ${}_1F_1(\alpha,\gamma,;z)$ defined as

$$_1F_1(\alpha;\gamma;z) \triangleq \sum_{n=0}^{\infty} \frac{\alpha^{\bar{n}}}{\gamma^{\bar{n}}} \frac{z^n}{n!}, \tag{6.50}$$

with $\alpha^{\bar{n}}, \gamma^{\bar{n}}$ being rising factorials, i.e. $x^{\bar{n}} = x(x+1)(x+2)\dots(x+n-1) = \prod_{k+1}^{n}(x+k-1)$. To compute the confluent hypergeometric function in practise, we use an alternative integral representation:

$$_1F_1(\alpha;\gamma;z) = \frac{\Gamma(b)}{\Gamma(b-a)\Gamma(a)} \cdot \int_0^1 \exp(zt)\; t^{a-1}(1-t)^{b-a-1}dt. \tag{6.51}$$

Putting everything together, we can now formulate the parameters change in closed-form. We will denote this as a corollary because the next tactic we will introduce will fall back on it.

Corollary 6.6.1.1 (Closed-form model parameters change of GP)**.** *Based on the above results, the model parameters change $\boldsymbol{C_\theta}(\tilde{\boldsymbol{x}}_*)$ of the GP-based model f_{ML} after adding $\tilde{\boldsymbol{x}}_*$ to the data set can be computed in closed-form:*

$$u_{EMC}(\tilde{\boldsymbol{x}}_*) := \Delta\boldsymbol{\beta}_{\tilde{\mathbf{x}}_*} = \frac{1}{\sigma_*^2 + \sigma_n^2} \begin{bmatrix} (\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{X},\tilde{\boldsymbol{x}}_*) \\ -1 \end{bmatrix}$$
$$\cdot\, \sigma_*^2(\tilde{\boldsymbol{x}}_*) \cdot 2^{\frac{P}{2}} \cdot \frac{\Gamma(\frac{1+P}{2})}{\sqrt[2]{\pi}} \cdot {}_1F_1\bigg(-\frac{P}{2}, \frac{1}{2}, -\frac{1}{2}\bigg(\frac{\widetilde{\mu}(\tilde{\boldsymbol{x}}_*)}{\sigma_*^2(\tilde{\boldsymbol{x}}_*)}\bigg)^2\bigg) \tag{6.52}$$

Tactic 2.3: Expected model output change maximisation

Expected model output change (EMOC) aims to select candidates that will result in the maximum change of the model outputs (Käding and Mothes 2018). This approach is closely related to the previously introduced tactic (6.6.2) of expected model change with the difference that EMOC does not measure the difference between model parameters but the change of model outputs. The authors (Käding and Mothes 2018) describe the intuition behind that strategy as selecting the candidate that *'shakes the current view on the world the most'*.

As previously seen when we derived EMC, we can compute the expectation of change given a single training example $\tilde{\mathbf{x}}$ with respect to the unknown target y_* as:

$$\Delta f_{\mathrm{ML}}(\tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}) = \mathbb{E}_{y_*|\tilde{\mathbf{x}}_*} \big\|[\mathrm{k}(\mathbf{X},\cdot), \mathrm{k}(\tilde{\mathbf{x}}_*,\cdot)]^\top g(\tilde{\mathbf{x}}_*)\big([\mathrm{k}(\mathbf{X},\cdot), \mathrm{k}(\tilde{\mathbf{x}}_*,\cdot)]^\top \boldsymbol{\beta} - y_*\big)\big\|_P \tag{6.53}$$

$$= \Bigg\|\underbrace{[\mathrm{k}(\mathbf{X},\cdot), \mathrm{k}(\tilde{\mathbf{x}}_*,\cdot)]^\top g(\tilde{\mathbf{x}}_*)}_{v}\Bigg\|_P \cdot \mathbb{E}_{y_*|\tilde{\mathbf{x}}_*} \Bigg\|\underbrace{[\mathrm{k}(\mathbf{X},\cdot), \mathrm{k}(\tilde{\mathbf{x}}_*,\cdot)]^\top \boldsymbol{\beta}}_{c} - y_*\Bigg\|_P, \tag{6.54}$$

where the terms v and c are independent of y_* and $\cdot$ denotes a placeholder for an argument of the kernel function. 7However, previously, we ignored the with the candidate input extended kernel values $[\mathbf{K}(\mathbf{X},\cdot), \mathbf{K}(\tilde{\mathbf{x}}_*,\cdot)]^\top$. Here, we use corollary (6.6.1.1) and multiply it with $[\mathbf{K}(\mathbf{X},\cdot), \mathbf{K}(\tilde{\mathbf{x}}_*,\cdot)]^\top$ such that we arrive at

$$u_{\text{EMOC}}(\tilde{\mathbf{x}}_*) := \Delta f_{\text{ML}}(\tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}) = \frac{[\mathbf{K}(\mathbf{X}, \cdot), \mathbf{K}(\tilde{\mathbf{x}}_*, \cdot)]^\top}{\sigma_*^2 + \sigma_n^2} \begin{bmatrix} (\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})^{-1} \mathbf{K}(\mathbf{X}, \tilde{\mathbf{x}}_*) \\ -1 \end{bmatrix} \cdot \sigma_*^2(\tilde{\mathbf{x}}_*) \cdot 2^{\frac{P}{2}} \cdot \frac{\Gamma(\frac{1+P}{2})}{\sqrt[2]{\pi}} \cdot {}_1F_1\left(-\frac{P}{2}, \frac{1}{2}, -\frac{1}{2} \left(\frac{\widetilde{\mu}(\tilde{\mathbf{x}}_*)}{\sigma_*^2(\tilde{\mathbf{x}}_*)} \right)^2 \right) \tag{6.55}$$

Tactic 2.4: Mutual information maximisation

This tactic follows the idea of selecting an input that possesses the highest mutual information between itself, the remaining candidates and all training inputs. It is inspired by the paper *'Near-Optimal Sensor Placements in Gaussian Processes'* from (Krause, Singh, and Guestrin 2008) who uses the same criterion to place sensors for monitoring spatial phenomena.

We start with introducing mutual information, a measure of the amount of information that one random variable contains about another one (Cover and J. A. Thomas 2006). The reader might be aware of the Pearson correlation coefficient (PCC): it shares the same motivation but the PCC is only captures linear correlations. Mutual information is a more general measure (e.g. allows discrete random variables) and captures non-linear relationships between two random variables.

We start with introducing the formal definition of mutual information. We then show an approximation algorithm for the maximisation of mutual information based on results from (Krause, Singh, and Guestrin 2008). To arrive at this approximation, we establish a link between mutual information and (conditional) entropy and use a standard result of the latter for Gaussian distributed random variables. We then end up with an intuitive utility function.

Definition 6.6.1 (Mutual information)**.** *Consider two discrete random variables X and Y with marginal probability mass functions $p(x), p(y)$ as well as the joint probability mass function $p(x, y)$. The mutual information $\mathbb{I}(X; Y)$ is the relative entropy between the joint distribution $p(x, y)$ and the factored distribution $p(X)p(Y)$:*

$$\mathbb{I}(X; Y) \triangleq \mathbb{KL}(p(X, Y) \parallel p(X)p(Y)) = \sum_{x\ \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \tag{6.56}$$

Consider candidate input points $\tilde{\mathbf{x}}_* \in \tilde{\mathcal{X}}_*$ and training inputs $\tilde{\mathbf{x}} \in \tilde{\mathcal{X}}$, as described in (6.6.2). Although we want to find an utility function for a single candidate point, imagine we had a budget of $b \in \mathbb{N}_+$ points we were to select and $\tilde{\mathcal{X}}_*^* \subset \tilde{\mathcal{X}}_*$ is the optimal set of candidate points with $|\tilde{\mathcal{X}}_*^*| = b$. We refer to $\tilde{\mathcal{X}}_- := (\tilde{\mathcal{X}} \cup \tilde{\mathcal{X}}_*) \backslash \tilde{\mathcal{X}}_*^*$ as the union of training inputs and candidate inputs with exception of the *optimal* candidates. The goal is to find $\tilde{\mathcal{X}}_*$ that captures most of the information over the remaining points $\tilde{\mathcal{X}}_-$, i.e., $\text{argmax}_{\tilde{\mathbf{x}}_*} \mathbb{I}(\tilde{\mathcal{X}}_*^*; \tilde{\mathcal{X}}_-)$. Intuitively, the reader might notice that this feels equivalent to selecting points $\tilde{\mathcal{X}}_*^*$ that maximally reduce uncertainty over the rest of the space. We can formalise this by

$$\underset{\tilde{\mathcal{X}}_*^*}{\text{argmax}}\, \mathbb{H}(\tilde{\mathcal{X}}_-) - \mathbb{H}(\tilde{\mathcal{X}}_- | \tilde{\mathcal{X}}_*^*), \tag{6.57}$$

where $\mathbb{H}(\tilde{\mathcal{X}}_- | \tilde{\mathcal{X}}_*^*)$ is the conditional entropy of $\tilde{\mathcal{X}}_-$ given the candidates $\tilde{\mathcal{X}}_*^*$. We now prove that assumed relationship between entropy and mutual information.

The following results are adapted from (Cover and J. A. Thomas 2006).

Definition 6.6.2 (Conditional entropy)**.** *Consider two discrete random variables X and Y with marginal probability mass functions $p(x), p(y)$ as well as the joint probability mass func-*

tion $p(x, y)$. The conditional entropy $\mathbb{H}(Y|X)$ is defined as

$$\mathbb{H}(Y|X) \triangleq \sum_{x \in \mathcal{X}} p(x)\mathbb{H}(Y|X) \tag{6.58}$$
$$= -\sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y|x) \log p(y|x) \tag{6.59}$$
$$= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y|x) \tag{6.60}$$
$$\tag{6.61}$$

Theorem 6.6.2 (Relationship between entropy and mutual information)**.** *Consider two discrete random variables X and Y with marginal probability mass functions $p(x), p(y)$ as well as the joint probability mass function $p(x, y)$. The mutual information $\mathbb{I}(X; Y)$ is a symmetric measure that can be equivalently expressed with entropies as:*

$$\mathbb{I}(X; Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X) = \mathbb{H}(X) - \mathbb{H}(X|Y). \tag{6.62}$$

Proof.

$$\mathbb{I}(X; Y) = \sum_{x\ \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \tag{6.63}$$
$$= \sum_{x\ \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)} - \sum_{x\ \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y) \tag{6.64}$$
$$= \sum_{x\ \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x)p(y|x) \log p(y|x) - \sum_{x\ \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y) \tag{6.65}$$
$$= \sum_{x\ \in \mathcal{X}} p(x)\Big(\sum_{y \in \mathcal{Y}} p(y|x) \log p(y|x)\Big) - \sum_{y \in \mathcal{Y}} \Big(\sum_{x\ \in \mathcal{X}} p(x, y)\Big) \log p(y) \tag{6.66}$$
$$= \sum_{x\ \in \mathcal{X}} p(x)\mathbb{H}(Y|X) - \sum_{y \in \mathcal{Y}} p(y) \log p(y) \tag{6.67}$$
$$= -\mathbb{H}(Y|X) + \mathbb{H}(Y) = \mathbb{H}(Y) - \mathbb{H}(Y|X). \tag{6.68}$$

The above holds analogously for $\mathbb{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y)$. □

Now, since we like to arrive at an utility function for one single candidate (due to our greedy selection requirement), we take the above (6.57) and pretend that we have already chosen some candidates, collected as $\tilde{\mathcal{X}}_*^*$ with $|\tilde{\mathcal{X}}_*^*| < b$ and we would like to add the next candidate $\tilde{\mathbf{x}}_*$. Hence, we like to maximise

$$\underset{\tilde{\mathbf{x}}_*}{\operatorname{argmax}}\ \mathbb{I}(\tilde{\mathcal{X}}_*^* \cup \tilde{\mathbf{x}}_*; \tilde{\mathcal{X}}_-) - \mathbb{I}(\tilde{\mathcal{X}}_*^*; \tilde{\mathcal{X}}_-) \tag{6.69}$$
$$= \mathbb{H}(\tilde{\mathcal{X}}_*^* \cup \tilde{\mathbf{x}}_*) - \mathbb{H}(\tilde{\mathcal{X}}_*^* \cup \tilde{\mathbf{x}}_* | \tilde{\mathcal{X}}_-^*) - \mathbb{H}(\tilde{\mathcal{X}}_*^*) + \mathbb{H}(\tilde{\mathcal{X}}_*^* | \tilde{\mathcal{X}}_-) \tag{6.70}$$
$$= \mathbb{H}(\tilde{\mathcal{X}}_*^* \cup \tilde{\mathbf{x}}_*) \cancel{- \mathbb{H}(\tilde{\mathcal{X}})} + \mathbb{H}(\tilde{\mathcal{X}}_-^*) \cancel{- \mathbb{H}(\tilde{\mathcal{X}}_*^*)} \cancel{+ \mathbb{H}(\tilde{\mathcal{X}})} \cancel{+ \mathbb{H}(\tilde{\mathcal{X}}_*^*)} \tag{6.71}$$
$$= \mathbb{H}(\tilde{\mathbf{x}}_* | \tilde{\mathcal{X}}_*^*) - \mathbb{H}(\tilde{\mathbf{x}}_* | \tilde{\mathcal{X}}_-^*), \tag{6.72}$$

where $\tilde{\mathcal{X}}_-^* := \tilde{\mathcal{X}}_- \setminus \tilde{\mathbf{x}}_*$ excludes the candidate under consideration. Now, linking this derivation back to our goal to select one single candidate, we just set $\tilde{\mathcal{X}}_*^* = \varnothing$ and pretend we aim to select the first initial candidate. That brings us

$$\underset{\tilde{\mathbf{x}}_*}{\operatorname{argmax}}\, \mathbb{H}(\tilde{\mathbf{x}}_*) - \mathbb{H}(\tilde{\mathbf{x}}_* | \tilde{\mathcal{X}} \setminus \tilde{\mathbf{x}}_*). \tag{6.73}$$

After translating our objective into the difference of two entropies, we can use the following standard result of the entropy of a Gaussian variable Y conditioned on a set of random variables $\mathcal{X}$:

$$\mathbb{H}(Y|A) = \frac{1}{2} \log(2\pi e \sigma^2_{Y|\mathcal{X}}), \tag{6.74}$$

where for GPs, we can interpret $\sigma^2_{Y|\mathcal{X}}$ as the predictive posterior variance, resulting in a closed-form solution. Note that the conditional entropy is a monotonic function of its variance (due to the $\log$). Also, mathematically speaking, the difference of two entropies, as in our objective (6.73), results in the difference of two $\log$ functions which can be rewritten as the $\log$ of the fraction of its arguments, i.e. $\log(A) - \log(B) = \log \frac{A}{B}$. Exploiting these both properties, we compute the score of a candidate by

$$u_{\text{MI}}(\tilde{\mathbf{x}}_*) := \frac{\mathbf{K}(\tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}_*)}{\mathbf{K}(\tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}_*) - \mathbf{K}(\tilde{\mathbf{x}}_*, \tilde{\mathcal{X}} \setminus \tilde{\mathbf{x}}_*)\mathbf{K}(\tilde{\mathcal{X}} \setminus \tilde{\mathbf{x}}_*, \tilde{\mathcal{X}} \setminus \tilde{\mathbf{x}}_*)^{-1}\mathbf{K}(\tilde{\mathcal{X}} \setminus \tilde{\mathbf{x}}_*, \tilde{\mathbf{x}}_*)}. \tag{6.75}$$

6.7 Stopping criteria

Lastly, we propose stopping criteria that determine when the PATA algorithm stops. In the context of meta-learning algorithms, the test time usually starts then. In general, determining when to stop to acquire more information and make a decision is called the *stopping problem*. Research in mathematics offers theories of *optimal stopping*. A famous example of optimal stopping theory is the *secretary problem* in which an administrator wants to hire the best secretary out of sequentially interviewed secretaries (in random order) but can only decide once for each candidate immediately after an interview was conducted (Bearden 2006). Note, that this does not imply that the more information the administrator gathers, the worse she gets in deciding. The problem is more about efficiency and opportunity costs. Similarly, it it is unlikely (but possible) that a badly-selected task decreases the predictive performance on test tasks. In information theory, there even exists an "information never hurts" theorem (Krause, Singh, and Guestrin 2008), implying that more having information is always better. This theorem does not directly apply here, but without theoretical justification, we argue that the more tasks the model learned the dynamics of, the better. Hence, the motivations for stopping criteria stems from the question of when to put the training model in production (or to test time) instead of requesting more trajectories of new tasks which can be costly and time-consuming. We will now propose three criteria that determine stopping:

- **Task budget**: A manually defined task budget is best if it makes no difference if the learner stops prematurely so as not to consume the entire budget. For instance, situations in which the user pays the "price" for interaction time or the number of tasks to acquire upfront are suitable to declare a task budget of $b \in \mathbb{N}^*$ for PATA.
- **Prediction error**: In theory, the model can predict a trajectory of some test tasks every time after it acquired a new task and feed the prediction error back. This "reward signal" informs the user how well the model advanced over an increasing number of observations. Straightforwardly, a user can incorporate such an error metric into the model so that it stops to learn after it provides a decent enough performance. However, note that this criterion seems more suitable in academic settings rather than in real-world environments - since we assume a set of test tasks is given on which we test model. In real-world environments, we typically do not know the tasks the system will face in production.
- **Predictive variance**: In (6.6.2), we introduced model-based selection policies that use the predictive posterior distribution of our meta-learning model f_{ML} to score a candidate. In particular, in variance-based uncertainty sampling (6.14), we derive the model's uncertainty about a candidate task by its posterior variance. One naive approach to determine whether the model has acquired enough tasks could be to define a numeric threshold for the candidates' variances. For instance, we could compute the scores for all tasks and if the maximum value, the mean or the sum of the scores is above the threshold, we acquire another task.

Chapter 7

Experiments

Previously, we derived a framework for actively acquiring tasks. Knowing how to explore the candidate space safely (SECS) as well as being aware of individual tactics falling into model-based or model-free strategies, we will validate these techniques on three continuous control tasks with increasing difficulty: Cart-pole, Cart-double-pole, Cart-triple-pole. Furthermore, we detail multiple limitations we experienced during the experiments to thoroughly prepare a prospect practitioner and propose ideas for future work.

To make the upcoming examinations more applicable to real-world situations, we incorporate the following assumptions into the experimental setup. We do not have domain knowledge nor a task distribution we sample from at hand. However, we know the ranges of possible configurations. We can then "request" a transition function from the real environment specified with the setting we define. We solely focus on exploring the latent space (not directly the configuration space) with the only additional ability to reject infeasible task configurations. We reject a task configuration if its parameter values lie outside the range of feasibility. For instance, an infeasible task is the pendulum with negative values defining its length. However, the system does not know which configuration variables are task-specific and infers that by itself. These assumptions are crucial to ensure transferability to systems with more complex, high-dimensional configuration spaces. Before we start with numerical comparisons, let us formalise three critical questions to evaluate the proposed methodology's performance:

1. What is the *quality* of the tasks acquired? Does PATA significantly outperform a random baseline that samples tasks from the latent space randomly?

2. How crucial is SECS in practice when exploring the latent space? Does it find reasonable regions within the latent space?

3. What are limitations a MRL practitioner has to consider when using PATA?

So that the reader keeps the overview, we include a table of all introduced and evaluated tactics abbreviations:

Table 7.1: Overview of the notation used in the subsequent sections.

Abbr.	Tactic	Strategy	Reference
EMC	Expected model change	MB	6.52
EMOC	Expected model output change	MB	6.55
GMM	Gaussian mixture model	MF	6.6
GSCS	Greedy sampling of configuration space	MF	6.4
GSLS	Greedy sampling of latent space	MF	6.3
MI	Mutual information	MB	6.75
RDN	Random baseline		
VAR	Variance sampling	MB	6.14

7.1 Quality of tasks acquired

This experiment attempts to answer the question of which quality the tasks our methods acquire are. The quality of a task describes by how much the model's prediction error on test tasks decreases. In all experiments, we will start with two randomly sampled configurations and corresponding embeddings. Then, we evaluate how the meta-learning model f_{ML} performs over an increasing number of tasks acquired.

7.1.1 Evaluation metrics

To quantify the prediction errors, we use the root mean squared error (RMSE) and the log-likelihood (LLH) which we defined below. RMSE is a common error metric but since we use probabilistic models, we argue, that the LLH is even more meaningful and should receive special attention. It is not unusual to model standardised data that is centered around 0 with a GP that assumes zero mean. Thas bares the risk of overestimating the model's performance: Even if your GP has no meaningful predictions but is "falling back to the prior", i.e., returning a predictive mean around that zero mean prior, the RMSE could look better than the situation actually is [1]. However, the LLH accounts for this since it considers the predictive variance which would be high in case the model does not know what to predict.

Definition 7.1.1 (Root mean squared error). *The root mean squared error is a measure of the performance of a model on a test set. It describes the differences between the model's predictions and the observed values by the square root of the Euclidean distance between the predictions and the targets:*

$$\mathit{RMSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) := \frac{1}{n} \sum_{n=1}^{N} \sqrt{\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2}, \tag{7.1}$$

with N being the size of the test set.

Definition 7.1.2 (Log-likelihood). *The likelihood models the model's uncertainty of the data, i.e., its probability for the observed values. It depends on the model parameter's $\boldsymbol{\theta}$ and tells us how likely a particular setting of $\boldsymbol{\theta}$ is for the observations $\boldsymbol{x}$ (Deisenroth, Faisal, and Ong 2020):*

$$\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{n} \sum_{n=1}^{N} \log p(y_n | \boldsymbol{x}_n, \boldsymbol{\theta}), \tag{7.2}$$

with N being the size of the test set and y being a single-dimensional target.

[1] I thank Steindór Sæmundsson for making me aware of this.

Remark 7.1.1 (Gaussian log-likelihood). *The log-likelihood of a Gaussian distribution (Deisenroth, Faisal, and Ong 2020), such as the predictive distribution our f_{ML} yields for an input $\tilde{\mathbf{x}}$, can be written as*

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log \mathcal{N}(y_n|\boldsymbol{x}_n^\top \boldsymbol{\theta}, \sigma^2) \tag{7.3}$$

$$= -\sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{(y_n - \boldsymbol{x}_n^\top \boldsymbol{\theta})^2}{2\sigma^2}\Big) \tag{7.4}$$

$$= -\sum_{n=1}^{N} \log \exp\Big(-\frac{-(y_n - \boldsymbol{x}_n^\top \boldsymbol{\theta})^2}{2\sigma^2}\Big) - \sum_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \tag{7.5}$$

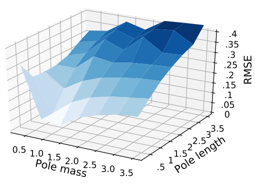
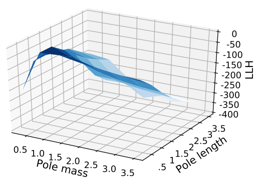
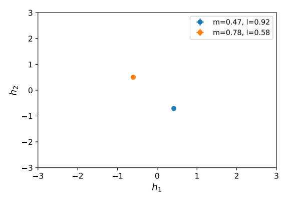
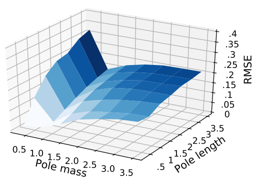
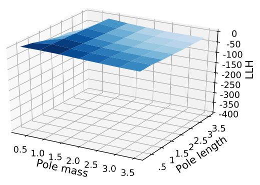
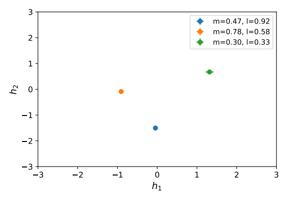
$$= \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \boldsymbol{x}_n^\top \boldsymbol{\theta})^2 - \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}}. \tag{7.6}$$

As one can see, the log-likelihood of a Gaussian resembles the RMSE measure but additionally, it considers the likelihood's variance.

7.1.2 Experimental setup

To analyse the applicability of PATA in common RL benchmarks, we adopted the overall experimental setup from the model-based RL experiment in (Galashov et al. 2019). In particular, the authors propose a distribution over tasks by uniformly sampling the pole mass $p_m \sim \mathcal{U}[0.01, 1.0]$ and cart mass $c_m \sim \mathcal{U}[0.1, 3.0]$. We modified their setup with three tweaks to increase the difficulty in learning the dynamics. Firstly, we empirically noticed that changing the cart mass across tasks is not as impactful in dynamics differences as changing the pole length or pole mass, so we interchanged the cart mass variable with the pole length variable and hold the cart mass constant for all experiments. Furthermore, the authors use the time discretisation parameter $\Delta t = 0.01$ which makes the transitions very smooth and easy to learn [2]. In comparison, we used $\Delta t = 0.2$ for the same cart-pole task and $\Delta t = 0.15$ for cart-double-pole as well as cart-triple-pole. Lastly, we increased the range of possible configurations and chose $[0.3, 3.5]$ for both pole mass and pole length.

To give the reader a better feeling of the experimental setup, i.e., how we observed the model's behaviour over time, we include an exemplary task acquisition sequence in Figure (7.2).

t	RMSE	LLH	Latent space
0			
1			

[2]This parameter is not mentioned in the paper, but I thank the author Jonathan Schwarz who gave me this information on request.

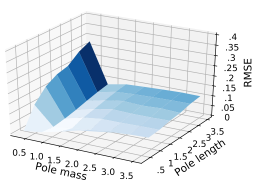
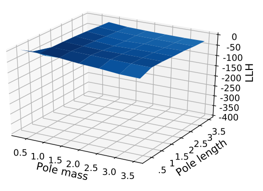
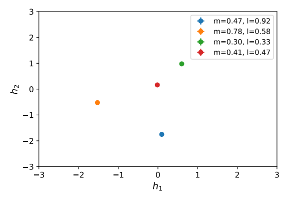
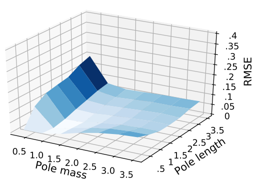
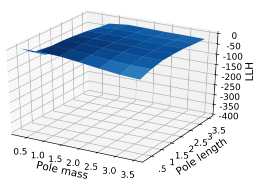
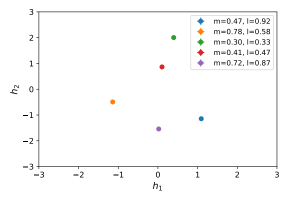
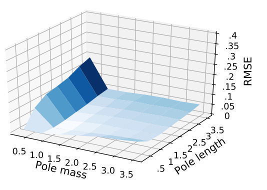
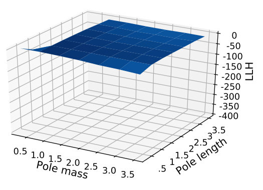
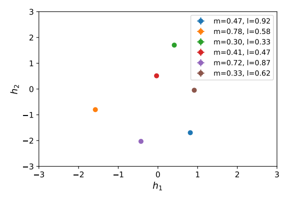
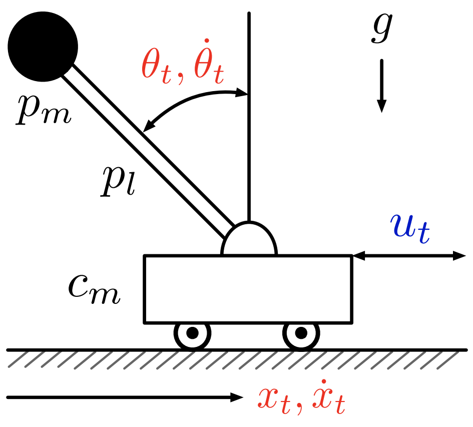
2			
3			
4			

Table 7.2: An illustration of our experimental setup. Here, GSLS acquires 4 tasks for the cart-pole environment. We can measure the RMSE and LLH on the test tasks in each iteration. Furthermore, after the model observed a new task, the latent space changes, including positions of the previously known tasks. Note that the axes of the latent space figures are static.

7.1.3 Continous control environments

Before diving into the results, we briefly introduce the three control task environments. In all experiments, PATA followed the high-level algorithm described in Algorithm (1).

Cart-pole In Section (2.3), we introduced the cart-pole task as an example of a typical dynamical system. It is an under-actuated system with a freely swinging pendulum mounted on cart (Kamthe and Deisenroth 2017). The position of the cart is x_1, the velocity of the cart is $\dot{x}_1$, the angle of the pendulum, measured anti-clockwise from hanging down, is θ and the angular velocity is $\dot{\theta}$.

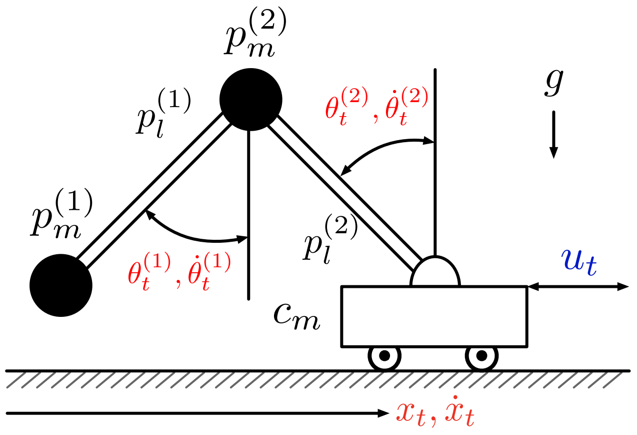


(a) The cart-pole setup as described in (Deisenroth 2010). State variables are marked in red, the control signal in blue and the constants in black.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \dot{x}_1 \\ \dot{\theta} \\ \sin\theta \\ \cos\theta \end{bmatrix} \in \mathbb{R}^5. \quad (7.7)$$

(b) The state representation of the cart-double-pole system.

Cart-double-pole The cart-double-pole setup is similar to the cart-pole system with the only difference of having an attached double pendlum instead of a single pendulum. Modelling this system is challenging since the under-actuated dynamic system is inherently unstable and the dynamics are chaotic (Deisenroth 2010).



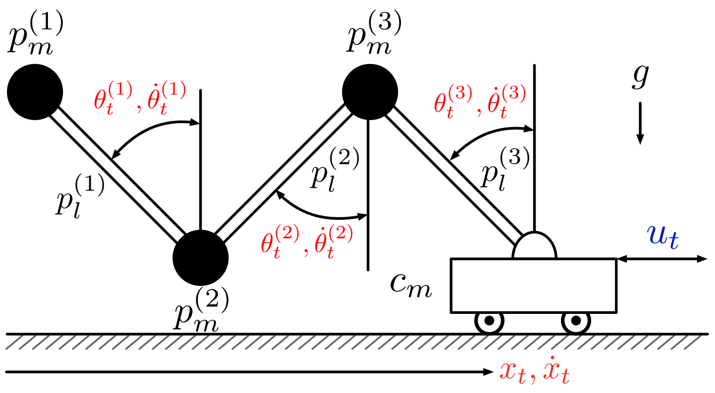
$p_m^{(2)}$
$p_l^{(1)}$
$\theta_t^{(2)}, \dot{\theta}_t^{(2)}$
g
$p_m^{(1)}$
$p_l^{(2)}$
u_t
$\theta_t^{(1)}, \dot{\theta}_t^{(1)}$
c_m
$x_t, \dot{x}_t$

(a) The cart-double-pole setup as described in (Deisenroth 2010). State variables are marked in red, the control signal in blue and the constants in black.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \dot{x}_1 \\ \dot{\theta}^{(1)} \\ \dot{\theta}^{(2)} \\ \sin\theta^{(1)} \\ \cos\theta^{(1)} \\ \sin\theta^{(2)} \\ \cos\theta^{(2)} \end{bmatrix} \in \mathbb{R}^8. \quad (7.8)$$

(b) The state representation of the cart-double-pole system.

Cart-triple-pole Analogously, the cart-triple-pole setup consists of three poles that are serially connected.



$p_m^{(1)}$
$\theta_t^{(1)}, \dot{\theta}_t^{(1)}$
$p_m^{(3)}$
$\theta_t^{(3)}, \dot{\theta}_t^{(3)}$
g
$p_l^{(1)}$
$p_l^{(2)}$
$p_l^{(3)}$
u_t
$\theta_t^{(2)}, \dot{\theta}_t^{(2)}$
c_m
$p_m^{(2)}$
$x_t, \dot{x}_t$

(a) The cart-triple-pole setup with three poles connected serially. State variables are marked in red, the control signal in blue and the constants in black.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \dot{x}_1 \\ \dot{\theta}^{(1)} \\ \dot{\theta}^{(2)} \\ \dot{\theta}^{(3)} \\ \sin\theta^{(1)} \\ \cos\theta^{(1)} \\ \sin\theta^{(2)} \\ \cos\theta^{(2)} \\ \sin\theta^{(3)} \\ \cos\theta^{(3)} \end{bmatrix} \in \mathbb{R}^{11}.$$

(b) The state representation of the cart-triple-pole system.

7.1.4 Experiment 1: Cart-pole

Table 7.3: The experimental setup for the **cart-pole** environment.

Evaluation parameters	
Training trajectory length	30 steps / 4.5s
Test trajectory length	30 steps / 4.5s
Number of different seeds	5
Initial configurations	2, Random
Dynamical configurations	
Time discretisation	$\Delta t = 0.2$s / 5 Hz
Mass of the cart	$c_m = 1$
Mass of the pendulum	$p_m \in [0.3, 3.5]$
Length of the pendulum	$p_l \in [0.3, 3.5]$
Test tasks	$7 \times 7 = 49$
State space	$\dim(\mathcal{S}) = 4$
Action space	$\dim(\mathcal{A}) = 1$
Observation space	$\dim(\mathcal{O}) = 5$
Model	
Dimensions of latent space	$H = 2$
Inducing points	350
Training steps	$7,000$
Inference steps on test tasks	200
Optimisation	
Optimiser	Adam (Kingma and Ba 2014)
Learning rate	0.02
β_1 [3]	$\beta_1 = 0.9$
β_2	$\beta_2 = 0.999$
PATA	
Candidate grid size [4]	$k = 40 \times 40 = 1600$
Candidate rejection ratio	$\alpha = 0.5$
Variance rejection ratio	$\sigma_{\text{MAX}} = \mu(\boldsymbol{\sigma}^2_{\boldsymbol{h}_*}) + \sigma(\boldsymbol{\sigma}^2_{\boldsymbol{h}_*})$
Maximum slack value	$\xi_{\text{MAX}} = 6$
Greedy Sampling distance function	Euclidean distance
BO initial trials	10
BO function evaluations	20

[3] β_1 and β_2 are the exponential decay rate for the 1st and 2nd moment estimates, respectively. For more information, see (Kingma and Ba 2014).

[4] Except for mutual information, where we used 25 per dimension, i.e., $k = 25 \times 25 = 625$, due to the high computational complexity.

Table 7.4: Root mean squared error (RMSE) of predicting test tasks with the **cart-pole** environment across all tactics compared with the random baseline (RDN). The error bars illustrate the standard deviation from the mean while running the experiment with 5 different random seeds. The lower the values, the better the respective tactic performed.

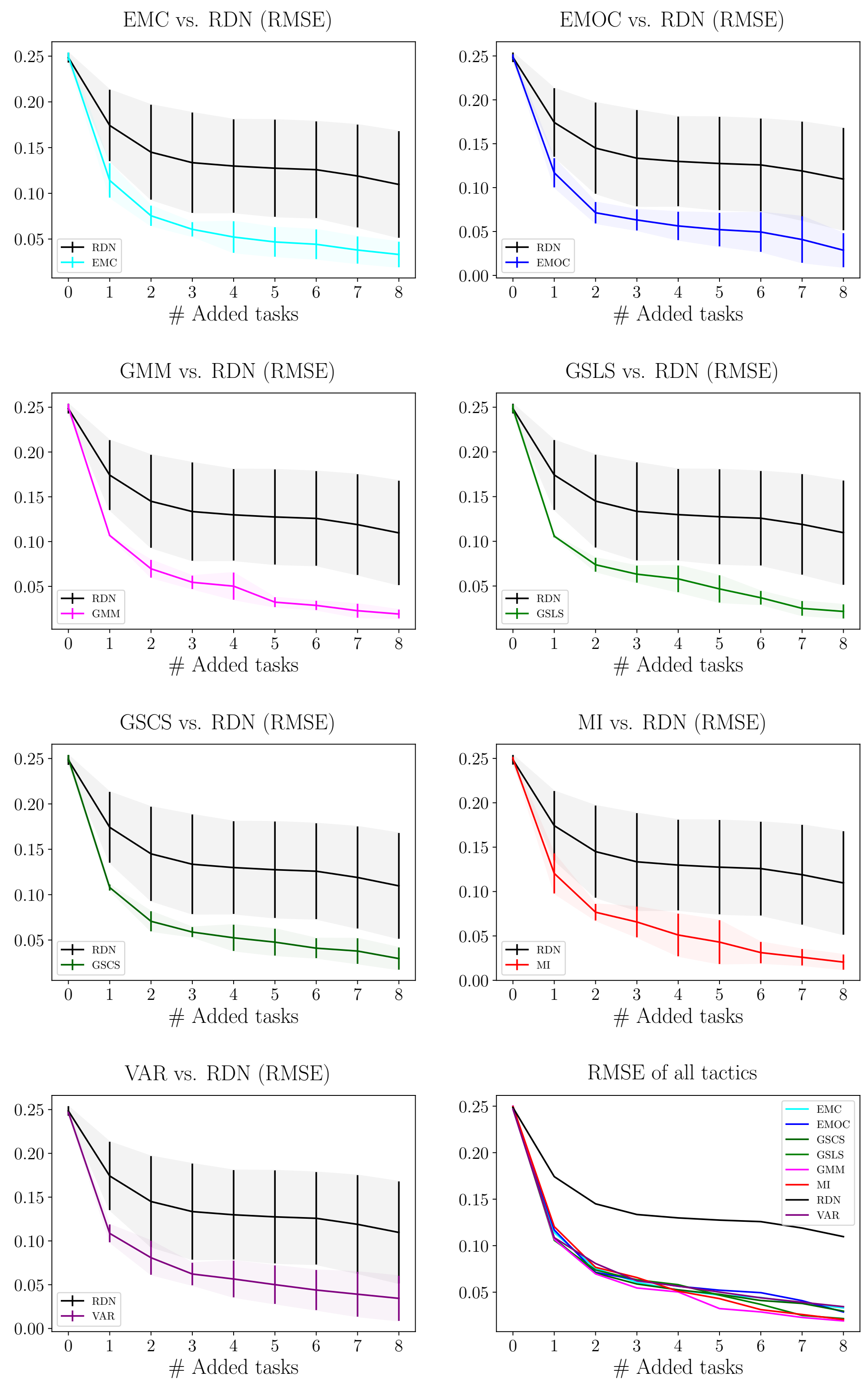
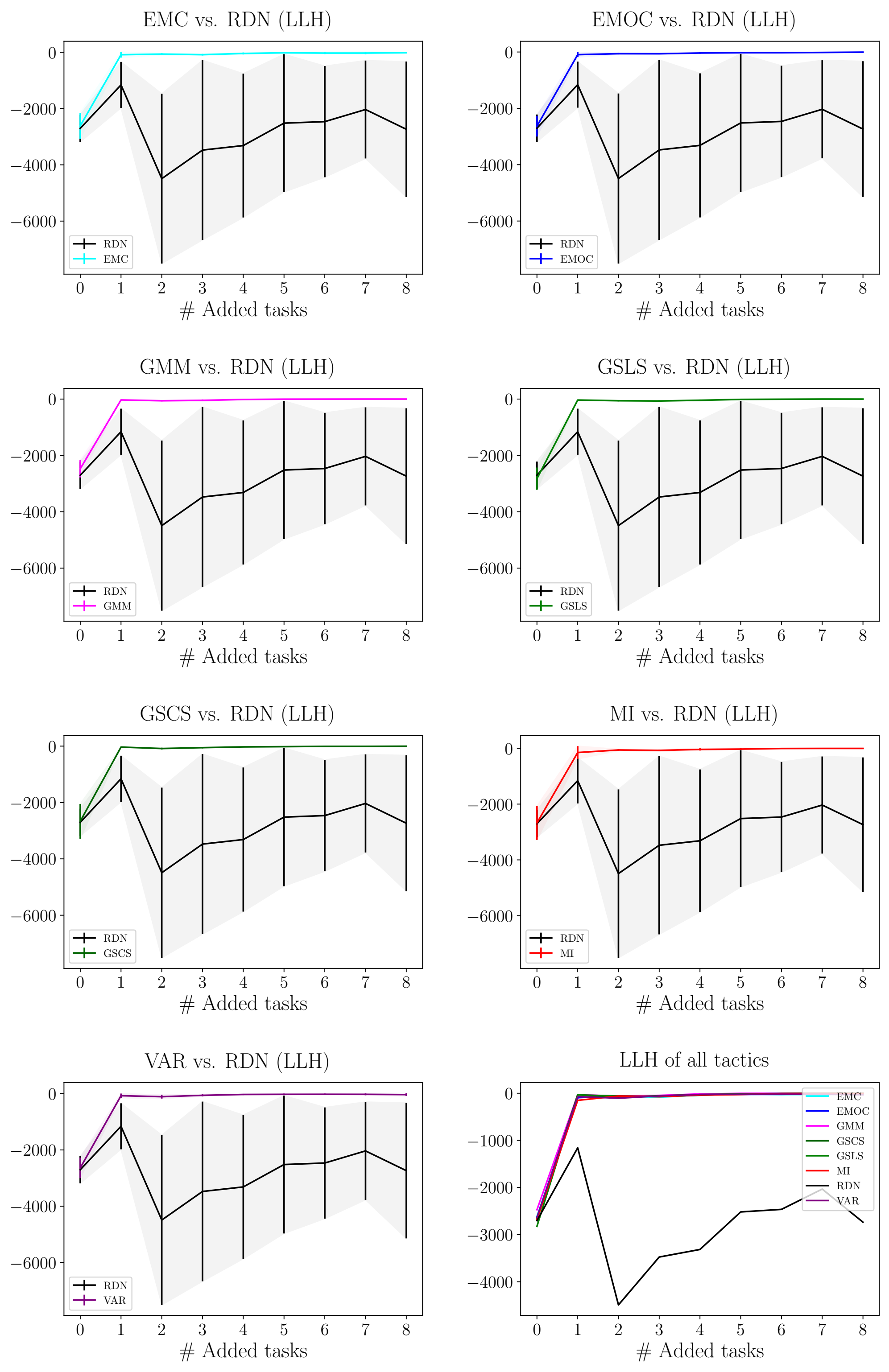


Table 7.5: Log-likelihood (LLH) of predicting test tasks with the **cart-pole** environment across all tactics compared with the random baseline (RDN). The error bars illustrate the standard deviation from the mean while running the experiment with 5 different random seeds. The higher the values, the better the respective tactic performed.



Discussion First of all, we note that the performance of the random baseline (RDN) varies significantly across different seeds. We interpret this as *hit-or-miss* behaviour: It is not impossible that RDN selects informative tasks and performs reasonable, but its performance is volatile. In practice, we would not trust that the tasks it picks cover a wide range of dynamics. Considering only the mean across all seeds after 8 acquired tasks, it significantly performs worse than all the proposed tactics as can be seen in the graph plotting the RMSE and LLH of all tactics. The second worst error stems from uncertainty sampling (VAR), which shows a $\sim 87\times$ improved LLH and roughly $2\times$ better RMSE. The median improvement, as defined by the comparison of RDN with the tactic whose performance is the median over all tactics, is $\sim 683\times$ better in terms of LLH (VAR) and $\sim 3\times$ better in terms of RMSE (EMOC).

Furthermore, as we can see in both RMSE and LLH metrics, all methods, including RDN improve over time. In theory, we expect RDN and all other methods to converge at some point. This event has not happened here during our limited task budget of 8 tasks. Next, we observe decreasing gains of performance improvement over time until all tactics converge to a fairly similar low error. This concavity of the model improvement as a function of tasks added is a desirable observation since it validates each utility function's effectiveness in general. At the same time, the variance of the acquisition tactics across different seeds is fairly low - indicating a very stable behaviour.Interestingly, the model-agnostic tactic GMM is the "winner" without even touching the model's posterior. Remember that tactics from this strategy require low computational overhead. However, the differences across all tactics overall are not high [5].

Table 7.6: The RMSE and LLH mean of all tactics averaged across 5 different seeds after 8 acquired tasks on predicting the transition dynamics of 49 evenly distributed test tasks of the cart-pole system.

Tactic	**RMSE**	**LLH**
RDN	0.1096	-2735.85
EMC	0.0331	-14.51
EMOC	0.0286	-5.76
GMM	$\mathbf{0.0191}$	$\mathbf{-0.49}$
GSCS	0.0295	-3.77
GSLS	0.0215	-1.40
MI	0.0205	-4.37
VAR	0.0344	-31.60

[5]This outcome is not surprising to us since we aimed to research in an agile fashion: We often tried out a tactic as fast as possible, and if it produced promising results, we stuck to it and looked deeper into the theory behind it. Multiple other tactics did not make it into this thesis. Detailing their ideas and investigating potential reasons for their inferior performance is beyond the scope of this report.

7.1.5 Experiment 2: Cart-double-pole

Table 7.7: The experimental setup for the **cart-double-pole** environment.

Evaluation parameters	
Training trajectory length	30 steps / 4.5s
Test trajectory length	30 steps / 4.5s
Number of different seeds	5
Initial configurations	2, Random
Dynamical configurations	
Time discretisation	$\Delta t = 0.15$s / $6.\bar{6}$ Hz
Mass of the cart	$c_m = 1$
Mass of the pendulum	$p_m \in [0.3, 3.5]$
Length of the pendulum	$p_l \in [0.3, 3.5]$
Test tasks	$7 \times 7 = 49$
State space	$\dim(\mathcal{S}) = 6$
Action space	$\dim(\mathcal{A}) = 1$
Observation space	$\dim(\mathcal{O}) = 8$
Model	
Dimensions of latent space	$H = 2$
Inducing points	350
Training steps	$9,000$
Inference steps on test tasks	200
Optimisation	
Optimiser	Adam (Kingma and Ba 2014)
Learning rate	0.02
β_1 [6]	$\beta_1 = 0.9$
β_2	$\beta_2 = 0.999$
PATA	
Candidate grid size [7]	$k = 40 \times 40 = 1600$
Candidate rejection ratio	$\alpha = 0.5$
Variance rejection ratio	$\sigma_{\text{MAX}} = \mu(\boldsymbol{\sigma}^2_{\boldsymbol{h}_*}) + \sigma(\boldsymbol{\sigma}^2_{\boldsymbol{h}_*})$
Maximum slack value	$\xi_{\text{MAX}} = 6$
Greedy Sampling distance function	Euclidean distance
BO initial trials	10
BO function evaluations	20

[6] β_1 and β_2 are the exponential decay rate for the 1st and 2nd moment estimates, respectively. For more information, see (Kingma and Ba 2014).

[7] Except for mutual information, where we used 25 per dimension, i.e., $k = 25 \times 25 = 625$, due to the high computational complexity.

Table 7.8: Root mean squared error (RMSE) of predicting test tasks with the **cart-double-pole** environment across all tactics compared with the random baseline (`RDN`). The error bars illustrate the deviation from the average of running the experiment with 5 different seeds. The lower the values, the better the respective tactic performed.

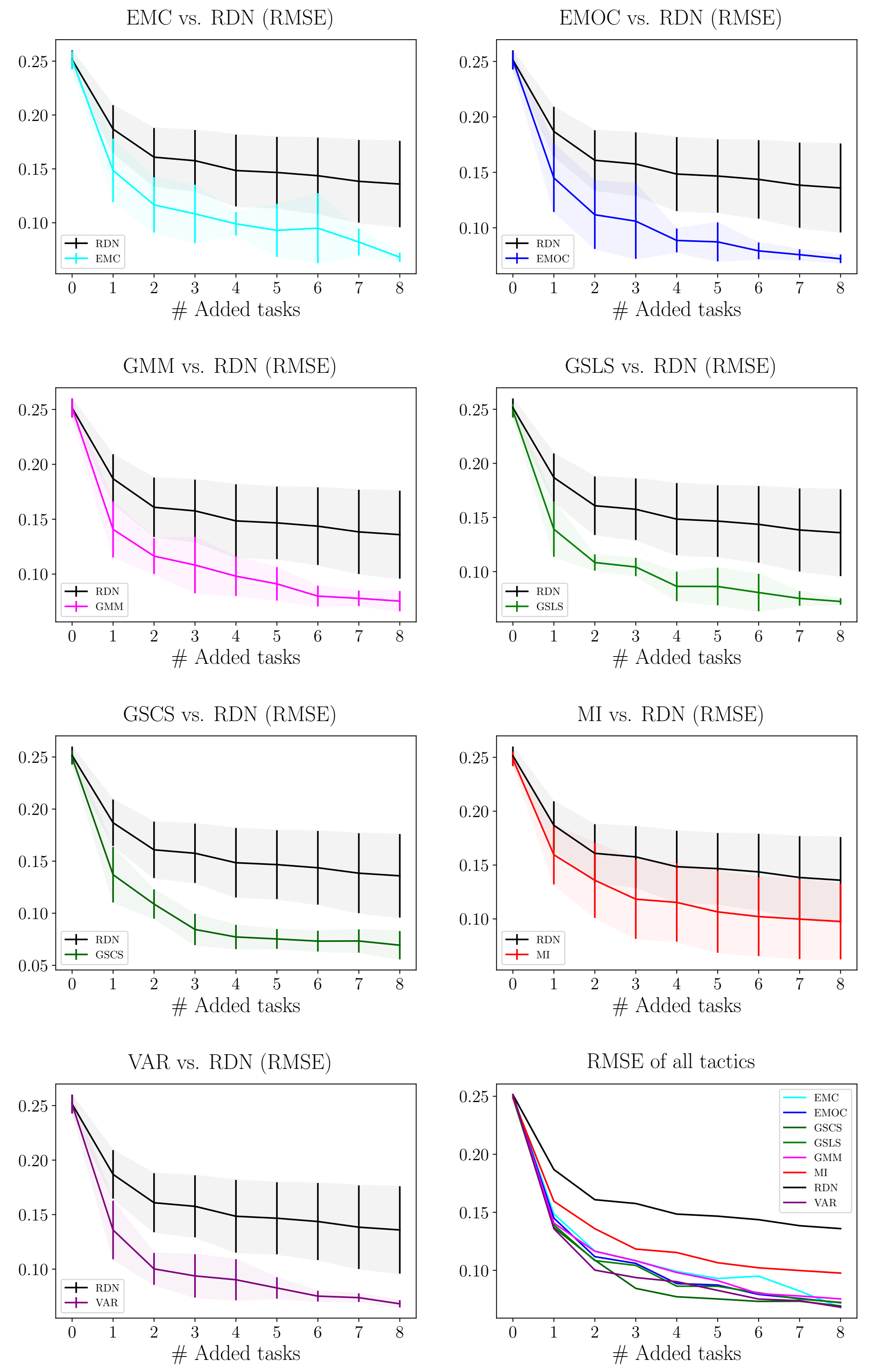
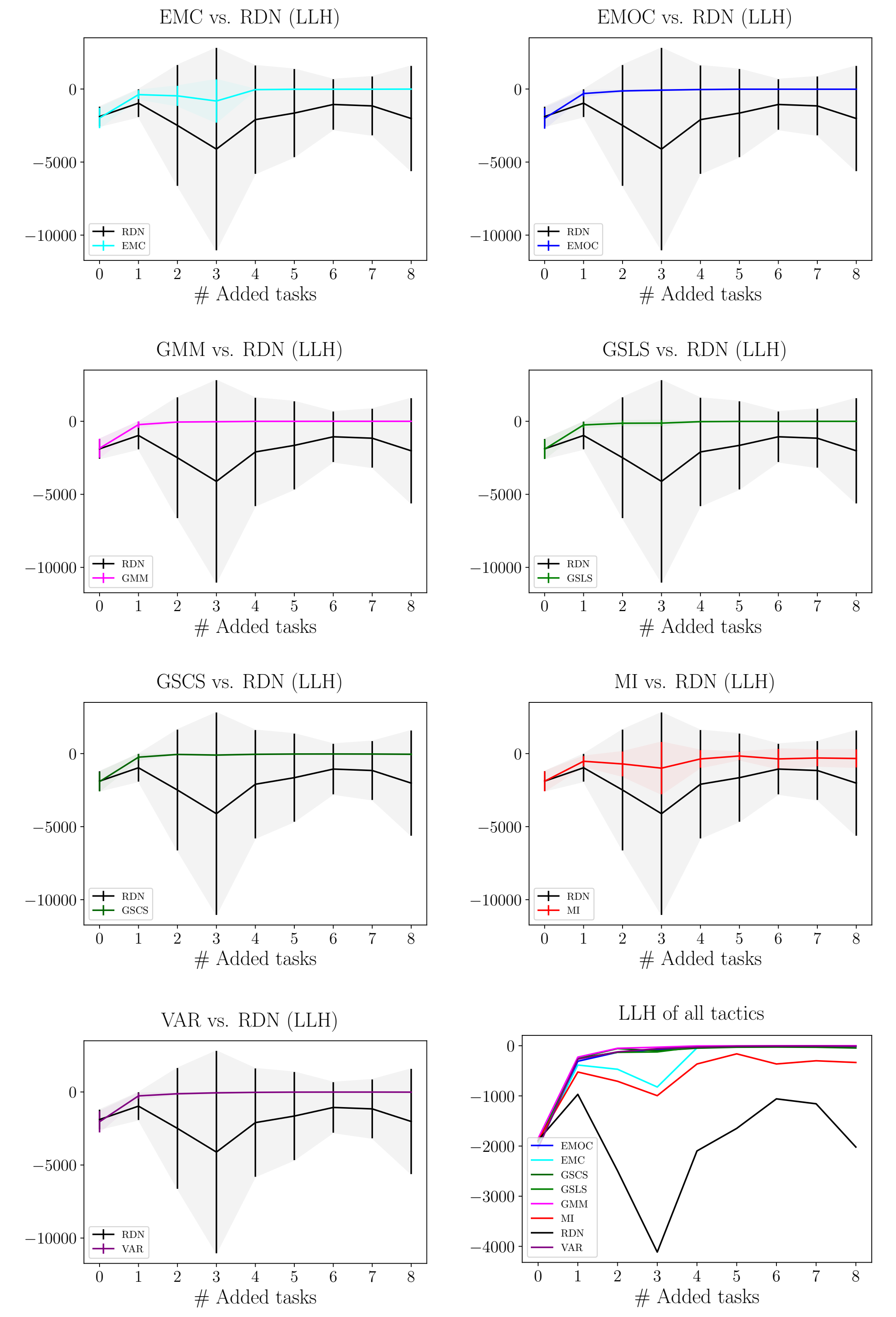
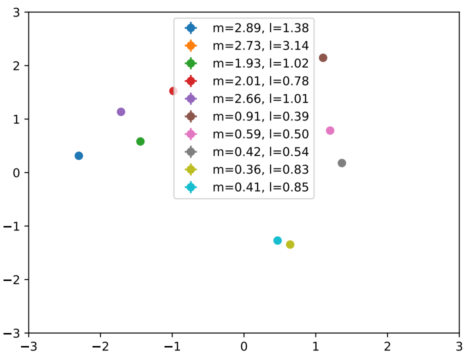


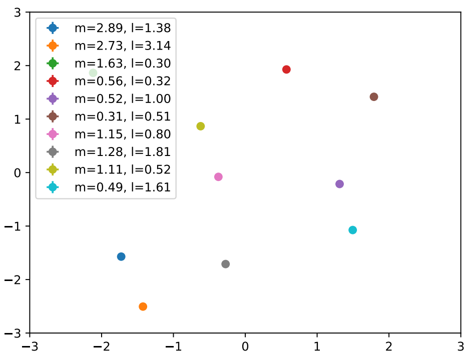
Table 7.9: Log-likelihood (LLH) of predicting test tasks with the **cart-double-pole** environment across all tactics compared with the random baseline (`RDN`). The error bars illustrate the deviation from the average of running the experiment with 5 different seeds. The higher the values, the better the respective tactic performed.



Discussion The results of this experiment share multiple similarities with the previous one, such as the large variance of `RDN`'s performance across different seeds. However, the overall performance improvement has slightly changed, with the median improvement only being $\sim 0.88\times$ better in terms of RMSE (`EMOC`) and $\sim 9180\times$ higher LLH (`GMM`). Again, `GMM` performed quite well and to give the reader an insight into its behaviour, why we included a direct comparison between `GMM`'s and `RDN`'s inferred latent space at the end of the PATA procedure. We also added labels, marking the corresponding configuration of each latent embedding.



(a) The latent space of `RDN` after 8 randomly sampled tasks (Seed = 1).



(b) The latent space of `GMM` after 8 actively acquired tasks (Seed = 1).

Figure 7.4: An illustrative comparison of the latent spaces of `RDN` with `GMM` after the task budget is used up.

Table 7.10: The RMSE and LLH mean of all tactics averaged across 5 different seeds after 8 acquired tasks on predicting the transition dynamics of 49 evenly distributed test tasks of the cart-double-pole system.

Tactic	**RMSE**	**LLH**
`RDN`	0.1359	-2020.02
`EMC`	$\mathbf{0.0679}$	-7.15
`EMOC`	0.0720	-15.08
`GMM`	0.0753	$\mathbf{0.22}$
`GSCS`	0.0693	-44.54
`GSLS`	0.0723	-2.22
`MI`	0.0976	-333.95
`VAR`	0.0681	-10.80

7.1.6 Experiment 3: Cart-triple-pole

Table 7.11: The experimental setup for the **cart-triple-pole** environment.

Evaluation parameters	
Training trajectory length	30 steps / 4.5s
Test trajectory length	30 steps / 4.5s
Number of different seeds	5
Initial configurations	2, Random
Dynamical configurations	
Time discretisation	$\Delta t = 0.15$s / $6.\bar{6}$ Hz
Mass of the cart	$c_m = 1$
Mass of the pendulum	$p_m \in [0.3, 3.5]$
Length of the pendulum	$p_l \in [0.3, 3.5]$
Test tasks	$7 \times 7 = 49$
State space	$\dim(\mathcal{S}) = 8$
Action space	$\dim(\mathcal{A}) = 1$
Observation space	$\dim(\mathcal{O}) = 11$
Model	
Dimensions of latent space	$H = 2$
Inducing points	350
Training steps	$12,000$
Inference steps on test tasks	200
Optimisation	
Optimiser	Adam (Kingma and Ba 2014)
Learning rate	0.02
β_1 [8]	$\beta_1 = 0.9$
β_2	$\beta_2 = 0.999$
PATA	
Candidate grid size [9]	$k = 40 \times 40 = 1600$
Candidate rejection ratio	$\alpha = 0.5$
Variance rejection ratio	$\sigma_{\text{MAX}} = \mu(\boldsymbol{\sigma}^2_{\boldsymbol{h}_*}) + \sigma(\boldsymbol{\sigma}^2_{\boldsymbol{h}_*})$
Maximum slack value	$\xi_{\text{MAX}} = 6$
Greedy Sampling distance function	Euclidean distance
BO initial trials	10
BO function evaluations	20

[8] β_1 and β_2 are the exponential decay rate for the 1st and 2nd moment estimates, respectively. For more information, see (Kingma and Ba 2014).

[9] Except for mutual information, where we used 25 per dimension, i.e., $k = 25 \times 25 = 625$, due to the high computational complexity.

Table 7.12: Root mean squared error (RMSE) of predicting test tasks with the **cart-triple-pole** environment across all tactics compared with the random baseline (RDN). The error bars illustrate the deviation from the average of running the experiment with 5 different seeds. The lower the values, the better the respective tactic performed.

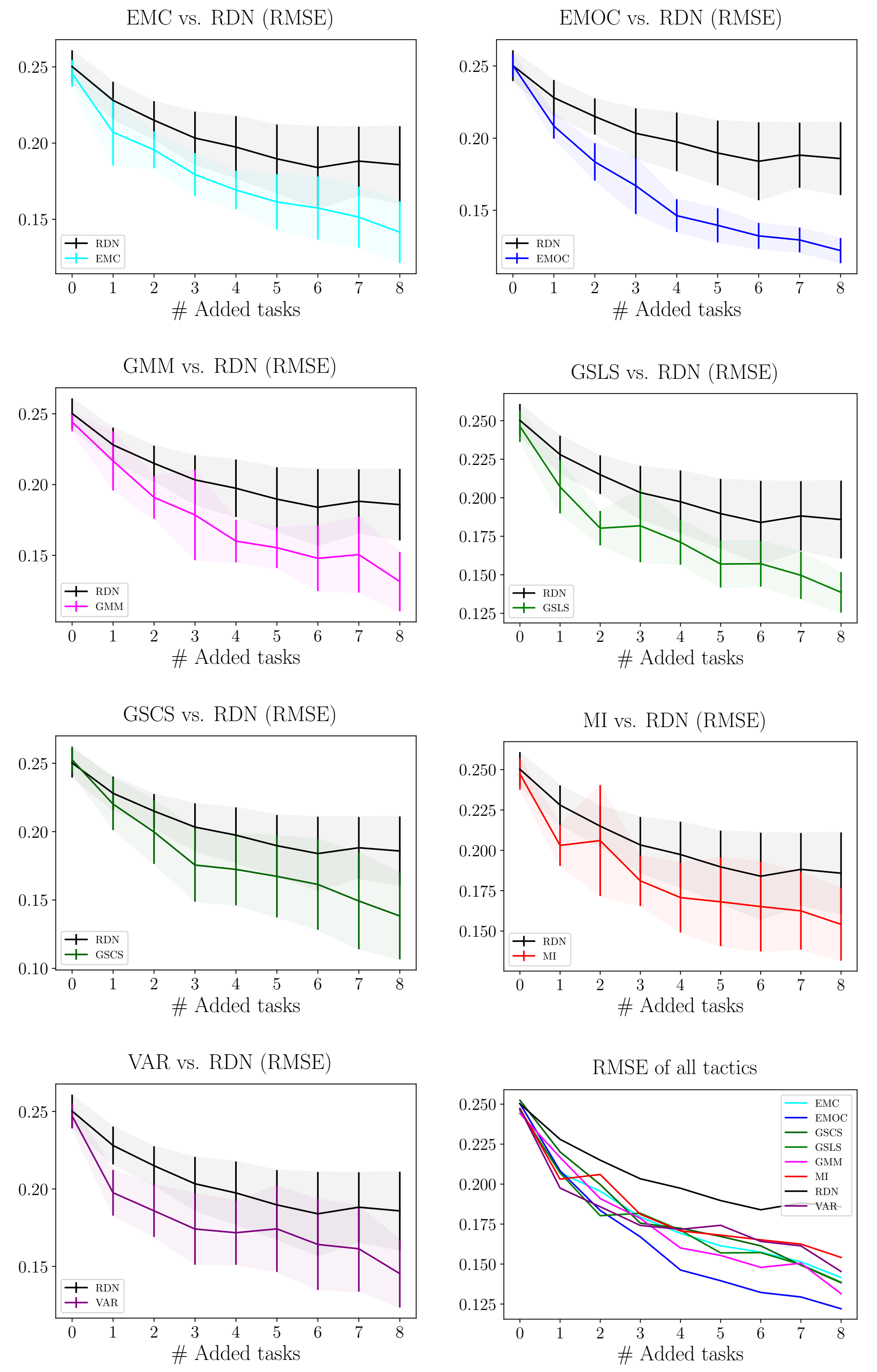
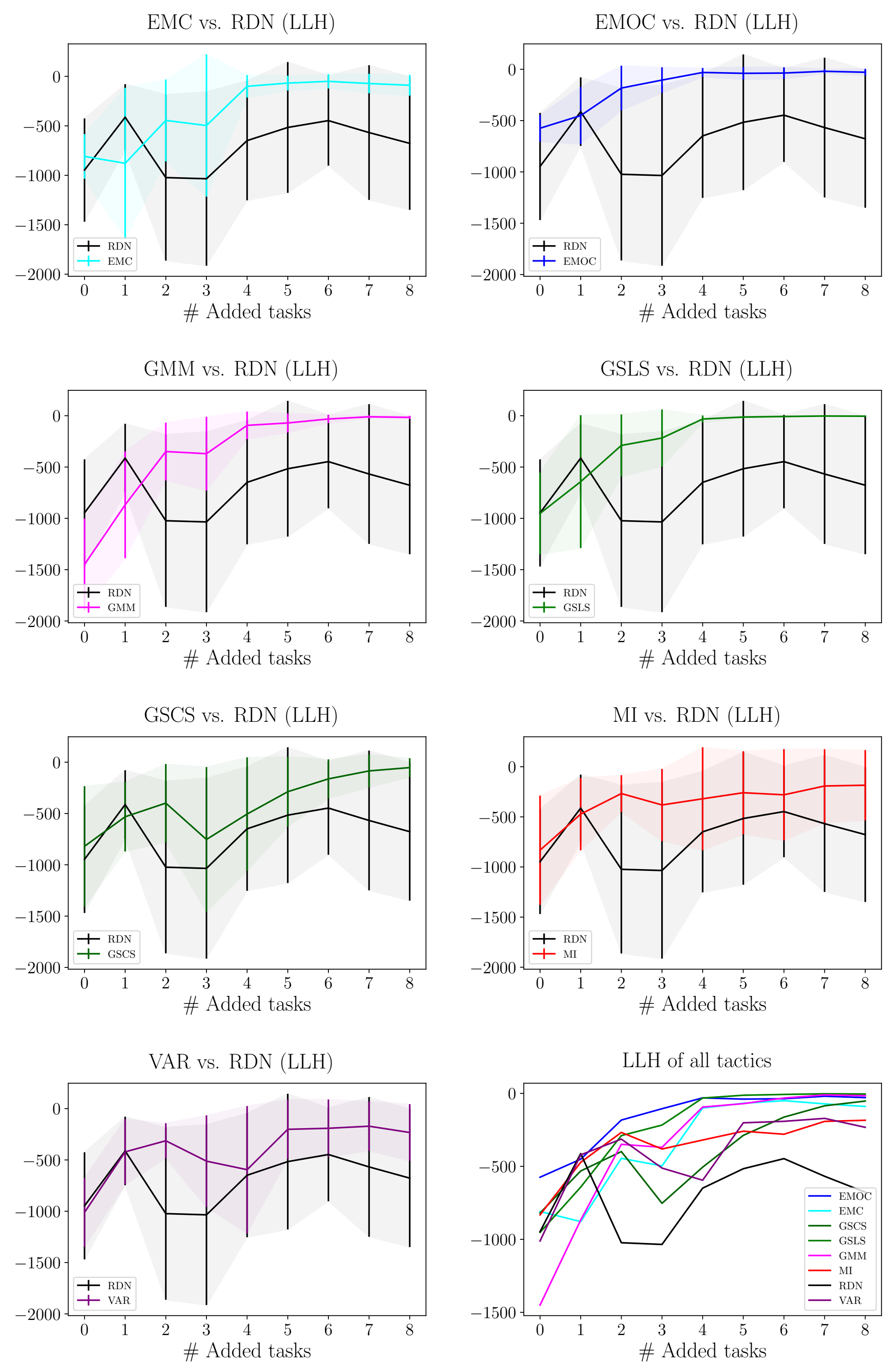


Table 7.13: Log-likelihood (LLH) of predicting test tasks with the **cart-triple-pole** environment across all tactics compared with the random baseline (`RDN`). The error bars illustrate the deviation from the average of running the experiment with 5 different seeds. The higher the values, the better the respective tactic performed.



Remark Before we discuss the results, we want to inform the reader that we had severe difficulties to conduct this experiment due to numerical instabilities. We hypothesise that the cause is the inherent unstable system, even more unstable than Cart-double-pole, but we have not investigated this problem further. Many times, computing the Cholesky decomposition during calculating the GP's kernel matrix was not successful. In the end, we tried many different seeds until we had 5 runs with each method. Note that the used seeds are not consistent across the tactics. Therefore, the initial errors of the runs at 0 added tasks, vary slightly.

Discussion Firstly, the performance differences between `RDN` and the other approaches have become much smaller. It seems that `RDN` is still the worst performing method but this time, the median improvement is only $\sim 0.34\ \times$ lower RMSE (`GSLS`) and $\sim 12\times$ higher LLH (`GSCS`). the variance of the tactics' performances across different seeds is high in comparison to the previous experiments. On average, it is still smaller than `RDN`'s variance. Secondly, there is not so much a drop in the error at the beginning of the task selection as previously seen. For instance, whereas the improvements of `GMM` are mostly flat after having acquired the first task, in this experiment, this flatness sets in around the fourth acquired task.

Table 7.14: The RMSE and LLH mean of all tactics averaged across 5 different seeds after 8 acquired tasks on predicting the transition dynamics of 49 evenly distributed test tasks of the cart-triple-pole system.

Tactic	**RMSE**	**LLH**
`RDN`	0.1858	-677.12
`EMC`	0.1415	-89.63
`EMOC`	0.1220	-28.95
`GMM`	$\mathbf{0.0753}$	-15.94
`GSCS`	0.1383	-52.78
`GSLS`	0.1386	$\mathbf{-4.34}$
`MI`	0.1541	-184.27
`VAR`	0.1453	-232.62

7.1.7 Reacher

In this experiment, instead of specifying the task distribution by different transition dynamics, we specified it by different reward functions. The `ReacherEnv` environment (Tassa et al. 2018) is a two-link planar reacher that aims to reach a target. The coordinates of the target state are defined as follows:

$$x_{\text{target}} = r \cdot \sin(\theta) \qquad y_{\text{target}} = r \cdot \cos(\theta) \tag{7.9}$$

The reward function is the distance between the finger and the target surface

$$R = \left\| \begin{bmatrix} x_{\text{target}} \\ y_{\text{target}} \end{bmatrix} - \begin{bmatrix} x_{\text{finger}} \\ y_{\text{finger}} \end{bmatrix} \right\|. \tag{7.10}$$

During our experiments, all configuration ranges of the reward function we tried were ’too easy’ for the model to learn. As one can see in Figure (7.5), the error distribution of different target locations is mostly flat. All tasks have almost the same errors. Furthermore, we noticed that these errors decrease almost equally for all tasks, regardless of which task was acquired. For the sake of brevity, we do not list all experimental parameters explicitly except for $\Delta t = 0.2$ instead of the default 0.02 (Tassa et al. 2018), an attempt to make the learning more challenging.

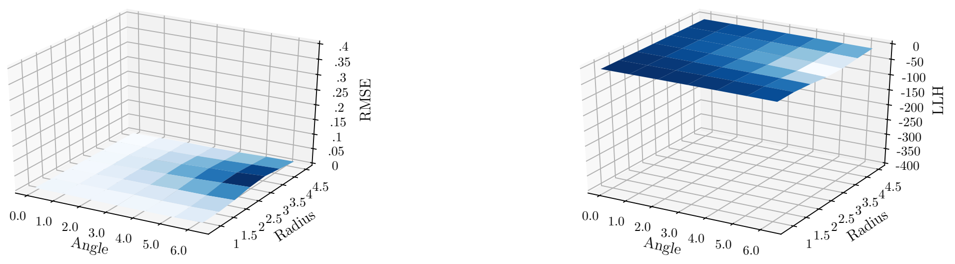
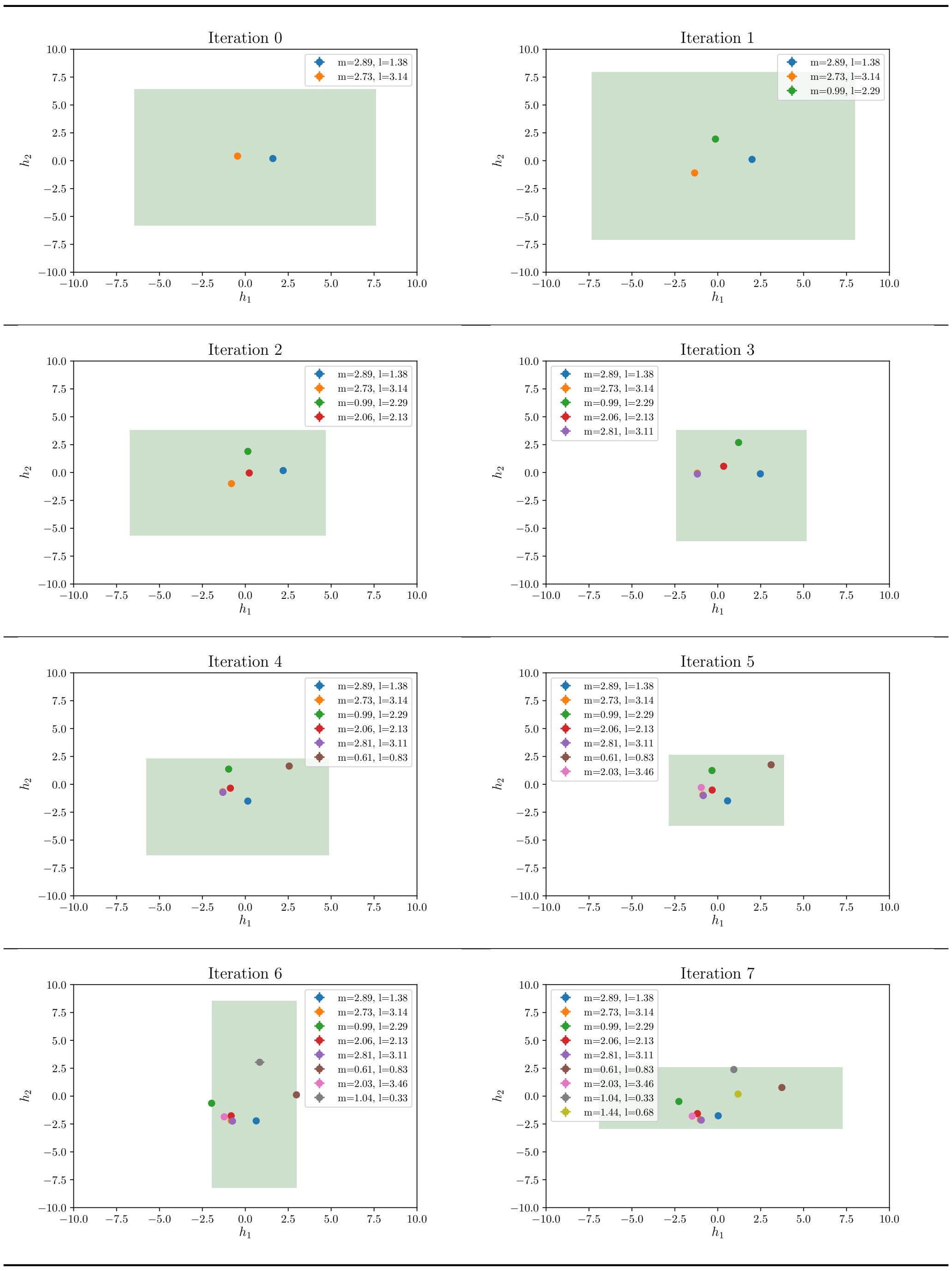


Figure 7.5: The attempt to actively learn different reward functions instead of different dynamics with the reacher environment.

7.2 SECS case study

In Section (6.5), we introduced our proposed technique *Safe exploration of candidate space* (SECS). Now, we want to see whether its proposed regions satisfy the exploration/exploitation trade-off reasonably: They should start large so that the used tactic can explore very different tasks from the beginning. Assuming that the latent space is then well explored, in theory, the optimised slack variable values should get smaller over time. This behaviour is equivalent to noticing the distances between the proposed region edges to the latent embeddings getting smaller. Indeed, we can observe this behaviour in the following experiment.

Table 7.15: `GMM` actively acquiring cart-pole tasks with $\alpha = 0.5, \xi_{\text{MAX}} = 6.0$. SECS proposes the green-shaded area to draw tasks from. The green-shaded area must fulfill the rejection ratio α: more than 50% of the latent candidates in the area's discretisatied set must map to feasible tasks. For the sake of brevity, the rejected tasks are not shown.



7.3 Limitations

After running a variety of experiments, we observed multiple hurdles. In the following we will detail the most common issues we faced, especially with the goal in mind to prepare the reader for her own experiments.

1. **Degenerate latent to configuration space mappings**
   The most frequent issue we had observed was yielding degenerate outputs by the regressor f_{LC}. That means that the outputted task configuration was disadvantageous for improving the model performance, e.g., when it returns a configuration that is very close to a previously learned one (or even the same). We assume that two drivers cause this kind of problem.

   Firstly, the latent candidate who is the input into f_{LC} is too distant to previously learned points in latent space on which f_{LC} is trained on. An illustration of this issue can be seen in Table (7.16). The regression model is then confused and returns the known configuration (its annotation) of the latent point which is closest to the test input.

   Secondly, we observed that embeddings are degenerate themselves, in the sense that they do not represent the underlying task configurations of the observed trajectories reasonably. This problem might co-occur with unlearnable dynamics, described at Point 3 (3), but here, we want to emphasise the lack of informational value of the embeddings due to task indistinguishability. For instance, f_{ML} may yield the (almost) same embedding for different task configurations. Another example is that they only partly capture the degrees of freedom of the dynamics, i.e., only a fraction of the variations of configurations so that they ignore other axes. Assume two latent dimensions $H = 2$ and tasks that only differ in one axis of variation. This setting results in latent embeddings whose means lie in a horizontal or vertical line. On the other hand, it can happen that f_{ML} has not learned a decoupled latent space even though the number of latent dimensions is the same as the degree of freedom among the tasks. For example, when the model observes trajectories of the cart-pole task, where only the mass and length of the pendulum changed, ideally, the inferrend latent dimensions do directly correspond to each of the configuration variable, respectively. However, it can happen that some dimensions of the configuration space do not correspond directly to dimensions in latent space but instead to a combination of them.

h_1	h_2	ψ_1	ψ_2	
0.65	−2.26	0.71	0.97	training configurations
1.00	−0.25	0.86	0.57	
0.92	0.51	0.84	0.40	
−2.76	−0.17	0.44	0.48	
−2.96	−0.17	0.42	0.48	predicted configurations
−3.76	−0.17	0.42	0.48	
−3.56	−0.17	0.42	0.48	
−3.76	−0.17	0.42	0.48	

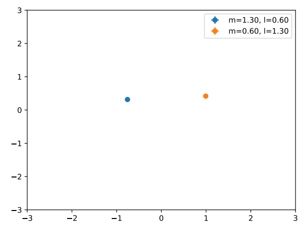
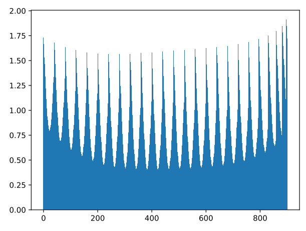
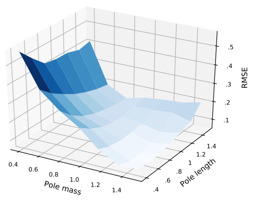
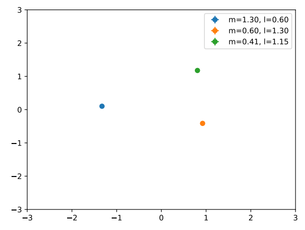
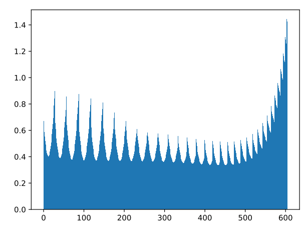
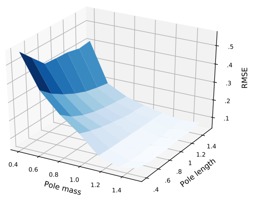
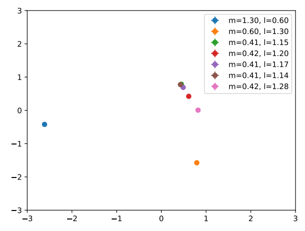
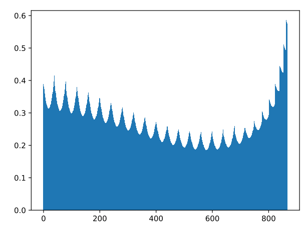
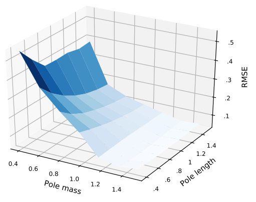
Table 7.16: Illustration of degenerate outputs caused by latent candidates too distant to known annotations.

2. **Initial settings with low prediction errors** In the previous experiments, we deliberately chose challenging settings. That includes that there is still an uneven distribution of task *difficulty* after having learned some initial tasks. With the here chosen environments, tasks closer to previously learned ones are *easier*, task configurations with higher euclidean distance to previously learned tasks are generally more *difficult*. This can also be seen at Table (7.17). Too good initial settings are settings in which the errors of predicting test tasks are already very low and there is not much room of improvement. For example, consider the cart-pole task with a setting with an initial error $\mu_{RMSE} < 0.1$ in iteration 0, where μ_{RMSE}

is the mean of the root mean squared error across all test tasks. The model can still improve, but the efficiency boots through PATA are marginal. Although we have not observed overfitting, i.e., an obvious worsening of the errors with an increasing number of tasks, we observed that the task selection policies do not noticeably outperform random in those scenarios. Moreover, the evolution of the errors over the number of tasks added is quite arbitrary.

We observed two drivers of too 'good' initial settings: Firstly, the range of admissible tasks is a decisive factor since a too small range does not allow for meaningful discoveries of other configurations. Secondly, a short period Δt can lead to almost linear transitions between two states, depending on the task configuration range. Intuitively, one can imagine that 'it does not change much' between two events that are temporally very close.

3. **Unlearnable dynamics** In some settings with high initial error on predicting some test tasks, we could not improve that error or even worsened it after adding new trajectories. We observed the causes for this behaviour are threefold: Firstly, chaotic systems can be complicated to learn (Deisenroth 2010; Kellert 1993). These systems are inherently unstable, and their behaviour over time depends heavily on its initial conditions. Even rounding errors due to too low float-arithmetical precision can render long-term predictions impossible. Secondly, we noticed that a long period Δt combined with an inadequate task configuration space could lead to unlearnable dynamics due to too disorderly behaviour between time-steps. Imagine a pendulum with a very small mass and a low temporal frequency: Between each time-step, the pendulum might have already swung twice around its own axis before the next observation is realised, leading to confusion of the model and high unpredictability of exact next state. Lastly, an adverse sequence of control signals might generate a hard-to-learn trajectory, independent of the actual system configuration. We observed that when initialising the model, there can exist significant differences in how well the model fits the initial trajectories among randomly sampled control signals. An indicator for that is how low the ELBO gets during model training.

t	**Latent space**	**Candidate scores**	**RMSE**
0			
1			
...	...	...	...
5			

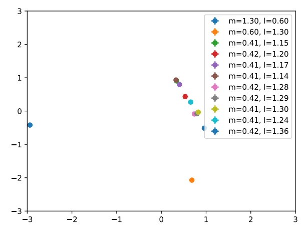
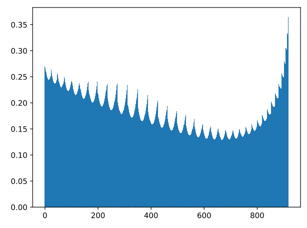
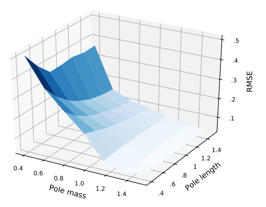
9			

Table 7.17: An illustration of the model-based task selection with EMOC utility function attempting to learn unlearnable dynamics. The dynamics stem from the cart-double-pole environment with $\Delta t = 0.2$ with 30 time steps per trajectory. We highlight that EMOC aims to select tasks near $p_m = 0.4$ (left side of the RMSE graph) and scores candidates at this region comparatively high to learn their dynamics but gets stuck. This results in repeatedly sampling proximate points from the same region.

Chapter 8

Conclusion and Future Directions

Motivated by the lack of a systematic way to select training tasks in meta-learning research, we developed PATA, a probabilistic active task acquisition framework. We tailored the scope of this work's tasks to the context of reinforcement learning (RL) problems, in which an agent aims to learn a policy from interactions that maximises rewards. Since finding an optimal policy is a lengthy trial-and-error procedure that typically requires many interactions with the environment, we informed the reader about the concept of model-based RL that reduces the amount of interacting with the real environment by simulating it through a model. To lay the theoretical foundation for such a model class, we elaborated on well-established ideas from Bayesian statistics up to statistical machine learning, so that we ultimately arrived at Gaussian Processes (GP). Having demonstrated the standard supervised learning GP setting, we guided the reader through a more scalable approximate inference GP model, enabling us to deal with with larger data sets. To initiate with the idea of inferring unobserved variables, we introduced GPLVMs and then, moved forward to a state-of-the-art meta-reinforcement learning model. Before we looked at the novel contributions, we conducted a brief literature review and concluded that there is no notion of active task acquisition for RL tasks in related work (active learning) as well as in the immediate problem space (MRL).

Finally, we proposed PATA to combat the ineffiency shortcoming: By exploiting the meta-learning model's knowledge of the relations between previously learned tasks, we proposed concrete strategies to sample tasks that are dissimilar to previous ones. Conceptually, we suggested two types of acquisition strategies, *model-based* and *model-free*, and shortly discussed the pros and cons of them. Next, we recommended concrete tactics one can use off-the-shelf, making PATA a practical framework. We provided experimental evidence that sampling tasks from the latent space systematically outperformed a random baseline. Now that we have gone through the past and arrived in the present, we now look to the future.

8.1 Future work

We believe there are numerous promising opportunities for future work that we want to work on and ideally, see others tackling. We break down future work into short-term and longer-term plans. Short-term plans are incremental improvements on PATA. Hereby, we can incorporate the learnings we have gained during this project. Based on experienced shortcomings, we derive goals we would have immediately continuned to work on if we were given more time. Concretely, short-term future work is summarised below:

1. **Apply PATA to higher-dimensional continuous control tasks**: We would run experiments with well-known physical environments that have high-dimensional observation spaces. Examples include tasks from the `dm_control` environment (Tassa et al. 2018), such as `Cheetah` ($\dim(\mathcal{O})=17$), `Fish` ($\dim(\mathcal{O})=24$), or `Humanoid` ($\dim(\mathcal{O})=67$).

2. **Incorporate planning**: With respect to model-based RL, this work is following the assumption that an incorrect model leads to sub-optimal policies (Barto 2018). Hence, PATA aims to improve the model actively. For the sake of brevity, we neglected the planning component and focused our efforts on exclusively improving the model's performance on predicting test

trajectories rather than actually solving any tasks by learning policies. It would be interesting to validate that assumption by putting a planning algorithm on top in place. Hereby, we could go back to (Sæmundsson, Hofmann, and Deisenroth 2018)'s approach, again.

3. **Combining PATA with other probabilistic ML models**: Until now, we have only considered (Sæmundsson, Hofmann, and Deisenroth 2018)'s model as foundation for PATA. However, we assume that with some tweaks we could combine it with other meta-learning models that are also suitable for MRL. Examples are models that are proposed or mentioned in: Galashov et al. 2019; Gordon et al. 2018; Petangoda et al. 2019; Rakelly et al. 2019; Springenberg et al. 2018; Zintgraf, Igl, et al. 2019.

4. **Extending depth and breath of tactics**: Regarding the tactics that can be used interchangeably within PATA's strategies, there are two questions worth to explore: First, how can we match a tactic to the learning setting we face at hand? Note that this mapping might not only take the underlying task environment into account but also other factors. Second, what are shortcomings of the current tactics we could combat by deriving new ones? Also, there might be other interesting ideas in related research fields we could explore.

Longer-term plans aim to go beyond what is reliably known, i.e., they are non-trivial and lasting unsolved in the future. Meta-learning is an actively evolving research field. These plans implicitly consider the uncertainty of change in the overall research scheme. Specifically, we propose three areas to explore:

1. **Non-probabilistic active task acquisition** So far, we focused on model-based meta reinforcement learning (MB-MRL) where the model is a probabilistic one. Recently, researchers proposed neural-network based MB-MRL approaches (Nagabandi et al. 2018), related to gradient-based and recurrence-based meta-learning models. There are no inferred task-specific properties one could explore. One benefit of these models is a potentially higher model capacity due to the computational complexity of GPs. Again, the authors randomly sample training tasks from a given task distribution. We think it is worthwhile to incorporate an active task acquisition functionality in these models as well.

2. **Active task acquisition within model-free MRL**: Hitherto, we aimed to improve the performance of an MB-MRL algorithm's model. In literature, there exist also model-free MRL approaches. For example, in (Zintgraf, Shiarlis, et al. 2019), the authors suggested a *fast context adaptation via meta-learning* (CAVIA) model, an extension to model-agnostic meta-learning, (C. Finn, Abbeel, and Levine 2017). This approach uses neural networks and partitions the model parameters into *context parameters* and *shared parameters*. Furthermore, the CAVIA for RL algorithm samples training tasks randomly. There might be a way to select tasks such that the policy gradient methods can converge to an optimal parameterised policy faster.

3. **Applying active task acquisition to fundamentally different task domains** Up to now, we were concerned with the domain of physical systems. How would one explore task embeddings in visual or hierarchical task domains? For example, in (Mnih et al. 2015), reinforcement learning agents reach human-level performance on video games by just observing pixels and game scores as model inputs.

Appendix A

User Guide

A.1 Dependencies

We listed all requirements in the file `requirements.txt`. We quickly enumerate some of the crucial libraries and explain their function.

- **Gaussian Processes**: `GPflow` is used to implement the meta-learning model f_{ML}. The original code is available here. Furthermore, a vanilla GP regression model from `GPy` implements f_{LC}. Lastly, we use `GPyTorch` to be compatible with the Bayesian optimisation library `BoTorch` and the experimental design library `ax-platform`.

- **Automatic differentiation**: Both GP libraries `GPflow` and `GPyTorch` use automatic differentiation libraries; here, `Tensorflow 1` and `PyTorch`, respectively.

- **Continuous control tasks**: To simulate the physical systems, we use `dm_control` which in turn uses `MuJoCo`. Note that `MuJoCo` is not a python library, but an advanced physics simulation engine built with C/C++ API. This website provides its download. Furthermore, `MuJoCo` requires an activivation key which is provided to licensed users. More information can be found on its License page.

A.2 Packages

In the following, we want to familiarise the reader with the structure of our code basis. Thereby, we quickly explain the functionality of each package.

- `models`: Meta-learning model training, inference, predictions

- `models.mrl`: f_{ML}

- `rl`: Wrapper methods for interacting with continuous control task enviroments and trajectory observation from dynamical systems

- `tactics`: Parent package of all tactics

- `tactics.mb`: Model-based tactic utility functions

- `tactics.mf`: Model-free tactic utility functions

- `utils`: Helper functions

The overall PATA algorithm is implemented in `run.py` in the parent package `PATA`.

A.3 Experimental parameters

To run the experiments, the user needs to set experimental parameters, as demonstrated in the experimental setup tables in chapter 7. All necessary parameters to-be-set are parsed from the command line.

Table A.1: This table contains the arguments to set the experimental parameters.

Argument	**Type**	**Default value**
`-env`	str	"CartPoleDMEnv"
`-model_name`	str	"MLSVGP"
`-seed`	int	4
`-control_signals`	str	"fixed"
`-evaluation`	bool	True
`-tactic`	str	"GMM"
`-candidate_grid_size`	int	40
`-evaluation_task_grid_size`	int	7
`-slack_min_const_dim_1`	float	-3
`-slack_max_const_dim_1`	float	3
`-slack_min_const_dim_2`	float	-3
`-slack_max_const_dim_2`	float	3
`-alpha`	float	.5
`-conf_space_dim`	int	2
`-eval_lower_bound_dim_1`	float	.3
`-eval_upper_bound_dim_1`	float	3.5
`-eval_lower_bound_dim_2`	float	.3
`-eval_upper_bound_dim_2`	float	3.5
`-latent_space_filtering`	str	"opt"
`-task_budget`	int	8
`-n_train_envs`	str	2
`-train_envs_rdn`	bool	True
`-train_envs`	list	[[1.0, 0.5],[0.5, 1.0]]
`-episode_length`	int	30
`-draw_plots`	bool	True
`-pred_traj_length`	int	5
`-dim_h`	int	2
`-n_inducing`	int	180
`-model_train_steps`	int	7000
`-model_infer_steps`	int	100
`-BO_init_trials`	int	10
`-BO_trials`	int	20

Non-obvious options to set the parameters correctly are:

- `-env`: "CartPoleDMEnv", "TwoPolesEnv", "ThreePolesEnv", "ReacherEnv"
- `-latent_space_filtering`: "opt" for using SECS, otherwise it will just add the specified

slack constants without optimising

- `-tactic`: "EMC", "EMOC", "GMM", "GSCS", "GSLS", "MI", "RDN", "VAR"
- `-train_envs_rdn` and `-train_envs`: if `-train_envs_rdn` is set to `True`, initial training tasks are randomly sampled, otherwise, the in `-train_envs` specified configurations are used
- `-draw_plots`: if set to `True`, plots are automatically shown and saved. Since we use `mplot3d` for the 3D plots which is not compatibile with all operating systems, this parameters, if set to `False`, saves all results numerically, the user then can plot by herself

A.4 Examples

For reproducing results and running experiments in general, please run commands like the following examples from the command line:

Listing A.1: Examples to run experiments from the command line.

```
python3 --tactic="GMM" --env="ThreePolesEnv" --evaluation true --alpha=0.2
 --task_budget=12 --model_train_steps=10000  --BO_init_trials=5 --BO_trials=5
  --seed=1

 python3 --tactic="MI" --env="CartPoleDMEnv" candidate_grid_size=25 --evaluation
  false --alpha=0.5  --task_budget=8 --model_train_steps=7000
   --BO_init_trials=10 --BO_trials=20 --seed=5

  python3 --tactic="RDN" --env="TwoPolesEnv" --candidate_grid_size=25
   --eval_lower_bound_dim_1=0 --eval_upper_bound_dim_1=6
    --eval_lower_bound_dim_2=0.01 --eval_upper_bound_dim_2=0.2 --evaluation
  false --alpha=0.5  --task_budget=8 --model_train_steps=2000 --seed=5
   --BO_init_trials=10 --BO_trials=20
```

Appendix B

Ethical considerations

Autonomous systems (AS) will undoubtedly lead to many paradigm changes across a variety of areas like education, healthcare, politics, renewable resources, transportation, and work. We will not look at the ethical implications of AS in general within this report. However, we will focus on two concerns: a) Has the conduction of this work been ethically considerate and b) how does it affect others?

Firstly, among other things, this work's research procedure included the incorporation of software from others and the computation of multiple experiments. Starting with the former, all incorporated software code stems exclusively from open-source libraries, marked as such and cited. For instance, `GPflow` and `dm_control` are two crucial external packages used in this work's system. Both are licensed under the Apache License 2.0, a permissive license that allows academic use, modification and distribution. Furthermore, the usage of computational resources for conducting experiments required electricity. Some ways of energy generation can damage the environment. In our case, we computed all experiments on the Google Cloud Plattform, a cloud service that states to use renewable energy exclusively (during the time of our usage) [1]. Accordingly, we aimed to reduce this work's environmental impact by the usage of sustainable resources.

Secondly, to assess how this work affects others, we will briefly touch upon job automation, military spheres and again, energy consumption. Reinforcement learning (RL) agents can perform many complex tasks, certainly including such that automate jobs currently performed by humans. On the other side, we believe that we will see many positive consequences of in-production reinforcement learning systems, including the creation of new jobs and better resource management (Mao et al. 2016). Moreover, we have not developed the methods in this work with military applications in mind. However, to the extent that the military benefits from more efficient meta-learning algorithms to train their agents, the contributions in this work may benefit military applications as well. Lastly, we want to emphasise that the key success metric of this work is data-efficiency. We proposed techniques to make meta-learning more efficient, in terms of fewer data collected (task trajectories) as well as the resulting shorter computation time. Both effects directly lead to less energy consumption.

[1] For more information, we refer the reader to Google Cloud's renewable energy page.

Appendix C

Legal and Ethics Checklist

Section 1: HUMAN EMBRYOS/FOETUSES	**Yes**	**No**
Does your project involve Human Embryonic Stem Cells?		✗
Does your project involve the use of human embryos?		✗
Does your project involve the use of human foetal tissues / cells?		✗
Section 2: HUMANS	**Yes**	**No**
Does your project involve human participants?		✗
Section 3: HUMAN CELLS / TISSUES	**Yes**	**No**
Does your project involve human cells or tissues? (Other than from “Human Embryos/Foetuses” i.e. Section 1)?		✗
Section 4: PROTECTION OF PERSONAL DATA	**Yes**	**No**
Does your project involve personal data collection and/or processing?		✗
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✗
Does it involve processing of genetic information?		✗
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✗
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✗
Section 5: ANIMALS	**Yes**	**No**
Does your project involve animals?		✗
Section 6: DEVELOPING COUNTRIES	**Yes**	**No**
Does your project involve developing countries?		✗

If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✗
Could the situation in the country put the individuals taking part in the project at risk?		✗
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY	**Yes**	**No**
Does your project involve the use of elements that may cause harm to the environment, animals or plants?	✗	
Does your project deal with endangered fauna and/or flora /protected areas?		✗
Does your project involve the use of elements that may cause harm to humans, including project staff?		✗
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✗
Section 8: DUAL USE	**Yes**	**No**
Does your project have the potential for military applications?	✗	
Does your project have an exclusive civilian application focus?		✗
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✗
Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✗
Section 9: MISUSE	**Yes**	**No**
Does your project have the potential for malevolent/criminal/terrorist abuse?		✗
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✗
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✗
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✗
SECTION 10: LEGAL ISSUES	**Yes**	**No**
Will your project use or produce software for which there are copyright licensing implications?		✗
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✗

SECTION 11: OTHER ETHICS ISSUES	**Yes**	**No**
Are there any other ethics issues that should be taken into consideration?		✗

Bibliography

Álvarez, Mauricio A., Lorenzo Rosasco, and Neil D. Lawrence (2012). “Kernels for Vector-Valued Functions: A Review”. In: *Foundations and Trends® in Machine Learning* 4.3, pp. 195–266. ISSN: 1935-8237. DOI: 10.1561/2200000036. arXiv: arXiv:1106.6251v2. URL: http://www.nowpublishers.com/article/Details/MAL-036.

Bachman, Philip, Alessandro Sordoni, and Adam Trischler (2016). “Learning Algorithms for Active Learning”. In:

Barto, Richard S. Sutton; Andrew G. (2018). *Reinforcement Learning - An Introduction*. Vol. 84. ISBN: 9780262039246. URL: http://ir.obihiro.ac.jp/dspace/handle/10322/3933.

Bauer, Matthias, Mark van der Wilk, and Carl Edward Rasmussen (2016). “Understanding Probabilistic Sparse Gaussian Process Approximations”. In: Nips. arXiv: 1606.04820. URL: http://arxiv.org/abs/1606.04820.

Bearden, J Neil (2006). “A new secretary problem with rank-based selection and cardinal payoffs”. In: *Journal of Mathematical Psychology* 50.1, pp. 58–59. ISSN: 0022-2496. DOI: https://doi.org/10.1016/j.jmp.2005.11.003. URL: http://www.sciencedirect.com/science/article/pii/S0022249605000933.

Bertsekas, Dimitri P. (2019). *Reinforcement Learning and Optimal Control*.

Bhattacharya, Rabi and Mukul Majumdar (Dec. 2003). “Random dynamical systems: a review”. In: *Economic Theory* 23.1, 13–38 (2004). ISSN: 1432-0479. DOI: 10.1007/s00199-003-0357-4. URL: https://doi.org/10.1007/s00199-003-0357-4.

Bishop, C M (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer. ISBN: 9780387310732. URL: https://books.google.co.uk/books?id=qWPwnQEACAAJ.

Blei, David M., Alp Kucukelbir, and Jon D. McAuliffe (2017). “Variational Inference: A Review for Statisticians”. In: *Journal of the American Statistical Association* 112.518, pp. 859–877. ISSN: 1537274X. DOI: 10.1080/01621459.2017.1285773. arXiv: arXiv:1601.00670v9.

Blei, David M., Rajesh Ranganath, and Shakir Mohamed (2016). “Variational Inference: Foundations and Modern Methods”. In: *NIPS 2016 Tutorial*.

Cai, Wenbin, Ya Zhang, and Jun Zhou (2013). “Maximizing expected model change for active learning in regression”. In: *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 51–60. ISSN: 15504786. DOI: 10.1109/ICDM.2013.104.

Cohn, David, Zoubin Ghahrami, and Michael Jordan (1996). “Active Learning with Statistical Models David”. In: *Proceedings - IEEE International Symposium on Circuits and Systems* 3, pp. 868–887. ISSN: 10769757. DOI: 10.1613/jair.295. arXiv: 9603104 [cs].

Cover, Thomas M and Joy A Thomas (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. New York, NY, USA: Wiley-Interscience. ISBN: 0471241954.

Deisenroth, Marc Peter (2010). *Efficient Reinforcement Learning using Gaussian Processes*. ISBN: 9783866445697. URL: http://eprints.pascal-network.org/archive/00007628/.

— (2019a). *argmax talks: Data-efficient reinforcement learning*. URL: https://www.youtube.com/watch?v=AVdx2hbcsfI.

— (2019b). *C493 - Bayesian optimization lecture slides*.

Deisenroth, Marc Peter, A Aldo Faisal, and Cheng Soon Ong (2020). *Mathematics for machine learning*.

Deisenroth, Marc Peter, Dieter Fox, and Carl Edward Rasmussen (2015). “Gaussian processes for data-efficient learning in robotics and control”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2, pp. 408–423. ISSN: 01628828. DOI: 10.1109/TPAMI.2013.218.

Deisenroth, Marc Peter, Yicheng Luo, and Mark Van der Wilk (n.d.). “A Practical Guide to Gaussian Processes”. In: ().

Deza, M M and E Deza (2009). *Encyclopedia of Distances*. Encyclopedia of Distances. Springer Berlin Heidelberg. ISBN: 9783642002342. URL: https://books.google.co.uk/books?id=LXEezzccwcoC.

Dorf, Richard C and Robert H Bishop (2000). *Modern Control Systems*. 9th. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. ISBN: 0130306606.

Duvenaud, David Kristjanson (2014). "Automatic Model Construction with Gaussian Processes". In: June, p. 144. URL: https://www.cs.toronto.edu/%7B~%7Dduvenaud/thesis.pdf.

Fang, Meng, Yuan Li, and Trevor Cohn (2017). "Learning how to Active Learn: A Deep Reinforcement Learning Approach". In: arXiv: 1708.02383. URL: http://arxiv.org/abs/1708.02383.

Finn, Chelsea B. (2018). "Learning to Learn with Gradients". PhD thesis. University of California, Berkeley.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: arXiv: 1703.03400. URL: http://arxiv.org/abs/1703.03400.

Frazier, Peter I. (2018). "A Tutorial on Bayesian Optimization". In: Section 5, pp. 1–22. arXiv: 1807.02811. URL: http://arxiv.org/abs/1807.02811.

Freytag, Alexander et al. (2013a). "Labeling examples that matter: Relevance-based active learning with Gaussian processes". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8142 LNCS, pp. 282–291. ISSN: 03029743. DOI: 10.1007/978-3-642-40602-7_31.

— (2013b). "Labeling examples that matter: Relevance-based active learning with Gaussian processes". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8142 LNCS.c, pp. 282–291. ISSN: 03029743. DOI: 10.1007/978-3-642-40602-7_31.

Gal, Yarin and Mark van der Wilk (2014). "Variational Inference in Sparse Gaussian Process Regression and Latent Variable Models - a Gentle Tutorial". In: pp. 1–9. arXiv: 1402.1412. URL: http://arxiv.org/abs/1402.1412.

Galashov, Alexandre et al. (2019). "Meta-Learning surrogate models for sequential decision making". In: arXiv: 1903.11907. URL: http://arxiv.org/abs/1903.11907.

Girard, Agathe et al. (2002). "Gaussian Process Priors with Uncertain Inputs - Application to Multiple-Step Ahead Time Series Forecasting". In: pp. 529–536.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Gordon, Jonathan et al. (2018). "Meta-Learning Probabilistic Inference For Prediction". In: pp. 1–22. arXiv: 1805.09921. URL: http://arxiv.org/abs/1805.09921.

Goulard, M. and M. Voltz (Apr. 1992). "Linear coregionalization model: Tools for estimation and choice of cross-variogram matrix". In: *Mathematical Geology* 24.3, pp. 269–286. ISSN: 08828121. DOI: 10.1007/BF00893750. URL: https://doi.org/10.1007/BF00893750.

Gupta, Abhishek et al. (2018). "Unsupervised Meta-Learning for Reinforcement Learning". In: arXiv: arXiv:1806.04640v1.

Hensman, James, Nicoló Fusi, and Neil D Lawrence (2013). "Gaussian Processes for Big Data". In: *CoRR* abs/1309.6. arXiv: 1309.6835. URL: http://arxiv.org/abs/1309.6835.

Hoef, Jay M Ver and Ronald Paul Barry (1998). "Constructing and fitting models for cokriging and multivariable spatial prediction". In: *Journal of Statistical Planning and Inference* 69.2, pp. 275–294. ISSN: 0378-3758. DOI: https://doi.org/10.1016/S0378-3758(97)00162-6. URL: http://www.sciencedirect.com/science/article/pii/S0378375897001626.

Hotelling, Harold (1936). "Relations Between Two Sets of Variates". In: *Biometrika* 28.3/4, pp. 321–377. ISSN: 00063444. DOI: 10.2307/2333955. URL: http://www.jstor.org/stable/2333955.

Käding, Christoph and Oliver Mothes (2018). "Active Learning for Regression Tasks with Expected Model Output Changes". In:

Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore (May 1996). "Reinforcement Learning: A Survey". In: *J. Artif. Int. Res.* 4.1, pp. 237–285. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622737.1622748.

Kamthe, Sanket and Marc Peter Deisenroth (2017). "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control". In: 84. arXiv: 1706.06491. URL: http://arxiv.org/abs/1706.06491.

Kellert, S H (1993). *In the Wake of Chaos: Unpredictable Order in Dynamical Systems*. In the Wake of Chaos: Unpredictable Order in Dynamical Systems. University of Chicago Press. ISBN: 9780226429748. URL: https://books.google.co.uk/books?id=6tFroUf6PcYC.

Kendall, Alex et al. (2018). “Learning to Drive in a Day”. In: arXiv: arXiv:1807.00412v2.
Kingma, Diederik P. and Jimmy Ba (2014). “Adam: A Method for Stochastic Optimization”. In: pp. 1–15. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.
Knuth, Donald E (1997). *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-89684-2.
Kober, Jens, J Andrew Bagnell, and Jan Peters (2009). “Reinforcement Learning in Robotics : A Survey”. In:
Konyushkova, Ksenia, Raphael Sznitman, and Pascal Fua (2017). “Learning Active Learning from Data”. In: Nips. arXiv: 1703.03365. URL: http://arxiv.org/abs/1703.03365.
Konyushkova, Ksenia, Raphael Sznitman, Pascal Fua, and Data-driven Al (2019). “Discovering General-Purpose Active Learning Strategies”. In: pp. 1–10. arXiv: arXiv:1810.04114v2.
Körding, Konrad P. and Daniel M. Wolpert (2006). “Bayesian decision theory in sensorimotor control”. In: *Trends in Cognitive Sciences* 10.7, pp. 319–326. ISSN: 13646613. DOI: 10.1016/j.tics.2006.05.003.
Krause, Andreas (2010). “Lecture 17.1 Uncertainty Sampling”. In: *CS 253: Advanced Topics in Machine Learning*, pp. 1–7.
Krause, Andreas, Ajit Singh, and Carlos Guestrin (June 2008). “Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies”. In: *J. Mach. Learn. Res.* 9, pp. 235–284. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1390681.1390689.
Kuss, Malte (2006). “Gaussian Process Models for Robust Regression, Classification, and Reinforcement Learning”. In: *Ph.D. Thesis*.
Lambert, B (2018). *A Student’s Guide to Bayesian Statistics*. SAGE Publications. ISBN: 9781526418265. URL: https://books.google.co.uk/books?id=CLZBDwAAQBAJ.
Landolfi, Nicholas C, Garrett Thomas, and Tengyu Ma (2017). “A Model-based Approach for Sample-efficient Multi-task Reinforcement Learning”. In: pp. 1–13. arXiv: arXiv:1907.04964v2.
Lawrence, Neil D (2003). “Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data”. In: *Proceedings of the 16th International Conference on Neural Information Processing Systems*. NIPS’03. Cambridge, MA, USA: MIT Press, pp. 329–336. URL: http://dl.acm.org/citation.cfm?id=2981345.2981387.
Lázaro-Gredilla, Miguel (2012). “Bayesian warped Gaussian processes”. In: *Advances in Neural Information Processing Systems* 2, pp. 1619–1627. ISSN: 10495258.
Lázaro-Gredilla, Miguel et al. (2010). “Sparse Spectrum Gaussian Process Regression Joaquin Quiñonero-Candela Aníbal R. Figueiras-Vidal”. In: *Journal of Machine Learning Research* 11, pp. 1865–1881.
Levine, Sergey (2019). “Imitation, Prediction, and Model-Based Reinforcement Learning for Autonomous Driving”. In: *ICML*. URL: https://slideslive.com/38917941/imitation-prediction-and-modelbased-reinforcement-learning-for-autonomous-driving.
Mao, Hongzi et al. (2016). “Resource management with deep reinforcement learning”. In: *HotNets 2016 - Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp. 50–56. DOI: 10.1145/3005745.3005750.
Mnih, Volodymyr et al. (Feb. 2015). “Human-level control through deep reinforcement learning”. In: *Nature* 518, p. 529. URL: https://doi.org/10.1038/nature14236%20http://10.0.4.14/nature14236%20https://www.nature.com/articles/nature14236%7B%5C#%7Dsupplementary-information.
Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. Cambridge, MA.
Nagabandi, Anusha et al. (2018). “Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning”. In: pp. 1–17. arXiv: 1803.11347. URL: http://arxiv.org/abs/1803.11347.
Ng, Annalyn and Kenneth Soo (2017). *Numsense! Data Science for the Layman: No Math Added*.
Pang, Kunkun et al. (2018). “Meta-Learning Transferable Active Learning Policies by Deep”. In: pp. 1–8. arXiv: arXiv:1806.04798v1.
Pearson, Karl (Nov. 1901). *LIII. On lines and planes of closest fit to systems of points in space*. DOI: 10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720.
Perolat, Julien, Charles Beattie, and Karl Tuyls (2017). “A multi-agent reinforcement learning model of common-pool resource appropriation”. In: Nips.

Petangoda, Janith C. et al. (2019). “Disentangled Skill Embeddings for Reinforcement Learning”. In: arXiv: 1906.09223. URL: http://arxiv.org/abs/1906.09223.

Quinonero-Candela, J, Agathe Girard, and CE Rasmussen (2003). “Prediction at an Uncertain Input for Gaussian Processes and Relevance Vector Machines Application to Multiple-Step Ahead Time-Series Forecasting”. In: *Technical Report* 1. URL: http://mlg.eng.cam.ac.uk/pub/pdf/QuiGirRas03.pdf.

Quiñonero-Candela, Joaquin and Carl Edward Rasmussen (2005). “A unifying view of sparse approximate Gaussian process regression”. In: *Journal of Machine Learning Research* 6, pp. 1939–1959. ISSN: 1533-7928.

Rakelly, Kate et al. (2019). “Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables”. In: arXiv: 1903.08254. URL: http://arxiv.org/abs/1903.08254.

Rasmussen, Carl Edward; and Christopher K I Williams (2004). *Gaussian processes for machine learning.* Vol. 14, pp. 69–106. ISBN: 026218253X. DOI: 10.1142/S0129065704001899. arXiv: 026218253X.

Rios, Gonzalo and Felipe Tobar (2019). “Compositionally-Warped Gaussian Processes”. In: DOI: 10.1016/j.neunet.2019.06.012. arXiv: 1906.09665. URL: http://arxiv.org/abs/1906.09665%7B%5C%%7D0Ahttp://dx.doi.org/10.1016/j.neunet.2019.06.012.

Sæmundsson, Steindór, Katja Hofmann, and Marc Peter Deisenroth (2018). “Meta Reinforcement Learning with Latent Variable Gaussian Processes”. In: arXiv: 1803.07551. URL: http://arxiv.org/abs/1803.07551.

Scholkopf, Bernhard and Alexander J Smola (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press. ISBN: 0262194759.

Seeger, Matthias, Christopher K I Williams, and Neil D Lawrence (2003). “Fast Forward Selection to Speed Up Sparse Gaussian Process Regression”. In: *IN WORKSHOP ON AI AND STATISTICS 9*.

Settles, Burr (2009). “Active Learning Literature Survey - TR 1648”. In: *Sciences-New York*. ISSN: 0167577X. DOI: 10.1016/j.matlet.2010.11.072. arXiv: 1206.5533.

Silver, David et al. (2018). “A general reinforcement learning algorithm that masters chess , shogi and Go through self-play”. In:

Snelson, Edward, Zoubin Ghahramani, and Carl E Rasmussen (2004). “Warped Gaussian Processes”. In: *Advances in Neural Information Processing Systems 16*. Ed. by S Thrun, L K Saul, and B Schölkopf. MIT Press, pp. 337–344. URL: http://papers.nips.cc/paper/2481-warped-gaussian-processes.pdf.

Springenberg, Jost Tobias et al. (2018). “Learning an Embedding Space for Transferable Robot Skills”. In:

Tassa, Yuval et al. (2018). “DeepMind Control Suite”. In: arXiv: 1801.00690. URL: http://arxiv.org/abs/1801.00690.

Tedrake, Russ (2009). “The Acrobot and Cart-Pole”. In: *Underactuated Robotics: Learning, Planning, and Control for Efficient and Agile Machines*, pp. 22–38. URL: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-832-underactuated-robotics-spring-2009/readings/MIT6%7B%5C_%7D832s09%7B%5C_%7Dread%7B%5C_%7Dch03.pdf.

Titsias, Michalis (2009). “Variational Learning of Inducing Variables in Sparse Gaussian Processes”. In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. by David van Dyk and Max Welling. Vol. 5. Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, pp. 567–574. URL: http://proceedings.mlr.press/v5/titsias09a.html.

Vinyals, Oriol, Alexander Sasha Vezhnevets, and David Silver (2017). “StarCraft II : A New Challenge for Reinforcement Learning”. In: arXiv: arXiv:1708.04782v1.

Wackernagel, H (1998). *Multivariate Geostatistics: An Introduction with Applications*. Environmental science. Springer. ISBN: 9783540647218. URL: https://books.google.co.uk/books?id=0CnwAAAAMAAJ.

Widrow, B. and F. Smith (1964). “Pattern-recognizing control systems”. In: *J. T. Tou and R. H. Wilcox (Eds.), Computer and Information Sciences*, pp. 288–317.

Wilson, James T., Frank Hutter, and Marc Peter Deisenroth (2018). “Maximizing acquisition functions for Bayesian optimization”. In: NeurIPS. arXiv: 1805.10196. URL: http://arxiv.org/abs/1805.10196.

Wilson, James T., Riccardo Moriconi, et al. (2017). “The reparameterization trick for acquisition functions”. In: 1, pp. 1–7. arXiv: 1712.00424. URL: http://arxiv.org/abs/1712.00424.

Wu, Dongrui, Chin Teng Lin, and Jian Huang (2019). “Active learning for regression using greedy sampling”. In: *Information Sciences* 474, pp. 90–105. ISSN: 00200255. DOI: 10.1016/j.ins.2018.09.060. arXiv: arXiv:1808.04245v1.

Yu, Hwanjo and Sungchul Kim (2010). “Passive sampling for regression”. In: *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 1151–1156. ISSN: 15504786. DOI: 10.1109/ICDM.2010.9.

Zintgraf, Luisa, Maximilian Igl, et al. (2019). “Variational Task Embeddings for Fast Adaptation in Deep Reinforcement Learning”. In: *Workshop on Structures & Priors in RL (ICLR)*, pp. 1–7.

Zintgraf, Luisa, Kyriacos Shiarlis, et al. (2019). “Fast Context Adaptation via Meta-Learning”. In: 2018.