Department of Informatics
Chair for Computer Vision and Artificial Intelligence
Dynamic Vision and Learning Group

Learning Gaussian Process Dynamics Models from Visual Observations for Control

Master's Thesis by Nathanael Bosch

Examiner:	Prof. Dr. Laura Leal-Taixé
Advisor:	Dr. Jörg Stückler, Max Planck Institute for Intelligent Systems, Tübingen
Submission Date:	October 15, 2019

I hereby confirm that this is my own work, and that I used only the cited sources and materials.

Garching, October 15, 2019

Nathanael Bosch

Abstract

Planning is a powerful approach to control problems with known environment dynamics, but in unknown environments the agent first needs to learn a model for the system dynamics. This is particularly challenging when the underlying states are only indirectly observable through high-dimensional visual observations, such as images.

We propose a latent Gaussian process dynamics model that learns low-dimensional environment dynamics entirely from images. The method infers latent state representations from observations using neural networks and models the system dynamics in the learned latent space with Gaussian processes. All parts of the model can be trained jointly by optimizing a lower bound on the likelihood of transitions in image space. Additionally, we present a simplified version in which state representations and system dynamics are learned separately.

We evaluate both approaches in two environments of different complexity, Pendulum and CartPole, and show that the proposed model outperforms the separately trained method on the Pendulum environment. Further, the agent is able to use the learned latent dynamics model to efficiently solve the Pendulum swing-up task by planning in latent space. Finally, we demonstrate fast adaptation capabilites of the trained agent to environments with modified system dynamics.

Zusammenfassung

Planung ist ein leistungsfähiger Ansatz zur Steuerung von Problemen mit bekannter Umgebungsdynamik. In unbekannten Umgebungen muss der Agent jedoch zunächst ein Modell für die Systemdynamik erlernen. Dies ist insbesondere dann eine Herausforderung, wenn die zugrunde liegenden Zustände nur indirekt durch hochdimensionale visuelle Beobachtungen, wie z. B. Bilder, erkennbar sind.

In dieser Arbeit stellen wir ein latentes Dynamikmodell vor, welches vollständig aus Bildern erlernt wird. Die latenten Zustandsdarstellungen werden mit neuronalen Netzen aus Beobachtungen abgeleitet, und wir modellieren die Systemdynamik im erlernten latenten Raum mit Gaußschen Prozessen. Alle Teile des Modells können gemeinsam darauf trainiert werden, eine Untergrenze für die Wahrscheinlichkeit der Bildübergänge zu optimieren. Wir präsentieren außerdem eine vereinfachte Version unseres Modells, in der Zustandsdarstellungen und Systemdynamik separat gelernt werden.

Wir bewerten beide Ansätze in zwei Umgebungen unterschiedlicher Komplexität, “Pendulum” und “CartPole”, und zeigen, dass das vorgeschlagene Modell die separat trainierte Methode auf dem Pendelsystem übertrifft. Darüber hinaus kann der Agent mit dem erlernten Modell im latenten Raum planen und kann damit das Pendel aufschwingen und balancieren. Zuletzt demonstrieren wir schnelle Anpassungsfähigkeiten des Agenten an Umgebungen mit geänderter Systemdynamik.

Contents

Chapter 1

Introduction

1.1 Motivation

Reinforcement learning (RL) has shown success for a number of applications, including Atari games (Mnih et al., 2015), robotic manipulation (Gu et al., 2017), navigation and reasoning tasks (Oh et al., 2016), and machine translation (II et al., 2014). Many such results were obtained with model-free deep RL, where the agent directly learns a policy function, in the form of a neural network, by interacting with the environment. However, such approaches commonly require a large number of interactions, which often hinders their application to real-world tasks: Performing actions in real environments, such as driving a vehicle or moving a robot, can be orders of magnitude slower than performing an update of the policy model, and mistakes can carry real-world costs.

Model-based RL is a promising direction to reduce this sample complexity. In model-based RL, the agent acquires a predictive model of the world and uses that model to make decisions. This offers several potential benefits over model-free approaches. First, learning a transition model enables the agent to leverage a richer training signal by using the observed transition instead of just propagating a scalar reward. Further, the learned dynamics can be independent of the specified task and could therefore potentially be transferred to other tasks in the same environment. Finally, instead of learning a policy function the agent can use the learned environment for planning to chose its actions.

For environments with only a few state variables, PILCO (Deisenroth and Rasmussen, 2011) achieves remarkable sample efficiency. A crucial component is its use of Gaussian processes to model the system dynamics, which allows PILCO to include the uncertainty of the transition model into its policy search. However, in many problems of interest the underlying state of the world is only indirectly observable through high-dimensional visual observations, such as images. In order to enable fast planning, the agent can learn low-dimensional state representations and model the system dynamics in the learned latent space. Models of this type have been successfully applied to simple tasks such as balancing cartpoles and controlling 2-link arms (Banijamali et al., 2018; Watter et al., 2015). However, model-based RL approaches are generally known to lag behind model-free methods in asymptotic performance for problems of this type. Recently,

PlaNet (Hafner et al., 2019) was able to match top model-free algorithms in complex image-based domains. PlaNet learns environment dynamics from pixels and chooses actions through online planning in latent space. Notably, all components in PlaNet are modeled through neural networks.

1.2 Contribution

In this thesis, we combine Gaussian processes with neural networks to learn latent dynamics models from visual observations. All parts of the proposed model can be trained jointly to optimize a lower bound on the likelihood of transitions in image space. To unify the different data requirements and limitations of both models, we motivate a subset-of-data approximation for the marginal log-likelihood of the Gaussian process. This allows us to provide more training data to the neural network while keeping the computational cost for the Gaussian process constant. The predictions of the learned dynamics model enable the agent to successfully solve a swing-up task in the Pendulum environment. Finally, the Gaussian processes allow the agent to quickly adapt to environments with modified system dynamics.

1.3 Thesis Structure

We begin with a review of related scientific literature in Chapter 2, covering related work on planning in known state spaces and on state representation learning. Our goal is to combine findings of both fields in order to learn low-dimensional latent dynamics models from images. With this motivation we further review previous approaches that combine neural networks and Gaussian processes in a joint model.

Chapter 3 provides theoretical background on machine learning, neural networks and Gaussian processes. It covers basic terms and definitions, as well as all required concepts for the proposed method, including stochastic gradient descent, convolutional layers and variational auto-encoders.

We introduce our proposed method in Chapter 4. Section 4.1 formally defines the considered state space models and the underlying generative process. In the simpler case of fully observable states, we can model the system dynamics directly through a Gaussian process, as shown in Section 4.2. In order to approach the more difficult setting with visual observations, we propose a combination of neural networks, for state inference and generation of observations, with Gaussian processes, to model the system dynamics in latent space. We present the proposed model in Section 4.3 and derive a training objective in Section 4.4. As a comparison, Section 4.5 provides a simplified version of our model with separately learned latent states and system dynamics. In order to apply the method to planning and control, we discuss two approaches to

sample sequences from the dynamics model (Section 4.6) and then introduce the chosen online planning algorithm in Section 4.7.

Chapter 5 covers the evaluation of our method. The chapter starts with general information on the considered environments, relevant metrics, and numerical considerations for training. We then present results of both the proposed method and the separately learned model and compare these to a Gaussian process dynamics model on the true physical states. The evaluation first covers the quality and accuracy of predictions independently of any control task and we visualize and analyze the learned latent spaces. Then, we evaluate the learned models for planning and control and demonstrate model adaptation to new physical properties.

We conclude the thesis in Chapter 6 by briefly reviewing the main results and we propose multiple starting points for future improvements.

Chapter 2

Related Work

Planning in state space

When measurements of the low-dimensional physical states of the environment are available it is possible to learn the dynamics directly in state space. PILCO (Deisenroth et al., 2015; Deisenroth and Rasmussen, 2011) models the system dynamics with a Gaussian process and achieves remarkable sample efficiency. One reason for its success is the good uncertainty estimation of the Gaussian process model predictions, which can be incorporated into the long-term planning. This approach has since also been extended to partially observable Markov decision problems (McAllister and Rasmussen, 2016). Deep PILCO (Gal et al., 2016) extends PILCO's framework to use Bayesian neural network dynamics models, allowing for better scaling with the number of trials and the dimensionality of the observation space. Deep PILCO has been successfully applied for learning swimming controllers for a 6-legged autonomous underwater vehicle (Higuera et al., 2018). Doerr et al. (2017) extend the original PILCO in order to optimize the model directly with respect to the likelihood of observed trajectories, as opposed to optimizing one-step-ahead predictions, leading to better long-term predicitons and higher resilience against noisy input and output data. Finally, Chua et al. (2018) use ensembles of neural networks to model system dynamics, matching performance of model-free approaches on the cheetah running task while being data-efficient.

State representation learning

The true low-dimensional physical states are often not directly available. Instead, we can often only indirectly observe the considered dynamical system through visual observations, such as images. The goal of *state representation learning* is to infer low-dimensional latent states from these high-dimensional visual observations. The learned representation is then often used in order to predict future states or observations, as well as for planning actions.

Mattner et al. (2012) control an inverted pendulum by embedding the visual input into a two-dimensional latent space using a standard auto-encoder. However the authors do not explicitly model the system dynamics to predict future states for planning. Wahlström et al. (2014) map images to lower-dimensional latent representations with a

deep auto-encoder and learn low-dimensional system dynamics with a second neural network, training both models jointly. They evaluate the results in image prediction tasks but do not use the learned models for planning or control. Embed-to-control (Watter et al., 2015) and RCE (Banijamali et al., 2018) are approaches which consider planning and control during the learning of latent representations. They both encode images into a linearizable latent representation, which then allows for classical optimal control methods such as LQR, but has limitations for environments which are difficult to linearize. Assael et al. (2015) improve data-efficiency and training times by concatenating latent representations instead of high-dimensional images to describe the current system state. Deep Variational Bayes Filters (DVBF) (Karl et al., 2016) are able to learn and identify latent representations of the state space of a dynamical system from image sequences. In their experiments the authors use locally linear state transitions, parametrized by a neural network, and were able to show good results for the dynamic pendulum and the bouncing ball experiments. In comparison, we want to explore general non-linear system dynamics in the learned latent space. Fraccaro et al. (2017) propose to model the non-linear system dynamics with a linear Gaussian state space model (LGSSM), combined with variational auto-encoders to map the high-dimensional images to lower-dimensional latent states, allowing to compute exact posterior distributions. A thorough review of state-of-the-art approaches of the recent years to state representation learning is given by Lesort et al. (2018), with comparisons of learning objectives, model architectures, used priors, and evaluation metrics and environments. More recently, SOLAR (Zhang et al., 2018), a Bayesian latent variable model with locally linear dynamics in latent space, has shown good performance and high data-efficiency on more complex robotic tasks, including manipulation tasks on a real Sawyer robotic arm directly from camera images.

Finally, we highlight PlaNet (Hafner et al., 2019), a purely model-based agent that learns the environment dynamics from pixels and chooses actions through online planning in latent space. The authors model the non-linear latent dynamics through both a deterministic and stochastic transition function, and the variational objective directly encourages multi-step predictions. PlaNet solves control task with contact dynamics, partial observability, and sparse rewards, using significantly fewer episodes than top model-free algorithms while reaching a similar and sometimes higher final performance. The proposed architecture has similarities with our approach, but instead of modeling the system dynamics through a combination of deterministic and stochastic transition functions we want to use Gaussian processes, motivated by their earlier successes for modeling system dynamics. Additionally, we want to investigate the general task of learning system dynamics and we do not directly train for control.

Combining Gaussian processes and deep neural networks

There are many different attempts to combine Gaussian processes with neural networks, trying to get the best of both worlds. One approach to use the expressiveness and flexibility of neural networks in Gaussian processes is *deep kernel learning*, proposed by Wilson et al. (2015b). The authors first map the inputs into a different representation space with a deep neural network before applying the GP kernel function, and jointly train both models through the marginal log-likelihood of the training data. In order to scale to the large datasets which are required to train the deep kernel the authors apply stochastic kernel interpolation (Wilson et al., 2015a; Wilson and Nickisch, 2015). This has since been extended (Wilson et al., 2016) by using stochastic variational inference (Hoffman et al., 2012), allowing for multi-task classification and mini-batch training.

Chapter 3

Theoretical Foundations

3.1 Machine Learning and Regression

Machine learning is a scientific discipline which investigates methods to provide knowledge to computers through data, observations and interacting with the world. With the appropriate algorithms and statistical models computers should then use this acquired knowledge to correctly generalize to new settings. A simplified example could be to consider images of cats and dogs, with the task consisting on correctly classifying new images into one of these two categories. Real-world usage of machine learning extends far beyond such a simple example and is widely used in fields such as computer vision and natural language processing, with applications such as autonomous driving, medical imaging, finance, climate modeling, and many more.

The goal in many machine learning problems is to learn the relation between an input $x \in \mathcal{X}$ and a target $y \in \mathcal{Y}$, which is often approached by learning a function $f : \mathcal{X} \to \mathcal{Y}$. We often consider parametric functions, such as for example the linear function $f(x; w, b) = x^\mathsf{T} w + b$. In *supervised learning* we chose this function using a given dataset of pairs of inputs and targets, which we also call the *training data*, and the task could then be reformulated as learning the parameters (w, b) in such a way that best explains the training data.

In *regression* we consider continuous targets $\mathcal{Y} \subset \mathbb{R}^m$, and in *classification* the targets are discrete values $\mathcal{Y} \subset \mathbb{N}_0$, called *labels*. A different learning approach is *unsupervised learning*, where we are not given any labels y and instead want to learn about the structure of the data X, with tasks such as clustering or outlier detection, or data generation. *Reinforcement learning* describes a third approach to machine learning, where models are typically trained by interacting with an environment and receiving reward. For a thorough overview on machine learning see for example *Pattern Recognition and Machine Learning* by Bishop (2006), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* by Hastie et al. (2013), or *Machine Learning: A Probabilistic Perspective* by Murphy (2012). More specifically for reinforcement learning we refer to *Reinforcement Learning: An Introduction* by Sutton and Barto (2018).

In this thesis, two specific methods are of particular interest: Neural networks

(Section 3.2) and Gaussian processes (Section 3.3). Neural networks describe a broad class of parametric functions. On the other hand, a Gaussian process is a non-parametric Bayesian method, which uses the given training data in order to provide probabilistic estimates over function values, including uncertainties of these estimates. In the following we introduce both of these methods.

3.2 Deep Neural Networks

Artificial Neural Networks have generated a lot of excitement both in research and industry, thanks to many breakthrough results in fields such as computer vision, speech recognition, and text processing (Krizhevsky et al., 2012). This can be attributed to a large number of factors, notably the increase of computational power and the development of efficient hardware for these types of models, together with the availability of large datasets. This increase in attention in turn lead to many scientific advances in the field of deep-learning, resulting in even more breakthroughs.

Neural networks themselves have been known for many decades. A historical precursor of the neural network, a linear binary classifier called the *perceptron*, has been introduced by Rosenblatt (1958). Neural networks generalize the perceptron by replacing the binary step function with general nonlinear functions, and modern deep learning considers models with many such stacked layers, each consisting of a parametric linear function and an element-wise non-linearity. A motivation for such models is given by the "universal approximation theorem" (Cybenko, 1989), which states that a two-layer neural network with sigmoid activation functions is able to represent any continuous function up to arbitrary precision. This has since been extended to a larger class of non-linear activations (Hornik, 1991), as well as to width-bounded but deep networks with ReLU activation functions (Hanin, 2017; Lu et al., 2017).

In the following we define neural networks and introduce common terms such as *layers* and *activation functions*. We explain how the parameters of these networks can be learned with gradient-descent. Finally we introduce more specialized concepts such as *convolutional layers*, *auto-encoders* and *variational auto-encoders*, all of which we will use in later chapters of this thesis for our proposed method.

3.2.1 Single layer neural networks

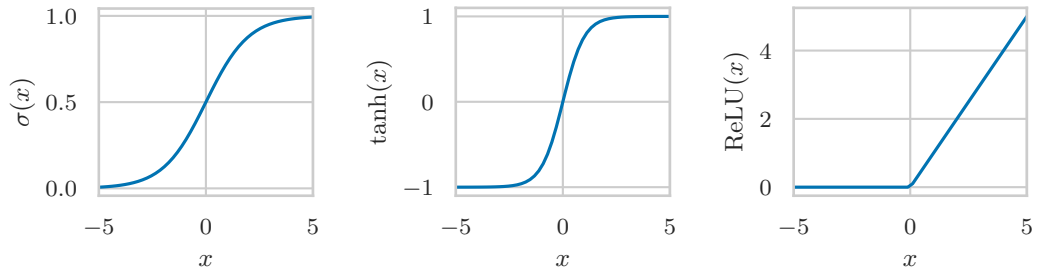
Consider a dataset consisting of inputs $X = \{x_1, \ldots, x_N\}$ and targets $\mathbf{y} = \{y_1, \ldots, y_N\}$ with $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^m$. A *dense* or *fully-connected* neural network with a single layer defines a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with the following operation:

$$f(x) = \sigma(Wx + b), \tag{3.2.1}$$

with *weight matrix* $W \in \mathbb{R}^{m \times n}$, *bias vector* $b \in \mathbb{R}^m$, and element wise application of the non-linear *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$.

3.2.2 Activation functions

Common choices of activation functions are the *sigmoid function* $\sigma(x) = \frac{1}{1+e^{-x}}$, the hyperbolic tangent $\tanh(x) = \frac{e^{2x}+1}{e^{2x}-1}$ and the *rectifier linear unit* (ReLU) $\text{relu}(x) = \max(x, 0)$ (Glorot et al., 2011), shown in Fig. 3.1. More recently proposed activation functions include the the leaky ReLU (Xu et al., 2015), the parametrized ReLU (He et al., 2015), Swish (Ramachandran et al., 2017), or the exponential linear unit (ELU) (Clevert et al., 2015).



(a) Sigmoid **(b)** tanh **(c)** ReLU

Figure 3.1: Examples of some commonly used activation functions.

3.2.3 Multi-layered neural networks

Deep neural networks consist of multiple layers $f_1, \dots, f_L$, each defined as in Eq. (3.2.1) with weights $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$, biases $b_l \in \mathbb{R}^{d_l}$ and activation function $\sigma_l : \mathbb{R} \to \mathbb{R}$, for each $l \in \{1, \dots, L\}$. Note that $d_0 = n$ and $d_L = m$ in order to match the dimensions of the dataset. A deep neural network is then defined as the sequential application of these layers to the input data, leading to a function of the form

$$f(x) = f_L\left(f_{L-1}\left(\dots\left(f_1\left(x\right)\right)\right)\right). \tag{3.2.2}$$

We compactly denote the set all parameters of the neural network with $\theta = \{W_l, b_l\}_{l=1}^{L}$. Note that while the activation functions $\{\sigma_l\}_{l=1}^{L}$ can be chosen independently from each other, it is often done in practice to consider a common activation function σ for all but the last layer.

3.2.4 Gradient-based learning

In a *supervised regression* task we consider a dataset of inputs $X = \{x_1, \dots, x_N\}$ and targets $\mathbf{y} = \{y_1, \dots, y_N\}$ with $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^m$, and we want to find a function $f : \mathbb{R}^n \to \mathbb{R}^m, x \mapsto \hat{y}$ which best explains the data. We approach this task by defining

a *loss function* to measure the dissimilarity of the predicted targets to the real targets on the dataset, such as for example the mean-squared error or L_2-loss

$$L(\mathbf{y}, f(X, \theta)) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - f(x_i, \theta)\|_2^2 , \tag{3.2.3}$$

where we write $f(X, \theta) := \{f(x_i, \theta)\}_{i=1}^N$ as the element-wise application of the neural network to each input.

Assume that the chosen loss function is fully differentiable with respect to $f(x_i, \theta)$ for all $i \in \{1, \dots, N\}$, and note that the deep neural network f is differentiable (almost everywhere) with respect to θ. We can thus compute the gradient of $L(\mathbf{y}, f(X, \theta))$ with respect to θ and we therefore chose to minimize this function using a *gradient descent* based optimization scheme.

In practice it is common to perform *stochastic gradient descent* (Bottou, 2010), where instead of computing the gradient over the whole dataset at once, we consider subsets of the data called *batches* and perform a gradient steps for each batch. This circumvents hardware limitations which one might encounter when training on very large datasets, but it has also been shown to improve the stability of the training (Ge et al., 2015; Masters and Luschi, 2018). We call a single training iteration over all batches an *epoch*. Further, gradient descent methods with momentum, most notably the Adam algorithm (Kingma and Ba, 2014), are very common in deep learning research.

3.2.5 Convolutional layers

In *fully-connected* layers as defined in Eq. (3.2.1) we consider the full input vector x for the computation of each single output dimension $f(x)_i$. In contrast, *convolutional layers* (LeCun et al., 1989; LeCun et al., 1998) define more structure in the weight matrix W by considering sparse matrices which share values in multiple entries. A practical motivation in the context of computer vision might be the concept of *translation invariance*: Shifting the whole input image by a single pixel should not lead to large changes in the output. Convolutional layers convolve the input data with filters, as depicted in Fig. 3.2, the parameters of which are learned during the training process. This convolution operation is thus translation equivariant. As with fully-connected layers we add a bias term to the resulting output and apply a non-linear activation function element-wise. Used in combination with *pooling layers*, which effectively downsample a tensor by aggregating values, we obtain neural network architectures which are invariant to small translations.

Convolutional layers are commonly employed in many image processing tasks and most architectures contain convolutional layers at least in the first few layers of the network.

In *image generation* we consider the inverse case: The dataset consists of images as targets and often of lower-dimensional vectors as inputs. In order to upsample

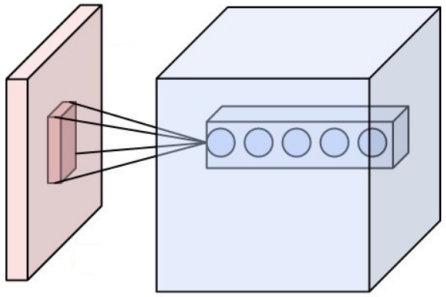


Figure 3.2: Neurons of a convolutional layer (blue) connected to their receptive field (red).
Source: `https://upload.wikimedia.org/wikipedia/commons/6/68/Conv_layer.png` by Aphex34 [CC BY-SA 4.0 (`https://creativecommons.org/licenses/by-sa/4.0`)]

low-dimensional representations into higher-dimensional images we can use *transposed convolutional layers*, which describe an operation with the same connectivity as a normal convolution but in the backward direction. This is also commonly known as *deconvolutional layer*, but note that this name name is misleading, since it does not provide an inverse operation to a given convolution. For more thorough definitions of both the convolution and tranposed convolution, as well as for visual examples, see (Dumoulin and Visin, 2016).

3.2.6 Auto-encoders

Auto-encoders are deep neural networks which are designed in a way to learn compact low-dimensional representations of data.

Given a dataset $X = \{x_1, \dots, x_N\}$ of inputs $x_i \in \mathbb{R}^n$ we consider a reconstruction task where the targets correspond to the data itself, that is $y_i = x_i$. In order to learn low-dimensional data representations we further consider a neural network $f : \mathbb{R}^n \to \mathbb{R}^n$ of the form $f(x) = f_d(f_e(x))$ with *encoder* $f_e : \mathbb{R}^n \to \mathbb{R}^m$ and *decoder* $f_d : \mathbb{R}^m \to \mathbb{R}^n$. We then train the network to learn to reconstruct the original data X by minimizing a loss function, such as the mean-squared error (MSE)

$$L_{\text{MSE}}(x, f(x)) = \frac{1}{N} \sum_{i=1}^{N} \|x_i - f(x_i, \theta)\|_2^2 , \tag{3.2.4}$$

or the binary cross-entropy (BCE) loss

$$L_{\text{BCE}}(x, f(x)) = \sum_i -(x_i \log(f(x)_i) + (1 - x_i) \log(1 - f(x)_i)). \tag{3.2.5}$$

By choosing $m \ll n$ the network needs to compress each data-point x_i into an m-dimensional *latent* representation $h_i := f_e(x_i)$ in such a way to allow for good reconstructions of the decoder. See Fig. 3.3 for a visualization of this model architecture.

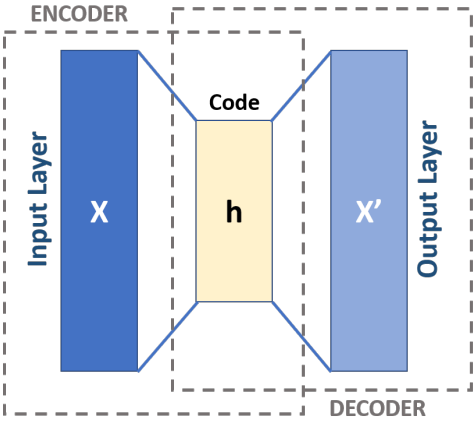


Figure 3.3: Schematic illustration of an auto-encoder. Both the encoder and decoder are deep neural networks with many layers. A main characteristic of an autoencoder is the *bottleneck*-layer in which the network has to encode all necessary information for reconstruction into a lower-dimensional representation h.
Source: `https://commons.wikimedia.org/wiki/File:Autoencoder_schema.png` by Michela Massi [CC BY-SA 4.0 (`https://creativecommons.org/licenses/by-sa/4.0`)]

To provide a small example consider a dataset consisting of images of hand-written digits, each with 16×16 pixels. Each datapoint x_i is thus a 256-dimensional vector, but it is well conceivable that the contained information could be described in a more compact way. Knowing just the shown digit might already allow for reasonable reconstructions of x_i. With further information regarding the writing style, such as how curly it is drawn or how cursively it is oriented, it might be possible to allow for near-perfect reconstructions.

Note that we can not control how exactly the network chooses to compress the original information, or which meaning the latent representations might have, since the objective of the network consists only of minimizing the chosen reconstruction loss. In the following section we present a different approach motivated by learning the underlying generative model.

3.2.7 Variational auto-encoders

Variational auto-encoders (VAEs) (Kingma and Welling, 2013; Rezende et al., 2014) seem similar to auto-encoders from a neural network perspective: They also consist of an encoder, which maps a datapoint to a lower-dimensional hidden representation, and a decoder, which tries to reconstruct the original datapoint. However, VAEs consider the underlying generative process and describe a more probabilistic approach.

Consider data $X = \{x_i\}_{i=1}^n$, consisting of i.i.d. samples of some random variable x. Assume the data is generated from an unobserved random variable z by first sampling z_i from the true prior distribution $p_{\theta^*}(z_i)$, then generating x_i from the conditional likelihood $p_{\theta^*}(x_i|z_i)$, assuming that the prior $p_{\theta^*}(z_i)$ and likelihood $p_{\theta^*}(x_i|z_i)$ come from parametric families of distributions $p_\theta(z_i)$ and $p_\theta(x_i|z_i)$ with parameter θ. We are interested in learning the true generative distribution $p(x)$. Unfortunately, we cannot compute the marginal likelihood directly since the integral $\int p_\theta(x|z)p_\theta(z)\mathrm{d}z$ is intractable. Additionally the true posterior distribution $p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$ is also intractable.

One approach to such problems, which will lead us to VAEs, is with *variational inference*. The main idea of variational inference is to approximate an intractable distribution, here $p_\theta(z|x)$, with some parametrized distribution $q_\phi(z|x)$. We then optimize the variational parameters ϕ of $q_\phi(z|w)$ such that the Kullback-Leibler divergence between the approximating variational distribution and the true posterior distribution is minimized.

We start by deriving a decomposition of the KL-divergence $D_{\mathrm{KL}}\left[q_\phi(z|x_i)||p_\theta(z|x_i)\right]$ between the variational distribution $q_\phi(z|x_i)$ and the true intractable $p_\theta(z|x_i)$:

$$
\begin{aligned}
D_{\mathrm{KL}}\left[q_\phi(z|x_i)||p_\theta(z|x_i)\right] &= \int q_\phi(z|x_i)\log\left(\frac{q_\phi(z|x_i)}{p_\theta(z|x_i)}\right)\mathrm{d}z && (3.2.6)\\
&= \int q_\phi(z|x_i)\log\left(\frac{q_\phi(z|x_i)}{\frac{p_\theta(x_i|z)p_\theta(z)}{p_\theta(x_i)}}\right)\mathrm{d}z && (3.2.7)\\
&= \int q_\phi(z|x_i)\log\left(\frac{q_\phi(z|x_i)}{p_\theta(z)}\right)\mathrm{d}z && (3.2.8)\\
&\quad - \int q_\phi(z|x_i)\log\left(p_\theta(x_i|z)\right)\mathrm{d}z && (3.2.9)\\
&\quad + \int q_\phi(z|x_i)\log\left(p_\theta(x_i)\right)\mathrm{d}z && (3.2.10)\\
&= \underbrace{D_{\mathrm{KL}}\left[q_\phi(z|x_i)||p_\theta(z)\right] - \mathbb{E}_{q_\phi(z|x_i)}\left[\log p_\theta(x_i|z)\right]}_{=:-\mathcal{L}(\theta,\phi;x_i)} && (3.2.11)\\
&\quad + \log\left(p_\theta(x_i)\right). && (3.2.12)
\end{aligned}
$$

This gives

$$
\log\left(p_\theta(x_i)\right) = D_{\mathrm{KL}}\left[q_\phi(z|x_i)||p_\theta(z|x_i)\right] + \mathcal{L}\left(\theta,\phi;x_i\right), \tag{3.2.13}
$$

with $\mathcal{L}\left(\theta,\phi;x_i\right)$ as defined previously. Since the KL-divergence is always positive, that is $D_{\mathrm{KL}}\left[q_\phi(z|x_i)||p_\theta(z|x_i)\right] \geq 0$, we get

$$
\log\left(p_\theta(x_i)\right) \geq \mathcal{L}\left(\theta,\phi;x_i\right). \tag{3.2.14}
$$

Further note that $\log p_\theta(x_1, \ldots, x_n) = \sum_{i=1}^{N} \log p_\theta(x_i)$, and thus

$$\log p_\theta(x_1, \ldots, x_n) \geq \sum_{i=1}^{N} \mathcal{L}(\theta, \phi; x_i) =: \mathcal{L}(\theta, \phi; X). \tag{3.2.15}$$

In order to maximize the data likelihood $p_\theta(x_1, \ldots, x_n)$ we want to optimize the lower bound $\mathcal{L}(\theta, \phi; X)$. We also call $\mathcal{L}(\theta, \phi; X)$ the *evidence lower bound* or ELBO. The ELBO contains two separate optimization objectives. In order to maximize the ELBO we need to minimize the KL divergence between the variational distribution and the prior distribution while maximizing the expected log-likelihood.

For insights into the general case and into the estimation of gradients of this lower bound we refer to (Kingma and Welling, 2013). In the following we will describe a special case of this setting where distributions are parametrized by neural networks.

We assume a centered isotropic multivariate Gaussian prior $p_\theta(z) = \mathcal{N}(z; 0, I)$, as well as a multivariate Gaussian $p_\theta(x|z)$ whose distribution parameters are computed from z with a neural network. We denote the dimensionality of the latent space with J. Note that the true posterior $p_\theta(z|x)$ is intractable. We further assume the true (but intractable) posterior takes on a approximate Gaussian form with an approximately diagonal covariance, and chose the variational distribution

$$\log q_\phi(z|x_i) = \log \mathcal{N}(z; \mu(x_i), \sigma(x_i)^2 I), \tag{3.2.16}$$

with $\mu(x_i), \sigma(x_i)$ being outputs of the encoding MLP. In this specific case we can estimate the ELBO with

$$\mathcal{L}(\theta, \phi; x_i) \simeq \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log(\sigma(x_i)_j^2) - \mu(x_i)_j^2 - \sigma(x_i)_j^2\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(x_i|z_i^{(l)}), \tag{3.2.17}$$

where $z_i^{(l)} = \mu(x_i) + \sigma(x_i) \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim \mathcal{N}(0, I)$. We call this sampling procedure, where we sample z by reparametrizing a standard normal sample ϵ, the *reparametrization-trick*. Note that a single sample, that is $L = 1$, is often sufficient in practice to estimate and optimize this lower bound. For the full derivation we refer to (Kingma and Welling, 2013).

Going back to a neural network perspective and considering the previously defined auto-encoders, we can interpret the training objective in Eq. (3.2.11), or more concretely for the neural network case in Eq. (3.2.17), as consisting of two parts. We first have a log-likelihood term $\mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i|z)]$, which essentially describes the reconstruction of the data point x_i. Assuming a Gaussian decoder with fixed variance, maximizing this likelihood is equivalent to minimizing the MSE between the reconstruction and the original x_i. This term thus corresponds to the AE objective, while considering the additional sampling step of the latent z_i. The second term $D_{\mathrm{KL}}[q_\phi(z|x_i)||p_\theta(z)]$ can be seen as a regularization term for the encoder network. In comparison to AEs, VAEs are commonly considered to produce more structured latent representations.

3.2.8 β-VAEs

Higgins et al. (2017) proposed the β-VAE, a modification of the variational auto-encoder. The authors introduce an adjustable hyperparameter β to balance latent channel capacity and independence constraints with reconstruction accuracy. This leads to a lower bound of the form

$$\mathcal{L}(\theta, \phi; x_i) = \mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - \beta D_{\mathrm{KL}}[q_\phi(z|x_i)||p_\theta(z)]. \tag{3.2.18}$$

Note that $\beta = 1$ corresponds to the standard VAE as previously defined. The authors demonstrate that β-VAE with appropriately tuned $\beta > 1$ qualitatively outperform VAE and argue for improved disentanglement in the learned latent representations.

3.3 Gaussian Processes

A Gaussian process (GP) can be seen as a generalization of the Gaussian probability distribution. Whereas a probability *distribution* describes random variables which are scalars or vectors, a stochastic *process* describes functions. By focussing on processes which are Gaussian, it turns out that many computations required for inference and learning become relatively easy and computationally tractable. Thinking of supervised learning as learning a function from examples can be cast directly into the Gaussian process framework as inference of a Gaussian process conditioned on the examples. Interestingly, Gaussian processes are mathematically equivalent to many well known models, including Bayesian linear models (Rasmussen and Williams, 2005), spline models (Kimeldorf and Wahba, 1970), large neural networks (under suitable conditions) (Lee et al., 2017; Neal, 1996; Novak et al., 2018; Williams, 1997), and are closely related to others, such as support vector machines (Seeger, 2002). In contrast to the previously described neural networks, Gaussian processes provide an estimate of their own uncertainty.

In the following we provide a brief introduction to Gaussian processes. We provide a formal definition, explain the conditioning of the GP over given training data, and describe the inference process while providing visual examples. We further show how we select models which maximize the data likelihood and how we use additional prior knowledge. We conclude by pointing to current research directions of interest.

3.3.1 Definition

We follow the introduction given in Rasmussen and Williams (2005, Section 2.2).

Definition 3.1 (Rasmussen and Williams (2005, Definition 2.1))
A *Gaussian process* describes a distribution over functions, such that any finite number of function values have a joint Gaussian distribution.

A Gaussian process f is completely specified by its *mean function* μ and *kernel function* or *covariance function* k, defined by

$$\mu(x) = \mathbb{E}\left[f(x)\right] \tag{3.3.1}$$

$$k(x, x') = \mathbb{E}\left[\left(f(x) - \mu(x)\right)\left(f(x') - \mu(x')\right)\right]. \tag{3.3.2}$$

We write the Gaussian process as $f \sim \mathcal{GP}(\mu, k)$.

Consider a finite number of input points $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathbb{R}^d$. The resulting function values $f(X) := [f(x_1), \ldots, f(x_n)]^T$ are then distributed according to a multivariate Gaussian distribution

$$f(X) \sim \mathcal{N}(\mu(X), K(X, X)), \tag{3.3.3}$$

with mean vector $\mu(X) = [\mu(x_1), \ldots, \mu(x_i)]^\mathsf{T}$ and covariance matrix $(K(X, X))_{ij} = k(x_i, x_j)$.

It is often sufficient to choose μ to be zero, since we can normalize the given dataset to have zero mean and unit standard-deviation. Further, having a fixed deterministic mean function is equivalent to modeling the difference between the observation and the mean function with a zero-mean Gaussian process (Rasmussen and Williams, 2005, Section 2.7). In the following, since we make use of this property for the GP dynamics models introduced in Section 4.2, we consider this case of a deterministic mean function.

Example 3.2 (Gaussian process with *squared exponential* kernel)
Consider a GP $f \sim \mathcal{GP}(0, k)$ with zero mean and the *squared exponential* function as covariance function, defined as follows:

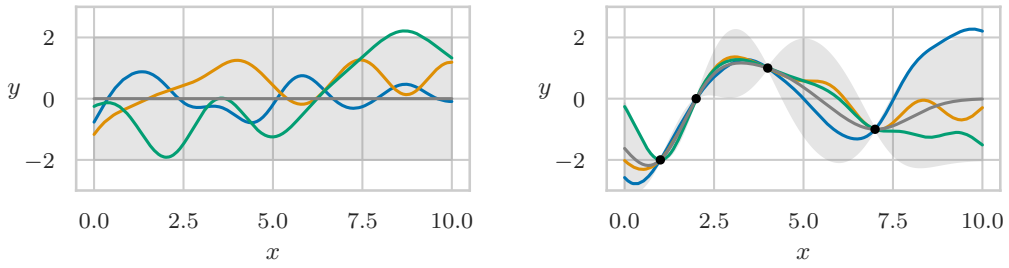
$$k(x_i, x_j) = \exp\left(-\frac{1}{2}\left\|x_i - x_j\right\|^2\right). \tag{3.3.4}$$

Given a collection of input points X_*, here the discretized one-dimensional interval $X_* = [0, 10] \cap 0.01 \cdot \mathbb{Z}$, we can compute the covariance matrix $K(X_*, X_*))$. We can then write the distribution over function values $\mathbf{f}_* = f(X_*)$ as $\mathbf{f}_* \sim \mathcal{N}(0, K(X_*, X_*))$. Finally, we can sample from this distribution to get different realizations of the Gaussian process. Figure 3.4a visualizes this example and shows three such samples, together with the mean function and the 95% confidence interval for the function values.

3.3.2 Prediction with noise-free observations

Instead of randomly sampling from the prior, we want to incorporate knowledge from training data. Consider a dataset of inputs and noise-free observations $\mathcal{D} = \{x_i, f_i\}_{i=1}^n$, with $f_i = f(x_i)$. In order to predict function values $f(X_*)$ over some test inputs X_* we write the joint distribution over training outputs $\mathbf{f} := f(X)$ and test outputs $\mathbf{f}_* := f(X_*)$ as

$$\begin{bmatrix}\mathbf{f} \\ \mathbf{f}_*\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mu(X) \\ \mu(X_*)\end{bmatrix}, \begin{bmatrix}K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*)\end{bmatrix}\right). \tag{3.3.5}$$



(a) **(b)**

Figure 3.4: Three different samples from a GP with squared exponential kernel. (a) shows samples from the prior distribution. In (b) we sampled from the posterior distribution, conditioned on the data shown as black dots. The shaded area represents the pointwise mean plus and minus two times the standard deviation for each input value, corresponding to the 95% confidence region of the underlying function values $\mathbf{f}_*$.

We can condition this joint Gaussian distribution on the known function values $\mathbf{f}$, leading to

$$\begin{aligned}\mathbf{f}_*|\mathbf{f} \sim \mathcal{N}\Big(&\mu(X_*) + K(X_*, X)K(X, X)^{-1}\left(\mathbf{f} - \mu(X)\right), \\ &K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*)\Big),\end{aligned} \tag{3.3.6}$$

where we took into account the possibly non-zero mean function μ. For more details on Gaussian identities see for example (Rasmussen and Williams, 2005, Section A.2).

Since these expressions contain many terms such as $K(X, X)$, $K(X, X_*)$, and $K(X_*, X_*)$, we will compactly denote these with $K = K(X, X)$, $K_* = K(X, X_*)$, and $K_{**} = K(X_*, X_*)$. Similarly, we write $\boldsymbol{\mu} = \mu(X)$ and $\boldsymbol{\mu}_* = \mu(X_*)$. We can then write Eq. (3.3.6) more compactly as

$$\mathbf{f}_*|\mathbf{f} \sim \mathcal{N}\Big(\boldsymbol{\mu}_* + K_*^\mathsf{T}K^{-1}\left(\mathbf{f} - \boldsymbol{\mu}\right), K_{**} - K_*^\mathsf{T}K^{-1}K_*\Big).\Big), \tag{3.3.7}$$

For any given set of test inputs X_* and training data $(X, \mathbf{f})$ we can then calculate the mean and covariance of this distribution and sample from it. Figure 3.4b shows such an example, where we again consider a discretized interval of test inputs $X_* = [0, 10] \cap 0.01 \cdot \mathbb{Z}$, but this time we condition on training data $\mathcal{D} = \{(1, -2), (2, 0), (4, 1), (7, -1)\}$ shown as black dots in the image.

3.3.3 Prediction with noisy observations

We now consider a dataset of inputs and noisy observations $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$, and we assume the relationship

$$y_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_n^2). \tag{3.3.8}$$

We still consider f to be a Gaussian process $f \sim \mathcal{GP}(\mu, k)$. Even though we do not explicitly know the true function values $\mathbf{f} = f(X)$ we still have $\mathbf{f}|X \sim \mathcal{N}(\boldsymbol{\mu}, K)$, by definition of a GP. Using $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$ we can use the formula for products of Gaussians (see again (Rasmussen and Williams, 2005, Section A.2) for more detailed information on Gaussian identities) to get $\mathbf{y}|X \sim \mathcal{N}(\boldsymbol{\mu}, K + \sigma_n^2 I)$.

In order to do inference we we again first write a joint distribution over the given training data and the test values of interest, this time accounting for the noise, and we get

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} K + \sigma_n^2 I & K_* \\ K_*^\mathsf{T} & K_{**} \end{bmatrix} \right). \tag{3.3.9}$$

This leads to the conditional distribution

$$\mathbf{f}_*|\mathbf{y} \sim \mathcal{N}\left(\bar{\mathbf{f}}_*, \operatorname{cov}(\mathbf{f}_*)\right), \tag{3.3.10}$$

where

$$\begin{aligned} \bar{\mathbf{f}}_* &= \boldsymbol{\mu}_* + K_*^\mathsf{T}[K + \sigma_n^2 I]^{-1}(\mathbf{y} - \boldsymbol{\mu}) \\ \operatorname{cov}(\mathbf{f}_*) &= K_{**} - K_*^\mathsf{T}[K + \sigma_n^2 I]^{-1} K_*. \end{aligned} \tag{3.3.11}$$

Equation (3.3.11) shows an interesting property of the Gaussian distribution: The covariance does not depend on the observed targets but only on the inputs. The term consists of the prior covariance K_{**}, minus a term which represents the information the observations gives us about the function. We can further very simply compute the predictive distribution of test targets y_* by adding a term $\sigma_n^2 I$ to the variance.

With Eqs. (3.3.10) and (3.3.11) we can calculate the resulting distribution and sample from it. Figure 3.5 visualizes the mean and variance of the result and shows three samples of this posterior distribution.

3.3.4 Parametrized kernels

In order to add more flexibility to the introduced Gaussian processes, we typically consider covariance functions with some free parameters. A variation of the squared exponential kernel function (see Eq. (3.3.4)) with additional parameters is the *radial basis function* (RBF) given by

$$k(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(x_i - x_j)^\mathsf{T} \Lambda^{-2} (x_i - x_j)\right), \tag{3.3.12}$$

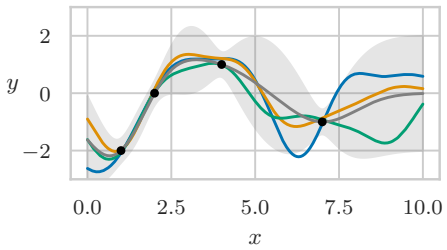


Figure 3.5: Samples from the posterior distribution of a GP conditioned on noisy data. The shaded area represents the pointwise mean plus and minus two times the standard deviation for each input value, corresponding to the 95% confidence region of the underlying function values $\mathbf{f}_*$.

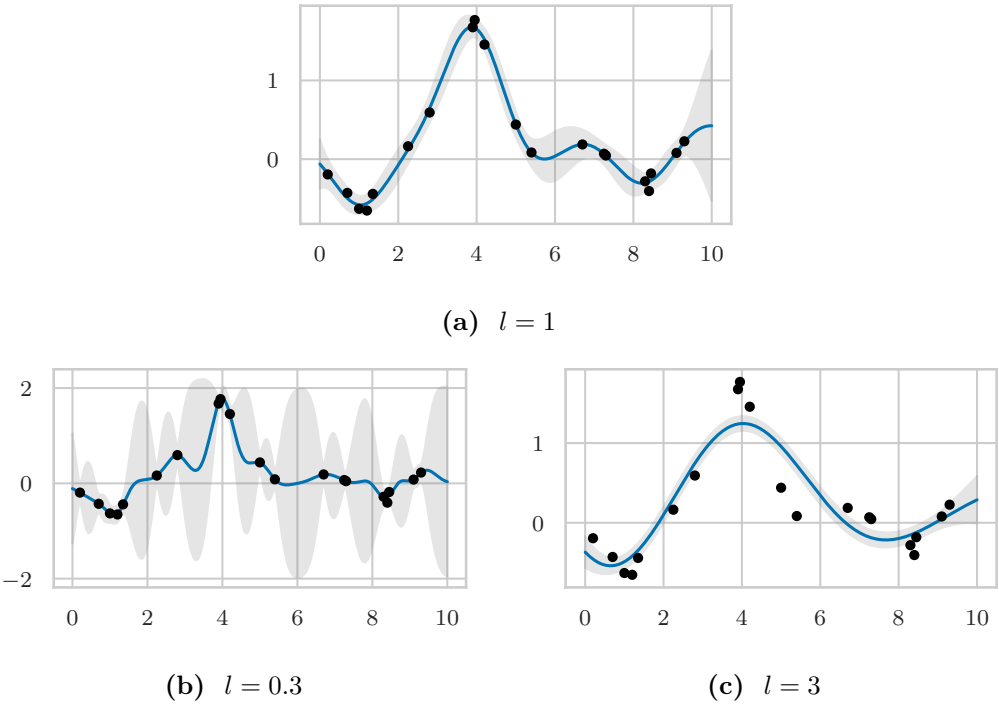
with diagonal Λ. We call σ_f^2 the *outputscale* and $(\Lambda)_{ii} = l_i$ the *lengthscales* of the kernel. When $\Lambda \neq lI$ we say that we use *automatic relevance detection* (ARD), since each input dimension is separately considered and could in theory be "turned off" by diverging $l_i \to \infty$.

For each choice of signal variance σ_f^2, lengthscales $\{l_i\}_i$ and noise variance σ_n^2 we obtain a different GP. We call these parameters the *hyperparameters* of the Gaussian process. As an example, we visualize the effect of the lengthscale parameter on the resulting GP in Fig. 3.6. The figure shows three GPs conditioned on the same set of training data, shown as black dots. In Fig. 3.6b we show the posterior of a Gaussian process with a smaller lengthscale of $l = 0.3$, all other hyperparameters being the same. The resulting model seems very flexible and is able to fit the points very well, but it seems to *overfit* the training data. The uncertainty between points is significantly larger than in the original GP. Basically, a shorter lengthscale means that the considered neighborhood for each test input is smaller, thus the prediction falls back to the prior distribution instead of interpolating between the training points. On the other hand, Fig. 3.6c shows an increased lengthscale and the resulting function varies more slowly in x. The curve looks smooth but it does not fit the points well and, in particular, the provided uncertainty estimates do not seem accurate.

3.3.5 Hyperparameter optimization

We want to find the hyperparameters θ of some Gaussian process $f \sim \mathcal{GP}(\mu, k)$ in such a way that the resulting GP is able to best describe some given, typically noisy, training data $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$. For example, if we consider a GP with zero mean and an RBF kernel the set of hyperparameters would consist of $\theta = \{\sigma_n^2, \sigma_f^2, \Lambda\}$.

Recall that $\mathbf{y}|\mathbf{f}, \theta \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$ and $\mathbf{f}|X, \theta \sim \mathcal{N}(\mu, K)$, leading to $\mathbf{y}|X, \theta \sim \mathcal{N}(\mu, K + \sigma_n^2 I)$, as shown in Section 3.3.3.



(a) $l = 1$

(b) $l = 0.3$

(c) $l = 3$

Figure 3.6: Comparison of different lengthscales. Panel (a) shows the GP from which we generated the data. Panels (b) and (c) show two different hyperparameter settings. The blue line shows the mean of the GP, and the gray area indicates the 95% interval for the underlying function f.

We can write the log-likelihood of $\mathbf{y}|X$ as

$$\log p(\mathbf{y}|X,\theta) = -\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^\mathsf{T}\left(K+\sigma_n^2 I\right)^{-1}(\mathbf{y}-\boldsymbol{\mu}) - \frac{1}{2}\log\left|K+\sigma_n^2 I\right| - \frac{n}{2}\log 2\pi. \tag{3.3.13}$$

We call this term the *marginal log-likelihood* (MLL) of the GP on the given training data. Note that the kernel matrix K depends on the hyperparameters of the GP. Further note that we consider the log-probability since it can be computed more easily and is often numerically more stable.

The different terms of Eq. (3.3.13) can be interpreted as an automatic trade-off of data-fit and model complexity. The only term involving the observed targets is $-(\mathbf{y}-\boldsymbol{\mu})^\mathsf{T}\left(K+\sigma_n^2 I\right)^{-1}(\mathbf{y}-\boldsymbol{\mu})$ describing the ability of the model to fit the data. On the other hand $\log|K+\sigma_n^2 I|$ grows as the covariance deviates from the identity matrix, therefore penalizing model complexity. This is a very interesting and remarkable property of Gaussian processes, and many other regression methods, in particular neural networks, are not able to consider such a trade-off automatically and therefore

tend to *overfit* if given enough training iterations. For a more complete discussion of this property in Gaussian processes, including visualizations of the different terms, see Rasmussen and Williams (2005, Section 5.4.1).

We want to chose the hyperparameters θ^* such that the MLL is maximized:

$$\theta^* = \arg\max_{\theta} \log p(\mathbf{y}|X, \theta), \tag{3.3.14}$$

with the MLL as in Eq. (3.3.13). Assuming a differentiable mean function μ and kernel k, the marginal log-likelihood is fully differentiable with respect to the hyperparameters θ. We can then maximize the MLL using a simple gradient ascent algorithm, or any other gradient-based optimizer (see also Section 3.2.4 for more information on gradient descent). Note that the marginal log-likelihood is non-convex in the parameters θ and that there often are multiple local optima. Gradient descent therefore does not provide a convergence guarantee. However these different optima can often be interpreted and correspond to particular interpretations of the data. One possibility of preventing convergence to bad local optima is to introduce prior belief over the parameters. We will discuss this approach in Section 3.3.6.

The computation of the marginal log-likelihood in Eq. (3.3.13) requires the inverse of the kernel matrix K. The typical computation uses the Cholesky decomposition, which requires $O(n^3)$ computations for training data of size n. This is not an issue for small datasets and can be easily computed, but it does not scale to larger datasets. There exist many different approaches to improve this issue, some of which we state briefly in Section 3.3.7. For a more thorough discussion see for example Liu et al. (2018).

3.3.6 Prior belief over hyperparameters

One of the advantages of GPs and of Bayesian approaches in general is that we can naturally incorporate some prior belief $p(\theta)$ over the hyperparameters. By doing so we can guide the MLL optimization towards preferable optima. However, it is important to note that we can also decrease the model performance by choosing unsuitable prior distributions.

The posterior distribution over the hyperparameters can be computed with Bayes' rule as

$$p(\theta|X, \mathbf{y}) = \frac{p(\mathbf{y}|X, \theta)p(\theta)}{p(\mathbf{y}|X)}. \tag{3.3.15}$$

Note that the normalization term $p(\mathbf{y}|X)$ does not depend on the hyperparameters θ. We can therefore compute the maximum a posteriori (MAP) estimate of the hyperparameters equivalently with

$$\theta^* = \arg\max_{\theta} \log p(\theta|X, \mathbf{y}) = \arg\max_{\theta} \log p(\mathbf{y}|X, \theta) + \log p(\theta). \tag{3.3.16}$$

The first term corresponds exactly to the MLL as computed in Eq. (3.3.13). The incorporation of prior information on the hyperparameters is therefore very straightforward and easy to implement: One simply has to add the log-probabilities of the hyperparameters to the marginal log-likelihood.

The choice of prior distribution is very flexible. A simple approach could be to use a uniform prior, or a smooth approximation thereof, but it does not provide meaningful gradients for most of the considered state space with mostly constant log-probabilities. A Gaussian distribution can also be a sensible choice, but most of the considered hyperparameters are bound to be positive while the Gaussian distribution allows for negative values. We therefore chose the Gamma distribution $\Gamma(\alpha, \beta)$ as the prior over hyperparameters, with density function defined as follows:

$$p(x; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}, \tag{3.3.17}$$

where $\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} \mathrm{d}x$ is the gamma function. We call α the *concentration* and β the *scale* of the Gamma distribution.

For a visualization of different Gamma prior distributions over lengthscales and their corresponding optimization results see Fig. 3.7.

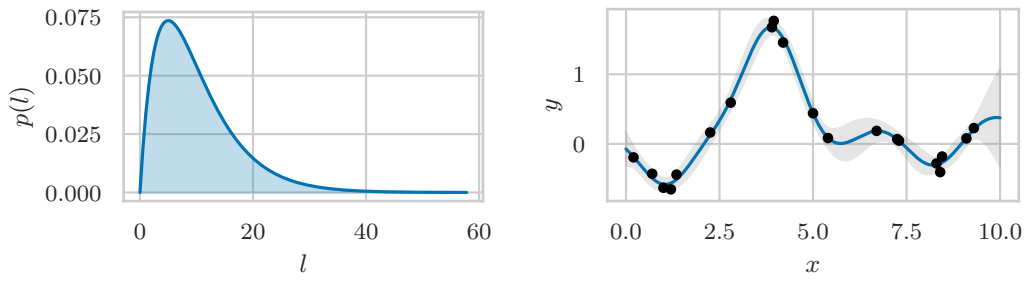
3.3.7 Extensions and current research

We conclude our brief introduction to Gaussian processes by discussing some extensions of standard Gaussian processes.

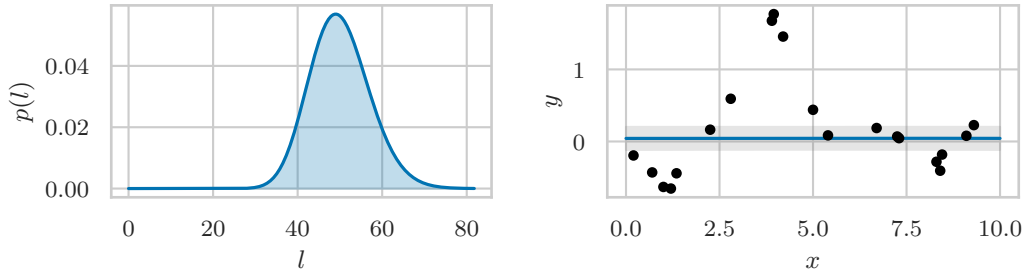
A very actively researched problem is the scaling of Gaussian processes to larger datasets. Liu et al. (2018) provide a general overview on the many different approaches. Some examples of such approaches include using only on a subset of the full training data (Hayashi et al., 2019), sparse approximations of the kernel matrix (Williams and Seeger, 2001), or by introducing a smaller set of inducing points (Snelson and Ghahramani, 2007). It is also possible to exploit structure in the kernel matrix (Wilson and Nickisch, 2015).

Notably, Titsias (2009) considered inducing points from a variational perspective. Together with stochastic variational inference (Hoffman et al., 2012) this allowed for training such a sparse variational GP on mini batches of data (Hensman et al., 2013).

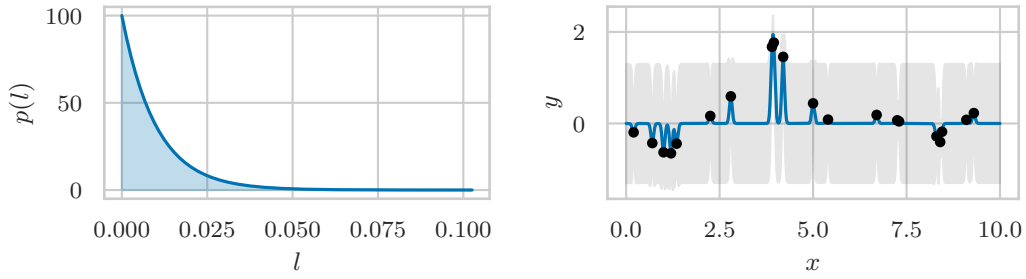
From a different perspective, there have been many approaches regarding the combination of neural networks and Gaussian processes. While both methods seem very different (deterministic vs. probabilistic, large data vs. small data) there has been ongoing research regarding their equivalence (Lee et al., 2017; Neal, 1996; Novak et al., 2018; Williams, 1997). One approach to combine neural networks with Gaussian processes is *deep kernel learning* (Wilson et al., 2015b), where the authors propose to first map the inputs into a different representation space with a deep neural network before applying the GP kernel function. This has since been combined with stochastic



(a) Gamma prior $\Gamma(2, 0.2)$. The prior is not very restrictive and distributes its mass over a large range of possible lengthscales. We could come up with such a prior by inspecting the data, shown as black dots on the right, or having some knowledge on its distribution and scale. Knowing that it lies in the $(0, 10)$ interval can already be useful to estimate which range of lengthscales might be reasonable.



(b) Gamma prior $\Gamma(50, 1)$. In this example we completely overestimated the lengthscale value. The learned hyperparameters lead to an almost constant model, underfitting the data and explaining all variations by the added noise term. Note that the shown confidence interval describes noise-less function values $\mathbf{f}$, but the learned noise parameter is indeed very large with $\sigma_n = 0.5043$.



(c) Gamma prior $\Gamma(1, 100)$. We restricted the lengthscales to very small values, thus overfitting to each data point and falling back to the zero-mean prior in-between.

Figure 3.7: Comparison of different prior distributions over lengthscales. The gray area indicates the 95% confidence interval for the function values.

variational inference to allow for a more practical mini-batch training setting (Wilson et al., 2016). A very different approach are *deep Gaussian processes* (Damianou and Lawrence, 2012), which consider multiple stacked GPs in a NN-like structure. This idea has been extended to a convolutional structure (Blomqvist et al., 2018) and to recurrence (Mattos et al., 2015).

Chapter 4

Latent Gaussian Process Dynamics Models from Visual Observations

We are interested in learning dynamical systems from visual observations. Such systems describe the evolution of a true physical state over time, possibly influenced by actions. However, we are not able to observe these physical states directly and instead only obtain a flux of images which show this system. While the physical states might follow simple rules of classical mechanics, the evolution of the observed images often behaves in a much more complex way. We therefore consider approaches which learn a latent, low-dimensional dynamics model, while learning different models to infer latent states from the visual observations as well as to generate observations from latent states.

In this chapter we first introduce state-space models which describe the considered problem more formally. We explain how we can model system dynamics with Gaussian processes and extend the approach to the case of visual observations. The derived training objectives allow for joint training of the introduced models. Finally, we discuss approaches to simulate trajectories in the learned dynamics model, which we then use for planning and control.

4.1 State-space models

We consider a dynamical system in which the true physical states $s \in \mathcal{S}$ evolve over time according to a *transition function* $f : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$ and *actions* $u \in \mathcal{U}$, with

$$s_{t+1} = f(s_t, u_t) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma_f^2), \tag{4.1.1}$$

and discrete time steps t. In general, we are not able to observe the states directly and instead observe them indirectly as *observations* $o \in \mathcal{O}$ through a *measurement function* $g : \mathcal{S} \to \mathcal{O}$, with

$$o_t = g(s_t) + \nu, \qquad \nu \sim \mathcal{N}(0, \sigma_g^2). \tag{4.1.2}$$

We call ϵ the *system noise* and ν the *observation noise*. Since we observe the true states s only indirectly we also call them the *latent states*. Figure 4.1 illustrates the dynamical system defined in Eqs. (4.1.1) and (4.1.2).

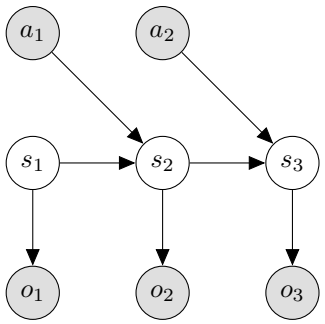


Figure 4.1: Generative process of a dynamical system. Gray filled circles denote observed variables, white circles show latent variables.

If f and g are both linear we call the above a *linear dynamical system* (LDS). In this thesis, we consider the more general case of non-linear f and g and we call the above a *nonlinear dynamical system* (NLDS). We are interested in both *inference*, where we want to find a posterior distribution over states s given observations o, as well as in *learning*, where our goal is to infer the functions f and g from observations o.

4.2 Gaussian process dynamics models on physical states

We first consider a simplified version of the NLDS specified in Eqs. (4.1.1) and (4.1.2), in which we are able to fully observe the true dynamical states s. This corresponds to a special case in which the observation function g is the identity function and where we have zero measurement noise $\sigma_g = 0$. Our goal in this simplified setting is to learn the transition function f.

There are multiple examples in literature on GP dynamics models in fully observable Markov decision processes (Deisenroth, 2010; Ko et al., 2007; Rasmussen and Kuss, 2004). We highlight PILCO by Deisenroth and Rasmussen (2011), a model-based policy search method which also learns system dynamics with GPs. PILCO showed unprecedented data-efficient learning from scratch and showcases the advantages of using a probabilistic dynamics model for planning and control.

We closely follow the approach of Deisenroth and Rasmussen in PILCO for system dynamics learning. We model the transition function f with a Gaussian process and write $f \sim \mathcal{GP}(\mu, k)$, with $\mu : (s, a) \mapsto s$ and k the RBF kernel as defined in Eq. (3.3.12). Recall that modeling GPs with fixed deterministic non-zero mean functions is equivalent to modeling the difference between the targets and the mean function with a GP with zero mean. Therefore, our model is equivalent to the proposed model in PILCO, where the authors model the state differences at each time step with a zero-mean Gaussian process. To highlight the advantage of this prior function over the common zero-mean prior for modeling dynamical systems, consider a state-action input which lies far away

from the known data. Since the RBF kernel is not able to extrapolate, a GP with a zero-mean prior would predict a next state of $s_{t+1} = 0$. In contrast, with our choice of mean function, the GP predicts an unchanged state $s_{t+1} = s_t$, which we argue is often a much better uninformed estimate.

In order to learn the GP hyperparameters and to perform posterior inference we collect training data by randomly interacting with the environment. This provides us with a dataset consisting of sequences $\{s_t, a_t\}_{t=1}^{T}$. Note that we could have many different sequences of this kind, possibly of different length, but we chose to omit the sequence index for notational clarity.

Since Gaussian processes model the correlation between data points, we always need to consider the whole dataset at once. We define

$$S = \begin{bmatrix} s_1 \\ \vdots \\ s_{T-1} \end{bmatrix}, \quad A = \begin{bmatrix} a_1 \\ \vdots \\ a_{T-1} \end{bmatrix} \quad \text{and} \quad S' = \begin{bmatrix} s_2 \\ \vdots \\ s_T \end{bmatrix}$$

as concatenations of states and actions, respectively. We then consider training inputs $X = [S, A]$ and training targets $y = S'$, such that each row in the respective matrices correspond to a single transition $((s_t, a_t), s_{t+1})$. Finally, we learn the GP hyperparameters by maximizing the marginal log-likelihood $\log p\left(y|X\right)$, as introduced in Section 3.3.5.

Next we consider the posterior distribution $p(s^*_{t+1}|s^*_t, a^*_t)$ of the GP dynamics model, given training data (X, y) as defined above. We follow the formulas for posterior inference from noisy observations (Section 3.3.3) while taking into account the non-zero mean function, as well as our formulation where inputs consist of both states and actions, i.e. $x = [s, a]$. We get

$$p(s^*_{t+1}|s^*_t, a^*_t) = \mathcal{N}\left(\bar{s}^*_{t+1}, \mathrm{cov}(s^*_{t+1}) + \sigma_n^2 I\right),$$

with

$$\begin{aligned} \bar{s}^*_{t+1} &= s^*_t + \mathbf{k}_*^\mathsf{T}\left[K + \sigma_n^2 I\right]^{-1}(S' - S) \\ \mathrm{cov}(s^*_{t+1}) &= k_{**} - \mathbf{k}_*^\mathsf{T}\left[K + \sigma_n^2 I\right]^{-1}\mathbf{k}_*, \end{aligned}$$

where $k_{**} = k\left([s^*_t, a^*_t], [s^*_t, a^*_t]\right)$, $\mathbf{k}_* = k\left([S, A], [s^*_t, a^*_t]\right)$ and $K = k\left([S, A], [S, A]\right)$. Note that we included the noise term $\sigma_n I$ in the posterior covariance since we provide the distribution for noisy state transitions.

Finally, for multi-dimensional states we follow the procedure in PILCO (Deisenroth and Rasmussen, 2011) and train conditionally independent GPs for each target state dimension. We present results of this approach in Section 5.6.

4.3 Gaussian process dynamics models from visual observations

Instead of having fully observable physical states, we are interested in the more general case of visual observations of the dynamical system. More formally, with the definition of a non-linear dynamical system in Section 4.1 (see also the graphical representation in Fig. 4.1), we want to learn both the transition function $f : s_t, a_t \mapsto s_{t+1}$ and the observation function $g : s_t \mapsto o_t$ given only sequences of observations $o_{1:T}$ and the corresponding actions $a_{1:T}$. Additionally, we want to infer states from observations, which we then propagate through the learned transition function in order to simulate new state sequences, as well as observation sequences using the observation function. In the following we introduce models for the transition function and observation function, and describe the inference process of the joint dynamics model, provided with visual observations.

In the previous section (Section 4.2) we have shown how Gaussian processes can be used to model system dynamics, provided the model has access to the true physical state representations, and PILCO (Deisenroth and Rasmussen, 2011) showed impressive results with this method. We therefore model the transition function $f : s_t, a_t \mapsto s_{t+1}$ as in PILCO, with a Gaussian process $f \sim \mathcal{GP}(\mu, k)$ with mean function $\mu : (s, a) \mapsto s$ and k the RBF kernel.

Next, the observation function $g : s_t \mapsto o_t$ generates high-dimensional visual observations, notably images, from low-dimensional physical states. A natural choice for models that deal with images are convolutional neural networks (see also Section 3.2.5). More concretely, since g *generates* images we chose to model it with a transposed convolutional neural network (see Section 3.2.5). We describe the chosen architecture in more detail in the corresponding experiments section (Section 5.5).

In order to apply the transition and observation functions, given only sequences of observations $o_{1:T}$ and actions $a_{1:T}$, we need to infer state sequences $s_{1:T}$. We chose to learn an encoder $q(s_t|o_t)$ to infer a belief over latent state representations from the current observation. We consider two different approaches to model $q(s_t|o_t)$: In our first approach we model $q(s_t|o_t)$ as a diagonal Gaussian where the mean and variance are parametrized by a convolutional neural network. Another option we explored is to model the encoder deterministically using a convolutional neural network, corresponding to a Dirac delta distribution where the location of the point mass is parametrized by a convolutional neural network. We provide the exact architecture in the experiments section (Section 5.5).

Note that we implicitly assume that it is possible to reconstruct the full state from the observation. For observations consisting of single images this assumption seems unreasonable since it is generally not possible to infer the velocity of objects from a single still RGB image. To circumvent this problem, we define an observation

as a sequence of the last k images $o_t = [i_t, i_{t-1}, \ldots, i_{t-k}]$, with $i_j \in \mathbb{R}^{H \times W \times 3}$ the generated frame at timestep j. The physical states in the environments we chose in our experiments (Chapter 5) consist only of positions and velocities of objects and show no long-term dependencies, such that we chose $k = 2$.

Figure 4.2 shows a graphical representation of the inference process. The different types of lines show the different parts of the model, namely the encoder, decoder, and the dynamics model.

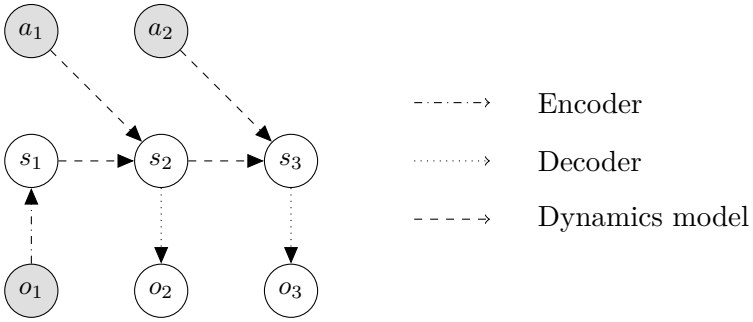


Figure 4.2: Inference process of the introduced model. The different lines denote the individual components of our model as detailed in the legend. The model uses the observed variables (gray) to compute a belief for the unobserved variables (white). Notably, the model is able to use its belief over the next state s_2 in order to estimate s_3 without observing o_2. We discuss the sampling of trajectories in more detail in Section 4.6.

4.4 Training objectives

In the following, we derive two different training objectives to train the proposed dynamics model by maximizing the likelihood of the given data under our model. We briefly recall the proposed state space model.

We consider sequences $\{o_t, a_t\}_{t=1}^T$, generated by a latent state-space model using a hidden state sequence $\{s_t\}_{t=1}^T$, with

$$\begin{aligned} &\text{Transition model:} && s_t \sim p(s_t|s_{t-1}, a_{t-1}), \\ &\text{Observation model:} && o_t \sim p(o_t|s_t), \end{aligned} \tag{4.4.1}$$

where we assume a fixed initial state s_0. As introduced in the last section (Section 4.3), we consider the transition model as given by a Gaussian process and the observation model given by a Gaussian with mean parametrized by a deconvolutional neural network and identity covariance. Note that the log-likelihood under a Gaussian distribution

with unit variance equals the mean squared error up to a constant. Further, we defined an encoder $q(s_{1:T}) = \prod_{t=1}^{T} q(s_t|o_t)$ to infer approximate state posteriors from observations, either as a diagonal Gaussian with both mean and variance parametrized by a convolutional neural network or as a Dirac delta with deterministic point mass location, parametrized by a convolutional neural network.

4.4.1 KL-divergence-based training objective

We construct a variational bound on the data likelihood $p(o_{1:T}|a_{1:T})$.

$$\log p(o_{1:T}|a_{1:T}) = \log \mathbb{E}_{p(s_{1:T}|a_{1:T})}\left[\prod_{t=1}^{T} p(o_t|s_t)\right] \tag{4.4.2}$$

$$= \log \mathbb{E}_{q(s_{1:T}|o_{1:T})}\left[\prod_{t=1}^{T} p(o_t|s_t)p(s_t|s_{t-1},a_{t-1})/q(s_t|o_t)\right] \tag{4.4.3}$$

$$\geq \mathbb{E}_{q(s_{1:T}|o_{1:T})}\left[\sum_{t=1}^{T}\left(\log p(o_t|s_t) + \log\left(\frac{p(s_t|s_{t-1},a_{t-1})}{q(s_t|o_t)}\right)\right)\right] \tag{4.4.4}$$

$$\begin{aligned} = \sum_{t=1}^{T}\Bigg(&\mathbb{E}_{q(s_t|o_t)}\left[\log p(o_t|s_t)\right] \\ &+ \mathbb{E}_{q(s_{t-1}|o_{t-1})}\left[\mathbb{E}_{q(s_t|o_t)}\left[\log\left(\frac{p(s_t|s_{t-1},a_{t-1})}{q(s_t|o_t)}\right)\right]\right]\Bigg) \end{aligned} \tag{4.4.5}$$

$$\begin{aligned} = \sum_{t=1}^{T}\Bigg(&\underbrace{\mathbb{E}_{q(s_t|o_t)}\left[\log p(o_t|s_t)\right]}_{\text{Reconstruction}} \\ &- \underbrace{\mathbb{E}_{q(s_{t-1}|o_{t-1})}\left[D_{\mathrm{KL}}\left(q(s_t|o_t)||p(s_t|s_{t-1},a_{t-1})\right)\right]}_{\text{Latent state prediction}}\Bigg), \end{aligned} \tag{4.4.6}$$

where we used the properties of the transition model $p(s_{1:T}|a_{1:T}) = \prod_{t=1}^{T} p(s_t|s_{t-1},a_{t-1})$, Jensen's inequality, the defined decoder $q(s_{1:T}|o_{1:T}) = \prod_{t=1}^{T} q(s_t|o_t)$, and the definition of the KL-divergence $D_{\mathrm{KL}}\left(p(x)||q(x)\right) = \mathbb{E}_{p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right]$. Estimating the outer expectations using a single reparameterized sample yields an efficient objective for inference and learning in non-linear latent variable models that can be optimized using gradient ascent (Kingma and Welling, 2013; Rezende et al., 2014).

For this training objective we only considered the probabilistic encoder, where $q(s_t|o_t)$ is a Gaussian distribution with mean and variance parametrized by a neural network. We additionally include a KL-divergence term between the encoder $q(s_t|o_t)$ and a standard Gaussian prior $\mathcal{N}(0, I)$.

The derivation of the training objective closely resembles the derivation done by Hafner et al. (2019) while taking into account the different model definitions. The appeal of this training objective is that the probabilistic encoder gets regularized by the prediction of the transition model, which we assume should be favorable for the training process compared to a regularization by KL-divergence to a standard Gaussian prior. A notable drawback of this training objective is that it factorizes over the individual training data points. In the context of neural networks, this factorization is necessary in order to train the model with mini-batch stochastic gradient descent. However, an important property of exact Gaussian processes is that they model the covariance between different data points. In order to get meaningful predictions for single transitions $p(s_t|s_{t-1}, a_{t-1})$ we therefore need to compute the posterior distribution given some evidence. Thus, we provide all inferred latent states and actions as evidence to the GP transition model. However, these considerations motivate the derivation of a different training objective which takes into account this GP property, and we present such an objective in the following section.

4.4.2 MLL-based training objective

We derive a second training objective while considering the GP viewpoint more closely. In particular, we aim to find a training objective which considers the covariances between individual training points and which contains a marginal log-likelihood term for the GP transition model. We introduce new notation regarding the given training data and define

$$X_o := \begin{bmatrix} o_1, & a_1 \\ \vdots & \vdots \\ o_{T-1}, & a_{T-1} \end{bmatrix}, \qquad y_o := \begin{bmatrix} o_2 \\ \vdots \\ o_T \end{bmatrix} \tag{4.4.7}$$

as the evidence in observation space as well as

$$X_s := \begin{bmatrix} s_1, & a_1 \\ \vdots & \vdots \\ s_{T-1}, & a_{T-1} \end{bmatrix}, \qquad y_s := \begin{bmatrix} s_2 \\ \vdots \\ s_T \end{bmatrix} \tag{4.4.8}$$

in state space.

Using these definitions, we can write the transition in state space as $p(y_s|X_s)$, provided by the GP dynamics model, for which we can compute the prior likelihood as described in Section 3.3. Further, we assume that the observation model and the

encoder factorize over the individual data points.

$$\text{Decoder:} \qquad p(y_o|y_s) = \prod_{t=2}^{T} p(o_t|s_t), \qquad p(X_o|X_s) = \prod_{t=1}^{T-1} p(o_t|s_t), \tag{4.4.9}$$

$$\text{Encoder:} \qquad q(y_s|y_o) = \prod_{t=2}^{T} q(s_t|o_t), \qquad q(X_s|X_o) = \prod_{t=1}^{T-1} q(s_t|o_t). \tag{4.4.10}$$

We marginalize the data likelihood $p(y_o|X_o)$ in the following way:

$$p(y_o|X_o) = \int\int p(y_o|y_s)p(y_s|X_s)q(X_s|X_o)\mathrm{d}y_s\mathrm{d}X_s \tag{4.4.11}$$

$$= \int\int \frac{p(y_o|y_s)}{q(y_s|y_o)} q(y_s|y_o)p(y_s|X_s)q(X_s|X_o)\mathrm{d}y_s\mathrm{d}X_s \tag{4.4.12}$$

$$= \mathbb{E}_{q(y_s|y_o)q(X_s|X_o)}\left[\frac{p(y_o|y_s)}{q(y_s|y_o)} p(y_s|X_s)\right]. \tag{4.4.13}$$

We then apply Jensen's inequality to derive our training objective as a lower bound on the log-likelihood:

$$\log p(y_o|X_o) \geq \mathbb{E}_{q(y_s|y_o)q(X_s|X_o)}\left[\log p(y_o|y_s) - \log q(y_s|y_o) + \log p(y_s|X_s)\right] \tag{4.4.14}$$

$$\begin{aligned} &= \underbrace{\mathbb{E}_{q(y_s|y_o)}\left[\log p(y_o|y_s)\right]}_{\text{I}} + \underbrace{\mathbb{E}_{q(y_s|y_o)}\left[-\log q(y_s|y_o)\right]}_{\text{II}} \\ &\quad + \underbrace{\mathbb{E}_{q(y_s|y_o)q(X_s|X_o)}\left[\log p(y_s|X_s)\right]}_{\text{III}}. \end{aligned} \tag{4.4.15}$$

We can interpret the three terms in the following way:

I. **Reconstruction likelihood.** This term corresponds exactly to the "reconstruction" term in the KL-divergence-based training objective. Since the decoder parametrizes the mean of a Gaussian distribution with unit variance this is equivalent to the negative mean squared error up to a constant.

II. **Encoder regularization.** This term corresponds to the definition of the *differential entropy* $h(p) := \mathbb{E}_{p(x)}\left[-\log p(x)\right]$. Given a multivariate normal distribution $p \sim \mathcal{N}(\mu, \Sigma)$ with diagonal covariance matrix Σ and $\sigma_i := (\Sigma)_{ii} > 0$, the differential entropy is given by

$$h(p) = \frac{1}{2}\log\left((2\pi e)^N\right) + \frac{1}{2}\sum_i \log\left(\sigma_i\right), \tag{4.4.16}$$

which is maximized as the variances σ_i increase. Thus, considering a probabilistic Gaussian encoder with parametrized mean and variance, this term can be interpreted as a regularization term which prevents small variances.

III. **Log-likelihood term of the state transitions.** The inner term $p(y_s|X_s)$ corresponds to the standard marginal log-likelihood, which is the default training objective to learn hyperparameters of a GP. However, we consider the MLL in expectation over the posterior distribution from the encoder.

As before, we estimate the outer expectation using a single reparametrized sample (Kingma and Welling, 2013; Rezende et al., 2014).

Subset-of-data approximation

In comparison to the KL-divergence-based training objective derived in Section 4.4.1, we can not factorize Eq. (4.4.15) over individual data points; It was also our main motivation to consider covariances between points in this derived loss term. Thus, we cannot minimize this objective by using mini-batch stochastic gradient-descent. However, stochastic gradient descent with small mini-batches is generally preferred for training neural networks (see also Section 3.2.4), circumventing memory limitations as well as often providing more robustness, faster convergence and better generalization (Bottou, 2010; Ge et al., 2015; Masters and Luschi, 2018).

Another motivation for a mini-batch stochastic optimization scheme comes from the data requirements of neural networks, which are known to require large amounts of data in order to provide good results and to generalize well to unseen data. Computing the MLL as required in Eq. (3.3.13) (III) scales $O(T^3)$ with the dataset size T. In a full-batch gradient descent training scheme, this effectively limits the amount of data we could consider for training. By computing the MLL over batches of size $B \ll T$ we have a computational scaling of $O(B^3)$ and we can increase the size of the dataset n in order to include more variety to train the NNs. This strategy is know as the *subset-of-data* (SoD) approach (Liu et al., 2018, Section III.A.). While there are many other approaches on scaling GPs to large data (Liu et al., 2018), SoD has been shown to produce reasonable predictions in the case of abundant or redundant data (Hayashi et al., 2019). Finally, we argue that GPs are known to require very little data in the considered environments in order to learn good dynamics models (Deisenroth and Rasmussen, 2011).

With these considerations, we introduce an approximation of the training objective defined in Eq. (3.3.13) over a sufficiently large subset of data or batch. We start by showing the factorization of the two terms of Eq. (3.3.13) which do not contain the GP dynamics model by using the assumed encoder and decoder factorizations from

Eq. (4.4.9). For the reconstruction term (I) we get

$$\begin{aligned}\mathbb{E}_{q(y_s|y_o)}\left[\log p(y_o|y_s)\right] &= \mathbb{E}_{\prod_{t=2}^{T} q(s_t|o_t)}\left[\log\left(\prod_{t=2}^{T} p(o_t|s_t)\right)\right] \\ &= \sum_{t=2}^{T} \mathbb{E}_{q(s_t|o_t)}\left[\log\left(p(o_t|s_t)\right)\right],\end{aligned}$$

and we similarly show the factorization for the regularization term (II) as

$$\mathbb{E}_{q(y_s|y_o)}\left[-\log q(y_s|y_o)\right] = \sum_{t=2}^{T} \mathbb{E}_{q(s_t|o_t)}\left[-\log q(s_t|o_t)\right].$$

Considering a mini-batch with indices $\mathcal{B} \subset \{1, \ldots, T\}$ of size $B = |\mathcal{B}|$, we approximate these two terms as

$$\mathbb{E}_{q(y_s|y_o)}\left[\log p(y_o|y_s)\right] \approx \frac{T}{B} \sum_{t \in \mathcal{B}} \mathbb{E}_{q(s_t|o_t)}\left[\log\left(p(o_t|s_t)\right)\right] \tag{4.4.17}$$

and

$$\mathbb{E}_{q(y_s|y_o)}\left[-\log q(y_s|y_o)\right] \approx \frac{T}{B} \sum_{t \in \mathcal{B}} \mathbb{E}_{q(s_t|o_t)}\left[-\log q(s_t|o_t)\right]. \tag{4.4.18}$$

Finally we assume

$$\mathbb{E}_{q(y_s|y_o)q(X_s|X_o)}\left[\log p(y_s|X_s)\right] \approx \mathbb{E}_{\prod_{t \in \mathcal{B}} q(s_t|o_t)q(s_{t-1}|o_{t-1})}\left[\log p(s_{\mathcal{B}}|s_{(\mathcal{B}-1)}, a_{(\mathcal{B}-1)})\right], \tag{4.4.19}$$

where we define $(\mathcal{B}-1) := \{t-1 | t \in \mathcal{B}\}$ element-wise by a slight abuse of notation. By randomly sampling a subset $\mathcal{B} \subset \{1, \ldots, T\}$ to compute the gradients for the gradient descent scheme we obtain the desired mini-batch training scheme.

We remind the reader once more that the approximation proposed in Eq. (4.4.19) represents a strong assumption which does not hold in general. While we explored both the exact full-batch training as well as training in mini-batches for varying batch sizes, we achieved our best results by training on a larger data set with the mini-batches. We again suspect that the reason this approach is sensible in this context is due to both the data-efficiency of GP dynamics models (which required data in the order of order of 20 rollouts or 1000 transitions) in combination with the abundance of available simulation data (we often considered datasets in the order of 200 rollouts or 10000 transitions).

4.5 GP dynamics models on dynamics-unaware embeddings

We propose a simplified approach to combine AEs or VAEs for state representation learning with GPs for modeling system dynamics in the following two-stage procedure. We first learn state representations s_t for each observation o_t by training an AE or VAE on a dataset of observations $\mathcal{D} = \{o_i\}_{i=1}^N$ without considering any system dynamics. This first stage corresponds to a standard AE or VAE training as described in Sections 3.2.6 and 3.2.7. We then use the learned encoder $g^{-1} : o_t \mapsto s_t$ to map the transitions in observation space (o_t, a_t, o_{t+1}) to observations in latent space (s_t, a_t, s_{t+1}) such that we obtain a low-dimensional dataset of latent state transitions $\mathcal{D} = \{(s_t, a_t, s_{t+1})\}_{t=1}^T$. Finally, we train a GP dynamics model on these state transitions in the learned, but fixed, latent space by proceeding exactly as for GP dynamics models on the true physical states (see Section 4.2).

We include results for this proposed simplified latent GP dynamics model in Section 5.7.1.

4.6 Sampling trajectories from a Gaussian process dynamics model

The goal of learning a dynamics model for a given environment is to use the learned model in order to simulate the interaction with the environment. Ideally, this simulation perfectly describes the real environment. More formally, we want to generate a sequence of states $\{s_t\}_{t=1}^T$ from an initial state s_0 and a sequence of actions $\{a_t\}_{t=0}^{T-1}$.

4.6.1 Mean propagation

In *mean propagation*, we select the state with maximum likelihood under the posterior distribution of the transition model $p(s_{t+1}|s_t, a_t)$. Since $p(s_{t+1}|s_t, a_t)$ is a Gaussian distribution, the maximum likelihood estimate corresponds to the mean. We can then recursively apply the transition model to the new predicted state s_{t+1} and action a_{t+1} in order to compute s_{t+2}.

This approach is very simple to implement and computationally very cheap since it only requires to compute the mean of the Gaussian process prediction, but not its covariance. However, this property is also the main drawback: We do not consider any uncertainty. We lose the advantages of having a probabilistic model which also models uncertainties.

4.6.2 Moment matching

In order to consider the uncertainties in $s_t \sim \mathcal{N}(u, S)$ when computing the posterior $p(s_{t+1}|s_t, a_t)$, we need to marginalize over s_t and compute

$$p(s_{t+1}|u, S, a_t) = \int p(s_{t+1}|s_t, a_t)p(s_t|u, S)\mathrm{d}s_t. \tag{4.6.1}$$

In the special case of Gaussian processes with RBF kernels, it is possible to obtain exact analytical expressions for the mean and variance of the marginalized predictive distribution (Quiñonero-Candela et al., 2003). This approach is also known as *moment matching* since we approximate the true distribution with a Gaussian while exactly matching the true mean and variance. Notably, Deisenroth and Rasmussen (2011) also applied moment matching in PILCO.

A main application which requires simulating new sequences under the learned dynamics model is planning and control. We used the cross-entropy method in a model-predictive control setting (Section 4.7). In order to apply this method, we need to generate state trajectories for a larger number of candidate action sequences (e.g. ~ 1000 sequences). We found this to be unfeasible due to large memory requirements. Vectorized computation of moment matching as described by Quiñonero-Candela et al. (2003) requires building an array with N^2d entries, where N is the number of data points in the GP evidence and d the dimension of each datapoint. With $N = 1000$ and $d = 3$ this gives an array with $3,000,000$ elements. Considering 32 bit float values, we get a memory requirement of ~ 0.9 GB for a single computation of the moment matching solution. It is therefore infeasible to apply moment matching on larger batches of data. Further, we found sequential application or even the use of mini-batches to be too time-consuming: A single moment-matching step takes ~ 0.1 seconds in our implementation. The chosen CEM-MPC control algorithm (see Section 4.7) with parameters $H = 20$, $I = 10$, $J = 1000$ requires $200,000$ such evaluations, such that the generation of a single action would require ~ 5.5 hours. With a GPU memory limit of ~ 10 GB, we could decrease this number by a factor of 10 using batches, but it stays unfeasible to apply moment matching for control. Note that the described memory and time considerations are specific to our implementation and our choice of using CEM for planning. Other approaches such as directly learning a policy function through gradient descent do not require the evaluation of a large number of candidate sequences and implementations which use CPUs for their computations instead of GPUs might have both different memory restrictions and computation times.

4.6.3 Conclusion

In conclusion, we used mean propagation for planning and control as well as for the evaluation of the learned dynamics models. However, moment matching is a theoretically well-founded approach which considers uncertainties of the dynamics

model when predicting new state sequences. We argue that it is almost always preferable to use moment matching over mean propagation if the respective task or the applied method does not require evaluation of a large number of candidate sequences.

4.7 Planning and Control

Our interest in learning dynamics models from high-dimensional visual observations is to solve specific tasks in the learned environments. Such tasks could be defined as balancing a pole which is attached to a cart (the so-called CartPole problem described by Barto et al. (1983)), getting the highest possible score in ATARI games (Mnih et al., 2013), or manipulating objects in the real world with a physical robot. While we as humans typically have a good understanding of our task of interest, we generally need to describe it using a cost function (or equivalently a reward function) which specifies good and bad states in order to approach the problem formally. Then, our goal is to select actions which minimize this cost or equivalently maximize the reward. In the CartPole example the cost could be proportional to the angle, where an angle of zero means that the pole points upwards, thus enforcing the desired balancing. In the example of ATARI games one could consider the score provided by the game engine as a reward function.

4.7.1 Modeling reward

We consider environments as introduced in Section 4.1 on which we learned a dynamics model from visual observations as defined in Sections 4.3 and 4.4. Additionally, we now consider some specific task in this environment defined by a corresponding true reward function $p(r_t|s_t)$. However, we do not know the true reward function and instead only have access to rewards which we collected by interacting with the environment, in the form of sequences $\{o_t, a_t, r_t\}$. We can use the encoder $q(s_t|o_t)$ of the learned dynamics model to estimate latent states s_t from observations o_t. These state estimations are then used to learn a reward model $r_t \sim p(r_t|s_t)$ which we chose as as a scalar Gaussian with mean parametrized by a feed-forward neural network and unit variance. Maximizing the data log-likelihood is equivalent to minimizing the mean-squared error between predictions and the ground-truth rewards. The neural network is trained using gradient descent. Note that since we consider a fixed encoder, this corresponds to a standard supervised regression task.

4.7.2 Planning with the cross entropy method

Consider an initial state s_t, possibly inferred by the learned encoder from an initial observation o_t, where t is the current time step of the agent. Our goal is to find the

action sequence $a_{t:t+H}$ that maximizes the expected sum of future rewards, obtained by applying the action sequence to the real environment.

In order to estimate the expected reward, we compute state trajectories $s_{t:t+H}$ from the dynamics model using mean-propagation as described in Section 4.6, for which we can then estimate rewards $r_{t:t+H}$ using the learned reward model. We denote this mapping from action sequences to the sum of expected rewards with $S(a_{t:t+H})$ and we call S a *performance function*.

Now, our goal in planning can be formulated as an optimization problem: We want to find the action sequence $a_{t:t+H}$ which maximizes the performance function S

$$a^*_{t:t+H} = \arg\max_{a_{t:t+H}} S(a_{t:t+H}). \tag{4.7.1}$$

One method to solve this optimization problem is the cross entropy method (CEM) (Boer et al., 2005; Rubinstein, 1996). CEM is a popular approach for planning of action sequences and has been applied both on classic control problems (Mannor et al., 2003; Szita and Lörincz, 2006) as well as in the context of deep reinforcement learning (Chua et al., 2018; Hafner et al., 2019). The main idea of CEM is to maintain a distribution over possible solutions, that is over possible action sequences, and update this distribution at each step such that good solutions become increasingly more likely.

Consider the family of time-dependent diagonal Gaussian distributions over action sequences $a_{t:t+H} \sim \mathcal{N}(\mu_{t:t+H}, \sigma^2_{t:t+H} I)$, parametrized with means $\mu_{t:t+H}$ and variances $\sigma^2_{t:t+H}$, and let $f_0 = \mathcal{N}(0, I)$ be the initial distribution over actions. We iteratively sample J candidate sequences $a^{(1)}_{t:t+H}, \dots, a^{(J)}_{t:t+H}$ from f_i and obtain their expected rewards $S(a^{(1)}_{t:t+H}), \dots, S(a^{(J)}_{t:t+H})$. After ordering these candidate sequences according to decreasing expected return, we select the best K sequences $a^{(1)}_{t:t+H}, \dots, a^{(K)}_{t:t+H}$. Note that all selected sequences have a reward equal or higher than $\gamma_i := S(a^{(K)}_{t:t+H})$. In order to update the belief over action sequences, consider $g_{\geq\gamma_i}$ the uniform distributions over the best K sequences $\left\{a^{(1)}_{t:t+H}, \dots, a^{(K)}_{t:t+H}\right\}$. We then update the current belief over action sequences by finding the distribution f_{i+1} in the considered family of Gaussian distributions closest to $g_{\geq\gamma_i}$ with regard to the *cross-entropy measure*, which is formally defined as

$$H(p, q) = -\int p(x) \log q(x) \mathrm{d}x. \tag{4.7.2}$$

Using the fact that $g_{\geq\gamma_i}$ is a discrete uniform distribution we get

$$H(g_{\geq\gamma_i}, q) = -\int g_{\geq\gamma_i}(a_{t:t+H}) \log f_{i+1}\left(a_{t:t+H}\right) \mathrm{d}a_{t:t+H} \tag{4.7.3}$$

$$= -\frac{1}{K} \sum_{j=1}^{K} \log f_{i+1}\left(a^{(j)}_{t:t+H}\right), \tag{4.7.4}$$

which is equivalent to the negative log-likelihood of $\left\{a_{t:t+H}^{(1)}, \ldots, a_{t:t+H}^{(K)}\right\}$ under f_{i+1}. Since f_{i+1} is a Gaussian distribution, its minimum can be analytically computed as

$$\mu_{t:t+H}^{(i+1)} = \frac{1}{K}\sum_{j=1}^{K} a_{t:t+H}^{(j)}, \qquad \left(\sigma_{t:t+H}^{(i+1)}\right)^2 = \frac{1}{K}\sum_{j=1}^{K}\left(a_{t:t+H}^{(j)} - \mu_{t:t+H}^{(i+1)}\right)^2. \tag{4.7.5}$$

We repeat this procedure for I optimization steps and return $a_{t:t+H} = \mu_{t:t+H}^{(I)}$. This iterative CEM algorithm is outlined in Algorithm 1.

Algorithm 1 Planning with CEM

Input: H Planning horizon distance
I Optimization iterations
J Candidates per iteration
K Number of top candidates to fit

Initialize distribution over action sequences $f_0 \leftarrow \mathcal{N}(0, I)$.
for optimization iteration $i = 1..I$ **do**
  Sample J candidate sequences $\left\{a_{t:t+H}^{(1)}, \ldots, a_{t:t+H}^{(J)}\right\}$ from f_{i-1}
  Compute expected rewards $\left\{S\left(a_{t:t+H}^{(1)}\right), \ldots, S\left(a_{t:t+H}^{(J)}\right)\right\}$
  // Sort the candidates by decreasing rewards
  sort $\left(\left\{S\left(a_{t:t+H}^{(1)}\right), \ldots, S\left(a_{t:t+H}^{(J)}\right)\right\}\right)$
  // Re-fit belief to the K best action sequences.
  $\mu_{t:t+H} = \frac{1}{K}\sum_{k=1}^{K} a_{t:t+H}^{(k)}, \quad \sigma_{t:t+H}^2 = \frac{1}{K}\sum_{k=1}^{K}\left(a_{t:t+H}^{(k)} - \mu_{t:t+H}\right)^2$.
  $f_i \leftarrow \mathcal{N}(\mu_{t:t+H}, \sigma_{t:t+H}^2 I)$
end for
return First action mean μ_t.

4.7.3 Model-predictive control

We apply the presented planning algorithm to the real environment using *model-predictive control* (MPC) (Garcia et al., 1989): For each current state belief s_t we compute an action sequence $a_{t:t+H}$ using CEM and then apply a_t to the real system. As a result we obtain a new observation o_{t+1} from which we can infer a state s_{t+1} and we repeat this process. Note that the belief over action sequences starts from zero mean and unit variance again to avoid local optima.

The described approach is a *closed-loop* control method since we receive feedback from the environment which we consider while choosing the subsequent action. This is in contrast to *open-loop* control where we would plan to apply a full sequence of actions $a_{0:T}$ without considering the resulting observations $o_{1:T}$. Further, MPC is not bound

to CEM for planning or even to the considered type of dynamics model. It generally describes the approach of planning control for a finite time-horizon using a model of the environment but only applying the first step of the planned control strategy.

We evaluated the proposed dynamics models for control using MPC with CEM as introduced in this section. We provide the results in Section 5.8.

Chapter 5

Experiments

In the following we present experimental results of our proposed method, learning latent Gaussian process dynamics models from visual observations.

We first introduce the two OpenAI gym (Brockman et al., 2016) environments on which we evaluate the proposed methods, `Pendulum-v0` and `CartPole-v1`, and define the data collecting process. In preparation of the main results we provide examples for the two different underlying approaches our method builds on, namely unsupervised representation learning from images and GP dynamics models on physical states. Our main results show latent GP dynamics models which we jointly trained with the encoder/decoder pair on the derived training objectives (see also Section 4.4). We compare our joint training method to the simplified approach of first learning state representations which are unaware of a transition model on the latent space, as AEs or VAEs, and then fitting a GP dynamics model to the fixed learned embeddings. Finally we apply the learned models to control tasks in these environments and demonstrate transfer to new physical conditions.

5.1 Environments

OpenAI gym (Brockman et al., 2016) provides well-established simulated environments with tasks of varying difficulty. We considered the following environments:

- `Pendulum-v0`:
  Description: This environment describes a single pendulum to which the actor can apply torque. The task consists in moving the pendulum such that it points upwards and to balance it in this position, which we also call performing a "swing-up". It is not possible to directly move the pendulum to the goal position and a minimum of one swing to each side are required in order to gather sufficient momentum. For a render of this environment see Fig. 5.1a.

  States and actions: The true physical states consist of the angle θ and the angular velocity $\dot{\theta}$ of the pendulum, with velocities $|\dot{\theta}| < 8$. The state is typically encoded in a three dimensional vector $s = (\cos(\theta), \sin(\theta), \dot{\theta})$. This encoding makes the state space continuous, eliminating the jump from 0 to 2π, which is beneficial

for many methods, notably for GPs with RBF kernels. Figure 5.2 shows a 3D visualization of the true physical state space. Finally, the action a consists of a single real scalar describing the torque applied to the joint, with $a \in [-2, 2]$.

Modifications: We removed the rendering of the last applied action from the environment.

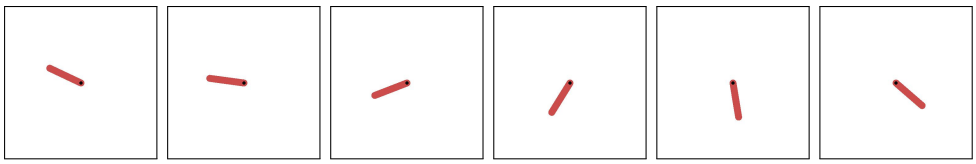
- `CartPole-v0`:
  Description: A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. Figure 5.1b shows a render of this dynamical system. The environment corresponds to the problem described by Barto et al. (1983). There are two commonly considered tasks in this environment: In *balancing* the pendulum starts upright, and the goal is to prevent it from falling over, and in the *swing-up* task the pendulum starts pointing downwards and the actor has to move the cart in a way to lift the pendulum and to balance it pointing up afterwards.

  States and actions: The true physical states consist of the cart position and velocity, as well as the pole angle and angular velocity. As for `Pendulum-v0` we encoded the pole angle into its cosine and sine, leading to a five-dimensional state representation. The action a consists of force applied to the cart in order to push it to the left or right, with $a \in [-10, 10]$.

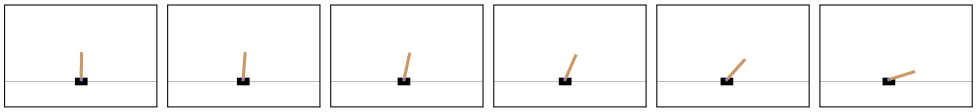
  Modifications: We modified the original version in order to have a continuous action space, using the applied force of the original `CartPole-v0` as the upper bound for the interval of valid actions. We further do not stop the environment if the pole angle crosses a certain threshold, as we are interested in the whole physical state space as well as in the swing-up task. We changed the sparse, binary reward function into the continuous reward used in the non-sparse CartPole version of the DeepMind Control Suite (Tassa et al., 2018). Lastly, we slightly modified the rendering to have a wider pole.

5.2 Data Collection

We generate data for learning the dynamics models by interacting with the environment and applying randomly chosen actions, uniformly sampled from the respective action space. The starting configuration depends on the environment. For the Pendulum we sample the starting state uniformly from the full state space $[0, 2\pi) \times [-8, 8]$. In the CartPole environment we start with the balancing position, with central cart, upright pole, and zero velocities, and we add uniform noise $\sigma_i \sim \mathcal{U}(-0.05, 0.05)$ to each state dimension s_i. With these random interactions we obtain N sequences of length T, giving us a dataset $\mathcal{D} = \left\{ \{(o_t^i, s_t^i, a_t^i, r_t^i)\}_{t=1}^{T} \right\}_{i=1}^{N}$ with observations o_t^i, true physical



(a) Pendulum.



(b) CartPole.

Figure 5.1: Rendered frames of the considered OpenAI gym environments.

states s_t^i, actions a_t^i and rewards r_t^i for sequence i at time t. We remind the reader that we consider observations which consist of two subsequent frames, in order to capture information on velocities, that is $o_t = [i_t, i_{t-1}]$ where i_t is the frame rendered at time step t. Additionally we added a small amount of Gaussian noise $\epsilon_{i,j} \sim \mathcal{N}(0, 0.01)$ to each pixel of the collected frames, thus equivalently to each observation.

Note that the system states are collected for evaluation purposes and for the results with GP dynamics models on the true physical states, but are not available during the training of the proposed method. Similarly, the collected rewards are only relevant in the context of planning and control. We describe the actual data used for training in more detail in each corresponding experiments section.

5.3 Model Evaluation

We generally want the model to provide *accurate predictions* of the learned environment. However, this notion is still very broad. Pixel-wise losses in image space do often not accurately reflect the model error. On the other hand, the scale and structure of the latent space is learned by the model and it might happen that it allows for low losses while not providing meaningful representations. Ideally we could use the learned dynamics model for planning and control and evaluate the model on task performance. However, we also want to learn system dynamics without considering a specific control task. With these general considerations in mind we present and discuss approaches to understand and evaluate the different aspects of the proposed model.

A simple metric to evaluate the learned *state representations* is already contained in both of the derived training objectives: A reconstruction loss for the encoder/decoder pair, for example the MSE or BCE. A complete loss of information due to a malformed

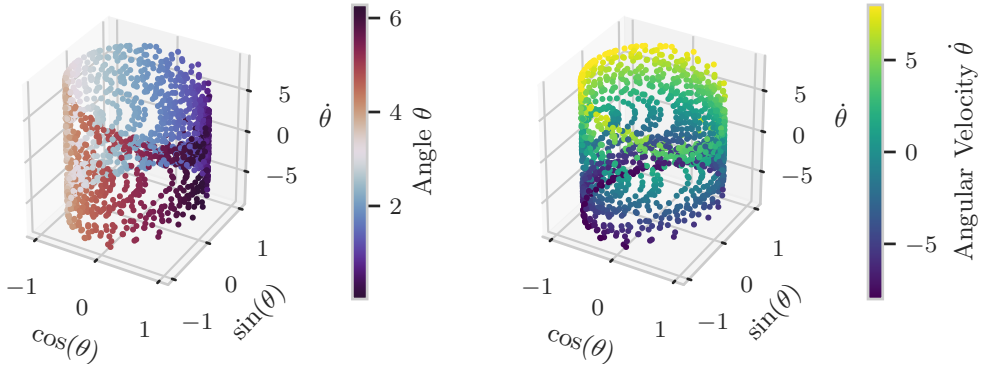


Figure 5.2: Visualization of the true physical state space in the pendulum environment. For each state $s_i = (\theta, \dot{\theta})$ we show a point at location $(\cos(\theta), \sin(\theta), \dot{\theta}))$. We additionally colored each point according to its angle θ in the left plot, and with its angular velocity $\dot{\theta}$ in the right plot, respectively.

latent space should become apparent and lead to very high reconstruction errors. However, the reconstruction loss does not capture our main motivation well. We want a *meaningful* latent space which enables a good dynamics model, and it is in our interest if the latent representations contain only controllable aspects of the environment instead of precisely describing details of the visual observations.

For latent spaces with dimensions $d \leq 3$, which holds for the Pendulum environment, we can visually inspect the encoded dataset of observations, similarly to Fig. 5.2. Since we know the true state space we can visually compare the shape of the learned manifold and intuitively build an opinion on its shape and structure. During our work on the proposed method this evaluation has been very insightful. For higher-dimensional state spaces we can visualize projections to a selection of three of the dimensions, allowing us to visualize CartPole, but due to the high dimensionality of the true physical state space the visualizations were less insightful and intuitive. A downside is the qualitative nature of this evaluation. Ideally we want a quantitative metric which coincides well with our intuitive judgement about the learned latent space.

A quantitative approach to evaluate the information contained in a latent representation is to regress the true physical states from the learned latent states (Lesort et al., 2018). Karl et al. (2016) regress the physical states of a pendulum with a linear regression. In our experiments the learned state representation did mostly not accurately recover the original state manifold and linear regression did not seem flexible enough. Since we model the system dynamics with Gaussian processes we opted for a support vector regression (SVR) (Drucker et al., 1997), a kernel method which closely relates to the support vector machine used for classification (Boser et al., 1992). Our

motivation for this choice was that the SVR does also use an RBF kernel and thus it might be more indicative of the quality of the learned embeddings for the GP dynamics model. We use the implementation provided by scikit-learn (Pedregosa et al., 2011). Indeed, we observed a meaningful correlation between low regression errors and our own visual and intuitive judgement of the learned representations.

To qualitatively evaluate the dynamical system we can visualize predicted observations, for multi-step predictions, through the learned decoder and compare them to the ground-truth observations. Quantitatively this might be best reflected with a MSE between predicted observations and true observations. However, note that our goal is not to solve a video prediction task. Errors in pixel space do also not accurately reflect the amplitude of errors in state space. For example, a small inaccuracy in the predicted angle of a pendulum might already lead to very little overlap of the rendered pendulums, thus leading to a high MSE in image space.

We can similarly evaluate prediction errors in latent space. Since the state representations are learned by the model they likely deviate in both structure and scale for different runs. Errors in latent space can therefore not be compared between different training runs, or between different methods. Still, the evolution of these latent prediction errors during the training can still be an interesting indicator, especially together with the errors in image space.

Finally, we can use the learned dynamics model for planning and control and evaluate the model on task performance. In order to achieve good performance the model needs to infer meaningful latent state representations, and accurately predict trajectories in state space. The learned state space also has to be informative enough to learn an accurate reward model. A downside of this evaluation is that the task performance might depend on the chosen planner and controller, as well as on their settings. However, the chosen cross-entropy method has been shown to provide good performance on complex environments, in combination with learned latent states from pixels (Hafner et al., 2019), and we therefore assume that task performance should be indicative of model performance.

5.4 Numerical Considerations for training Gaussian Processes

During our experiments we regularly encountered numerical issues when training Gaussian processes, both in combination with learning latent embeddings as well as on the true physical states. This is likely due to the structure of the kernel matrix during the Cholesky decomposition, which is required in order to compute the marginal log-likelihood (Eq. (3.3.13)). More concisely, the GPyTorch library (Gardner et al., 2018), which we used to implement the Gaussian processes in this work, replaces the exact Cholesky decomposition with a conjugate gradient descent (for more details on this numerical choice we refer to the GPyTorch paper by Gardner et al.). We therefore

obtained warnings of failed convergence during the conjugate gradient descent step. Additionally we were able to confirm that using exact Cholesky decompositions instead also lead to numerical errors.

From a practical perspective, we encountered these issues during later stages of the training process when the GP seemed close to convergence. We further suspect that this issue is in parts connected to the fact the we consider noise-less data: Indeed, we do not encounter this problem when adding a high amount of noise to the data, or raising the lower bound on the learned noise level. Since the Cholesky decomposition is performed on the covariance matrix of noisy predictions ($K + \sigma_n^2 I$) the added noise corresponds to an added diagonal jitter on the matrix, which is commonly applied to help numerical stability by keeping the eigenvalues large enough. However, since we consider noise-less data the noise level quickly diminishes and converges to the chosen lower bound of 10^{-4}.

Another consideration is the learned outputscale parameter, which multiplies the whole kernel matrix by some factor σ_f^2. By separating this multiplicative factor from the squared-exponential function, we could write the covariance matrix of noisy predictions as ($\sigma_f^2 K + \sigma_n^2 I$). Thus, large outputscale parameters $\sigma_f \gg 1$ effectively inversely scale the added diagonal matrix, while small outputscale parameters $\sigma_f \ll 1$ make the added noise more effective and help numerical stability. Indeed, we were able to overcome these numerical issues both by raising the lower bound on the noise parameter σ_n as well as by introducing an upper bound on the outputscale σ_f.

Finally, we researched related projects on GP dynamics models and encountered a hyperparameter penalty term in the original PILCO (Deisenroth and Rasmussen, 2011) implementation[1], which contained a *signal-to-noise*-term of the form

$$\text{penalty} = \left(\frac{\log\left(\frac{\sigma_f}{\sigma_n}\right)}{\log(1000)} \right)^{30} . \tag{5.4.1}$$

Instead of defining upper or lower boundaries on the parameters this term enforces both terms to be in a specified range of each other. We tried including this penalty as an additional loss term in the training objective and we found that the term helped to keep numerically stable values for the noise and outputscale. We chose this approach over setting stricter limits for the noise or outputscale term. All following experiments include this penalty as an additional loss term.

5.5 Dynamics-unaware state representation learning

We investigate different variations of auto-encoders and variational auto-encoders and evaluate their capability of learning low dimensional representations for the high-

[1] The original PILCO implementation can be found at `http://mlg.eng.cam.ac.uk/pilco/`.

dimensional observations. One of our motivations in this experiment is to validate the chosen network structure regarding its capabilities of inferring meaningful states from observations as well as in generating meaningful observations from these states. Additionally we are interested in the learned structure in the latent space and in its relation to the true physical state space.

Data

Since we do not consider a dynamics model in this task there is no notion of *time* for the model. We can combine the observations of the collected sequences into a single dataset of observations: $\mathcal{D} = \{o_i\}_{i=1}^{N \cdot T}$ with $o_i \in \mathbb{R}^{64 \times 64 \times 6}$. We considered $N = 100$ rollouts of length $T = 75$ on both the Pendulum and CartPole environment for training, and $N = 50$ additional rollouts of the same length for validation and testing, respectively.

Model Architecture

We closely followed a model architecture proposed by Ha and Schmidhuber (2018), shown in Fig. 5.3. In its original work this convolutional neural network was used to infer latent states from RGB images of size 64×64 which would then be used by an internal model of the environment in order to simulate future rollouts or to best interact with the real environment. Since we consider observations consisting of two images we modified the input and output layers accordingly.

We further chose the dimension of the bottleneck layer depending on the true physical dimension of the system. The pendulum environment can be described by the angle and angular velocity of the pendulum. Due to the common encoding of angles as its sine and cosine, in order to provide a continuous state space, we chose a three-dimensional bottleneck layer for the Pendulum environment. Similarly, we chose five dimensions for the CartPole data, the state of which consists of the position and velocity of the cart, together with the angle and angular velocity.

We explored different variations of this model, including β-VAEs (see Section 3.2.7) with varying β ($\beta = [0, 0.1, 1, 10, 100]$). Additionally, we compared the VAE to a standard non-variational deterministic auto-encoder with the same model architecture but where we omitted the sampling step $z \sim \mathcal{N}(\mu, \sigma)$, and instead chose the mean $z = \mu$.

Training

For both AEs and VAEs we use the binary cross-entropy (see Eq. (3.2.5)) as reconstruction loss. For VAEs we include the KL-divergence to the standard Gaussian prior as an additional loss term, and we scaled it by a factor β to compare β-VAEs (see Section 3.2.7).

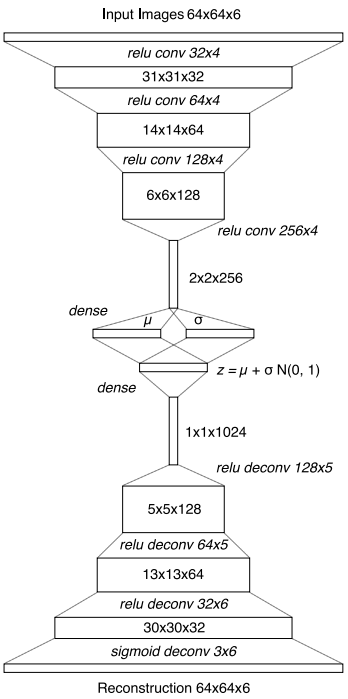


Figure 5.3: VAE architecture. For the deterministic (that is non-variational) auto-encoder we omit the sampling step and use $z = \mu$. Original figure by Ha and Schmidhuber (2018).

We used Adam (Kingma and Ba, 2014) as our optimization algorithm with a learning rate of $\eta = 10^{-3}$. We trained in batches consisting of 1024 observations and stopped the training after 2000 epochs. With our train/validation/test split of 100/50/50 rollouts we chose the models according to the lowest loss on the validation data. The presented results were then obtained by evaluating the models on the held-out test set.

Evaluation and Results

Tables 5.1 and 5.2 show metrics on the reconstruction of observations and regression from latent codes to the true physical states for the different variants (see Section 5.3). We further show visualizations of the learned three-dimensional latent space for the pendulum environment in Fig. 5.4 and Fig. 5.5 for an AE and a VAE, respectively.

Note that these images represent only a single experiment, and that in other experiments the learned representations differ, due to randomness such as a different parameter initialization or the batch sampling. However the images present characteristic features which we observed in all experiments: Comparing the learned representations in Fig. 5.4 and the true state space shown in Fig. 5.2 we see that

the model is able to recover the general structure of the original state manifold but fails to recover the meaning of the different axes, such that the result is a distorted version of the original open cylinder. In comparison, the states learned by a VAE shown in Fig. 5.4 also show signs of this distorted open cylinder, but located and folded in a way around the origin. This location is likely due to the additional objective of minimizing the KL-divergence with respect to a standard Gaussian distribution. Between these two learned representations we opine that for a human observer the former state representations exhibit a clearer structure. The quantitative results shown in Table 5.1 suggest that the latent representations learned by AEs allow for better regression of the true physical states, compared to the representations learned by VAEs.

On the other hand, for the CartPole environment we can not visually inspect the learned embeddings in the same way and we therefore have to rely more on the quantitative metrics. The results shown in Table 5.2 differ from those on the Pendulum environment: VAEs lead to a slightly better regression to the true physical states, but overall the values are very similar for the AE and for β-VAEs with values $\beta \in \{1, 0.1, 0\}$. However, large values of β lead to both worse reconstructions and worse regressions to the true physical states.

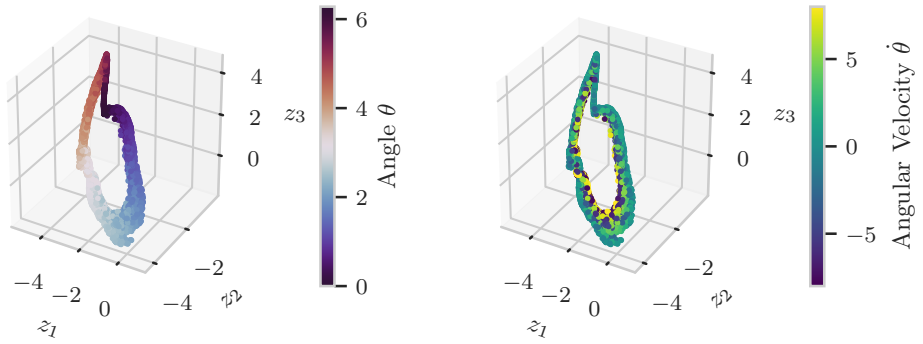
Method	Reconstruction Error	Regression Error
VAE	22.43 ± 0.06	0.54 ± 0.13
β-VAE, β=10	26.59 ± 0.32	0.62 ± 0.10
β-VAE, β=100	79.01 ± 0.01	1.17 ± 0.10
β-VAE, β=0.1	22.57 ± 0.28	0.35 ± 0.02
β-VAE, β=0	22.52 ± 0.30	0.39 ± 0.04
AE	**22.00 ± 0.1**	**0.29 ± 0.05**

Table 5.1: Test results on the Pendulum environment. "Reconstruction error" refers to a mean-squared error in image space between the true observations and reconstructions. The "regression error" is the error between the true physical states and the attempted regression of these, obtained by a support-vector regression.

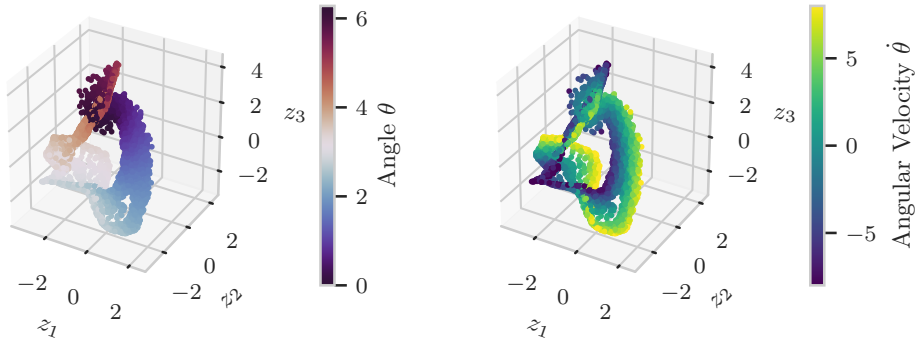
Conclusion

We trained both auto-encoders and variational auto-encoders on a standard representation learning task, given visual observations. The chosen architectures for encoder and decoder were able to learn latent representations which allow for very good reconstructions of the observations on the test data. Further, the visualizations of the learned latent embeddings seem structured and we argue that they can be interpreted from a human perspective.

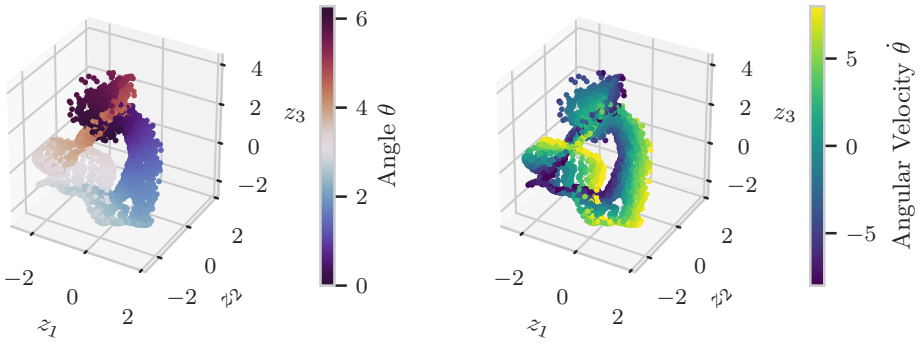
For the pendulum environment our experiments show better results for AEs compared to VAEs, not only on reconstruction of observations but in particular also regarding the



(a) Epoch 500

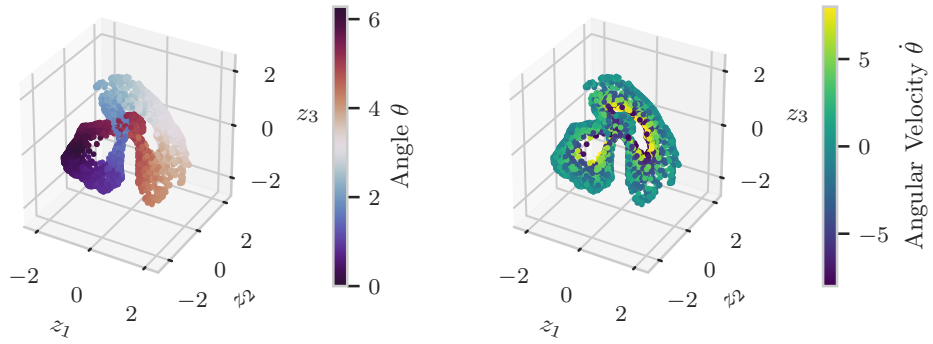


(b) Epoch 3000

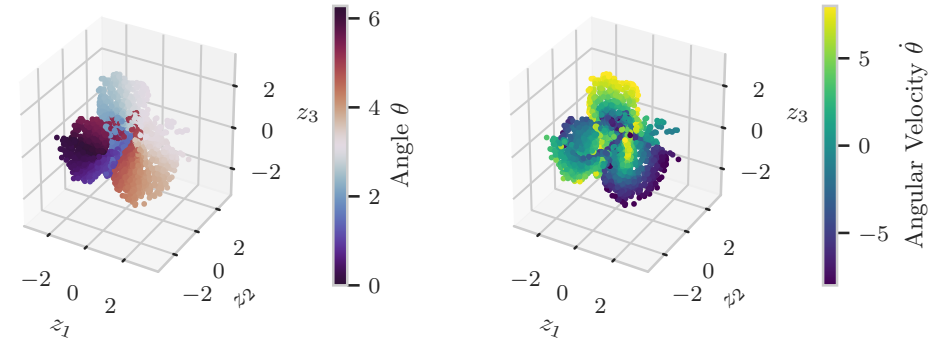


(c) Epoch 9000

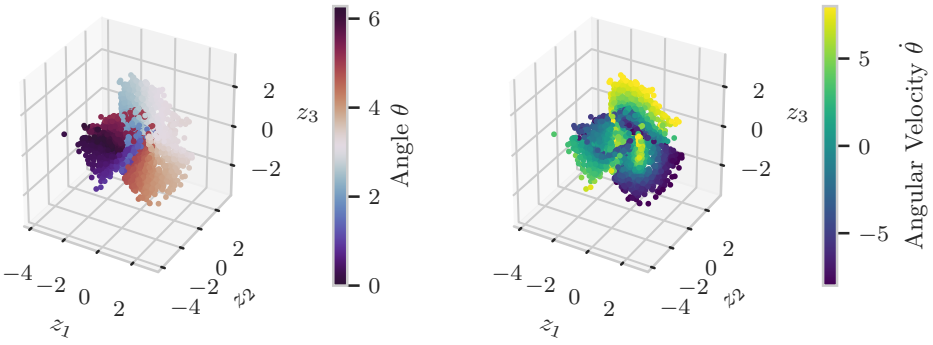
Figure 5.4: Learned representations of an auto-encoder for the Pendulum environment, during different stages of the training. Each point corresponds to the latent representation of a observation. Note how the angular velocity is not yet well separated at epoch 500, but is easily distinguishable at epoch 3000.



(a) Epoch 500



(b) Epoch 3000



(c) Epoch 9000

Figure 5.5: Learned representations of a variational auto-encoder for the Pendulum environment, during different stages of the training. Each point corresponds to the latent representation of a observation. As in Fig. 5.4 the angular velocity is hard to infer at epoch 500, whereas epoch 3000 shows it more clearly.

Method	Reconstruction Error	Regression Error
VAE	6.12 ± 0.19	**0.52 ± 0.02**
β-VAE, β=10	10.79 ± 0.33	0.63 ± 0.02
β-VAE, β=100	60.5 ± 0.86	0.75 ± 0.01
β-VAE, β=0.1	5.30 ± 0.47	0.53 ± 0.02
β-VAE, β=0	4.56 ± 0.33	0.54 ± 0.04
AE	**4.21 ± 0.40**	0.53 ± 0.01

Table 5.2: Test results on the CartPole environment. "Reconstruction error" refers to a mean-squared error in image space between the true observations and reconstructions. The "regression error" is the error between the true physical states and the attempted regression of these, obtained by a support-vector regression.

structure of the learned state representations: In a supervised support-vector regression (SVR) from the learned state representations to the true physical states we achieve much lower errors with the representations of standard AEs than with VAEs. We did not expect this outcome and instead assumed that VAEs would generally provide better state representations, since general consensus on AEs seems to be that they tend to produce unstructured latent state representations, whereas VAEs tend to learn a more structured and coherent latent space. We suspect that this result is due to the simplicity of the chosen environments, and indeed we do not observe this behavior on the CartPole environment. Instead, the regression error behaves similar for both AEs and β-VAEs with small $\beta \leq 1$. We suspect that for even more difficult environments, with more complex state spaces and more realistic images, VAEs would outperform AEs.

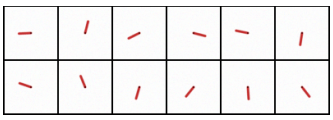
Overall, unsupervised representation learning seems to work very well for the chosen environments, and the models are able to recover the general structure of the state space.

5.6 GP dynamics models on fully observed physical states

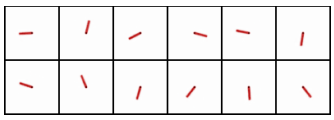
In this section we present results of learning dynamics models with Gaussian processes given the exact physical states of the observed dynamical system. For a thorough description of the approach and its theoretic foundations we refer to Section 4.2.

Data

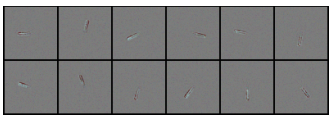
The method corresponds to a supervised Gaussian process regression with inputs (s_t, a_t) and targets s_{t+1}. We collect the training data by randomly interacting with the environment, as described in Section 5.2, for $N = 15$ interaction episodes of length $T = 75$, for both the Pendulum and CartPole environment.



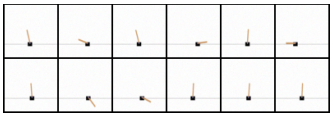
(a) Original images of the Pendulum environment.



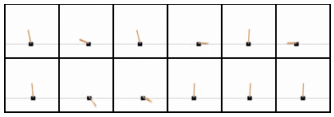
(b) Reconstructions of the images shown in (a).



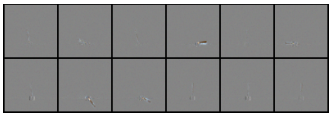
(c) Differences between the original (a) and the reconstructions (b).



(d) Original images of the CartPole environment.



(e) Reconstructions of the images shown in (d).



(f) Differences between the original (c) and the reconstructions (d).

Figure 5.6: Visualization of the learned image reconstruction capabilities on both the Pendulum and CartPole environments. We show the original images to be reconstructed on the left, the reconstruction of the corresponding encoder/decoder pair inthe middle, and the differences between these two images on the right.

Model

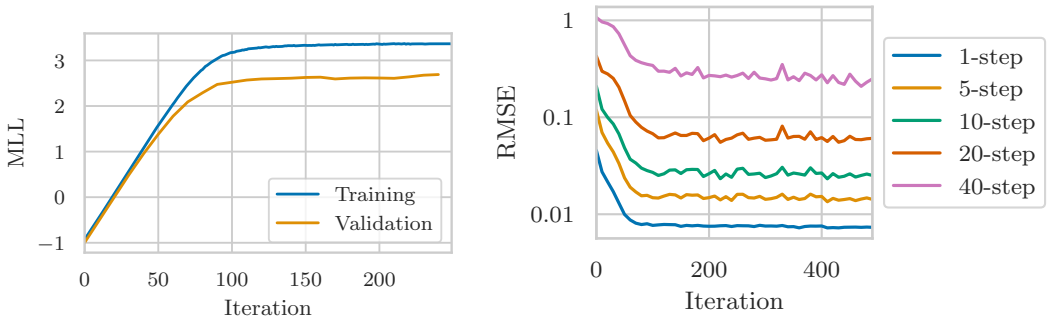
We model each target dimension as an independent Gaussian process, with its covariance function defined by an RBF kernel with automatic relevance determination. As proposed in Section 4.2 we chose the identity function as the mean function. Since each target dimension i is modeled by a separate GP f_i this mean is implemented as the projection along this respective dimension i. Additionally, we used a $\sigma_f \sim \Gamma(1, 5)$ prior (see Eq. (3.3.17)) on the outputscales and a $l_i \sim \Gamma(2, 0.5)$ prior on the lengthscales of the RBF kernel. The resulting model consists of D Gaussian processes $f_i \sim \mathcal{GP}\left(\text{proj}_i, k_{\text{RBF}}^{(i)}\right)$ for dimensions $i \in \{1, \ldots, D\}$, with $D = 3$ on the pendulum data and $D = 5$ on the CartPole data.

Training

We train all hyperparameters of the model with gradient descent using the Adam optimizer (Kingma and Ba, 2014) and a learning rate of $\eta = 0.1$, maximizing the marginal log-likelihood on the training data set. Additionally we consider the signal-to-noise penalty term as introduced in Section 5.4. Figure 5.7 shows the evolution of the training process. Both the marginal log-likelihood as well as the RMSE of the maximum-likelihood prediction of the next latent state converge quickly, reaching a plateau after around 150 iterations.

Additionally, Fig. 5.8 shows the evolution of the different GP hyperparameters over the course of the training. While we should rely on the values of the lengthscales

in order to rank the importance of different features (Paananen et al., 2017) we can interpret their convergence or divergence to state if the feature is considered at all. For both "GP-1" and "GP-2", which output the cosine and sine of the next angle, respectively, the cosine and sine of the current angle seem very important. However, the cosine of the current angle does not seem important in order to predict the next angular velocity. This corresponds to the true physical setting, gravity influences the velocity only by the sine of the current angle. Further, the action seems only to be of importance for the next velocity ("GP-3") while not influencing the next angle ("GP-1" and "GP-2"). This is again plausible in the true dynamical system, since the torque modifies the future velocity while the influence to the change in angle is very small. The noise converges to the defined lower bound of $10^{-}4$ which is expected since we consider noise-less data. Finally, the outputscale also converges to small values $\sigma_f \in (0.01, 0.1)$. However this is due to the signal-to-noise penalty (Eq. (5.4.1)). Indeed, without this penalty the GPs adopted higher outputscales, but this also lead to numerical issues during training.



(a) Marginal log-likelihood.

(b) Means of the prediction losses in latent space over the specified time horizon.

Figure 5.7: Visualization of the training process of GP dynamics models on the true physical states of the Pendulum environment.

Results

We evaluate the quality of the resulting model by simulating episodes in the learned dynamics model. We present the results both qualitatively and quantitatively: Using the rendering engines of the corresponding OpenAI gym environments (`Pendulum-v0` and `CartPole-v0`) we can visualize the proposed sequences and compare them to the ground truth, as shown in Fig. 5.10 and Fig. 5.11. We further quantify the losses in both latent space and image space over the prediction horizon on a test-set of true sequences, shown in Fig. 5.9. Finally, we provide values on the MLL during training as

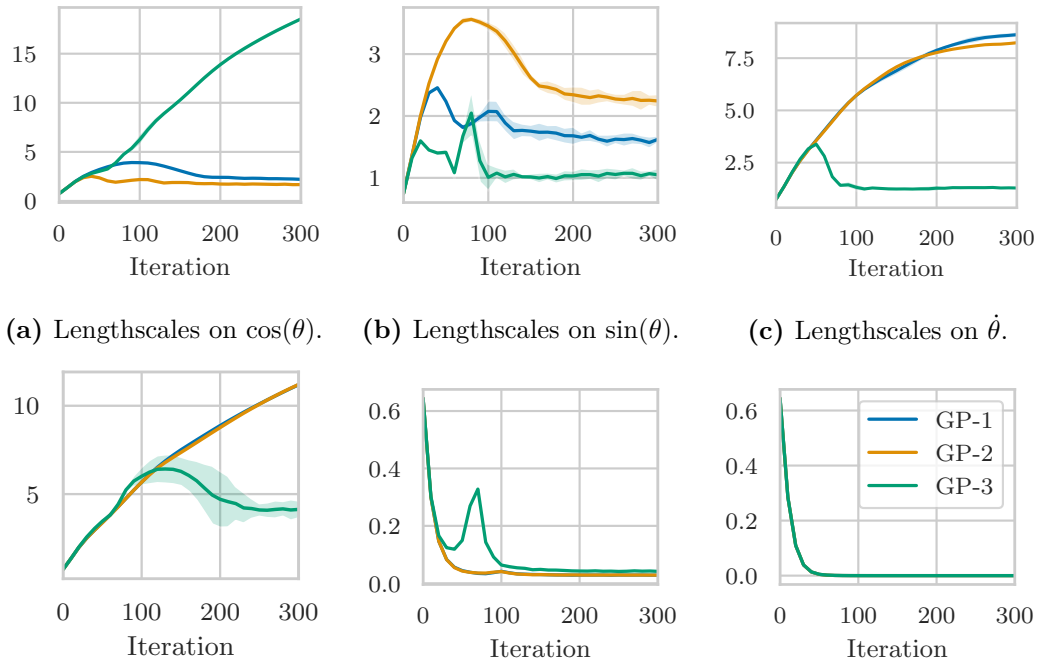
well as on the RMSE on the predicted latent states in Table 5.3.

	MLL on train data	RMSE on test data
Pendulum	3.43 ± 0.0024	0.007 ± 0.0001
CartPole	3.54 ± 0.0021	0.0013 ± 0.0001

Table 5.3: Results for GP dynamics models trained on true physical states. The MLL shown is obtained during training, and the RMSE meansures errors for single-step predictions in latent space on held-out test data.

Conclusion

We learned a Gaussian process dynamics model from 15 sequences of interactions with a fully observable dynamical system. Both the training and validation losses converge quickly to a plateau, thus the model seems to generalize well and does not show signs of over fitting. The mean-square errors of the maximum likelihood predictions in both latent space and image space decrease as the marginal log-likelihood increases. Visually, the rendered simulated rollouts seem close to the ground truth, even for prediction horizons of up to 20 or 30 steps, on CartPole or Pendulum, respectively. For examples in which the prediction deviates from the ground truth the predicted trajectory still seems physically reasonable. The model is sample efficient, converges fast, and provides accurate predicitons.



(a) Lengthscales on $\cos(\theta)$.

(b) Lengthscales on $\sin(\theta)$.

(c) Lengthscales on $\dot{\theta}$.

(d) Lengthscales on the action.

(e) Outputscales.

(f) Noise.

Figure 5.8: Hyperparameter evolution of GP dynamics models trained on the true physical states of the Pendulum environment. The colors denote the different Gaussian processes for each output dimension, respectively, as declared in the legend shown in Fig. 5.8f.

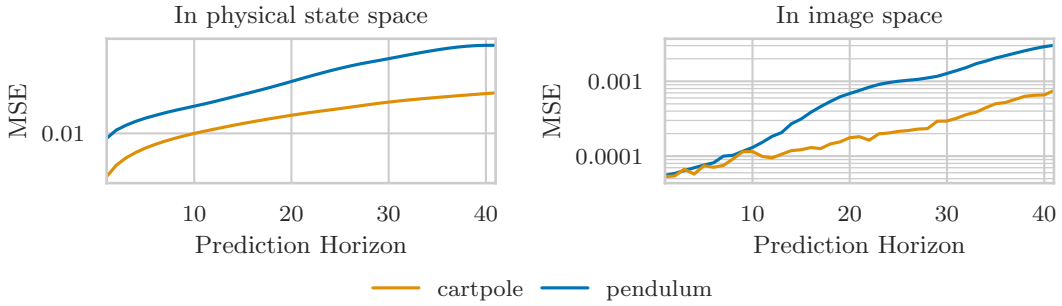


Figure 5.9: Errors of predicted latent states and observations over the prediction horizon.

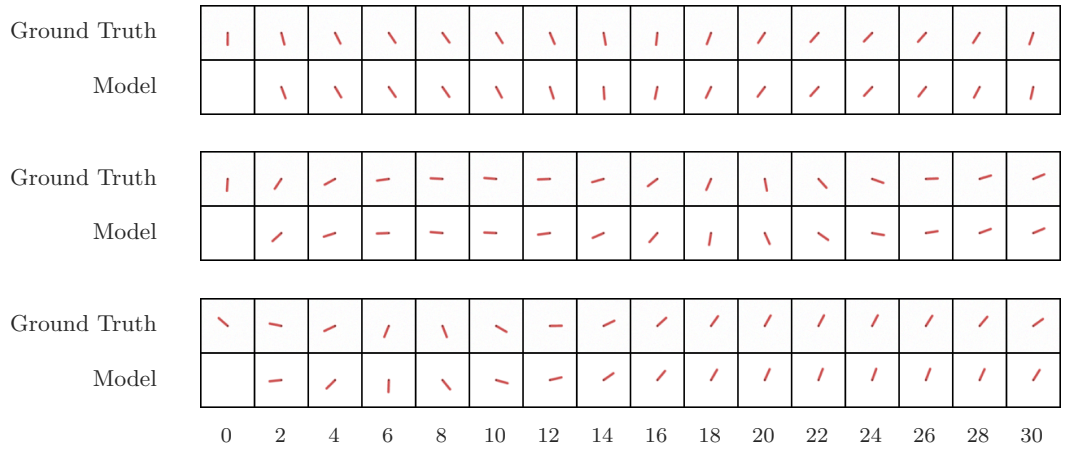


Figure 5.10: Example rollouts from a GP dynamics model in the Pendulum environment. The GP dynamics model recieves the initial physical state at time $t = 0$ together with a sequence of actions $a_{0:29}$ and generates a sequencec of states $s_{1:30}$. We then decode these predicted true physical states using the true original renderer of the OpenAI gym environment.

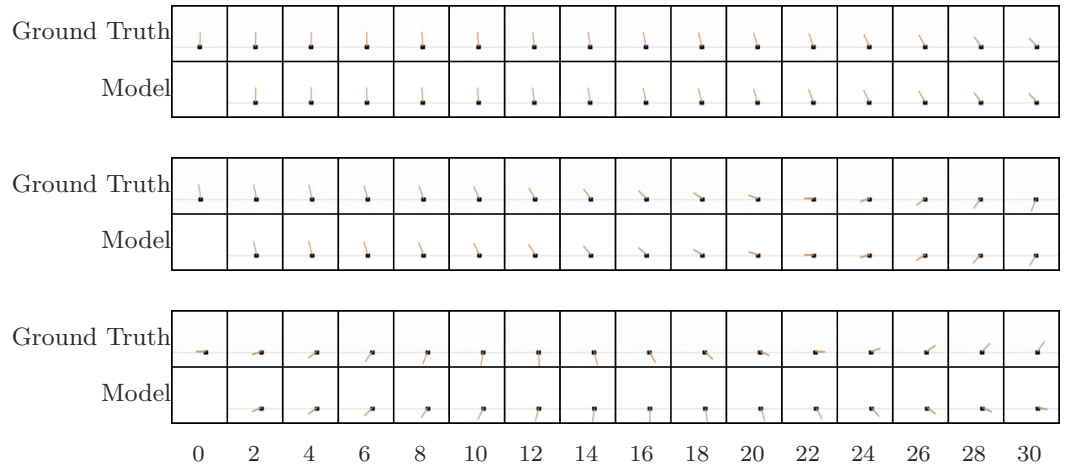


Figure 5.11: Example rollouts from a GP dynamics model in the CartPole environment. We proceed as in Fig. 5.10 and predict future states from an initial state and an action sequence with the learned GP dynamics model. The shown observations are generated with the true renderer of the CartPole environment.

5.7 Learning System Dynamics from Visual Input

In the following we present the main results of this thesis: We learn latent GP dynamics models from visual observations. We start with the simplified approach of first learning state representations with AEs or VAEs, unaware of a transition model, on which we then fit a GP dynamics model. We then jointly learn the latent representations and the latent transition model, as proposed in Section 4.3. We compare the results to the previous models and discuss benefits and shortcomings of the proposed method.

5.7.1 GP dynamics models on pre-trained dynamics-unaware embeddings

We follow the two-stage training approach as described in Section 4.5.

First stage: Learning state representations

In the first stage we learn state representations in an unsupervised way with an auto-encoder. This corresponds exactly to the setting we investigated in Section 5.5. We therefore reuse the presented models and results, choosing a deterministic auto-encoder for the Pendulum environment and a variational auto-encoder for the CartPole environment, respectively.

The models are trained on $N = 100$ rollouts of length $T = 75$, minimizing the respective training objective using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of $\eta = 0.001$. For more details, such as regarding the architecture, the exact training objectives, or model selection, we refer to Section 5.5.

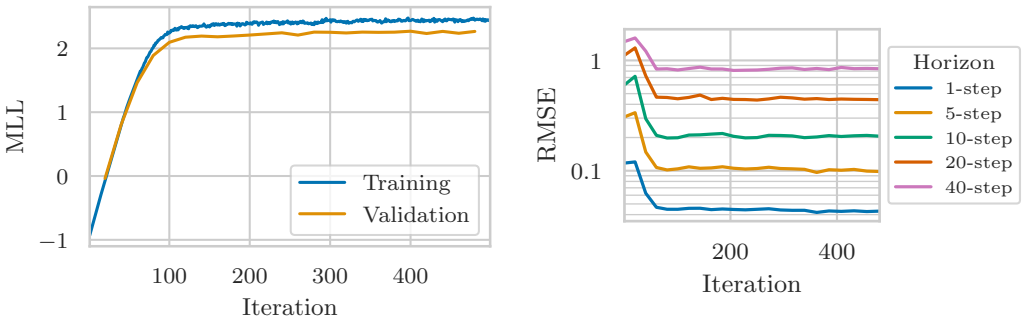
Second stage: Learning latent GP dynamics

In the second stage we learn a Gaussian process dynamics on the learned, but fixed, state representations. This step essentially corresponds to learning a Gaussian process dynamics model on physical states, as introduced in Section 4.2. In order to learn this dynamics model we can proceed as in Section 5.6 while considering learned state representations $\hat{s}_t = g^{-1}(o_t)$ instead of the true physical states s_t given by the fixed encoder g^{-1}.

The model specifications follow those in Section 5.6. We use $N = 15$ randomly selected rollouts of length $T = 75$ to learn the hyperparameters of the GP dynamics model, by minimizing the marginal log-likelihood of the transitions in latent space using the Adam algorithm (Kingma and Ba, 2014) with a learning rate of $\eta = 0.1$.

We trained the GP dynamics model for 500 epochs. Figure 5.12 shows the evolution of the GP training process. Both the marginal log-likelihood on the training data as well as the RMSE between the maximum-likelihood prediction and the true latent transition on the validation set converge after ~100 iterations. Since the validation loss

does not increase during this extended training period the model does not seem to overfit.



(a) Marginal log-likelihood.

(b) Agreggated latent prediction losses.

Figure 5.12: Evolution of errors during the training process. The left plot (a) shows the MLL over both the training and validation set which both converge after ~ 100 iterations and show no sign of overfitting. On the right plot (b) we present RMSEs of the latent predictions of the dynamics model against the true latent states, for different prediction horizons.

Evaluation and results

The learned state representations correspond to those presented and evaluated in Section 5.5. Further, since these representations are fixed for this experiment, we can evaluate the GP dynamics model very similarly to our evaluation of the GP dynamics models on true physical inputs in Section 5.6.

We evaluate the learned dynamics model both qualitatively and quantitatively in latent and image space, using the fixed decoder to generate images from latent states. Figures 5.13 and 5.14, show example sequences generated from the learned dynamics model, for the Pendulum and CartPole environment, respecitvely. Figure 5.15 visualizes errors in latent space and image space over the prediction horizon. We remind the reader that the errors in state space can not directly be compared, since the latent representations are learned by the model and their scale might be very different to the true physical states. The losses in image space on the other hand can be directly compared.

The GP dynamics model on true physical states seems to provide smaller errors in the predicted observations. This difference can to some degree be attributed to the fact that the model uses a decoder for image generation, instead of having access to the true renderer, but we believe that a large part of this difference is due to the transition model. Looking specifically at the loss curves for CartPole, we see that both methods

start with a similar error, thus indicating near-perfect decoding since for a single-step prediction the changes in state space are minimal. With increasing prediction horizon the inaccuracies increase much more rapidly for the GP dynamics model on learned state representations, indicating a less exact transition model.

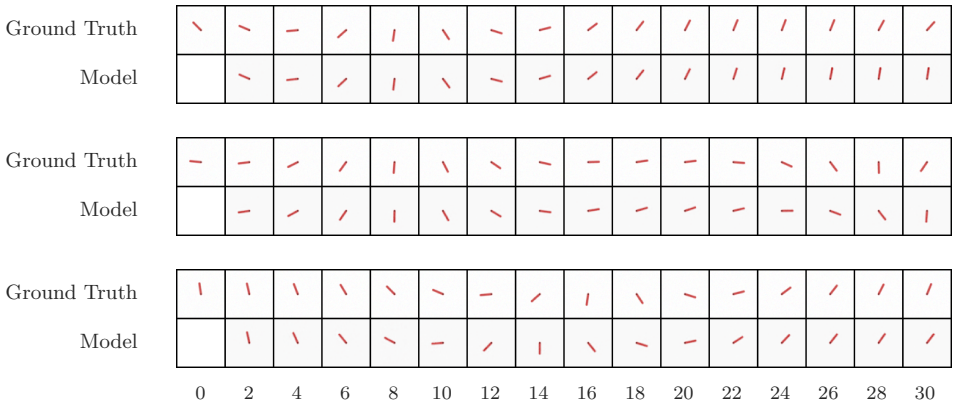


Figure 5.13: Simulated sequences on the Pendulum environment, predicted from the GP dynamics model trained on dynamics-unaware latent state embeddings. In all three examples the model has only access to the initial observation at timestep $t = 0$. We infer a latent state with the learned encoder and, given a random set of actions, generate a sequence of latent states. We can then use the learned decoder to infer observations from these latent states for each time step t which we can then compare to the ground-truth observations.
In the first example we show very accurate predictions which coincide well with the ground truth even for larger time horizons. Both the second and third example show more inaccuracies, after around 20 and 10 steps, respectively. Afterwards the predicted observations deviate from the ground truth, but the predicted trajectory still seems physically plausible.

Conclusion

We trained a Gaussian process dynamics model on learned, but fixed, latent state representations. The training loss converges quickly while the validation loss does not increase, indicating that the model does not overfit. The decoded simulated rollouts appear to be physically reasonable for both the Pendulum and CartPole environment, but they seem to become less accurate for larger prediction horizons in comparison to GP dynamics models on the true physical states (as presented in Section 5.6). We confirm this difference quantitatively (Fig. 5.15) and we argue that a large part of this difference should not be attributed to the imperfect decoder but to inaccuracies in the transition model itself.

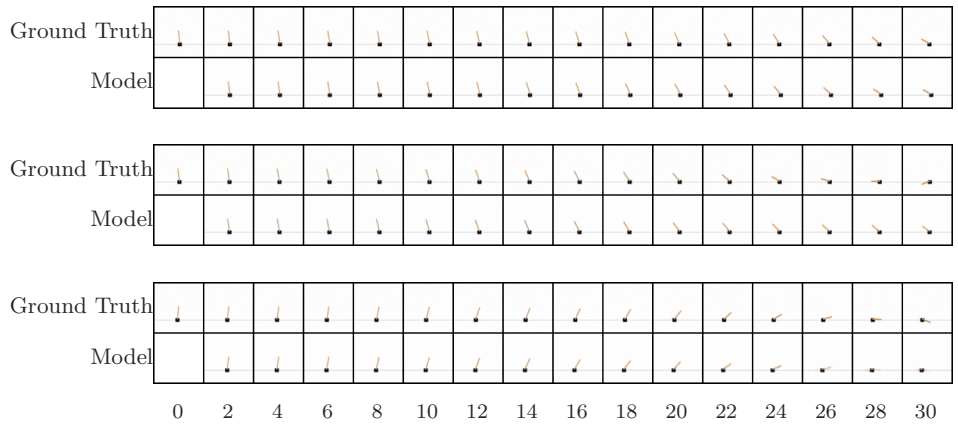


Figure 5.14: As in Fig. 5.13 we infer an initial latent state from the initial observation with the decoder, and then generate sequences of states withgiven actions and the learned dynamics model. We present both the ground-truth observations together with the decoded predicted states.

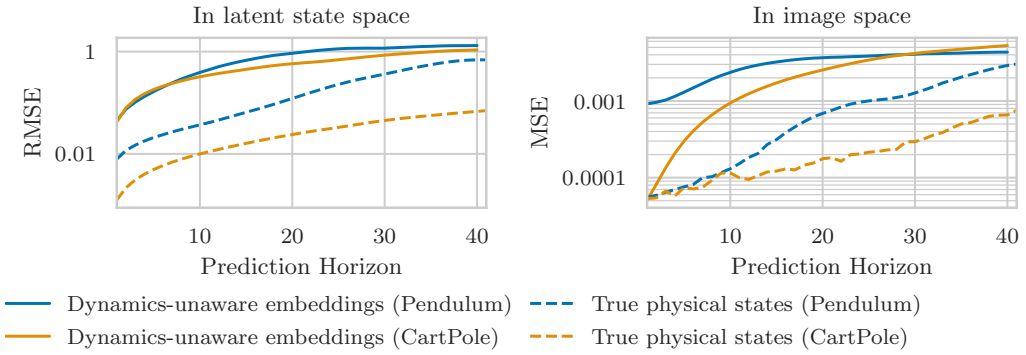


Figure 5.15: Errors of predicted latent states and visual observations over the prediction horizon. The continuous line shows the obtained results with the GP dynamics model on dynamics-unaware emebddings and the dashed line shows the previous results of a GP dynamics model on the true physical states, as presented in Fig. 5.9. We show results for both the Pendulum environment in blue and CartPole in orange.
Note that the learned dynamics-unaware latent states and the true physical states do not live in the same space or scale. The left plot should therefore not be overly interpreted in order to judge model quality, but serves to compare the performance of a single model over different time horizons, as well as to relate the errors in latent space to the errors in the predicted observations.

We highlight that both transition models use the same approach, modeling system dynamics with a separate Gaussian process for each dimension, and both models are trained with the same training procedure on a comparable amount of training data. The differences in performance should therefore be attributed to the learned state representations, which seem to be less beneficial for GP dynamics models, compared to the true physical states. This realization motivates the proposed method and the next experiments: A joint training schedule might enforce a more helpful structure in the latent representation.

5.7.2 Joint training of GP dynamics models and latent state embeddings

We present latent GP dynamics models, jointly trained with the encoder and decoder on the MLL-based training objective which we derived in Section 4.4.2. For a thorough motivation and definition of the method we refer to Section 4.3.

Model Architecture

The encoder/decoder pair follows the previous architecture as specified in Section 5.5, consisting of a convolutional neural network for the encoder and a deconvolutional network to generate images from latent states. See also Fig. 5.3 for an illustration of the architecture. We again chose the latent space to be three-dimensional on the Pendulum environment and five-dimensional on CartPole. For both the Pendulum environment and the CartPole environment we chose a regular deterministic auto-encoder.

Similarly, the Gaussian process dynamics model is as previously specified in Sections 4.2 and 5.6. We model each dimension of the predicted latent state with an independent Gaussian process, using an an RBF kernel with automatic relevance determination. We chose $\sigma_f \sim \Gamma(1, 5)$ (see Eq. (3.3.17)) as the prior distribution of the outputscales and $l_i \sim \Gamma(2, 0.5)$ for the lengthscales.

Data

During our development and experimentation we explored a vast number of configurations regarding the used training data. The $O(n^3)$ scaling of Gaussian processes with the number of inputs sets an upper bound on the amount of training data we can use for full-batch training. The proposed mini-batch training approach (Section 4.4.2) circumvents this issue and enables us to include larger amounts of data to increase the diversity of visual observations for the encoder and decoder, while keeping a smaller amount of data in the GP evidence. We generally observed that using a larger dataset together with the mini-batch training lead to increased performance of the resulting model. This was especially notable in the more complex CartPole environment. The final results we report were obtained with a training dataset consisting of 500 sequences, each with 30 transitions, for both the Pendulum and CartPole environment. We

additionally used 50 sequences for validation, as well as 50 sequences as held-out test data.

We also considered an additional hyperparameter which is common in reinforcement learning: *Action repeat*. Explained with an example, an action repeat value of 5 means that we apply the chosen action 5 times before collecting the next observation, while summing all intermediate rewards. This effectively corresponds to downsampling the temporal dimension and the resolution of the collected data gets more coarse. An increased action repeat reduces the planning horizon of the transition model and makes long-term predictions easier, while at the same time increasing the influence of the chosen action on the resulting transition and trajectory, which supports model learning (Mnih et al., 2016, 2015). During our exploration phase we found action repeat to be influential for the resulting model, regarding the training, the learned embeddings, and overall performance. Notably, Hafner et al. (2019) report action repeat as an important hyperparameter in PlaNet. On the Pendulum environment we achieved good results with the default action repeat of 1, but we chose an action repeat of 3 on CartPole.

Training

We jointly train encoder, decoder, and the hyperparameters of the GP dynamics model on the MLL-based training objective as derived in Section 4.4.2. We apply the proposed mini-batch training scheme as motivated in Section 4.4.2, allowing the encoder/decoder pair to experience a larger variety of observations while keeping a constant complexity for the GP. We fixed the number of data points in the GP evidence to 2000 transitions, and we chose batch sizes of 512 and 2000 for the Pendulum and CartPole, respectively.

We optimize the training objective with the Adam algorithm (Kingma and Ba, 2014). We chose a learning rate of $\eta = 10^{-3}$ for the encoder/decoder pair, and we use a learning rate schedule for the GP dynamics model to smoothly transition from a low initial learning rate of $\eta_0 = 10^{-5}$ to a high goal learning rate of $\eta_* = 0.05$. In the following we motivate such a learning rate schedule and we define the chosen function. During early iterations of the training the latent codes are not yet very structured and they might resemble noise. However, we do not want the GP to fit its hyperparameters in a way to explain everything with the learned noise, since we might not be able to get out of this local optimum in later stages when latents are more structured. On the other hand, earlier experiments have shown that GP dynamics models can be learned with relatively high learning rates, such as $\eta = 0.1$, and we observed that smaller choices significantly increase the required time for convergence. We therefore chose a learning rate schedule to transition from a very low initial learning rate $\eta_0 = 10^{-5}$ to a high goal learning rate $\eta_* = 0.05$. In order to get an almost exponential initial growth, together with a smooth transition to the goal learning rate, we chose to model this

transition with a logistic function of the form

$$\eta(t) = \frac{\eta_* \cdot \eta_0}{\eta_0 + (\eta_* - \eta_0) \cdot e^{-\eta_* k t}}, \tag{5.7.1}$$

where k is defined such that we can set the turning point in the logistic function to a chosen epoch E, which leads to $k := \frac{\log(\frac{S}{a-1})}{E*S}$. With this choice we achieve a learning rate of $\eta(E) = \frac{\eta_* - \eta_0}{2} \approx \frac{\eta_*}{2}$ at the chosen turning epoch E, and we have $\eta(2E) \approx \eta_*$. Figure 5.16 shows this learning-rate schedule for $\eta_0 = 10^{-5}$, $\eta_* = 0.05$ and $E = 1000$ both on a standard and logarithmic scale.

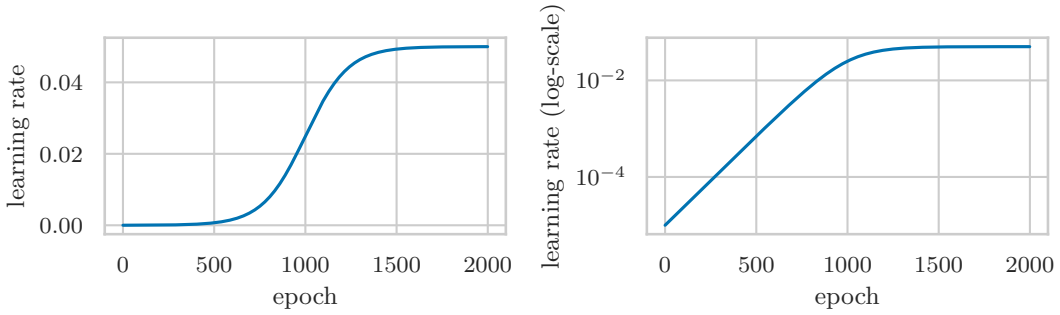


Figure 5.16: Learning rate schedule, shown both with a normal y-axis on the left and with a logarithmic y-axis on the right to show the early exponential growth.

Finally, we trained the model for 3000 and 5000 epochs, on Pendulum and CartPole, respectively. The models for which we report the results were chosen as the models with lowest loss on the validation data.

Evaluation and Results

We start by evaluating the results on the Pendulum environment. Figure 5.17 shows the learned latent space. Comparing these representations with the true physical state space shown in Fig. 5.2 the model seems to recover the original structure of the state space very well. In contrast with previously learned latent space manifolds (see Section 5.5) there do not seem to be large distortions, folds, or twists, and instead it can be clearly seen as a rotated and scaled version of the original open cylinder.

To better visualize the training process we visualize the GP hyperparameters in Fig. 5.18. We can not directly interpret the shown values, or even the separate GPs for the different output dimensions, since the latent state is not interpretable. By the rotation of the state manifold the different coordinates are entangled, and each dimension contains, to some degree, information on the sine and cosine of the pole angle and on the angular velocity. However, looking at the latent space more precisely

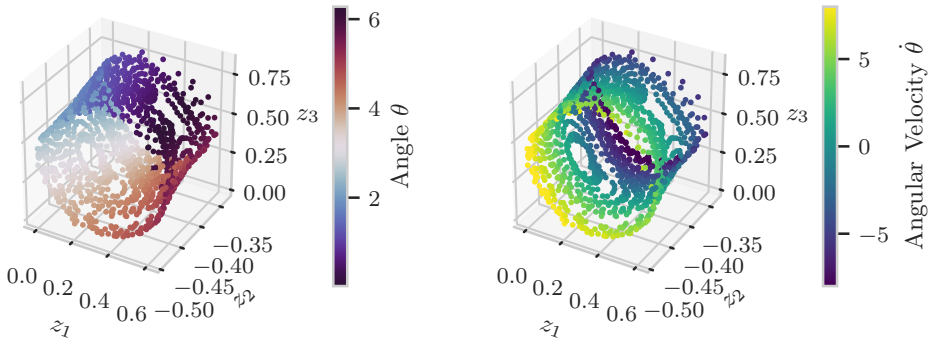


Figure 5.17: Learned representations on the Pendulum environment. This looks very similar compared to the true physical state space, shown in Fig. 5.2. The model seems to recover the original structure of the latent space up to rotation and scaling.

it seems that for example the second dimension z_2 correlates more strongly with the angular velocity. Additionally, the final value of the action lengthscales (Fig. 5.18d) is lowest for "GP-2", which predicts z_2, and higher for the other dimensions. We already observed a similar behavior for GP dynamics models on true physical states (Fig. 5.8). Finally, no observed lengthscale value seems to diverge during the training.

We show example sequences generated from the learned dynamics model, for the Pendulum environment, in Fig. 5.19. Figure 5.20 shows a quantitative evaluation of the losses in both latent and image space and includes a comparison to GP dynamics models on physical states (see Section 5.6) and on pre-trained dynamics-unaware states (see Section 5.7.1). We remind the reader that the errors in state space can not directly be compared, since the latent representations are learned by the model and their scale might be very different to the true physical states. The losses in image space on the other hand can be directly compared.

We see large improvements for the Pendulum environments, for both short-term and long-term predictions. The initial predictions almost matches those of the physical dynamics model, and we are able to predict up to 10 steps before reaching the single-step error of the GP on dynamics-unaware embeddings. This generally shows a very large improvement over the previous results. However, the difference to the true physical states still seems significant, especially for larger prediction horizons. We were not able to produce similar results on CartPole. The rollouts shown in Fig. 5.21 seem physically plausible, but the quantitative evaluation (Fig. 5.20) shows that we could not significantly outperform the previously presented GP dynamics model on pre-trained dynamics-unaware embeddings.

During our development and experimentation we tried many different settings, regarding the data, training schedule, and even the model, but we found it very difficult

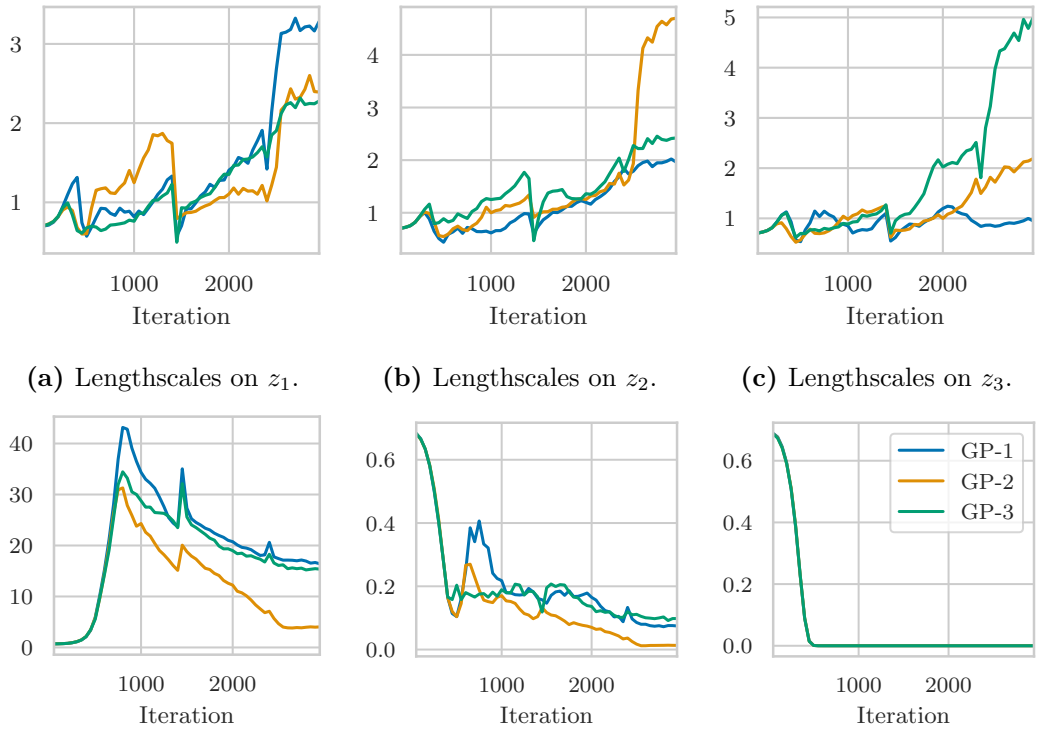
to obtain good results for CartPole. These findings motivate both a more exhaustive experimentation, as well as changes and improvements to the proposed method, which could be an interesting topic for future work. We discuss some ideas for modifications of the training schedule and objective in the context of control following section.

Training Stability

Another shortcoming we discovered is the instability of the training, and we found it difficult to recreate results. To better visualize this statement we refer to Fig. 5.22, which shows the mean-squared error of one-step predictions in image space over the training iterations, for 8 different training runs. The plot contains very stable runs with low errors, such as those shown in brown, green, and orange, but also runs that oscillate around a mean error which is an order of magnitude higher. Then, we chose the training iteration with the lowest validation loss for each of those runs and computed their loss for different training horizons. Figure 5.23 shows the aggregation of these losses after applying a logarithmic scaling, and includes comparisons to GP dynamics models on both fixed pre-trained latent states, as well as on physical states. The "min" line roughly corresponds to the presented main results, outperforming the separately trained model. However, if we consider the mean over these 8 runs we match the predictive performance in image space of the separate model after already 5 steps.

Conclusion

The proposed method shows a very strong performance on the pendulum environment. It recovered the structure of the true physical state space up to scaling and rotation and and significantly outperforms the separately trained model. However, we were not able to extend these results to the CartPole environment. While the added complexity in image and state space seem to be difficult to learn, the issue could also be linked to a general instability of the method. We found it difficult to recreate results and observed a large variance when training multiple runs with the same settings. These issues might provide interesting starting points for future research.



(a) Lengthscales on z_1.

(b) Lengthscales on z_2.

(c) Lengthscales on z_3.

(d) Lengthscales on the action.

(e) Outputscales.

(f) Noise.

Figure 5.18: Hyperparameter evolution during the training. The colors denote the different Gaussian processes for each output dimension, respectively, as declared in the legend shown in Fig. 5.18f.

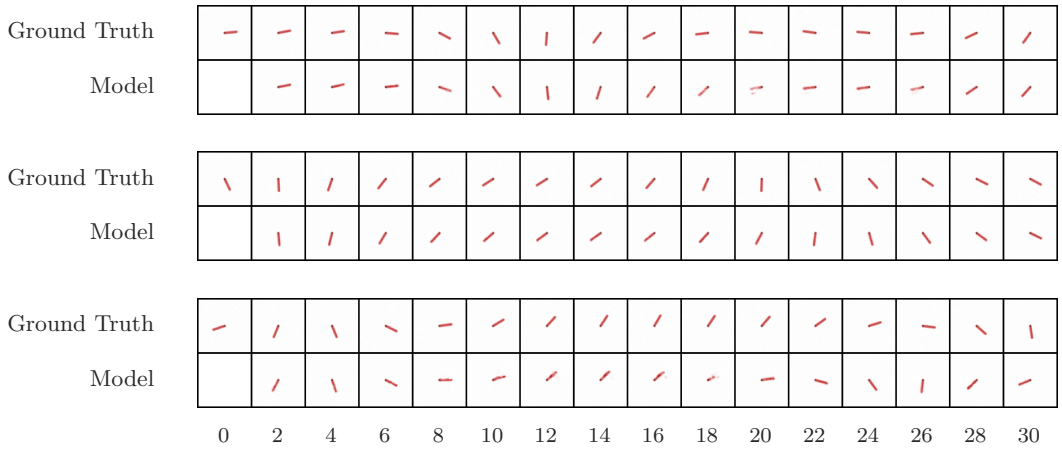


Figure 5.19: Simulated sequences on the Pendulum environment, predicted with jointly trained model. In all three examples the model has only access to the initial observation at timestep $t = 0$. We infer a latent state with the learned encoder and, given a random set of actions, generate a sequence of latent states. We can then use the learned decoder to infer observations from these latent states for each time step t which we can then compare to the ground-truth observations.

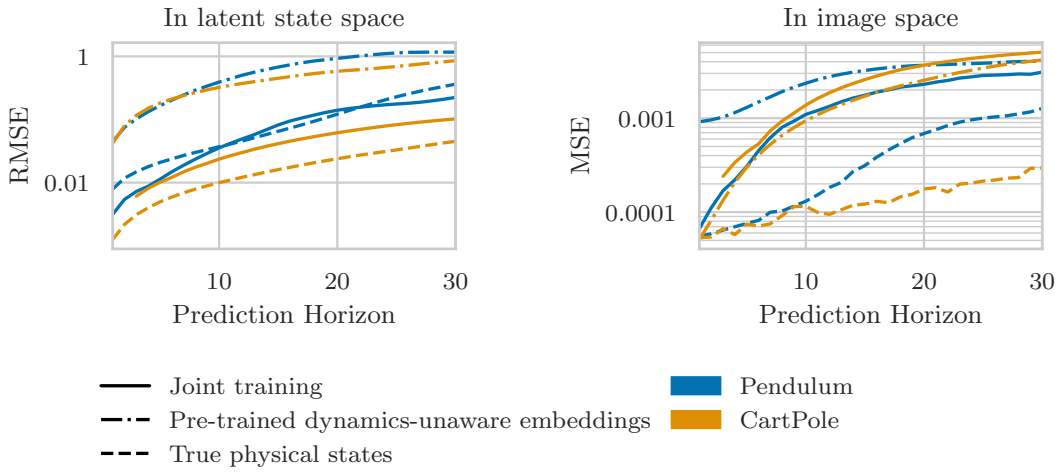


Figure 5.20: Errors of predicted latent states and visual observations over the prediction horizon. The continuous lines show the obtained results with the jointly trained latent GP dynamics model. The dash-dotted line shows the previous results of a GP dynamics model on dynamics-unaware emebddings and the dashed line shows results of a GP dynamics model on the true physical states. We show results for both the Pendulum environment in blue and CartPole in orange. Note that the learned dynamics-unaware latent states and the true physical states do not live in the same space or scale. The left plot should therefore not be overly interpreted in order to judge model quality, but serves to compare the performance of a single model over different time horizons, as well as to relate the errors in latent space to the errors in the predicted observations.

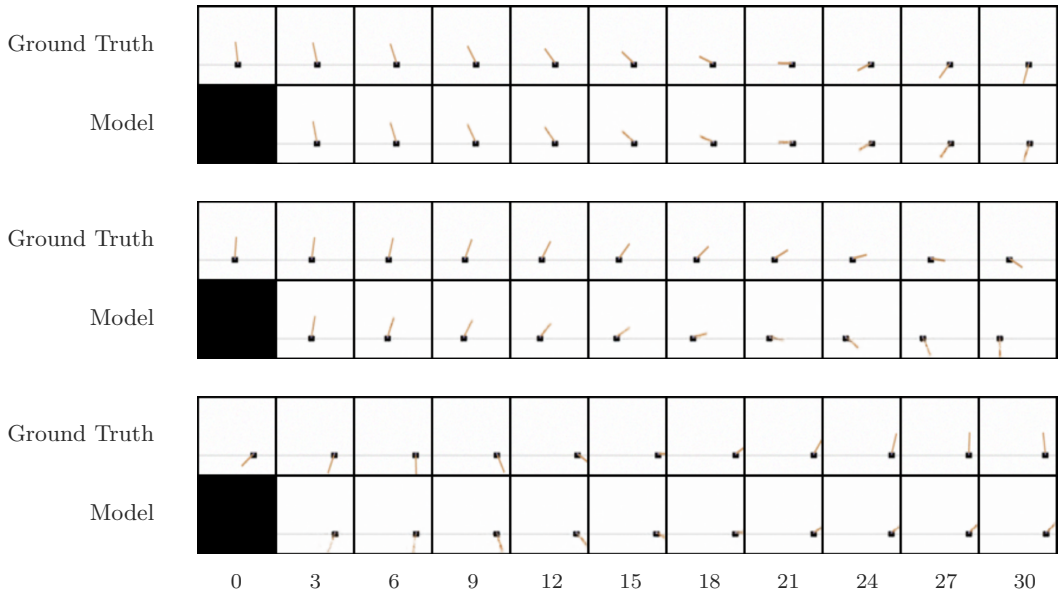


Figure 5.21: Simulated sequences on the CartPole environment, predicted with jointly trained model. Note that the model was trained with an action repeat of 3. The shown 30-step predictions thus effectively correspond to 10 recursive predictions of the learned dynamics model.

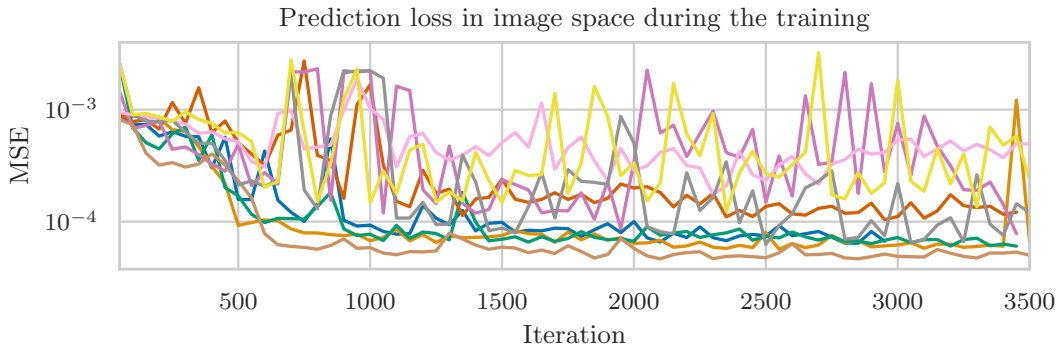


Figure 5.22: Mean-squared errors of 1-step prediction in image space over the course of the training for 8 separate runs with the same specifications. The plot contains very stable runs with low errors, such as those shown in brown, green, and orange, but also runs that oscillate around a mean error which is higher by an order of magnitude. This visualizes an unstable training and low reproducibility.

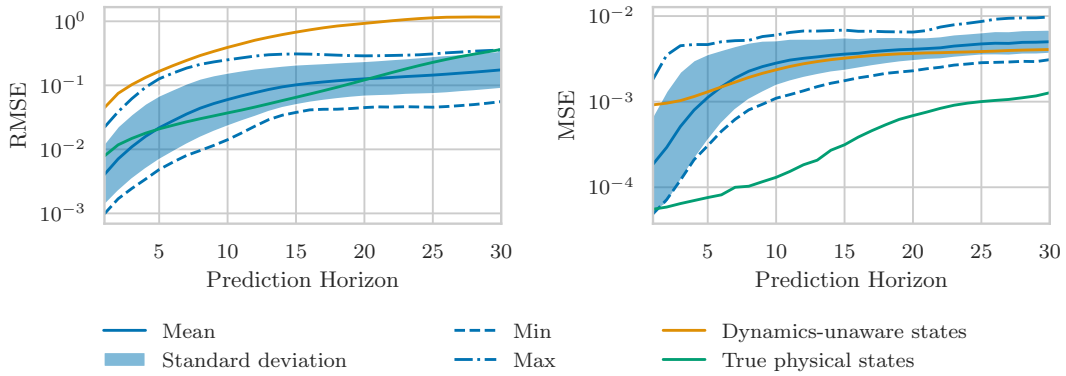


Figure 5.23: Errors of predicted latent states and visual observations over the prediction horizon. This plot is very similar to Fig. 5.20 but includes the variance of multiple training runs with the same specification, visualized with mean and standard deviation, as well as by the minimum and maximum values. Note that the standard deviation was computed and visualized in logarithmic scale.
The dashed line corresponding to the minimum value roughly corresponds to the presented main results, outperforming the separately trained model. However, the mean matches the predictive performance in image space of the separate model after already 5 steps.

5.8 Control

We apply the previously trained models to specific tasks in the considered environments, notably a swing-up task in the Pendulum environment and a pole balancing task in the CartPole environment. For more information on the environments or the tasks see Section 5.1.

Actions are chosen by planning in the learned latent space with the cross-entropy method as described in Section 4.7, while using the learned GP dynamics model to generate state sequences given an initial observation and a sequence of actions. In order to evaluate the proposed trajectories we learn a reward-model $p(r_t|s_t, a_t)$, parametrized by a neural network. We train the network on a set of training data, consisting of sequences of observations, actions, and rewards. Latent states are inferred using the trained encoder. We visualize the planning and show successful results in the Pendulum environment, confirming that the learned dynamics models can serve for planning in latent space.

Reward model

We consider collected episodes $\mathcal{D} = \left\{ \{(o_t^i, a_t^i, r_t^i)\}_{t=1}^T \right\}_{i=1}^N$, including the true rewards r_t^i. We want to learn a reward model $r \sim p(r|s, a)$, where the latent state s is inferred by the fixed encoder $p(s|o)$. We model the reward function with a standard feed forward neural network, with states and actions as inputs and the estimated scalar reward as output. Since we consider a fixed encoder this corresponds to a standard supervised regression task and we train the neural network to minimize the mean-squared error between the predicted reward and the true reward.

The chosen architecture consists of two fully connected layers of size 100 with ReLU activations, as well as a fully connected layer to the single scalar output without activation function. We use 20% of the available data as validation data and stop the training once the validation loss increases for 5 subsequent epochs. Finally, we choose Adam (Kingma and Ba, 2014) as the optimization algorithm. The resulting reward landscape for the pendulum task can be seen in Fig. 5.24. Figure 5.25 shows the equivalent visualization for the CartPole environment, but due to the 5-dimensional latent space we can only visualize projections to a selection of three of the dimensions.

Planning

For planning we use the CEM (see Section 4.7) with a planning horizon of $H = 20$, $I = 10$ optimization iterations, $J = 10000$ candidate samples, and we refit the belief to the best $K = 100$ candidates.

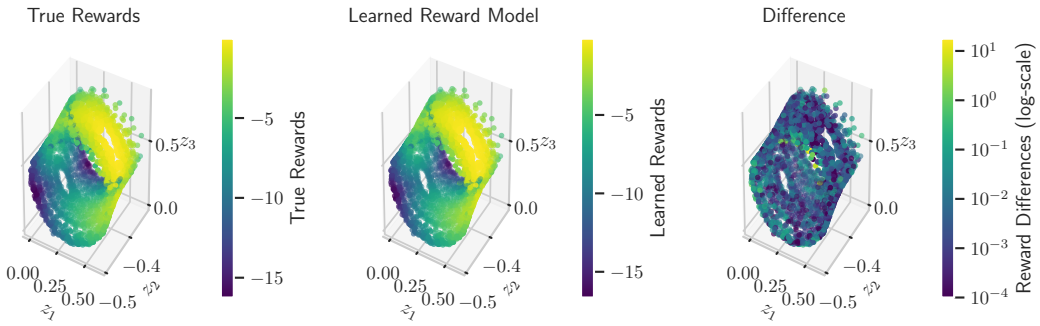


Figure 5.24: Rewards in the learned latent space for the pendulum task, using the learned model from Section 5.7.2. The differences (plot to the right) are visualized in logarithmic scale and are bounded below by 10^{-4}.

Results

Both learned reward models as shown in Fig. 5.24 and Fig. 5.25 seem reasonable. They show the same general structure as the true rewards, and the visualized differences show low values and do not reveal any structure in the residuals.

We were able to solve the swing-up task in the Pendulum environment. Figure 5.26b visualizes the resulting sequence. Note that the two preparatory swings which the agent performs are necessary in order to solve the task, since it first needs to gain sufficient momentum before being able to do the full swing-up movement. It is also not possible to achieve a swing-up with a lower number of preparatory swings. Figure 5.26a shows the latent state representations with both the candidates proposed by the CEM planner, as well as the actual traversed trajectory. The planned trajectory seems to correspond closely to the actual trajectory. Note that it is necessary in the pendulum environment to perform at least two preparatory swings in order to gain momentum before being able to do the full swing-up movement. Table 5.4 shows collected reward values over a sequence of 100 steps. We include a comparison to an agent which has access to the true physical states and who is given the same amount of evidence data. This agent is able to slightly outperform our proposed method.

Method	Reward
GP dynamics on visual observations	-348.91
GP dynamics on physical states	-335.68

Table 5.4: Collected return for the swing-up task in the Pendulum environment, over 100 steps.

We were not able to produce successful results for the CartPole environment. This

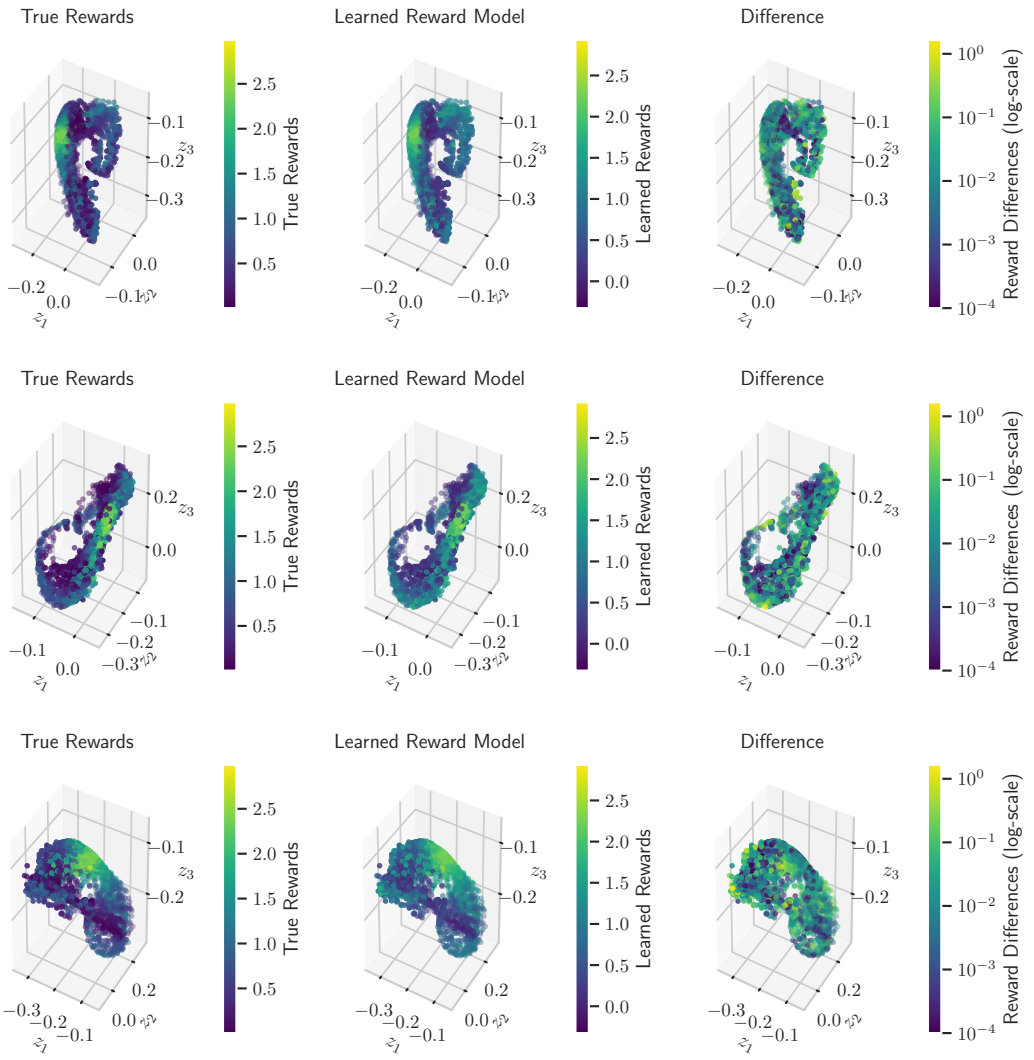


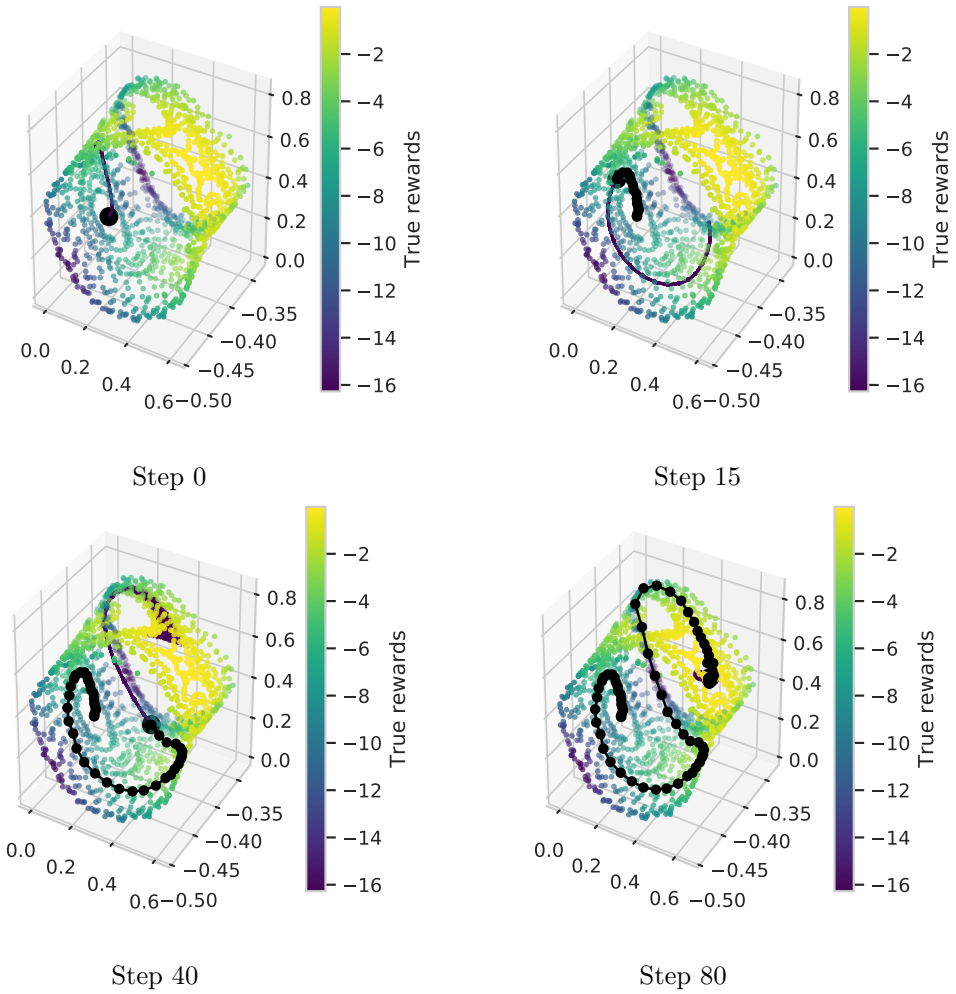
Figure 5.25: Visualization of the learned reward model for the CartPole task. The latent representations are learned as specified in Section 5.7.2, the model corresponds to the selected model for which we reported the results. Each row corresponds to a different projection to a selection of three of the five dimensions. From top to bottom we have $(1, 2, 3)$, $(2, 3, 4)$, and $(3, 4, 5)$. The left column shows the true rewards obtained by interacting with the environment. The middle column shows the reward values predicted by the learned reward model. We present differences between these two values in the right column, visualized in logarithmic scale and bounded below by 10^{-4}.

might be due to a number of factors. The reward model shown in Fig. 5.25 seems to show higher errors in comparison with the Pendulum reward model (see Fig. 5.24), but since it still captures the general structure of the reward distribution we suspect that this is not the driving factor for the failure. On the other hand, the corresponding dynamics model as presented in Section 5.7.2 did not show as good predictions as we were able to obtain on the Pendulum, thus the learned model is likely to be less accurate. Another consideration is in the balancing task itself. In order to solve the task it would already be sufficient to only have accurate dynamics around the starting state, with the cart close to the middle position and an upright pole, together with a sufficiently exact reward model which guides the agent towards the optimal state. This however is a very different requirement to the general goal of learning system dynamics which we considered during the training of the model. The provided dataset might not be fine-grained enough around the optimal state to allow for concise control. This effect might be increased by our choice to apply action repeat. The more diverse data and stronger influence of the chosen action seemed to be beneficial to learn a general dynamics model, but the resulting data might badly reflect the balancing task. A good balancing trajectory would show only very little movement between each time step. It might therefore be beneficial, or even necessary, to train the full model directly in a way to solve the specified task, for example in a similar training scheme as done in PILCO (Deisenroth and Rasmussen, 2011) or PlaNet (Hafner et al., 2019).

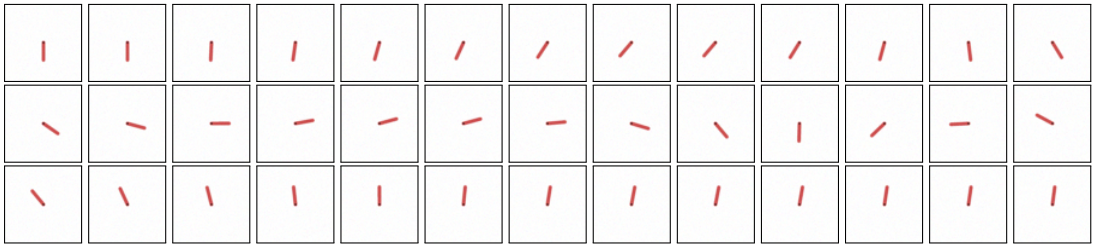
Conclusion

We have shown successful results on the Pendulum environment, confirming that the proposed state representations and dynamics model allow for good reward models and accurate planning in latent space. The agent was able to find a close to optimal trajectory, using the minimal number of preparatory swings the environment allows, before performing the swing-up.

We were not able to extend these results to the CartPole environment. This might be due to an inferior reward model, dynamics model, or encoder, but we suspect that the proposed training approach might also not be beneficial for good balancing performance. A different training scheme which includes the considered task, such as the schemes used in PILCO (Deisenroth and Rasmussen, 2011) or PlaNet (Hafner et al., 2019), might alleviate or solve even this issue. This would be an interesting topic for future work.

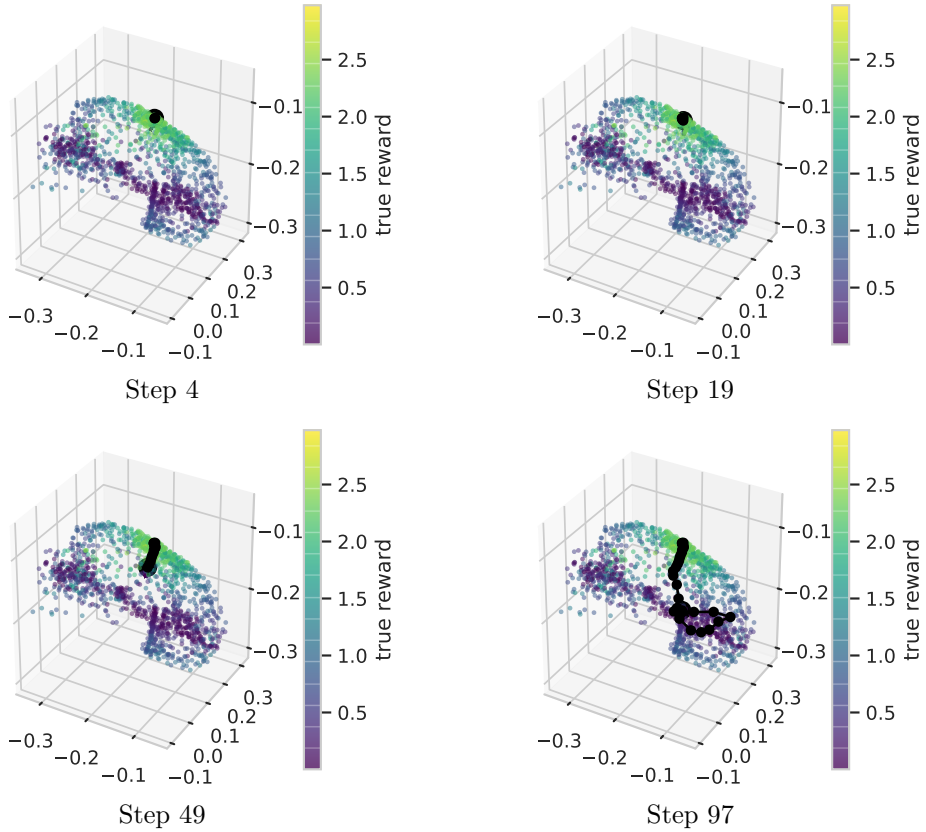


(a) Planning visualization with a planning horizon of 20 steps. For the corresponding pendulum states see Fig. 5.26b.

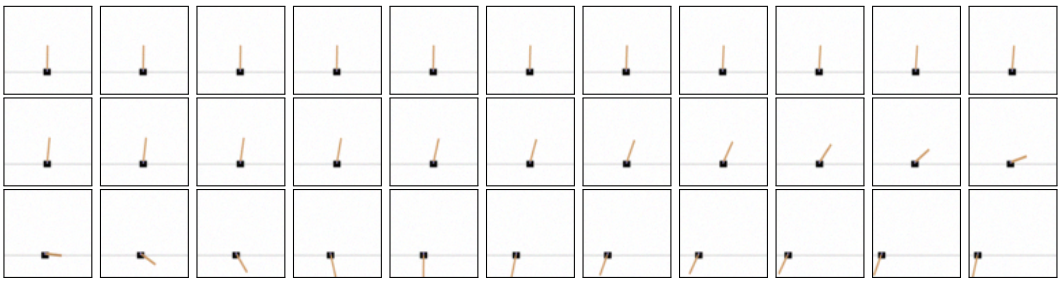


(b) Rollout corresponding to Fig. 5.26a, at time steps $t \in \{0, 2, \ldots, 78\}$.

Figure 5.26



(a) Planning and trajectory in latent space, showing only dimensions $3, 4, 5$. The planned trajectories are barely visible, which is expected for the balancing task since the state is not supposed to change much in order to maximize reward. However, the model fails to balance the pole and the actual trajectory shown as the line in black moves away from the high-reward region. For the corresponding cartpole observations see Fig. 5.27b.



(b) Rollout corresponding to Fig. 5.27a, at time steps $t \in \{1, 4, \ldots, 97\}$.

5.9 Transfer to previously unseen physical properties

Gaussian processes dynamics models have been shown to be very data efficient (Deisenroth and Rasmussen, 2011). We were so far not able to extend this data efficiency to the proposed model, since the neural networks of the encoder and decoder benefit from large training data. However, we found interesting settings in which we can make use of the data-efficiency of Gaussian processes: We can transfer to modified system dynamics while requiring only very little additional training data. In such scenarios the visual properties of the observed environment do not change, thus we can continue to use the encoder for state inference, but the transitions could be very different to those on which the model was originally trained. We found that by using a small number of rollouts collected in the modified environment as new evidence in the GP dynamics model we obtain accurate predictions, allowing for accurate planning in the modified environment.

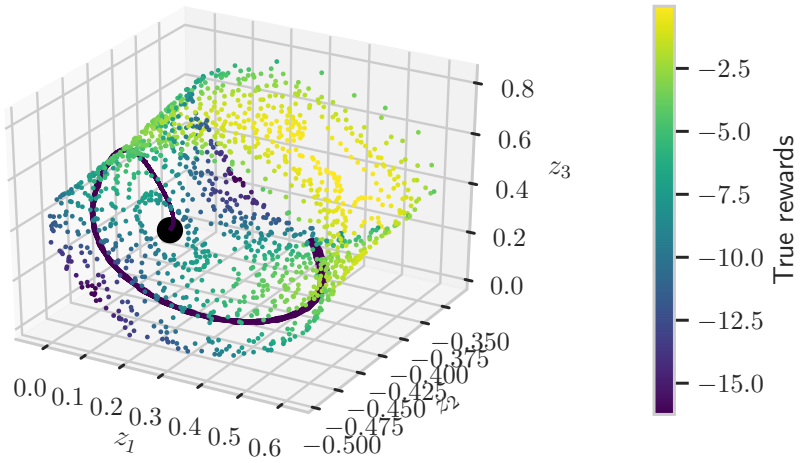
Examples for changes of physical properties could be the addition of a friction coefficient, or modifications of masses. We present examples for a modified Pendulum environment in which we decreased the pole mass from 1 to 0.2. The environment thus effectively becomes easier and it can be solved directly without any preparatory swings to the sides to gather momentum. Figure 5.28 shows the plan and actual trajectory of an agent with an unchanged dynamics model, trained on the original Pendulum environment with default mass. The shown plan accurately corresponds to actual trajectories in the original environment (see Fig. 5.26a), but it deviates strongly from the actual trajectory. The actual trajectory is still very efficient and the agent lifted the Pendulum without any swings, but this is largely due to MPC. By re-planning at every time step it continues to apply a large torque in order to gather more momentum, until it eventually suffices in order to reach the upright position. It can still be seen that the environment is unfamiliar to the agent and the sharp spikes in the trajectory show overly large applied actions.

On the other hand, Fig. 5.28 shows the plan and actual trajectory of an agent with a modified dynamics model. This model has access to 30 rollouts in the new environment, each of length 30, collected by randomly interacting with the environment in the same way as in previous experiments (see also Section 5.2). Encoder, decoder and GP of this model were trained on the original Pendulum environment, corresponding to the results presented in Section 5.7.2, but the evidence in the GP dynamics model now contains the transitions in the new environment. The updated dynamics model seems to accurately predict transitions since the shown plan matches the actual trajectory much more closely. It also does not show the previously observed spikes and the applied torques are smaller, indicating a better understanding of the physics of the environment.

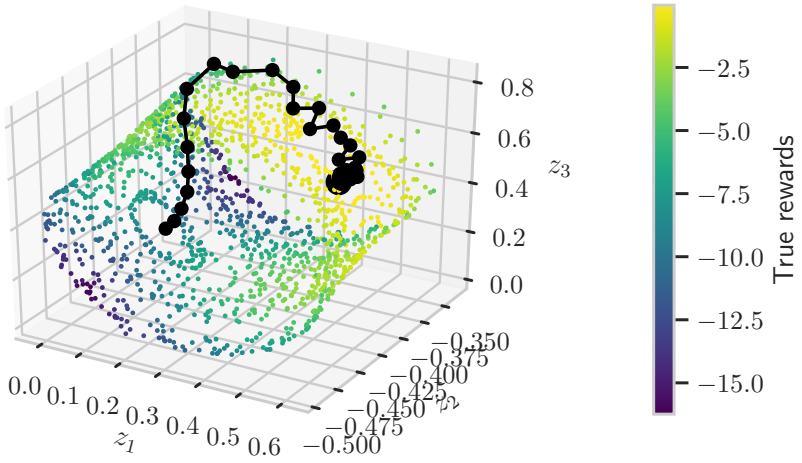
The chosen example allows for successful solutions of the task for both the original and the updated model, but it visualized the differences of both dynamics models well.

Another simple modification would be an inversion of the applied torque, by a simple change in the sign of the action. This completely prevents the original model from solving the task, and results in always applying the *wrong* action, such that the pole points downwards with small oscillations. The modified dynamics model learns the dynamics with inverted action and has no added difficulty for solving the task.

There are many possibilities to come up with other modifications of mass and friction which might prevent agents with the original dynamics model from solving the task while leading to solutions with updated dynamics, but all of these examples come down to exploiting a certain inaccuracy in the planned trajectories. We believe that the presented data-efficiency with regards to dynamics shows an interesting property of the proposed latent GP dynamics model, and it demonstrates an advantage over models consisting purely of neural networks.

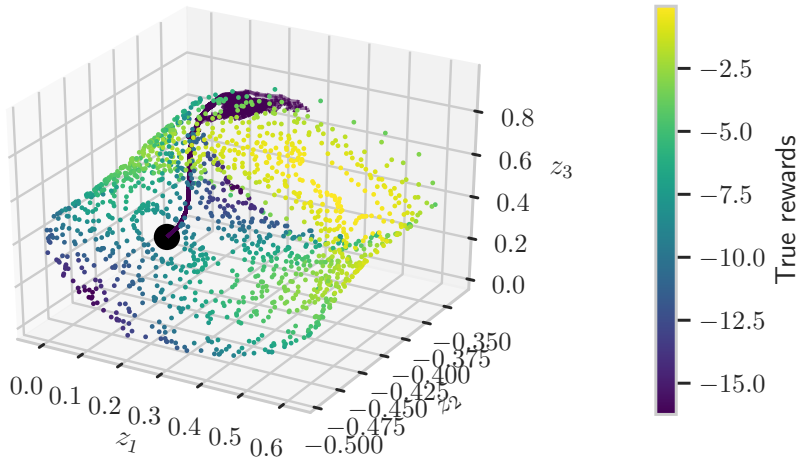


(a) Planned trajectory for a 50 step planning horizon of a model trained on the original Pendulum environment with standard pole mass. The larger planning horizon is motivated purely by visualization purposes in order to show the learned dynamics, and the planned trajectory is in line with the observed trajectories in Section 5.8.

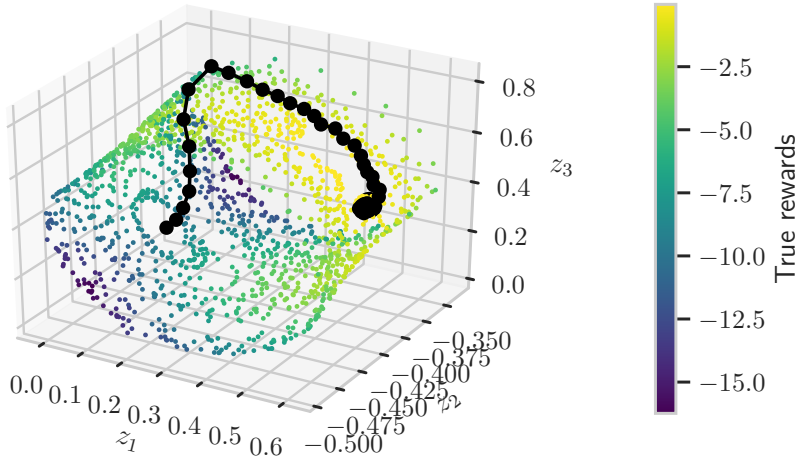


(b) Actual trajectory after 50 steps. The planning and control is done as described in Sections 4.7 and 5.8, but the environment is modified with a decreased pole mass.

Figure 5.28: Visualization of the planning of a model trained on the standard Pendulum environment as specified in Section 5.1. The planned trajectory differs vastly from the actual trajectory. The agent still manages to find an efficient trajectory in order to solve the task, but this is largely due to the MPC algorithm which allows the agent to plan again after each applied action. The sharp corners in the actual trajectory show that the agent applies overly large torques, showing that it overestimates the mass of the pole.



(a) Planned trajectory for a 20 step planning horizon of a model trained on the modified Pendulum environment with decreased pole mass.



(b) Actual trajectory after 50 steps, with planning and control as described in Sections 4.7 and 5.8.

Figure 5.29: Visualization of the planning of a model trained on the modified Pendulum environment with decreased pole width, together with the actual resulting trajectory after 50 steps. The planned trajectory is accurate and clearly shows how the learned dynamics differ from those in the default Pendulum environment, which can be seen in Fig. 5.28a. In comparison, the trajectory does not show sharp corners since the applied actions are much smaller, showing more accurate control over the pendulum and a better estimation of the effect of the chosen actions.

Chapter 6

Conclusion

We learn latent Gaussian process dynamics models from visual observations. The underlying physical states of the observed dynamical system might be concise and low-dimensional, but in our considered problem setting we do not have access to these states and instead only obtain high-dimensional observations, such as images. Our proposed model combines neural networks with Gaussian processes to model system dynamics in a learned low-dimensional latent space. The novelty of the proposed approach does not lie in the individual components: Gaussian process dynamics models are known to allow for very data-efficient learning of dynamical systems and show desirable properties such as accurate uncertainty estimation, and both auto-encoders and variational auto-encoders have been used for state representation learning. However, to the best of our knowledge, Gaussian processes have not previously been combined with neural networks to learn state representations and latent dynamics models.

We derive a lower bound on the likelihood of transitions in image space to train all parts of the proposed model jointly on this objective. We also motivate a subset-of-data approximation for the marginal log-likelihood of the Gaussian process. This allows us to provide more training data to the neural network while keeping the computational cost for the Gaussian process constant. In addition to our proposed model, we present a simplified version in which state representations are learned independently from the system dynamics. We evaluate both approaches on two environments of different complexity, Pendulum and CartPole, and compare our results to a Gaussian process dynamics model on the true physical states.

The separately trained model is able to produce seemingly physically reasonable predictions in both environments. For larger prediction horizons we observe inaccuracies and the model is outperformed by the physical Gaussian process dynamics model. A difference in performance is generally to be expected, since the decoder performs worse than the true rendering engine, but we argue that the difference is also due to the learned dynamics model. However, the Gaussian processes themselves are trained with exactly the same specifications, with the same training scheme and on the same amount of data. The different performance is therefore to be attributed in large parts to the latent state representations, and it seems that the learned representation is inferior to the true physical states for learning GP dynamics models. This provides further

motivation for a joint training of both the dynamics and the state representations.

Our proposed method, trained jointly on the derived training objective, is able to outperform the simplified model on the Pendulum environment. The predicted observations show much smaller errors over larger prediction horizons. 10-step predictions of the joint model are more accurate than 1-step predictions of the separately trained model. The auto-encoder even recovers the structure of the true physical state space, up to scaling and rotation. However, we find the training to be very unstable and we had difficulties to reproduce results during experimentation and development. We also did not manage to outperform the separately trained model on CartPole and only achieved to match the previous performance.

Finally, we use the trained model for planning and control. We train a reward model on the learned state representations and use it together with the transition model to plan in the low-dimensional latent space. This allows for accurate planning in the Pendulum environment and the agent is able to find a close to optimal trajectory for the swing-up task. We were again not able to extend these results to a pole balancing task in the CartPole environment. However, we can leverage the data-efficiency of the Gaussian process dynamics model in order to plan accurately in environments with modified physical properties, such as modified masses or frictions, without additional training and with only little additional experience collection. We compare the model with updated experience to an unchanged model, trained on the original system dynamics, in a modified Pendulum environment with reduced pole mass. While the unchanged model predicts trajectories according to the original physical properties, the planned trajectories of the updated model correspond more closely to the actual sequence, obtained during execution.

The different performance on Pendulum and CartPole, both for learning system dynamics and for control, shows limitations of our proposed method. CartPole is a more complex dynamical system with a higher state dimensionality, and we therefore require significantly more data to obtain similar information about the dynamical system as in the Pendulum environment. To enable larger datasets we propose a subset-of-data approximation, but we believe that this aspect of our method can be further improved. Scaling Gaussian processes to big data is an active field of research and there are a variety of approaches, many of which include sparse approximations of the kernel matrix. We believe that approaches which combine Gaussian process approximations with stochastic variational inference could be especially interesting for future improvements. Such approaches allow for optimization with stochastic gradient descent, which is commonly considered to be beneficial for neural networks.

Instead of increasing the amount of data, we believe that control performance could also be improved with an active learning approach. By following the current best policy for data collection the model receives trajectories which are likely to be of more interest than randomly selected trajectories. For example, the pole-balancing task on CartPole requires very fine-grained control and the optimal trajectory does not show

much movement, but a random policy will lead to very different trajectories with lots of movement and cover states which are far away from the optimal state. Notably, related methods such as PILCO and PlaNet both use training schemes in which data is collected during training according to the current policy. Additionally, training the reward model jointly with the rest of the dynamics model provides an additional learning signal and might lead to better state representations.

Finally, we believe that we could further improve our method by making better use of the uncertainty of the Gaussian process dynamics model. Sensible uncertainty estimates are indeed a large advantage of Gaussian processes over neural networks, and approaches to Gaussian process dynamics models, such as PILCO, often use moment-matching to propagate uncertainties through the recursive multi-step predictions. However, due to computational limitations we were not able to apply moment matching and instead propagated only the predicted mean, leading to overconfident uncertainty estimates of the transitions. One approach to overcome this issue might be to propagate uncertainties through Monte-Carlo sampling. Alternatively, sparse Gaussian process approximations might lead to different computational requirements for moment-matching and could therefore be of additional interest. Together with improved uncertainty estimates, a final suggestion for future research might be the inclusion of multi-step predictions into the training objective. Previous results in literature have shown benefits of such approaches for both neural networks and Gaussian process models.

Bibliography

Assael, J.-A. M., N. Wahlström, T. B. Schön, and M. P. Deisenroth (2015). "Data-Efficient Learning of Feedback Policies From Image Pixels Using Deep Dynamical Models". In: *CoRR*. arXiv: 1510.02173 [cs.AI].

Banijamali, E., R. Shu, M. Ghavamzadeh, H. H. Bui, and A. Ghodsi (2018). "Robust Locally-Linear Controllable Embedding". In: *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, pp. 1751–1759.

Barto, A. G., R. S. Sutton, and C. W. Anderson (Sept. 1983). "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5, pp. 834–846. DOI: 10.1109/TSMC.1983.6313077.

Bengio, S., H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds. (2018). *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer. ISBN: 9780387310732.

Blomqvist, K., S. Kaski, and M. Heinonen (2018). "Deep Convolutional Gaussian Processes". In: *CoRR*. arXiv: 1810.03052 [cs.LG].

Boer, P.-T. de, D. P. Kroese, S. Mannor, and R. Y. Rubinstein (Feb. 2005). "A Tutorial on the Cross-Entropy Method". In: *Annals of Operations Research* 134.1, pp. 19–67. ISSN: 1572-9338. DOI: 10.1007/s10479-005-5724-z.

Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*. ACM Press, pp. 144–152.

Bottou, L. (2010). "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proceedings of COMPSTAT'2010*. Ed. by Y. Lechevallier and G. Saporta. Heidelberg: Physica-Verlag HD, pp. 177–186. ISBN: 978-3-7908-2604-3.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016). *OpenAI Gym*. eprint: arXiv:1606.01540.

Chua, K., R. Calandra, R. McAllister, and S. Levine (2018). "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 4759–4770.

Clevert, D.-A., T. Unterthiner, and S. Hochreiter (2015). "Fast and Accurate Deep Network Learning By Exponential Linear Units (ELUs)". In: *CoRR*. arXiv: 1511.07289 [cs.LG].

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2, pp. 303–314.

Damianou, A. C. and N. D. Lawrence (2012). "Deep Gaussian Processes". In: *CoRR*. arXiv: 1211.0358 [stat.ML].

Deisenroth, M. (2010). "Efficient Reinforcement Learning Using Gaussian Processes". In: Karlsruhe series on intelligent sensor actuator systems. KIT Scientific Publ. ISBN: 9783866445697.

Deisenroth, M. P., D. Fox, and C. E. Rasmussen (Feb. 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: 2160-9292. DOI: 10.1109/tpami.2013.218.

Deisenroth, M. P. and C. E. Rasmussen (2011). "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *in International Conference on Machine Learning*.

Doerr, A., C. Daniel, D. Nguyen-Tuong, A. Marco, S. Schaal, T. Marc, and S. Trimpe (13–15 Nov 2017). "Optimizing Long-term Predictions for Model-based Policy Search". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by S. Levine, V. Vanhoucke, and K. Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, pp. 227–238.

Drucker, H., C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik (1997). "Support Vector Regression Machines". In: *Advances in Neural Information Processing Systems 9*. Ed. by M. C. Mozer, M. I. Jordan, and T. Petsche. MIT Press, pp. 155–161.

Dumoulin, V. and F. Visin (2016). "A Guide To Convolution Arithmetic for Deep Learning". In: *CoRR*. arXiv: 1603.07285 [stat.ML].

Fraccaro, M., S. Kamronn, U. Paquet, and O. Winther (2017). *A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning*. arXiv: 1710.05741v2 [stat.ML].

Gal, Y., R. McAllister, and C. E. Rasmussen (Apr. 2016). "Improving PILCO with Bayesian Neural Network Dynamics Models". In: *Data-Efficient Machine Learning workshop, ICML*.

Garcia, C. E., D. M. Prett, and M. Morari (May 1989). "Model predictive control: Theory and practice—A survey". In: *Automatica* 25.3, pp. 335–348. DOI: 10.1016/0005-1098(89)90002-2.

Gardner, J. R., G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson (2018). "GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 7587–7597.

Ge, R., F. Huang, C. Jin, and Y. Yuan (2015). "Escaping From Saddle Points - Online Stochastic Gradient for Tensor Decomposition". In: *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*. Ed. by P. Grünwald, E. Hazan, and S. Kale. Vol. 40. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 797–842.

Glorot, X., A. Bordes, and Y. Bengio (Nov. 2011). "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson, and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, pp. 315–323.

Gu, S., E. Holly, T. P. Lillicrap, and S. Levine (2017). "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pp. 3389–3396. DOI: 10.1109/ICRA.2017.7989385.

Ha, D. and J. Schmidhuber (2018). "World Models". In: *CoRR*. arXiv: 1803.10122 [cs.LG].

Hafner, D., T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson (2019). "Learning Latent Dynamics for Planning from Pixels". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 2555–2565.

Hanin, B. (2017). "Universal Function Approximation By Deep Neural Nets With Bounded Width and Relu Activations". In: *CoRR*. arXiv: 1708.02691 [stat.ML].

Hastie, T., R. Tibshirani, and J. Friedman (2013). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York. ISBN: 9780387216065.

Hayashi, K., M. Imaizumi, and Y. Yoshida (2019). "on random subsampling of gaussian process regression: a graphon-based analysis". In: *CoRR*. arXiv: 1901.09541 [stat.ML].

He, K., X. Zhang, S. Ren, and J. Sun (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, pp. 1026–1034. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.123.

Hensman, J., N. Fusi, and N. D. Lawrence (2013). "Gaussian Processes for Big Data". In: *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. Ed. by A. Nicholson and P. Smyth. AUAI Press.

Higgins, I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner (2017). "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *ICLR*.

Higuera, J. C. G., D. Meger, and G. Dudek (2018). "Synthesizing Neural Network Controllers With Probabilistic Model Based Reinforcement Learning". In: *CoRR*. arXiv: 1803.02291 [cs.RO].

Hoffman, M., D. M. Blei, C. Wang, and J. Paisley (2012). "Stochastic Variational Inference". In: *CoRR*. arXiv: 1206.7051 [stat.ML].

Hornik, K. (1991). "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2, pp. 251–257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T.

II, A. G., H. He, J. L. Boyd-Graber, J. Morgan, and H. D. III (2014). "Don't Until the Final Verb Wait: Reinforcement Learning for Simultaneous Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1342–1352.

Karl, M., M. Soelch, J. Bayer, and P. v. d. Smagt (2016). "Deep Variational Bayes Filters: Unsupervised Learning of State Space Models From Raw Data". In: *CoRR*. arXiv: 1605.06432 [stat.ML].

Kimeldorf, G. S. and G. Wahba (Apr. 1970). “A Correspondence Between Bayesian Estimation on Stochastic Processes and Smoothing by Splines”. In: *Ann. Math. Statist.* 41.2, pp. 495–502. DOI: 10.1214/aoms/1177697089.

Kingma, D. P. and J. Ba (2014). “Adam: a Method for Stochastic Optimization”. In: *CoRR*. arXiv: 1412.6980 [cs.LG].

Kingma, D. P. and M. Welling (2013). “Auto-Encoding Variational Bayes”. In: *CoRR*. arXiv: 1312.6114 [stat.ML].

Ko, J., D. J. Klein, D. Fox, and D. Hähnel (2007). “Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp”. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 742–747.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 1097–1105.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (Dec. 1989). “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Comput.* 1.4, pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*, pp. 2278–2324.

Lee, J., Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein (2017). “Deep Neural Networks As Gaussian Processes”. In: *CoRR*. arXiv: 1711.00165 [stat.ML].

Lesort, T., N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat (2018). “State Representation Learning for Control: an Overview”. In: *CoRR*. arXiv: 1802.04181 [cs.AI].

Liu, H., Y.-S. Ong, X. Shen, and J. Cai (2018). “When Gaussian Process Meets Big Data: a Review of Scalable Gps”. In: *CoRR*. arXiv: 1807.01065 [stat.ML].

Lu, Z., H. Pu, F. Wang, Z. Hu, and L. Wang (2017). “The Expressive Power of Neural Networks: A View from the Width”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 6231–6239.

Mannor, S., R. Rubinstein, and Y. Gat (2003). “The Cross Entropy Method for Fast Policy Search”. In: *Proceedings of the Twentieth International Conference on*

International Conference on Machine Learning. ICML'03. Washington, DC, USA: AAAI Press, pp. 512–519. ISBN: 1-57735-189-4.

Masters, D. and C. Luschi (2018). "Revisiting Small Batch Training for Deep Neural Networks". In: *CoRR*. arXiv: 1804.07612 [cs.LG].

Mattner, J., S. Lange, and M. Riedmiller (2012). "Learn to Swing Up and Balance a Real Pole Based on Raw Visual Input Data". In: *Neural Information Processing*. Ed. by T. Huang, Z. Zeng, C. Li, and C. S. Leung. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 126–133. ISBN: 978-3-642-34500-5.

Mattos, C. L. C., Z. Dai, A. Damianou, J. Forth, G. A. Barreto, and N. D. Lawrence (2015). "Recurrent Gaussian Processes". In: *CoRR*. arXiv: 1511.06644 [cs.LG].

McAllister, R. and C. E. Rasmussen (2016). "Data-Efficient Reinforcement Learning in Continuous-State Pomdps". In: *CoRR*. arXiv: 1602.02523 [stat.ML].

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *CoRR*. arXiv: 1602.01783 [cs.LG].

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). "Playing Atari With Deep Reinforcement Learning". In: *CoRR*. arXiv: 1312.5602 [cs.LG].

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (Feb. 2015). "Human-level control through deep reinforcement learning". In: *Nature* 518, 529 EP -.

Murphy, K. (2012). *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series. MIT Press. ISBN: 9780262018029.

Neal, R. M. (1996). "Priors for Infinite Networks". In: *Bayesian Learning for Neural Networks*. New York, NY: Springer New York, pp. 29–53. ISBN: 978-1-4612-0745-0. DOI: 10.1007/978-1-4612-0745-0_2.

Novak, R., L. Xiao, J. Lee, Y. Bahri, G. Yang, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein (2018). "Bayesian Deep Convolutional Networks With Many Channels Are Gaussian Processes". In: *CoRR*. arXiv: 1810.05148 [stat.ML].

Oh, J., V. Chockalingam, S. P. Singh, and H. Lee (2016). "Control of Memory, Active Perception, and Action in Minecraft". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 2790–2799.

Paananen, T., J. Piironen, M. R. Andersen, and A. Vehtari (2017). "Variable Selection for Gaussian Processes Via Sensitivity Analysis of the Posterior Predictive Distribution". In: *CoRR*. arXiv: 1712.08048 [stat.ME].

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Quiñonero-Candela, J., A. Girard, J. Larsen, and C. Rasmussen (2003). "Propagation of Uncertainty in Bayesian Kernel Models - Application to Multiple-Step Ahead Forecasting". In: *IEEE International Conference on Acoustics, Speech and Signal Processing* 2, pp. 701–704.

Ramachandran, P., B. Zoph, and Q. V. Le (2017). "Searching for Activation Functions". In: *CoRR*. arXiv: 1710.05941 [cs.NE].

Rasmussen, C. E. and C. K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press. ISBN: 026218253X.

Rasmussen, C. and M. Kuss (June 2004). "Gaussian Processes in Reinforcement Learning". In: *Advances in Neural Information Processing Systems 16*. Max-Planck-Gesellschaft. Cambridge, MA, USA: MIT Press, pp. 751–759.

Rezende, D. J., S. Mohamed, and D. Wierstra (2014). "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *CoRR*. arXiv: 1401.4082 [stat.ML].

Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review*, pp. 65–386.

Rubinstein, R. Y. (1996). "Optimization of Computer Simulation Models with Rare Events". In: *European Journal of Operations Research* 99, pp. 89–112.

Seeger, M. (2002). *Relationships between Gaussian processes, Support Vector machines and Smoothing Splines*.

Snelson, E. and Z. Ghahramani (21–24 Mar 2007). "Local and global sparse Gaussian process approximations". In: *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*. Ed. by M. Meila and X. Shen. Vol. 2. Proceedings of Machine Learning Research. San Juan, Puerto Rico: PMLR, pp. 524–531.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press.

Szita, I. and A. Lörincz (Dec. 2006). "Learning Tetris Using the Noisy Cross-Entropy Method". In: *Neural Computation* 18.12, pp. 2936–2941. DOI: 10.1162/neco.2006.18.12.2936.

Tassa, Y., Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller (Jan. 2018). *DeepMind Control Suite*. Tech. rep. DeepMind.

Titsias, M. (16–18 Apr 2009). "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. by D. van Dyk and M. Welling. Vol. 5. Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, pp. 567–574.

Wahlström, N., T. B. Schön, and M. P. Deisenroth (2014). "Learning deep dynamical models from image pixels". In: *CoRR*. arXiv: 1410.7550 [stat.ML].

Watter, M., J. T. Springenberg, J. Boedecker, and M. Riedmiller (2015). *Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images*. arXiv: 1506.07365v3 [cs.LG].

Williams, C. K. I. (1997). "Computing with Infinite Networks". In: *Advances in Neural Information Processing Systems 9*. Ed. by M. C. Mozer, M. I. Jordan, and T. Petsche. MIT Press, pp. 295–301.

Williams, C. and M. Seeger (2001). "Using the Nyström Method to Speed Up Kernel Machines". In: *Advances in Neural Information Processing Systems 13*. MIT Press, pp. 682–688.

Wilson, A. G., C. Dann, and H. Nickisch (2015a). "Thoughts on Massively Scalable Gaussian Processes". In: *CoRR*. arXiv: 1511.01870 [cs.LG].

Wilson, A. G., Z. Hu, R. Salakhutdinov, and E. P. Xing (2015b). *Deep Kernel Learning*. arXiv: 1511.02222v1 [cs.LG].

Wilson, A. G., Z. Hu, R. Salakhutdinov, and E. P. Xing (2016). "Stochastic Variational Deep Kernel Learning". In: *CoRR*. arXiv: 1611.00336 [stat.ML].

Wilson, A. G. and H. Nickisch (2015). "Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP)". In: *CoRR*. arXiv: 1503.01057v1 [cs.LG].

Xu, B., N. Wang, T. Chen, and M. Li (2015). "Empirical Evaluation of Rectified Activations in Convolutional Network". In: *CoRR*. arXiv: 1505.00853 [cs.LG].

Zhang, M., S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine (2018). "Solar: Deep Structured Latent Representations for Model-Based Reinforcement Learning". In: *CoRR*. arXiv: 1808.09105 [cs.LG].