UNIVERSITY OF CALIFORNIA SAN DIEGO

Uncertainty Estimation in Continuous Models applied to Reinforcement Learning

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Electrical Engineering (Intelligent Systems, Robotics, and Control)

by

Ibrahim Karim Akbar

Committee in charge:

Professor Nikolay Atanasov, Chair
Professor Sicun Gao
Professor Kenneth Kreutz-Delgado

2019

I dedicate this work to my dearest friends, family, and mother Isa Hinrichs. All your support is what has brought me here.

Ferda.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Professor Nikolay Atanasov for all the thoughtful suggestions and patience in letting me explore various research topics. I appreciated all the thorough demonstration & considerations during our meetings. Your support and guidance helped me develop better critical thinking skills and awareness. I especially appreciate the Existential Robotics Lab for being so constructive and supportive. Whether in furthering creative discussions or conceptual understanding the lab made full effort in making sure that no stone was left unturned.

ABSTRACT OF THE THESIS

Uncertainty Estimation in Continuous Models applied to Reinforcement Learning

by

Ibrahim Karim Akbar

Master of Science in Electrical Engineering (Intelligent Systems, Robotics, & Control)

University of California San Diego, 2019

Professor Nikolay Atanasov, Chair

We consider the model-based reinforcement learning framework where we are interested in learning a model and control policy for a given objective. We consider modeling the dynamics of an environment using Gaussian Processes or a Bayesian neural network. For Bayesian neural networks we must define how to estimate uncertainty through a neural network and propagate distributions in time. Once we have a continuous model we can apply standard optimal control techniques to learn a policy. We consider the policy to be a radial basis policy and compare it's performance given the different models on a pendulum environment.

Chapter 1

Model-Based Reinforcement Learning

1.1 Introduction

Intelligent systems such as robots are projected to be able to accomplish many tasks that would be inefficient in both resources or time, and potentially labor intensive or dangerous for humans. They further provide novel solutions to search and rescue, waste or recycle management, mapping & localization, transportation, and medical delivery problems. For these systems to operate intelligently they must be able learn from the data generated which brings us to the field of machine learning.

Machine learning (ML) is a branch of artificial intelligence that leverages statistics in order to extract information and infer relevant structures from the empirical data. These structures can be approximated through statistical models that attempt to represent the given data compactly. Reinforcement learning (RL) as a branch of machine learning restricts how this data is gathered and further uses it in decision making.

Reinforcement learning can be considered a combination of machine learning and control theory that considers learning through interactions. In this setting data is generated by an agent's engagement with an unknown environment. Furthermore the agent is concerned with using this information in sequential decision making to achieve some higher-order goals.

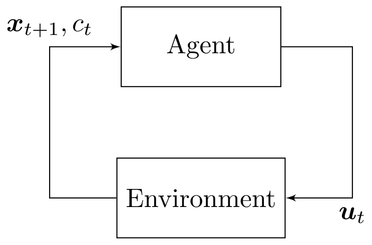

Figure 1.1: General Reinforcement Learning Framework: The agent takes an control $\boldsymbol{u}_t$ and receives a state $\boldsymbol{x}_t$ & cost c_t from the environment as a response.

An agent will engage with the environment accumulating rewards or costs and observations for the respective actions taken. This accumulation of information is the agent's experience and what dictates the future decisions the agent will make. Given this experience the objective of the agent is then to produce a sequence of decisions (policy) that minimizes the expected cost of a goal.

Similarly, in control theory we are concerned with producing the optimal policy for a given goal; however, we know the environment's dynamics and the reward function specifying our goal. Since these are known, interactions with the environment as in Fig. 1.1 are not necessary and the problem reduces to a simulated optimization problem. The learning aspect thus plays the role of trying to represent the environment's dynamics and/or the reward function through statistical modeling based on the agent's experience.

Since the agent can only generate policies based on it's current experience a dilemma occurs between exploration and expoitation. The agent must trade-off exploring the environment through making suboptimal decisions and exploiting the current learned representation. Since the experience of the agent is only partially reflective of the environment's dynamics exploration may lead the agent to develop a better policy. The corollary then being that the agent may explore undesirable regions of the environment. Hence the decision of when to explore and when to not explore becomes non-trivial. Since the assumptions being made on the environment and agent are very general the agent often requires a large amount of experience to produce a good policy. These assumptions also make the framework intuitive for autonomous learning and decision making under uncertainty.

Given that an agent must interact with the environment data efficiency becomes a central concern. Data efficiency can be quantified as the amount of necessary interactions to determine a good policy. As environments may be large and complex a means of extracting useful information from the agent's experience would help increase data efficiency. A statistical model tries to compactly represent the given data and can be used to generalize and infer the cost of a given action. Iterative improvements upon this model as the agent's experience grows would then increase the data efficiency. Thus a model-based approach to reinforcement learning address this issue well. However, in addition to addressing data efficiency the use of a model presents the problem of model bias.

Generally, dynamical systems are functions that map a state-action pair to another state. If we try to model the environment through choosing the most likely function that describes our experience we introduce the notion of model bias. Given a state-action input the model's bias is the offset of the model's expected state from the true state. This is further observed if we consider that the agent's experience is relatively small with respect to the environment. As the dataset is small there exist many plausible dynamics that would satisfy what the agent has experienced. Using only the most likely model makes the assumption that these dynamics describe the whole environment exactly. The chance that this is true is quite low, especially since we have a limited amount of data. An agent considering functions that may be insufficient representations of the environment's dynamics in certain relevant regions will have detrimental effects on learning a good policy. To avoid this pitfall we can use a probabilistic model that takes into account all the plausible dynamics given by an experience.

Gaussian Processes (GPs) and Bayesian Neural Networks (BNNs) are non-parametric and parametric probabilistic models, respectively, that can account for the uncertainty in the model effectively reducing the model bias. Given such models the agent's reasoning about a goal can incorporate model uncertainty increasing data efficiency. Along with addressing data efficiency and model bias concerns a model should scale both in terms of amount of experience and the complexity of the environment. Although GPs provide analytical results the complexity in time and space hinder them from being applied to more interesting environments.

This thesis considers learning a BNN and presents it's application as a dynamics model in a Bayesian RL framework with a continuous domain. The framework assumes a fully observable environment and episodic tasks. Given a BNN we observe that the uncertainty of the model is limited causing the agent to be overconfident in unexplored regions of the state space. This presents the consideration of learning a more expressive BNN that takes into accound more uncertainty. The remainder of the contents of this thesis are summarized as follows:

Chapter 1: 1.2: We present the notation used throughout the thesis and formulate the model-based reinforcement learning problem.

Chapter 2: 2.1: We introduce the necessary background for GPs and consider the case of prediction given an input with uncertainty.

Chapter 2: 2.2: We introduce an extension of the GP to the multi-dimensional regression case and present a preliminary extension of propagating distributions through them.

Chapter 2: 2.4: We introduce the background for BNNs and consider different algorithms for estimating the posterior distribution of these models.

Chapter 3: We discuss the general approach to optimizing a parameterized policy and present the variations to this approach when considering the dynamic model to be a GP or BNN.

Chapter 4: 4.2: We evaluate the models independently given datasets and discuss differences in terms of training time, accuracy, uncertainty estimates.

Chapter 4: 4.3: Now we consider the algorithms as a whole and evaluate the performance given different models of a pendulum environment.

1.2 Problem Formulation

1.2.1 Symbols & Notation

Matrices are capitalized and vectors are in bold font.

Symbols	Meaning
$\triangleq$	an equality acting as a definition
$\propto$	propotional to; e.g. $y \propto x$ means that $y = Cx$ where $C \in \mathbb{R}$ and independent of x
$\sim$	distributed according to; e.g. $x \sim \mathcal{N}(\mu, \sigma^2)$
$\#\|\cdot\|$	cardinality of; e.g. $\#\|\mathbb{R}\| = \infty$; the cardinality of the real number line is infinite
∇_ϕ	gradient with respect to ϕ of; e.g. $\nabla_\phi f$ is the gradient of the scalar function f with respect to ϕ
$\boldsymbol{x}^T$	transpose of the vector $\boldsymbol{x}$
$\mathbf{0}_n$	vector of all 0's with length n
$\mathbf{1}_n$	vector of all 1's with length n
δ_{xy}	1 if $x = y$ and 0 otherwise
$\text{diag}(\boldsymbol{x})$	(vector argument) a diagonal matrix containing the elements $\boldsymbol{x}$
$\text{diag}(\boldsymbol{X})$	(matrix argument) a vector containing the diagonal elements of $\boldsymbol{X}$
$\mathcal{D}$ or $\mathcal{D}^*$	train (test) data set: $\mathcal{D} \triangleq \{(\boldsymbol{X}, \boldsymbol{Y})\} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i : \boldsymbol{x}_i \in \mathbb{R}^n, \boldsymbol{y}_i \in \mathbb{R}^m, i = 1, \ldots, k, k \in \mathbb{N}\}$
$\mathbb{E}_{x\sim p(x)}[z(x)]$	expectation of $z(x)$ when $x \sim p(x)$
$\mathbb{V}_{x\sim p(x)}[z(x)]$	variance of $z(x)$ when $x \sim p(x)$
$\text{Cov}[x, x']$	covariance between x and x'
$\mathcal{M}$	model structure either being a Gaussian Process or Bayesian Neural Network
$\boldsymbol{\omega}$	the parameters of a model $\mathcal{M}$
$\boldsymbol{\theta}$	the hyper-parameters of a model $\mathcal{M}$
$f(\boldsymbol{x}_t, \boldsymbol{u}_t)$	the dynamics of a continuous system
$g(\boldsymbol{x}_t, \boldsymbol{u}_t)$	the approximate gradient of the $f(\boldsymbol{x}_t, \boldsymbol{u}_t)$
$f_\mathcal{M}(\boldsymbol{x})$ or $\boldsymbol{f}_\mathcal{M}$	model (or vector) of latent function values, $\boldsymbol{f}_\mathcal{M} = [f_\mathcal{M}(\boldsymbol{x}_1), \ldots, f_\mathcal{M}(\boldsymbol{x}_k)]^T$
$\boldsymbol{f}^*_\mathcal{M}$	Model prediction (random variable), $\boldsymbol{f}^*_\mathcal{M} = [f_\mathcal{M}(\boldsymbol{x}^*_1), \ldots, f_\mathcal{M}(\boldsymbol{x}^*_k)]$
$\mathcal{GP}$	Gaussian Process: $f \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$, the scalar function f is distributed as a Gaussian Process with mean function $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$
$m(\boldsymbol{x})$	mean function of a Gaussian Process
$k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}')$	covariance (kernel) function of a Gaussian Process parameterized by $\boldsymbol{\theta}$ evaluated at $\boldsymbol{x}$ and $\boldsymbol{x}'$
$\boldsymbol{K}_{\boldsymbol{\theta}}$ or $K_{\boldsymbol{\theta}}(\boldsymbol{X}, \boldsymbol{X})$	$k \times k$ Gram (covariance) matrix parameterized by $\boldsymbol{\theta}$
$\boldsymbol{K}^*_{\boldsymbol{\theta}}$	$k \times k^*$ $K_{\boldsymbol{\theta}}(\boldsymbol{X}, \boldsymbol{X}^*)$ matrix being the covariance between the training and test inputs
$\boldsymbol{k}_{\boldsymbol{\theta}}(\boldsymbol{x}*)$ or $\boldsymbol{k}^*_{\boldsymbol{\theta}}$	vector, short for $K_{\boldsymbol{\theta}}(\boldsymbol{X}, \boldsymbol{x}^*)$ where there is only a single test input
$h(\boldsymbol{x})$	dimensionality perserving nonlinearity; e.g. Sigmoid, Rectifying Linear Unit (ReLU)
$\boldsymbol{I}$ or $\boldsymbol{I}_n$	Identity matrix (of size n)
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ or $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$	(random variable $\boldsymbol{x}$) is Gaussian with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
$\boldsymbol{\phi}$	Normal distribution i.e. $\boldsymbol{\phi} = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
$\boldsymbol{\Phi}(x \mid \mu, \sigma^2)$	Cumulative density function of univariate gaussian distribution with mean μ and variance σ^2 evaluated till x i.e. $\mathbb{P}(X \leq x)$ where $X \sim \mathcal{N}(\mu, \sigma^2)$
$\boldsymbol{\Phi}(x)$	Cumulative density function of univariate normal distribution evaluated till x, i.e. $X \sim \boldsymbol{\phi}$
$\boldsymbol{y} \mid \boldsymbol{x}$ and $p(\boldsymbol{y} \mid \boldsymbol{x})$	conditional random variable $\boldsymbol{y}$ given $\boldsymbol{x}$ and its density
$\pi_{\boldsymbol{\psi}}(\cdot)$	deterministic control policy parameterized by $\boldsymbol{\psi}$

$\mathcal{O}(\cdot)$	big-Oh notation; for functions f and g on $\mathbb{N}$, we write $f(n) = \mathcal{O}(g(n))$ if $f(n)/g(n) < \infty$ as $n \to \infty$
$\mathcal{L}$	loss function
k and k^*	number of training and test inputs
$\log(z)$	natural logarithm (base e) of z
L	number of layers in a neural network
V_l	number of hidden units at layer l of a neural network

1.2.2 Model-based Reinforcement Learning

Model-based reinforcement learning revolves around formulating a Markov Decision Process (MDP) over which we can run a policy optimization procedure. An MDP can be formulated as a tuple $(\mathcal{X}, \mathcal{U}, p_f, p_0, c, \gamma, T)$, where $\mathcal{X}$ is the state space, $\mathcal{U}$ is the control space, $p_f(\cdot \mid \boldsymbol{x}, \boldsymbol{u})$ is the true unknown dynamic model for $\boldsymbol{x} \in \mathcal{X}, \boldsymbol{u} \in \mathcal{U}$, p_0 is the prior distribution over the state space, $c(\boldsymbol{x}, \boldsymbol{u}) : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ is the cost function, $\gamma \in [0, 1)$ is the discounting term, and T is the planning horizon. Specifically, we consider a continuous state-action finite horizon MDP where $\mathcal{X} \subseteq \mathbb{R}^n$, $\mathcal{U} \subseteq \mathbb{R}^m$, and $T < \infty$. Given a system's unknown deterministic dynamics $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t)$ from which we can sample, we are interested in the two components that comprise model-based reinforcement learning: model learning and control policy optimization. First we want to find a probabilistic model, $\mathcal{M}$, that is representative of the system's dynamics based on the experience the agent has observed. Onced we have learned this model we are interested in finding a good control policy π that achieves the desired goal. Thus the main objective in model learning is finding a representation of the dynamics that allows for good policy optimization. We define a policy as a function from the state space to the control space, $\pi : \mathcal{X} \mapsto \mathcal{U}$ where $\pi \in \Pi$ is a collection of policies. Given a policy we want to determine how well it perform in regards to our goal; which is embedded in the cost function. Thus we define the expected long-term culumative cost given a policy,

$$J^\pi(\mathbf{x}_0) = \mathbb{E}_{\mathbf{x}_t \sim p_{\hat{f}}}\left[\sum_{t=0}^{T} \gamma^t c_t(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t)) \middle| \boldsymbol{x}_0 \sim p_0\right] \tag{1.1}$$

where $p_{\hat{f}}(\cdot \mid \boldsymbol{x}, \boldsymbol{u})$ is generated from the model, $\mathcal{M}$. With this objective we consider the policy optimization problem,

$$\begin{gathered} \min_{\pi \in \Pi} \quad J^\pi(\boldsymbol{x}) \quad \forall \;\; \boldsymbol{x} \in \mathcal{X} \;\; s.t. \\ \boldsymbol{x}_0 \sim p_0, \quad \boldsymbol{x}_{t+1} \sim p_{\hat{f}}(\cdot \mid \boldsymbol{x}_t, \boldsymbol{u}_t) \\ \boldsymbol{x}_t \in \mathcal{X}, \quad \boldsymbol{u}_t = \pi(\boldsymbol{x}_t) \in \mathcal{U}(\boldsymbol{x}_t) \end{gathered} \tag{1.2}$$

where $\mathcal{U}(\boldsymbol{x}) \subseteq \mathcal{U}$ is the set of admissible controls in state $\boldsymbol{x}$. We can iteratively consider model learning with experienced observed when interacting with the environment and policy optimization with regards to the information provided by the model. In particular we can consider using a parameterized policy π_{ψ} such that we can perform a gradient based policy search. The appeal of considering GPs and BNNs as models is that we can exploit their derivatives through time when computing J^{π}. This allows the policy optimization to that take into consideration the uncertainty in the model through the gradients. Before discussing how to exploit these gradients we have to consider the models. In the next chapter we will go introduce two forms of models; the Gaussian Process and the Bayesian Neural Network. For each one we will discuss how to train them given a dataset $\mathcal{D}$ and subquentially how to do inference with an uncertain input.

Chapter 2

Model Learning & Inference

We are interested in learning a deterministic dynamic function $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Instead of directly learning the dynamic we consider learning the approximate state difference of the dynamics.

$$g(\boldsymbol{x}_t, \boldsymbol{u}_t) \triangleq f(\boldsymbol{x}_t, \boldsymbol{u}_t) - \boldsymbol{x}_t = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t \tag{2.1}$$

Thus the training dataset $\mathcal{D}$ contains inputs $\tilde{\boldsymbol{x}}_t \triangleq [\boldsymbol{x}^T, \boldsymbol{u}^T]^T \in \mathbb{R}^{n+m}$ and noisy targets $\Delta\boldsymbol{x}_t \in \mathbb{R}^n$ where $\Delta\boldsymbol{x}_t = g(\boldsymbol{x}_t, \boldsymbol{u}_t) + \epsilon$ and $\epsilon \sim \mathcal{N}(\boldsymbol{0}_n, \sigma_\epsilon^2 \boldsymbol{I}_n)$. Given a training and test dataset $\mathcal{D}, \mathcal{D}^*$ we can consider how to learn a model through a Gaussian process framework and through a Bayesian neural network framework. Once we have trained a model on $\mathcal{D}$ we want to evaluate the current policy π on the test set $\mathcal{D}^*$ which requires being able to propagate distributions through the models. Chapter two is organized such that we first introduce the Gaussian process framework and how to estimate propagating distributions through them. We extend this theoretical work to consider multi-output Gaussian processes. This is followed by a literature review regard this non-parametric framework. Next, we introduce the Bayesian neural network framework and one algorithm for approximating the posterior of the parameters. We then discuss propagating distributions through this model as well. We conclude the chapter with a literature review regarding the BNN framework.

2.1 Gaussian Process

A Gaussian process (GP) is a stochastic process where a finite subset of random variables is jointly Gaussian. Consider the real-valued function $y(\boldsymbol{v})$ on $\mathbb{R}^v$ to be governed by a Gaussian Process; it is fully described by it's prior mean and kernel function,

$$
\begin{aligned}
y_{gp}(\boldsymbol{v}) &\sim GP(m(\boldsymbol{v}), k(\boldsymbol{v}, \boldsymbol{v}')) \\
m(\boldsymbol{v}) &= \mathbb{E}[y(\boldsymbol{v})] \\
k(\boldsymbol{v}, \boldsymbol{v}') &= \mathbb{E}[((\boldsymbol{v}) - m(\boldsymbol{v}))(y(\boldsymbol{v}') - m(\boldsymbol{v}'))]
\end{aligned}
\tag{2.2}
$$

We will omit the gp subscript throughout this section for readibility. Since the model is described functionally we consider it to be a non-parametric model i.e. $\#|\boldsymbol{\omega}| = \infty$; it has infinitely many direct parameters. A dilemma that arises in this framework when considering dynamics is that dynamics are vector-valued functions for most interesting cases while a GP is formulated for real-valued functions. We can assume the next state values are all conditionally independent of each other given the current state. This allows us to decompose the dynamics f into a set of dynamics, $\{f_i(\tilde{\boldsymbol{x}}) : i = 1, \ldots, n\}$ and have each governed by a GP. Alternatively, instead of assuming independence we can formulate a Convolution Processes (CP) that can govern vector-valued functions. See Section 2.2 for an introduction. For demonstrations throughout this section we consider the GP governing the i^{th} dimension dynamics to have a zero mean function, $m(\tilde{\boldsymbol{x}}) = 0 \ \ \forall \tilde{\boldsymbol{x}}$ and the squared exponential (SE) kernel. This kernel measure the covariance between two states $\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{x}}'$ as

$$
k_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}') = \alpha_i^2 \exp{(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}')^T \boldsymbol{\Lambda}_i^{-1} (\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}')} \tag{2.3}
$$

where $\boldsymbol{\theta}_i = \{\alpha_i, \boldsymbol{\Lambda}_i,\}$ are the hyper-parameters of the model where $\boldsymbol{\Lambda}_i \in \mathbb{R}^{n+m \times n+m}$.

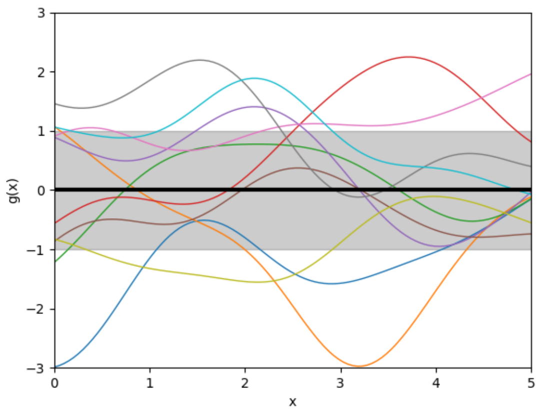

Figure 2.1: GP Prior with zero mean (black line) and SE Kernel. The gray region corresponds to the 95% confidence interval while the colored lines are samples drawn from the prior.

The hyper-parameters, $\boldsymbol{\theta}_i$ can be inferred through Bayesian inference as shown in Subsection 2.1.1. For any finite subset of states $\tilde{\boldsymbol{X}} \in \mathbb{R}^{n+m \times k}$ we can compute the covariance matrix, $K_{\boldsymbol{\theta}}(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}}) \in \mathbb{R}^{k \times k}$, through elementwise application of the kernel function. Thus we can specify a Gaussian distribution from which we can draw samples corresponding to the output of the function.

$$\boldsymbol{g}_i \sim \mathcal{N}\big(\mathbf{0}, K_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}})\big) \tag{2.4}$$

This is of particularly little interest as we don't want to draw samples from a prior distribution but rather a posterior conditioned on observed data. In Fig. 2.1 we see that the prior is flat causing the samples to be uniformly distributed. Thus making estimates about the quality of a policy is uniformative. Through introducing $\mathcal{D}$ we can reduce the uncertainty in the model and generate a posterior. Specificially, if we assume we are given a dataset $\mathcal{D}$ and known test inputs $\tilde{\boldsymbol{X}}^*$ then we know a joint distribution must satisfy,

$$\begin{bmatrix} \Delta \boldsymbol{x} \\ \boldsymbol{g}_i^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}}) + \sigma_\epsilon \boldsymbol{I} & K_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{X}}^*) \\ K_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{X}}^*, \tilde{\boldsymbol{X}}) & K_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{X}}^*, \tilde{\boldsymbol{X}}^*) \end{bmatrix}\right) \tag{2.5}$$

Given that the joint distribution and marginal distributions are Gaussian we can express the conditional predictive distribution as a Gaussian as well allowing for analytic inference.

$$\boldsymbol{g}_i^* \mid \mathcal{D}, \tilde{\boldsymbol{X}}^*, \boldsymbol{\theta}_i \sim \mathcal{N}\big(\boldsymbol{K}_{\boldsymbol{\theta}_i}^{*T}(\boldsymbol{K}_{\boldsymbol{\theta}_i} + \sigma_\epsilon \boldsymbol{I})^{-1}\Delta \boldsymbol{x}, K_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{X}}^*, \tilde{\boldsymbol{X}}^*) - \boldsymbol{K}_{\boldsymbol{\theta}_i}^{*T}(\boldsymbol{K}_{\boldsymbol{\theta}} + \sigma_\epsilon \boldsymbol{I})^{-1}\boldsymbol{K}_{\boldsymbol{\theta}_i}^*\big) \tag{2.6}$$

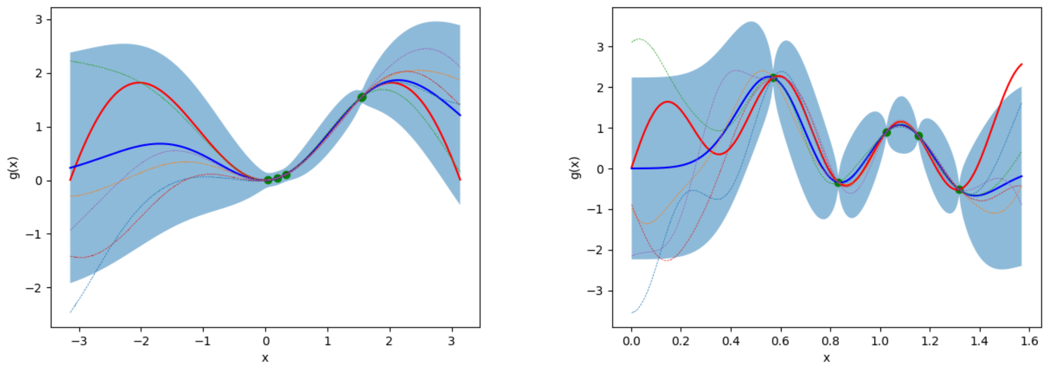

Figure 2.2: GP Posterior on $x\sin(x)$ and $x + \sin(4x) + \sin(13x)$ where the true functions are given in red while the model predicts a posterior mean in blue given the training inputs points in green. The shaded region shows the 95% confidence interval of the model while the transparent lines are samples drawn from this posterior GP.

Before we can compute the posterior we have to concern ourselves with $\boldsymbol{\theta}_i$. Given the dataset and an arbitrary initial $\boldsymbol{\theta}_i$ the posterior is generally not be the most probable. Thus we can consider learning the hyper-parameters $\boldsymbol{\theta}_i$ with regards to $\mathcal{D}$.

2.1.1 Training

Generally, machine learning allows for varying approaches for determining the optimal values of the parameters $\boldsymbol{\omega}$ and hyper-parameters $\boldsymbol{\theta}$. The Bayesian inference framework neatly encompasses these methods in a hierarchical framework. Bayesian inference is interested in learning the posterior over the respective parameters in order to take into account all possible values rather than the most likely. This directly address concerns of overfitting and bias by considering distributions rather than point-wise estimates. We can formulate the posterior over $\boldsymbol{\omega}$ and $\boldsymbol{\theta}$ through Baye's rule.

$$p(\boldsymbol{\omega} \mid \mathcal{D}, \boldsymbol{\theta}) = \frac{p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\omega}, \boldsymbol{\theta}) p(\boldsymbol{\omega} \mid \boldsymbol{\theta})}{p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta})} \tag{2.7}$$

The first term here is the likelihood of the data followed by the prior distribution over $\boldsymbol{\omega}$. In the denumerator we have the marginal likelihood (evidence) that is independent of the weights $\boldsymbol{\omega}$.

$$p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta}) = \int p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\omega}, \boldsymbol{\theta}) p(\boldsymbol{\omega} \mid \boldsymbol{\theta}) d\boldsymbol{\omega} \tag{2.8}$$

At the next level we can consider the posterior of the hyper-parameters $\boldsymbol{\theta}$.

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}})} \tag{2.9}$$

$$p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}) = \int p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{2.10}$$

Depending on the model structure the integrals in Eq. 2.8, 2.10 may be intractable leading to approximations through Markov Chain Monte Carlo (MCMC) methods. Rather than these approximations we can assume an uninformative prior $p(\boldsymbol{\theta})$ so that $p(\boldsymbol{\theta} \mid \mathcal{D}) \propto p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta})$. We can then move away from estimating the posterior and instead maximize the marginal likelihood with respect to $\boldsymbol{\theta}$. This is known as type-II maximum likelihood estimation (MLE) and is analytical given the formulation of a GP. We consider the optimization problem,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta}) \tag{2.11}$$

From the formulation of a Gaussian Process and Eq. 2.5 we know the marginal log-likelihood takes the form of,

$$\log p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta}) = -\frac{1}{2} \Delta \boldsymbol{x}^T (K_{\boldsymbol{\theta}} + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \Delta \boldsymbol{x} - \frac{1}{2} \log \left(|2\pi|^{n+m} \det \left(K_{\boldsymbol{\theta}} + \sigma_\epsilon^2 \boldsymbol{I} \right) \right) \tag{2.12}$$

Here the terms can be regarded as the data fitting and complexity regularization plus normalization term, respectively. We can consider finding the gradient in order to iteratively improve our estimate of the hyper-parameters. Thus we look at the partial derivatives of the marginal log-likelihood with respect to θ_j which corresponds to one of the parameters in the set.

$$\frac{\partial}{\partial \theta_j} \log p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta}) = \frac{1}{2}\operatorname{tr}\left(\left(\boldsymbol{\alpha}\boldsymbol{\alpha}^T - \left(K_{\boldsymbol{\theta}} + \sigma_\epsilon^2 \boldsymbol{I}\right)^{-1}\right)\frac{\partial\left(K_{\boldsymbol{\theta}} + \sigma_\epsilon^2 \boldsymbol{I}\right)}{\partial \theta_j}\right) \quad \boldsymbol{\alpha} = \left(K_{\boldsymbol{\theta}} + \sigma_\epsilon^2 \boldsymbol{I}\right)^{-1} \Delta \boldsymbol{x} \tag{2.13}$$

We write $\frac{\partial}{\partial \theta_j}\left(\boldsymbol{K}_{\boldsymbol{\theta}} + \sigma_\epsilon^2 \boldsymbol{I}\right)$ rather than $\frac{\partial}{\partial \theta_j}\boldsymbol{K}_{\boldsymbol{\theta}}$ as we can consider the noise σ_ϵ^2 as a hyper-parameter. Using standard matrix inversion techniques for positive definite matrices requires $\mathcal{O}(n^3)$ time complexity and $\mathcal{O}(n^2)$ memory complexity. Due to this time complexity standard GPs are impractical for datasets where more than a thousand samples are collected. In addition it should be recalled we are finding the mode of the distribution on $\boldsymbol{\theta}$ which makes the optimization more susceptible to overfitting.

2.1.2 Inference

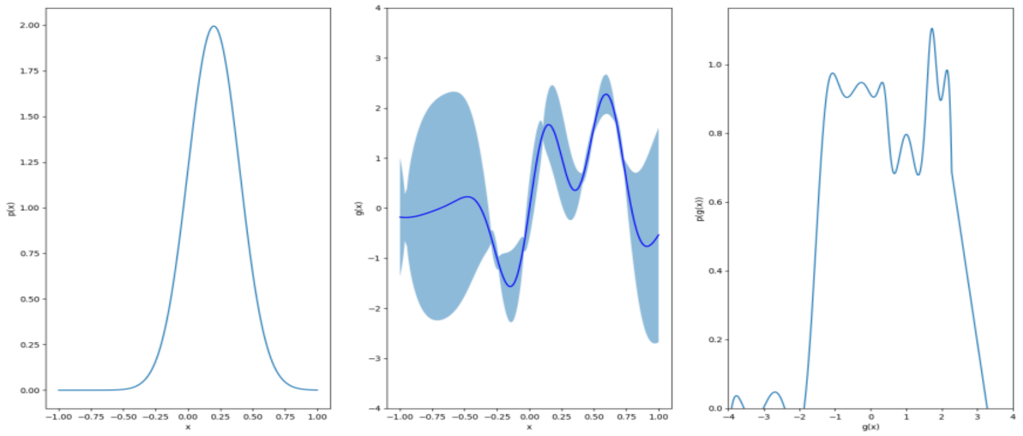

Figure 2.3: Propagation of a Gaussian distribution through a GP generally results in a non-Gaussian multimodal distribution. From left to right we have the state distribution $x \sim \mathcal{N}(\mu, \sigma^2)$, a Posterior GP on $g(x) = x + \sin(4x) + \sin(13x)$, and the approximate resulting distribution on $g(x)$ when propagating $p(x)$.

Now that we have learned a GP model from a dataset, we can use it to make more informed decisions. Given a deterministic known test point $\tilde{\boldsymbol{x}}^*$ we have already shown in Eq. 2.6 the resulting unitary predictive distribution. Since we are concerned with reinforcement learning the state may not be observable but rather a distribution can be specified. Therefore given a distribution as an input we want to infer the resulting distribution through the GP. Specifically, if $\tilde{\boldsymbol{x}}^* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ what can we say about the distribution over $\boldsymbol{g}^* \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathcal{D}$? These derivations have be shown by Diesenroth [1] following the work of

Kuss [2] and Quiñonero-Candela et al. [3], [4] but are re-iterated here for completeness. The exact predictive distribution through the i^{th} GP,

$$p(g_i^* \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathcal{D}) = \int p(g_i(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*, \mathcal{D}) p(\tilde{\boldsymbol{x}}^* \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\tilde{\boldsymbol{x}}^* \tag{2.14}$$

is generally not Gaussian or unimodal. See Fig. 2.3 for a visualization. If a Gaussian kernel is used for the GP we are able to compute the mean and variance of the distribution analytically. Therefore, we choose to approximate this distribution with the best-fit Gaussian that minimizes the Kullback-Lieber (KL) Divergence. This is equivalent to moment matching between the unknown true distribution and the mean and variance of the Gaussian. First we present the univariate case, i.e. given the i^{th} dimension of the dynamics, and then proceed to the multi-dimensional case.

Univariate Prediction

We want to determine the moments of $p(g_i(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ given that $\tilde{\boldsymbol{x}}^* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. We define,

$$\begin{aligned} m_{g_i}(\tilde{\boldsymbol{x}}^*) &\triangleq \boldsymbol{k}_{\boldsymbol{\theta}}^{*T}(\boldsymbol{K}_{\boldsymbol{\theta}_i} + \sigma_{\epsilon_i}\boldsymbol{I})^{-1}\Delta\boldsymbol{x} \\ \sigma_{g_i}(\tilde{\boldsymbol{x}}^*) &\triangleq k_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}^*) - \boldsymbol{k}_{\boldsymbol{\theta}_i}^{*T}(\boldsymbol{K}_{\boldsymbol{\theta}_i} + \sigma_{\epsilon_i}\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{\theta}_i}^*) \end{aligned} \tag{2.15}$$

to be the posterior mean and variance of the i^{th} GP given $\mathcal{D}$. We can compute the moments of the predictive distribution using Fubini's Theorem.

$$\mu_i^* \triangleq \int\int g_i(\tilde{\boldsymbol{x}}^*) p(g_i^*, \tilde{\boldsymbol{x}}^*) dg_i^* d\tilde{\boldsymbol{x}}^* = \mathbb{E}_{g_i, \tilde{\boldsymbol{x}}^*}[g_i(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\mathbb{E}_{g_i}[g_i(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*] \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}]$$

$$= \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[m_{g_i}(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int m_{g_i}(\tilde{\boldsymbol{x}}^*) \mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\tilde{\boldsymbol{x}}^* = \boldsymbol{\beta}_i^T \boldsymbol{q}_i \tag{2.16}$$

where

$$\boldsymbol{\beta}_i = (K_{\boldsymbol{\theta}_i} + \sigma_{\epsilon_i}\boldsymbol{I})^{-1}\Delta\boldsymbol{x}_i \quad \boldsymbol{q}_i = [q_{i1}, \ldots, q_{ik}]^T \in \mathbb{R}^k \tag{2.17}$$

$$\begin{aligned} q_{ij} &\triangleq \int k_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}^*) \mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\tilde{\boldsymbol{x}}^* \\ &= \alpha_i^2 \det(\boldsymbol{\Sigma}\boldsymbol{\Lambda}_i^{-1} + \boldsymbol{I})^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(\tilde{\boldsymbol{x}}_j - \boldsymbol{\mu})^T(\boldsymbol{\Sigma} + \boldsymbol{\Lambda}_i)^{-1}(\tilde{\boldsymbol{x}}_j - \boldsymbol{\mu})\Big) \end{aligned} \tag{2.18}$$

If we set $\boldsymbol{\Sigma} \equiv \boldsymbol{0}$ we get $q_{ij} = k_{\boldsymbol{\theta}_i}(\boldsymbol{x}_j, \boldsymbol{x}^*)$ which recovers the deterministic prediction through a GP.

$$\sigma_i^{2*} = \mathrm{Var}_{g_i, \tilde{\boldsymbol{x}}^*}[g_i(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\mathrm{Var}_{g_i}[g_i(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*] \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] + \mathrm{Var}_{\tilde{\boldsymbol{x}}^*}[\mathbb{E}_{g_i}[g_i(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*] \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] \tag{2.19}$$

$$= \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\sigma_{g_i}(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] + (\mathbb{E}_{\tilde{\boldsymbol{x}}^*}[m_{g_i}(\tilde{\boldsymbol{x}}^*)^2 \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[m_{g_i}(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}]^2) \tag{2.20}$$

$$= \int (\sigma_{g_i}(\tilde{\boldsymbol{x}}^*) + m_{g_i}(\tilde{\boldsymbol{x}}^*)^2) p(\tilde{\boldsymbol{x}}^*) d\tilde{\boldsymbol{x}}^* - \left(\int m_{g_i}(\tilde{\boldsymbol{x}}^*) p(\tilde{\boldsymbol{x}}^*) d\tilde{\boldsymbol{x}}^* \right)^2 \tag{2.21}$$

$$\sigma_i^{2*} = \alpha_i^2 - \mathrm{tr}\big((K_{\boldsymbol{\theta}_i} + \sigma_{\epsilon_i}^2 \boldsymbol{I})^{-1} \hat{\boldsymbol{Q}}_i\big) + \boldsymbol{\beta}_i^T \hat{\boldsymbol{Q}}_i \boldsymbol{\beta}_i - (\mu^*)^2 \tag{2.22}$$

where the entries of $\hat{\boldsymbol{Q}}_i \in \mathbb{R}^{k \times k}$ are given as,

$$(\hat{\boldsymbol{Q}}_i)_{jk} = \frac{k_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{x}}_j, \mu) k_{\boldsymbol{\theta}_i}(\tilde{\boldsymbol{x}}_k, \boldsymbol{\mu})}{\det\left(2\boldsymbol{\Sigma}\boldsymbol{\Lambda}_i^{-1} + \boldsymbol{I}\right)^{-\frac{1}{2}}} \exp\left((\hat{\boldsymbol{z}}_{jk} - \boldsymbol{\mu})^T (\boldsymbol{\Sigma} + \frac{1}{2}\boldsymbol{\Lambda}_i)^{-1} \boldsymbol{\Sigma} \boldsymbol{\Lambda}_i^{-1} (\hat{\boldsymbol{z}}_{jk} - \boldsymbol{u}) \right) \tag{2.23}$$

Thus for given Gaussian $p(\boldsymbol{x}^*)$ we approximate $p(g_i(\boldsymbol{x}^*)) \sim \mathcal{N}(\mu^*, \sigma^{2*})$.

Multivariate Prediction

Now we consider the full dynamics represented through n independent GPs. This means we would consider our hyper-parameters to be $\boldsymbol{\theta} = \{\alpha_i, \Lambda_i, \sigma_{\epsilon_i}\}_{i=1}^n$. using this we can again estimate the predictive multivariate moments. Due to independence we can apply the univariate predictive mean in Eq. 2.16 dimension-wise.

$$\boldsymbol{\mu}^* = [\boldsymbol{\beta}_1^T \boldsymbol{q}_1, \ldots, \boldsymbol{\beta}_n^T \boldsymbol{q}_n]^T \tag{2.24}$$

The key difference arises with determining the covariance matrix. The output dimensions can covary with each other and should be taken into account. Thus when looking at the structure of the predictive covariance matrix we have two options: variance and covariance along dimensions of the output.

$$\boldsymbol{\Sigma}^* = \begin{bmatrix} \mathbb{V}[g_1(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] & \ldots & \mathrm{Cov}[g_1(\tilde{\boldsymbol{x}}^*), g_n(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}[g_n(\tilde{\boldsymbol{x}}^*), g_1(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})] & \ldots & \mathbb{V}[g_n(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] \end{bmatrix} \tag{2.25}$$

The variance along the diagonal is derived in Eq. 2.22. For the cross-covariance terms we start with,

$$\mathrm{Cov}[g_a(\tilde{\boldsymbol{x}}^*), g_b(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{g, \tilde{\boldsymbol{x}}^*}[g_a(\tilde{\boldsymbol{x}}^*) g_b(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] - \mu_a^* \mu_b^* \tag{2.26}$$

$$\mathbb{E}_{g,\tilde{\boldsymbol{x}}^*}[g_a(\tilde{\boldsymbol{x}}^*)g_b(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\mathbb{E}_{g_a}[g_a(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*]\mathbb{E}_{g_b}[g_b(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*] \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int m_{g_a}(\tilde{\boldsymbol{x}}^*)m_{g_b}(\tilde{\boldsymbol{x}}^*)p(\tilde{\boldsymbol{x}}^*)d\tilde{\boldsymbol{x}}^* \tag{2.27}$$

Through substituting in the definitions from Eq. 2.15 we can determine that,

$$\mathbb{E}_{g,\tilde{\boldsymbol{x}}^*}[g_a(\tilde{\boldsymbol{x}}^*)g_b(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \boldsymbol{\beta}_a^T \underbrace{\int k_{\boldsymbol{\theta}_a}(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{X}})^T k_{\boldsymbol{\theta}_b}(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{X}})p(\tilde{\boldsymbol{x}}^*)d\tilde{\boldsymbol{x}}^*}_{\boldsymbol{Q}} \boldsymbol{\beta}_b \tag{2.28}$$

We can write $\boldsymbol{Q} \in \mathbb{R}^{k \times k}$ in closed form through it's elements which have an exponential form,

$$Q_{ij} = \frac{k_{\boldsymbol{\theta}_a}(\tilde{\boldsymbol{x}}_i, \boldsymbol{\mu})k_{\boldsymbol{\theta}_b}(\tilde{\boldsymbol{x}}_j, \boldsymbol{\mu})}{\sqrt{\det(\boldsymbol{R})}} \exp(\frac{1}{2}\boldsymbol{z}_{ij}^T \boldsymbol{R}^{-1}\boldsymbol{\Sigma}\boldsymbol{z}_{ij}) = \frac{\exp(\eta_{ij}^2)}{\sqrt{\det(\boldsymbol{R})}} \tag{2.29}$$

$$\eta_{ij}^2 = 2(\log(\alpha_a) + \log(\alpha_b)) - \frac{\boldsymbol{\eta}_i^T \boldsymbol{\Lambda}_a^{-1}\boldsymbol{\eta}_i + \boldsymbol{\eta}_j^T \boldsymbol{\Lambda}^{-1}\boldsymbol{\eta}_j - \boldsymbol{z}_{ij}^T \boldsymbol{R}^{-1}\boldsymbol{\Sigma}\boldsymbol{z}_{ij}}{2} \tag{2.30}$$

where $\boldsymbol{R} = \boldsymbol{\Sigma}(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1}) + \boldsymbol{I}$, $\boldsymbol{z}_{ij} = \boldsymbol{\Lambda}_a^{-1}\boldsymbol{\eta}_i + \boldsymbol{\Lambda}_b^{-1}\boldsymbol{\eta}_j$, and $\boldsymbol{\eta}_i = \tilde{\boldsymbol{x}}_i - \boldsymbol{\mu}$.

Thus the predictive covariance is fully described as,

$$\text{Cov}[g_a(\tilde{\boldsymbol{x}}^*), g_b(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \begin{cases} \boldsymbol{\beta}_a^T \boldsymbol{Q}\boldsymbol{\beta}_b - \mu_a^* \mu_b^* & a \neq b \\ \boldsymbol{\beta}_a^T \boldsymbol{Q}\boldsymbol{\beta}_a - (\mu_a^*)^2 + \alpha_a^2 - \text{tr}((\boldsymbol{K}_{\boldsymbol{\theta}_a} + \sigma_{\epsilon_a}\boldsymbol{I})^{-1} & a = b \end{cases} \tag{2.31}$$

When making predictions and propagating uncertainty it is also important to consider how the input varies with respect to the output of the model. Given that $\tilde{\boldsymbol{x}}^* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $g(\tilde{\boldsymbol{x}}^*) \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$ we assume that the joint is also Gaussian.

$$\begin{bmatrix} g(\tilde{\boldsymbol{x}}^*) \\ \tilde{\boldsymbol{x}}^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}^* \\ \boldsymbol{\mu} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}^* & \boldsymbol{\Sigma}_{g,\tilde{\boldsymbol{x}}^*} \\ \boldsymbol{\Sigma}_{\tilde{\boldsymbol{x}}^*,g} & \boldsymbol{\Sigma} \end{bmatrix} \right)$$

In order to fully decribe the distribution we have to derive input-output covariance $\boldsymbol{\Sigma}_{g,\tilde{\boldsymbol{x}}^*}$.

$$\boldsymbol{\Sigma}_{g,\tilde{\boldsymbol{x}}^*} = \mathbb{E}_{g,\tilde{\boldsymbol{x}}^*}[g(\tilde{\boldsymbol{x}}^*)\tilde{\boldsymbol{x}}^{*T}] - \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\tilde{\boldsymbol{x}}^*]\mathbb{E}_{g,\tilde{\boldsymbol{x}}^*}[g(\tilde{\boldsymbol{x}}^*)]^T = \mathbb{E}_{g,\tilde{\boldsymbol{x}}^*}[g(\tilde{\boldsymbol{x}}^*)\tilde{\boldsymbol{x}}^{*T}] - \boldsymbol{\mu}\boldsymbol{\mu}^{*T}$$

We can derive close form expressions for $\mathbb{E}_{g,\tilde{\boldsymbol{x}}^*}[g(\tilde{\boldsymbol{x}}^*)\tilde{\boldsymbol{x}}^{*T}]$ dimension-wise for $a = 1, \ldots, n$.

$$\mathbb{E}_{g,\tilde{\boldsymbol{x}}^*}[g_a(\tilde{\boldsymbol{x}}^*)\tilde{\boldsymbol{x}}^*] = \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\tilde{\boldsymbol{x}}^*\mathbb{E}_g[g_a(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*] \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \int \tilde{\boldsymbol{x}}^* m_{g_a}(\tilde{\boldsymbol{x}}^*)p(\tilde{\boldsymbol{x}}^*)d\tilde{\boldsymbol{x}}^*$$

Through some manipulations of the exponential forms and the variables we arrive at the form,

$$\mathrm{Cov}_{g,\tilde{\boldsymbol{x}}^*}[\tilde{\boldsymbol{x}}^*, g_a(\tilde{\boldsymbol{x}}^*) \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}] = \sum_{i=1}^{n} \beta_{ai} q_{ai} \boldsymbol{\Sigma}(\boldsymbol{\Sigma} + \boldsymbol{\Lambda}_a)^{-1}(\tilde{\boldsymbol{x}}_i - \boldsymbol{\mu}) \tag{2.32}$$

2.2 Convolution Process

A GP presents a flexible and fast model for learning single output regression well. This limitation to real-valued functions poses a problem when considering dynamics which are generally vector-valued functions. An alternative to considering the outputs of g to be conditionally independent given $\tilde{\boldsymbol{x}}$ is to consider the dynamics to have a convolutional functional form. Through this formulation we can consider a Convolution Process (CP) that governs a vector-valued function. This is particularly appealing for systems where various output states are simple transformation of another. Let the dynamics be decomposed along the output dimensions $g(\cdot) = \{g_i(\cdot)\}_{i=1}^{n}$. Let g_i have the form,

$$g_i(\tilde{\boldsymbol{x}}) = \sum_{r=1}^{R} \int k_{ir}(\tilde{\boldsymbol{x}} - \boldsymbol{z}) u_r(\boldsymbol{z}) d\boldsymbol{z} \tag{2.33}$$

where $k_{ir}(\cdot)$ is a smoothing kernels and $\{u_r(\cdot)\}_{r=1}^{R}$ are the latent functions. We can thuse recover the covariance between the two functions g_i and g_j as,

$$\mathrm{Cov}[g_i(\tilde{\boldsymbol{x}}), g_j(\tilde{\boldsymbol{x}}')] = \sum_{s=1}^{R}\sum_{r=1}^{R} \int k_{is}(\tilde{\boldsymbol{x}} - \boldsymbol{z}) \int k_{jr}(\tilde{\boldsymbol{x}}' - \boldsymbol{z}') \mathrm{Cov}[u_s(\boldsymbol{z}), u_r(\boldsymbol{z}')] d\boldsymbol{z}' d\boldsymbol{z} \tag{2.34}$$

This describes the covariance in the most general sense without any assumptions on $u_r(\cdot)$. If we consider these latent functions to be independent GPs, i.e. $\mathrm{Cov}[u_s(\boldsymbol{z}), u_r(\boldsymbol{z}')] = k_{u_s,u_r}(\boldsymbol{z}, \boldsymbol{z}')\delta_{sr}$ then the covariance simplifies to

$$\mathrm{Cov}[g_i(\tilde{\boldsymbol{x}}), g_j(\tilde{\boldsymbol{x}}')] = \sum_{r=1}^{R} \int k_{ir}(\tilde{\boldsymbol{x}} - \boldsymbol{z}) \int k_{jr}(\tilde{\boldsymbol{x}}' - \boldsymbol{z}') k_{u_r}(\boldsymbol{z}, \boldsymbol{z}') d\boldsymbol{z}' d\boldsymbol{z} \tag{2.35}$$

Additionally the covariance between latent function and output can be computed as,

$$\mathrm{Cov}[g_i(\tilde{\boldsymbol{x}}), u_r(\boldsymbol{z})] = \int k_{ir}(\tilde{\boldsymbol{x}} - \boldsymbol{z}') k_{u_r}(\boldsymbol{z}, \boldsymbol{z}') d\boldsymbol{z}' \tag{2.36}$$

Given this formulation and the dataset $\mathcal{D}$ we can construct a GP over the set of outputs with prior,

$$\Delta \boldsymbol{X} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_{\boldsymbol{\theta}} + \boldsymbol{\Sigma}_{\epsilon}) \tag{2.37}$$

where $\Delta \boldsymbol{X} = [\Delta \boldsymbol{x}_1^T, \ldots, \Delta \boldsymbol{x}_n^T]^T \in \mathbb{R}^{nk}$ is the set of output functions with $\Delta \boldsymbol{x}_i = [g_i(\tilde{\boldsymbol{x}}_1), \ldots, g_i(\tilde{\boldsymbol{x}}_k)]^T$, $\boldsymbol{K_\theta} \in \mathbb{R}^{nk \times nk}$ is the covariance matrix given training inputs $\tilde{\boldsymbol{X}}$ and hyper-parameters $\boldsymbol{\theta}$ with elements $\mathrm{Cov}[g_i(\tilde{\boldsymbol{x}}), g_j(\tilde{\boldsymbol{x}}')]$, and $\boldsymbol{\Sigma}_\epsilon = \sigma_\epsilon^2 \boldsymbol{I} \otimes \boldsymbol{I}_k$ is the noise. Here $\otimes$ denotes the Kronecker product.

Equivalently, the predictive distribution given the test set $\mathcal{D}^*$ is,

$$\Delta \boldsymbol{X}^* \mid \mathcal{D}, \tilde{\boldsymbol{X}}^*, \boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{K}_{\boldsymbol{\theta}}^{*T}(\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\Delta \boldsymbol{X}, K_{\boldsymbol{\theta}}(\tilde{\boldsymbol{X}}^*, \tilde{\boldsymbol{X}}^*) - \boldsymbol{K}_{\boldsymbol{\theta}}^{*T}(\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\boldsymbol{K}_{\boldsymbol{\theta}}^{*} + \boldsymbol{\Sigma}_\epsilon) \tag{2.38}$$

Due to the functional form of the covariance certain choices of smoothing kernels or latent kernels will cause it to be intractable. For the case where $k_{u_r}(\cdot, \cdot)$ is a squared exponential kernel and $k_{ir}(\cdot)$ is a Gaussian kernel we can produce a closed form expression for the covariance,

$$\mathrm{Cov}[g_i(\tilde{\boldsymbol{x}}), g_j(\tilde{\boldsymbol{x}}')] = \sum_{r=1}^{R} \frac{\alpha_r^2 \sqrt{\det(\boldsymbol{\Lambda}_{u_r})}}{\sqrt{\det(\boldsymbol{\Lambda}_{u_r} + \boldsymbol{\Lambda}_{ir} + \boldsymbol{\Lambda}_{jr})}} \exp\big(-\frac{1}{2}(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}')^T(\boldsymbol{\Lambda}_{u_r} + \boldsymbol{\Lambda}_{ir} + \boldsymbol{\Lambda}_{jr})^{-1}(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}})\big) \tag{2.39}$$

See Appendix C for a derivation of the kernel. Thus we can consider the set of hyper-parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_r\}_{r=1}^R$ where $\boldsymbol{\theta}_r = \{\alpha_r, \boldsymbol{\Lambda}_{u_r}, \{\boldsymbol{\Lambda}_{ir}\}_{i=1}^n\}$. Here we keep the hyper-parameter set for this kernel as general as possible through allowing each latent function to have a set of smoothing kernels for each dimension. Considering this formulation we now extend Diesenroth's results for prediction through a GP to a CP.

2.2.1 Multivariate Prediction

Given that $\tilde{\boldsymbol{x}}^* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we would like to approximate the predictive state distribution $p(\Delta \boldsymbol{x})$ with a Gaussian distribution. We can follow the same procedure in deriving the GP predictions and define,

$$\begin{aligned} \boldsymbol{m}_g(\tilde{\boldsymbol{x}}^*) &= \boldsymbol{K}_{\boldsymbol{\theta}}^{*T}(\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\Delta \boldsymbol{X} \\ \boldsymbol{V}_g(\tilde{\boldsymbol{x}}^*) &= K_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}^*) - \boldsymbol{K}_{\boldsymbol{\theta}}^{*T}(\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\boldsymbol{K}_{\boldsymbol{\theta}}^{*} \end{aligned} \tag{2.40}$$

$$\boldsymbol{\mu}^* = \mathbb{E}_{g,\tilde{\boldsymbol{x}}^*}[g(\tilde{\boldsymbol{x}}^*)] = \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\mathbb{E}_g[g(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}]] = \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{m}_g(\tilde{\boldsymbol{x}}^*)] = \int \boldsymbol{m}_g(\tilde{\boldsymbol{x}}^*)\mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})d\tilde{\boldsymbol{x}}^* \tag{2.41}$$

$$\begin{aligned} \boldsymbol{\mu}^* &= \underbrace{\int \boldsymbol{K}_{\boldsymbol{\theta}}^{*T}\mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})d\tilde{\boldsymbol{x}}^*}_{\boldsymbol{P}}(\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\Delta \boldsymbol{X} \\ &= \boldsymbol{P}\boldsymbol{\beta} \end{aligned} \tag{2.42}$$

where with a slight abuse of notation $\boldsymbol{\beta} = (\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\Delta\boldsymbol{X} \in \mathbb{R}^{nk}$ rather than $\mathbb{R}^k$ as with the GP. Here $\boldsymbol{P} \in \mathbb{R}^{n\times nk}$ and can be viewed as,

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}_{11} & \cdots & \boldsymbol{p}_{1n} \\ \vdots & \ddots & \vdots \\ \boldsymbol{p}_{n1} & \cdots & \boldsymbol{p}_{nn} \end{bmatrix} \tag{2.43}$$

such that $\boldsymbol{p}_{ij} \in \mathbb{R}^{1\times k}$ specifies the predictive covariance between output dimensions i and j for $\tilde{\boldsymbol{x}}^*$ and $\tilde{\boldsymbol{X}}$. For the l^{th} sample $\tilde{\boldsymbol{x}}_l$ the predictive covariance is given as,

$$\boldsymbol{p}_{ij}^{(l)} = \sum_{r=1}^{R} \frac{\alpha_r^2 \sqrt{\det(\boldsymbol{\Lambda}_{u_r})}}{\sqrt{\det(\boldsymbol{\Lambda}_{u_r} + \boldsymbol{\Lambda}_{ir} + \boldsymbol{\Lambda}_{jr} + \boldsymbol{\Sigma})}} \exp\big(-\frac{1}{2}(\tilde{\boldsymbol{x}}_l - \boldsymbol{\mu})^T(\boldsymbol{\Lambda}_{u_r} + \boldsymbol{\Lambda}_{ir} + \boldsymbol{\Lambda}_{jr} + \boldsymbol{\Sigma})^{-1}(\tilde{\boldsymbol{x}}_l - \boldsymbol{\mu})\big) \tag{2.44}$$

For the covariance, $\boldsymbol{\Sigma}^*$ we consider using the law of total covariance,

$$\begin{aligned} \mathrm{Cov}_{g,\tilde{\boldsymbol{x}}^*}[g(\tilde{\boldsymbol{x}}^*), g(\tilde{\boldsymbol{x}}^*)] &= \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\mathrm{Cov}_g[g(\tilde{\boldsymbol{x}}^*), g(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*]] + \mathrm{Cov}_{\tilde{\boldsymbol{x}}^*}[\mathbb{E}_g[g(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*], \mathbb{E}_g[g(\tilde{\boldsymbol{x}}^*) \mid \tilde{\boldsymbol{x}}^*]] \\ &= \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{V}_g(\tilde{\boldsymbol{x}}^*)] + \mathrm{Cov}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{m}_g(\tilde{\boldsymbol{x}}^*), \boldsymbol{m}_g(\tilde{\boldsymbol{x}}^*)] \\ &= \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{V}_g(\tilde{\boldsymbol{x}}^*) + \boldsymbol{m}_g(\tilde{\boldsymbol{x}}^*)\boldsymbol{m}_g(\tilde{\boldsymbol{x}}^*)^T] - \boldsymbol{\mu}^*\boldsymbol{\mu}^{*T} \end{aligned} \tag{2.45}$$

Expanding these with regards to Eqs. 2.40 we arrive at,

$$\begin{aligned} \mathrm{Cov}_{g,\tilde{\boldsymbol{x}}^*}[g(\tilde{\boldsymbol{x}}^*), g(\tilde{\boldsymbol{x}}^*)] &= \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[k_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}^*, \tilde{\boldsymbol{x}}^*)] - \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{k}_{\boldsymbol{\theta}}^{*T}(\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\boldsymbol{k}_{\boldsymbol{\theta}}^*] + \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{k}_{\boldsymbol{\theta}}^{*T}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{k}_{\boldsymbol{\theta}}^*] - \boldsymbol{\mu}^*\boldsymbol{\mu}^{*T} \\ &= \boldsymbol{\Upsilon} - \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{k}_{\boldsymbol{\theta}}^{*T}(\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\boldsymbol{k}_{\boldsymbol{\theta}}^*] + \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{k}_{\boldsymbol{\theta}}^{*T}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{k}_{\boldsymbol{\theta}}^*] - \boldsymbol{\mu}^*\boldsymbol{\mu}^{*T} \end{aligned} \tag{2.46}$$

where $\boldsymbol{\Upsilon} \in \mathbb{R}^{n\times n}$ and it's elements are given by,

$$\Upsilon_{ij} = \sum_{r=1}^{R} \frac{\alpha_r^2 \sqrt{\det(\boldsymbol{\Lambda}_{u_r})}}{\sqrt{\det(\boldsymbol{\Lambda}_{u_r} + \boldsymbol{\Lambda}_{ir} + \boldsymbol{\Lambda}_{jr})}} \tag{2.47}$$

We now consider looking at the i^{th} and j^{th} dimension of the output to continue the derivation.

$$\begin{aligned} \mathrm{Cov}_{g,\tilde{\boldsymbol{x}}^*}[g_i(\tilde{\boldsymbol{x}}^*), g_j(\tilde{\boldsymbol{x}}^*)] &= \Upsilon_{ij} - \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{k}_{\boldsymbol{\theta},i}^{*T}(\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\boldsymbol{k}_{\boldsymbol{\theta},j}^*] + \mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{k}_{\boldsymbol{\theta},i}^{*T}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{k}_{\boldsymbol{\theta},j}^*] - \mu_i^*\mu_j \\ &= \Upsilon_{ij} - \mathrm{tr}\bigg((\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{k}_{\boldsymbol{\theta},j}^*\boldsymbol{k}_{\boldsymbol{\theta},i}^{*T}]\bigg) + \boldsymbol{\beta}^T\mathbb{E}_{\tilde{\boldsymbol{x}}^*}[\boldsymbol{k}_{\boldsymbol{\theta},j}^*\boldsymbol{k}_{\boldsymbol{\theta},i}^{*T}]\boldsymbol{\beta} - \mu_i^*\mu_j^* \\ &= \Upsilon_{ij} - \mathrm{tr}\bigg((\boldsymbol{K_\theta} + \boldsymbol{\Sigma}_\epsilon)^{-1}\boldsymbol{O}_{ij}\bigg) + \boldsymbol{\beta}^T\boldsymbol{O}_{ij}\boldsymbol{\beta} - \mu_i^*\mu_j^* \end{aligned} \tag{2.48}$$

where $\boldsymbol{k}^*_{\boldsymbol{\theta},j} \in \mathbb{R}^{nk}$ denotes the j^{th} column of the matrix. With this we have derived an element-wise expression with which we can evaluate the predictive covariance. The only variable that needs to be defined is $\boldsymbol{O}_{ij} \in \mathbb{R}^{nk\times nk}$.

$$\boldsymbol{O}_{ij} = \int \boldsymbol{k}^*_{\boldsymbol{\theta},j}\boldsymbol{k}^{*T}_{\boldsymbol{\theta},i} p(\tilde{\boldsymbol{x}}^*) d\tilde{\boldsymbol{x}}* = \int \begin{bmatrix} \boldsymbol{k}_{\boldsymbol{\theta},j1}(\tilde{\boldsymbol{X}},\tilde{\boldsymbol{x}}^*)\boldsymbol{k}_{\boldsymbol{\theta},i1}(\tilde{\boldsymbol{x}}^*,\tilde{\boldsymbol{X}}) & \dots & \boldsymbol{k}_{\boldsymbol{\theta},j1}(\tilde{\boldsymbol{X}},\tilde{\boldsymbol{x}}^*)\boldsymbol{k}_{\boldsymbol{\theta},in}(\tilde{\boldsymbol{x}}^*,\tilde{\boldsymbol{X}}) \\ \vdots & \ddots & \vdots \\ \boldsymbol{k}_{\boldsymbol{\theta},jn}(\tilde{\boldsymbol{X}},\tilde{\boldsymbol{x}}^*)\boldsymbol{k}_{\boldsymbol{\theta},i1}(\tilde{\boldsymbol{x}}^*,\tilde{\boldsymbol{X}}) & \dots & \boldsymbol{k}_{\boldsymbol{\theta},jn}(\tilde{\boldsymbol{X}},\tilde{\boldsymbol{x}}^*)\boldsymbol{k}_{\boldsymbol{\theta},in}(\tilde{\boldsymbol{x}}^*,\tilde{\boldsymbol{X}}) \end{bmatrix} p(\tilde{\boldsymbol{x}}^*) d\tilde{\boldsymbol{x}}^* \tag{2.49}$$

Here $\boldsymbol{k}_{\boldsymbol{\theta},jl}(\tilde{\boldsymbol{X}},\tilde{\boldsymbol{x}}^*)$ denotes the covariance between the j^{th} and l^{th} output dimension in regards to $\tilde{\boldsymbol{X}}$ and $\tilde{\boldsymbol{x}}^*$. We further consider looking at the p^{th} and q^{th} sample from $\tilde{\boldsymbol{X}}$.

$$\begin{aligned}
(\boldsymbol{o}^{lk}_{ij})_{pq} &\triangleq \int \boldsymbol{k}_{\boldsymbol{\theta},jl}(\tilde{\boldsymbol{x}}_p,\tilde{\boldsymbol{x}}^*)\boldsymbol{k}_{\boldsymbol{\theta},ik}(\tilde{\boldsymbol{x}}^*,\tilde{\boldsymbol{x}}_q)p(\tilde{\boldsymbol{x}}^*)d\tilde{\boldsymbol{x}}^* \\
&= \sum_{r=1}^{R}\sum_{s=1}^{R} \alpha_r^2\alpha_s^2(2\pi)^{n+m}\sqrt{\det(\boldsymbol{\Lambda}_{\boldsymbol{u_r}}\boldsymbol{\Lambda}_{u_s})}\int \mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \tilde{\boldsymbol{x}}_p,\boldsymbol{\Lambda}_{rjl})\mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \tilde{\boldsymbol{x}}_q,\boldsymbol{\Lambda}_{sik})\mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \boldsymbol{\mu},\boldsymbol{\Sigma})d\tilde{\boldsymbol{x}}^* \\
&= \sum_{r=1}^{R}\sum_{s=1}^{R} \alpha_r^2\alpha_s^2(2\pi)^{n+m}\sqrt{\det(\boldsymbol{\Lambda}_{\boldsymbol{u_r}}\boldsymbol{\Lambda}_{u_s})}\mathcal{N}(\tilde{\boldsymbol{x}}_q \mid \boldsymbol{\mu},\boldsymbol{\Lambda}_{sik}+\boldsymbol{\Sigma})\int \mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \tilde{\boldsymbol{x}}_p,\boldsymbol{\Lambda}_{rjl})\mathcal{N}(\tilde{\boldsymbol{x}}^* \mid \boldsymbol{\mu}_n,\boldsymbol{\Sigma}_n)d\tilde{\boldsymbol{x}}^* \\
&= \sum_{r=1}^{R}\sum_{s=1}^{R} \alpha_r^2\alpha_s^2(2\pi)^{n+m}\sqrt{\det(\boldsymbol{\Lambda}_{u_r}\boldsymbol{\Lambda}_{u_s})}\mathcal{N}(\tilde{\boldsymbol{x}}_q \mid \boldsymbol{\mu},\boldsymbol{\Lambda}_{sik}+\boldsymbol{\Sigma})\mathcal{N}(\tilde{\boldsymbol{x}}_p \mid \boldsymbol{\mu}_n,\boldsymbol{\Lambda}_{rjl}+\boldsymbol{\Sigma}_n)
\end{aligned} \tag{2.50}$$

where $\boldsymbol{\Lambda}_{rjl} = \boldsymbol{\Lambda}_{\boldsymbol{u_r}} + \boldsymbol{\Lambda}_{jr} + \boldsymbol{\Lambda}_{lr}$, $\boldsymbol{\mu}_n = \boldsymbol{\Sigma}_n(\boldsymbol{\Lambda}^{-1}_{sik}\tilde{\boldsymbol{x}}_q + \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})$, and $\boldsymbol{\Sigma}_n = (\boldsymbol{\Lambda}^{-1}_{sik} + \boldsymbol{\Sigma}^{-1})^{-1}$. With this we have formalized the moments of the Gaussian distribution we use to approximate the true predictive state's distribution.

Proposition 1. *Given an input state $\tilde{\boldsymbol{x}}$ distributed as a Gaussian with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ we specified a CP with Gaussian smoothing kernels and SE latent kernels parameterized by hyper-parameters $\boldsymbol{\theta}$. We can then approximate the predictive distribution when propagating $p(\tilde{\boldsymbol{x}})$ through the CP with a Gaussian distribution. The mean $\boldsymbol{\mu}^*$ is derived from Eq. 2.42 where $\boldsymbol{P}$ has elements in the form of Eq. 2.44. The covariance matrix $\boldsymbol{\Sigma}^*$ is fully described through Eq. 2.48 where $\boldsymbol{O}_{ij}$ for each output dimension pair is specified element-wise through Eq. 2.50.*

2.2.2 Remarks

Due to the complexity of this model for both training and inference we do not consider practical results in this thesis but leave it for future work. The novelty provided with this section are the derivations of the predictive Gaussian distribution's moments. We believe that due to the increased capabilities of hardware in reducing the complexity of learning models like Gaussian Processes such a model will be able to scale to higher dimensions where more interesting dynamics can be considered. It is at this point the coupling effect that this model induces naturally will begin to show it's merits by being more sample efficient.

1. The time and memory complexity of this model for training is $\mathcal{O}((nk)^3)$ and $\mathcal{O}((nk)^2)$, respectively. This is extremely prohibitive to systems with high dimensional states or even large sample sizes.

2. The time complexity for predictions of the mean and covariance through a CP is $\mathcal{O}(n^2kR(n+m)^3)$ and $\mathcal{O}((nkR)^2(n+m)^3)$.

2.3 Gaussian Processes Related Works

GPs have garnered much attention over the past few years because of there analytical derivations, flexibility, and ability to represent uncertainty naturally. The main concern for using such models is it's scalability to higher dimensions; as it scales cubicly with respect to the data size. In conjunction with improving the scalability of these models, researchers are working on developing the expressibility through kernel design and developing different models for applications.

Neal [5] instigates the excitement for a non-parametric model (GP) through showing that the properties of a single layer neural network converge to those of a GP as the width tends to infinity. The covariance function of the GP is then determined through the prior placed on the weights and activation functions of the hidden layers. William & Rasmussen [6] and Neal [7] began to thoroughly explore the efficacy of this approach and build the foundation for the research community. For an in-depth introduction please refer to the book by Rasmussen [8]. Recently, Lee et al. [9] derive the exact equivalence relationship between deep neural networks and GPs. Using this relation they efficiently compute the kernel functions of a proposed network and perform Bayesian inference for wide deep neural networks.

2.3.1 Efficiency Improvement

The main initial drawback of using GPs despite being highly flexible is that training on large datasets is prohibitive due to the inversion of the covariance matrix. Standard techniques for matrix inversion require $\mathcal{O}(n^3)$ operations making datasets with sample size larger than a few thousand too time intensive. To remedy this, Snelson & Ghahramani [10], [11] propose inducing point methods that use a set of $m << n$ sparse pseudo-inputs to approximate the kernel with a decomposition allowing for $\mathcal{O}(nm^2)$ time complexity during training and $\mathcal{O}(m^2)$ during predictions. Due to the sparse selection of inducing points these methods sacrifice predictive capabilities and the expressibility of the kernels. Rather than considering the set of induced points as the parameters for a degenerate prior, Titsias [12] considers them as the parameters of a variational distribution and maximize the lower bound of the marginal log-likelihood. This leads to the model being more robust to overfitting and a training time complexity of $\mathcal{O}(m^3)$. The main difference between these approaches is in them approximating the prior or posterior distribution respectively. Burt et al. [13] show that the true time complexity grows sublinearly with the number of training samples indicating that posteriors can be approximated cheaply. Alternative to using inducing points, structure exploitation through Kronecker products [14] or Toeplitz methods [15] also allow for scalable inference and expressive kernel learning making. The main hinderance regarding structural approaches is that they require the data to lie on a multidimensional lattice. Wilson & Nickisch [16] propose the structured kernel interpolation (SKI) framework that applies both concepts of inducing points and structural exploitation to further reduce the time and storage complexity while maintaining the expressiveness of the kernel.

2.3.2 Expressive Kernels

The choice of kernel for a GP determines largely it's inference capabilities after training as it is placing assumptions on the properties of the data i.e. smoothness, stationarity, etc. Along with ridge regression and support vector machines (SVMs), GP motivate the problem of automatic kernel learning [17]. In addition to kernel learning, allowing for a general structure ensures that the learning approach is not limited in it's search space. Bach [18] considers a heirachical kernel learning (HKL) method that considers additive composition of kernels over subsets of the variables. Duvenaud et al. [19] extended this HKL method to be Bayesian and propose the Additive Guassian processes. Duvenaud et al. [20] alternatively presents a context-free grammar of kernel structures based on the composition of kernels. Lloyd et al. [21] extend this work to an Automatic Statistician that generates natural language reports on the models assessed. Sun et al. [17] accelerate the learning of the automatic statistician approach by making it differentiable in the form of a neural kernel network (NKN). Additionally, they show that the NKN is universal for stationary kernels. An alternative approach to kernel learning is through the spectral approaches that rely on Bochner's Theorem [22]. Lazaro-Gredilla et al. [23] propose sparse spectrum kernels where they limit the number of Fourier coefficients. Wilson & Adam [24] consider spectral mixture kernels where the each spectral dimension is modelled as an independent Gaussian. Samo & Roberts [25] extend this work to a tractable family of spectral kernels that can approximate any continuous bounded nonstationary kernel. Parra & Tobar [26] consider spectral mixture kernels for multi-output gaussian processes through employing Cramér's Theorem to provide a principled approach to multivariate covariance functions.

2.3.3 Model Design

An exact Gaussian process is limited to the one dimensional regression case. Thus there has been large consideration of extending this to multi-dimensional regression, and deep learning. When considering multi-dimensional regression the standard approach is to apply a GP per output dimension and consider the outputs to be conditionally independent given the training data. Boyle & Frean [27] model the dependencies through considering the outputs to be a convolution of the kernel and a latent function represented as a GP thereby generating a convolution process (CP). Alvarez & Lawrence [28] introduce sparse approximations analogous to fully independent training conditional (FITC) [29] to reduce the time and storage complexity of a CP. Alvarez et al. [30] further extend Titsias [12] work of variational inference to the multi-dimensional case and introduce functions for handling potentially non-smooth kernels in the CP. Muńoz et al. [31] consider extending the multi-output framework by allowing heterogenous outputs through assuming different likelihood functions. Considering the benefits of deep learning, Damianou & Lawrence [32] [33] introduce deep gaussian processes (DGP) which extend the GP formulation through layering and is trained with approximate variational inference. Thus through process composition of GPs a richer class of process priors can be learned while maintaining theoretical properties of the underlying processes [34]. Salimbeni & Deisenroth [35] present a variational inference approach that extends previous mean field variational approach through not inforcing independence or Gaussianity between layers. This reduces the concerns of underestimating the variance as was done with the mean-field variational approaches [36].

2.4 Bayesian Neural Networks

In contrast to a GP, Bayesian neural networks (BNN) are parametric models with a finite set of parameters. A large appeal for neural networks is their modularity and expressive structural designs. Furthermore, applying the Bayesian inference approach to neural networks is motivationed by robustness to overfitting, sense of uncertainty, and natural hyperparameter learning. For reinforcement learning uncertainty is necessitated through the exploration vs. exploitation tradeoff. Since the motion model is unknown exploring the environment generates a better understanding of the dynamics allowing for better policies to be determined. Exploration requires the agent to have a sense of uncertainty as a deterministic model causes the agent to act over-confident even in regions of the environment that are previously unseen. We consider Hernandez-Lobato's Probabilistic Backpropagation (PBP) [37] as methods for learning the posterior distribution. For the case of dynamic modeling a feed forward network is generally sufficient given that the state space is observable by the agent. Throughout this section we consider the structural flow of the network at the l^{th} layer to follow,

$$\boldsymbol{z}^{l-1} \rightarrow \boldsymbol{W}^l \boldsymbol{z}^{l-1} + \boldsymbol{b}^l \rightarrow h(\boldsymbol{W}^l \boldsymbol{z}^{l-1} + \boldsymbol{b}^l) = \boldsymbol{z}^l$$

where $\boldsymbol{\omega} \triangleq \{(\boldsymbol{W}^l, \boldsymbol{b}^l) : \boldsymbol{W}^l \in \mathbb{R}^{V_l \times V_{l-1}}, \boldsymbol{b} \in \mathbb{R}^{V_l}, l = 1, \dots, L\}$ are the model parameters, $h(\cdot)$ is the non-linearity, and $\boldsymbol{z}^l$ the output vector to the l^{th} layer. Furthmore we define the hidden units as $\boldsymbol{a}^l \triangleq \boldsymbol{W}^l \boldsymbol{z}^{l-1} + \boldsymbol{b}^l$ and $\boldsymbol{z}^L \triangleq \Delta\hat{\boldsymbol{x}} = g(\tilde{\boldsymbol{x}}; \boldsymbol{\omega}, \boldsymbol{\theta}) + \epsilon$ where $g(\cdot; \boldsymbol{\omega}, \boldsymbol{\theta})$ is the learned approximate state difference of the dynamics. A neural network will learn the mode of the parameters $\boldsymbol{\omega}^*$, $\boldsymbol{\theta}^*$ such that the distribution $p(\Delta\boldsymbol{x} \mid \tilde{\boldsymbol{x}}, \boldsymbol{\omega}^*, \boldsymbol{\theta}^*)$ has the best generalization results. In order to consider a network bayesian we are interested in learning a posterior distribution $p(\boldsymbol{\omega} \mid \mathcal{D}, \boldsymbol{\theta})$ instead. We can then apply marginalization for inference given the test set $\mathcal{D}^*$.

$$p(\Delta\boldsymbol{x}^* \mid \tilde{\boldsymbol{X}}^*, \boldsymbol{\theta}) = \int p(\Delta\boldsymbol{x}^* \mid \tilde{\boldsymbol{X}}^*, \boldsymbol{\omega}, \boldsymbol{\theta}) p(\boldsymbol{\omega} \mid \mathcal{D}, \boldsymbol{\theta}) d\boldsymbol{\omega} \tag{2.51}$$

Hernandez-Lobato [37] demonstrates learning a posterior distribution through assuming that it is of a Gaussian form with diagonal covariance. This allows for an iterative minimization of the Kullback-Liebler (KL) Divergence through the idea of an *assumed density filter* approach [38].

2.4.1 Probabilistic Back-Propagation

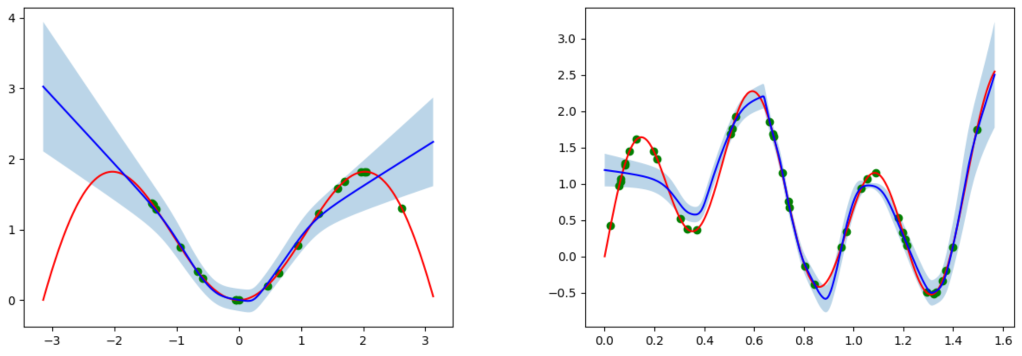

Figure 2.4: Evaluation of a bayesian neural network trained with probabilistic backpropagation on $x\sin(x)$ and $x + \sin(4x) + \sin(13x)$ respectively. In these figures the red line is the true function while the blue is the mean of the estimated function, and the green points are the training inputs. The shaded region corresponds to the 95% confidence interval.

At a high level filtering and estimation is the processes of updating our state distribution based on predictions through a model and observations from the environment. In the assumed density approach we consider the true posterior family to be known and try to learning an approximate distribution that minimizes the KL Divergence. If we assume the prior distribution $p(\boldsymbol{\omega})$ to be Gaussian the resulting posterior $p(\boldsymbol{\omega} \mid \mathcal{D})$

is generally of complex form. This makes minimizing the divergence intractable. However, if we assume the posterior to be Gaussian as well we can derive update steps for the moments of the prior. Specifically, we consider the optimization problem,

$$\begin{gathered}
\underset{\boldsymbol{\mu},\sigma^2}{\arg\min}\, KL(p(\boldsymbol{\omega} \mid \mathcal{D})||q(\boldsymbol{\omega})) = \int p(\boldsymbol{\omega} \mid \mathcal{D}) \log \frac{p(\boldsymbol{\omega} \mid \mathcal{D})}{q(\boldsymbol{\omega})} d\boldsymbol{\omega} \;\; s.t. \\
p(\boldsymbol{\omega} \mid \mathcal{D}) \propto Z^{-1} p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\omega}) q(\boldsymbol{\omega}) \\
q(\boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\omega} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad p(\boldsymbol{\omega} \mid \mathcal{D}) = \mathcal{N}(\boldsymbol{\omega} \mid \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n) \\
Z = \int p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\omega}) q(\boldsymbol{\omega}) d\boldsymbol{\omega}
\end{gathered} \tag{2.52}$$

where $p(\boldsymbol{\omega} \mid \mathcal{D})$ is the true posterior while $q(\boldsymbol{\omega})$ is the approximate distribution. We assume that $\boldsymbol{\Sigma} = \text{diag}([\sigma_1^2, \ldots, \sigma_p^2])$ where $p = \#|\boldsymbol{\omega}|$. Because of the independence assumption on the weights element-wise update steps follow,

$$\mu_{ni} = \mu_i + \frac{\partial \log Z}{\partial \mu_i}, \qquad \sigma_{ni}^2 = \sigma_i^2 - \sigma_i^4 \left[\left(\frac{\partial \log Z}{\partial \mu_i} \right)^2 - 2 \frac{\partial \log Z}{\partial \sigma_i^2} \right] \tag{2.53}$$

Considering again the dataset $\mathcal{D}$. From this we can determine the likelihood function for the weights of the network,

$$p(\Delta \boldsymbol{x} | \tilde{\boldsymbol{X}}, \boldsymbol{\omega}, \gamma) = \prod_{n=1}^{k} \mathcal{N}(\Delta \boldsymbol{x}_n \mid g(\tilde{\boldsymbol{x}}_n, \boldsymbol{\omega}), \gamma^{-1}) \tag{2.54}$$

where $\gamma^{-1} = \sigma_\epsilon^2$. We consider the weights to be indepdent and specify a prior over the weights as a product of univariate gaussian distributions,

$$p(\boldsymbol{\omega}_d | \lambda) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(\omega_{ijl}; 0, \lambda^{-1}) \tag{2.55}$$

where $\boldsymbol{\omega} = \boldsymbol{\omega}_d \cup \boldsymbol{\omega}_i$ are the direct and indirect parameters. The indirect parameters $\boldsymbol{\omega}_i = \{\gamma, \lambda\}$ are chosen to be distributed according to Gamma distributions. We consider the joint posterior to have the a functional form,

$$p(\boldsymbol{\omega} \mid \mathcal{D}) = \frac{p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}}, \boldsymbol{\omega}, \gamma) p(\boldsymbol{\omega}_d \mid \lambda) p(\lambda) p(\gamma)}{p(\Delta \boldsymbol{x} \mid \tilde{\boldsymbol{X}})} \tag{2.56}$$

Since the posterior is proportional to the numerator which is Gaussian we can assume that the functional form of the posterior is also Gaussian with a diagonal covariance. This leads to the consideration of the variational distribution form of,

$$q(\boldsymbol{\omega}) = \prod_{l=1}^{L}\prod_{i=1}^{V_l}\prod_{j=1}^{V_{l-1}+1} \mathcal{N}(\omega_{ijl} \mid \mu_{ijl}, \sigma^2_{ijl})\Gamma(\gamma \mid \alpha^\gamma, \beta^\gamma)\Gamma(\lambda \mid \alpha^\lambda, \beta^\lambda) \tag{2.57}$$

where $\Gamma(\cdot \mid \alpha, \beta)$ is the Gamma distribution with shape $\alpha > 0$ and rate β and $\boldsymbol{\theta} = \{\mu_{ijl}, \sigma^2_{ijl}, \alpha^\gamma, \beta^\gamma, \alpha^\lambda, \beta^\lambda\}$ for $l = 1, \ldots, L$, $i = 1, \ldots, V_l$, and $j = 1, \ldots, V_{l-1}+1$. We initialize the approximate distribution as uniform, i.e. $\mu_{ijl} = 0, \sigma^2_{ijl} = \infty, \alpha^\gamma_0 = \alpha^\lambda_0 = 1, \beta^\gamma_0 = \beta^\lambda_0 = 0$. We then iteratively incorporate the factors in the numerator of Eq. 2.56 and derive the update steps for parameters governing $\boldsymbol{\omega}$. The number of factors that must be incorporated are: two factors for the priors on γ and λ, $\prod_{l=1}^{L} V_l(V_{l-1}+1)$ factors for the prior on $\boldsymbol{\omega}$, and k factors for the likelihood. We have seen that the updates for the parameters of the prior on $\boldsymbol{\omega}$ follow according to Eqs. 2.53. Thus what remains to be computed are the updates for the hyper-parameters, $\alpha^\lambda, \beta^\lambda, \alpha^\gamma, \beta^\gamma$ when incorporating the all the factors and how to approximate the marginal likelihood.

First we incorporate the priors on γ and λ. The Gamma distribution is closed under multiplication and produces a new distribution, $\Gamma(\cdot \mid \alpha_1 + \alpha_2 - 1, \beta_1 + \beta_2) = \Gamma(\cdot \mid \alpha_1, \beta_1)\Gamma(\cdot \mid \alpha_2, \beta_2)$. Therefore we do not have to concern ourselves with the minimization of the KL divergence and the update for the parameters is straightforward: $\alpha^\lambda_n = \alpha^\lambda_0, \beta^\lambda_n = \beta^\lambda_0, \alpha^\gamma_n = \alpha^\gamma_0, \beta^\gamma_n = \beta^\gamma_0$.

Next, we incorporate the $\prod_{l=1}^{L} V_l(V_{l-1}+1)$ prior factors. The update for for μ_{ijl} and σ^2_{ijl} are from Eq. 2.53. We can derive update steps for $\alpha^\lambda, \beta^\lambda$ through matching the moments of λ rather than the minimizing the KL divergence as it does not have a closed form solution [38].

$$\alpha^{\lambda^n} = [Z Z_2 Z_1^{-2}(\alpha^\lambda + 1)/\alpha^\lambda - 1]^{-1} \tag{2.58}$$

$$\beta^{\lambda^n} = [Z_2 Z_1^{-1}(\alpha^\lambda + 1)/\beta^\lambda - Z_1 Z^{-1}\alpha^\lambda/\beta^\lambda]^{-1} \tag{2.59}$$

Here Z_1, Z_2 denote the marginal likelihood when α^λ is increased by 1 or 2, respectively. In order to apply this update rule though Z must be determined which given the functional forms does not have a closed form solution. Thus it is approximated through a Gaussian,

$$Z = \int\int\int \mathcal{N}(\omega_{ijl}; 0, \lambda^{-1}) q(\boldsymbol{\omega}, \lambda, \gamma) d\boldsymbol{\omega} d\lambda d\gamma \approx \mathcal{N}\left(\mu_{ijl} \middle| \mathbf{0}, \frac{\beta^\lambda}{\alpha^\lambda + 1} + \sigma^2_{ijl}\right) \tag{2.60}$$

Finally, there are k likelihood factors that we have to incorporate into our estimate of the posterior. Similar to our approximation of the normalizer for the weight factors we can approximate it with a Gaussian,

$$Z = \int\int\int p(\Delta\boldsymbol{x}_n \mid g(\tilde{\boldsymbol{x}}_n; \boldsymbol{\omega}), \gamma^{-1}) q(\boldsymbol{\omega}, \gamma, \lambda) d\boldsymbol{\omega} d\gamma d\lambda \approx \mathcal{N}\left(\Delta\boldsymbol{x}_n \middle| \boldsymbol{m}^L, \frac{\beta^\gamma}{\alpha^\gamma + 1} \oplus \boldsymbol{v}^L\right) \tag{2.61}$$

where we assume that $\boldsymbol{z}^L = g(\tilde{\boldsymbol{x}}; \boldsymbol{\omega}) \sim \mathcal{N}(\boldsymbol{m}^L, \text{diag}(\boldsymbol{v}^L))$ and $\oplus$ denotes an element-wise addition of a scalar to a vector. We can apply the same update rules for the parameters of our posterior using this approximation. What remains to be determined are the values $\boldsymbol{m}^L, \boldsymbol{v}^L$ which requires propagating an initial distribution through the network. For this we assume that $\boldsymbol{z}^l$ is a Gaussian with diagonal covariance represented by the vectors, $\boldsymbol{m}^l, \boldsymbol{v}^l$. Furthermore we consider the normalized the linear output, $\boldsymbol{a}^l = \boldsymbol{W}^l \boldsymbol{z}^{l-1}/\sqrt{V_{l-1}+1}$ and let $\boldsymbol{m}^{l-}, \boldsymbol{v}^{l-}$ be the hidden units mean and variance. We can represent the hidden layers distributions as,

$$\boldsymbol{m}^{l-} = \frac{\boldsymbol{M}^l \boldsymbol{m}^{l-1}}{\sqrt{V_{l-1}+1}} \tag{2.62}$$

$$\boldsymbol{v}^{l-} = \frac{(\boldsymbol{M}_l \circ \boldsymbol{M}_l)\boldsymbol{v}^{l-1} + \boldsymbol{V}_l(\boldsymbol{m}^{l-1} \circ \boldsymbol{m}^{l-1})}{V_{l-1}+1} \tag{2.63}$$

where $\boldsymbol{M}^l, \boldsymbol{V}^l \in \mathbb{R}^{V_l \times V_{l-1}+1}$ are the mean and variance matrices at each layer with elements $\mu_{ijl}, \sigma^2_{ijl}$, respectively and $\circ$ denotes the Hadamard product. Now we consider the activation function $\boldsymbol{z}^l = h(\boldsymbol{a}^l) \triangleq \max\{0, \boldsymbol{a}^l\}$ as a rectifying linear unit (RELU) function. We can then formulate the mean of the i^{th} element as,

$$m_i^{l+} = \Phi(\alpha_i)\nu_i \tag{2.64}$$

$$v_i^{l+} = m_i^{l+}\nu_i\Phi(-\alpha_i) + \Phi(\alpha_i)v_i^{l-}(1 - \tau_i(\tau_i + \alpha_i)) \tag{2.65}$$

$$\nu_i = m_i^{l-} + \sqrt{v_i^{l-}}\tau_i, \quad \alpha_i = \frac{m_i^{l-}}{v_i^{l-}}, \quad \tau_i = \frac{\phi(-\alpha_i)}{\Phi(\alpha_i)} \tag{2.66}$$

Here $\phi(x)$ and $\Phi(x)$ are the probability density function and cumulative density function evaluated with x, respectively. Finally to determine the parameters of $\boldsymbol{z}^l$ we concatenate a degenerate prior for the bias to $\boldsymbol{m}^{l+}$ and $\boldsymbol{v}^{l+}$. This mean and variance indicate that the bias of each layer is deterministic.

$$\boldsymbol{m}^l = [\boldsymbol{m}^{l+}; 1], \quad \boldsymbol{v}^l = [\boldsymbol{v}^{l+}; 0] \tag{2.67}$$

Another term that is of interest the cross-covariance between an input $\tilde{\boldsymbol{x}} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}, \text{diag}(\tilde{\boldsymbol{v}}))$ and output $\Delta\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{m}^L, \text{diag}(\boldsymbol{v}^L))$. Since $\Delta\boldsymbol{x} \mid \tilde{\boldsymbol{x}}, \boldsymbol{\omega}, \boldsymbol{\theta} \sim \mathcal{N}(\Delta\boldsymbol{x} \mid g(\tilde{\boldsymbol{x}}; \boldsymbol{\omega}), \gamma^{-1}\boldsymbol{I})$ it implies that the joint distribution is Gaussian as well. Thus we know the relationship,

$$\text{diag}(\boldsymbol{v}^L) - \gamma^{-1}\boldsymbol{I} = \boldsymbol{\Sigma}_{io}\text{diag}(\tilde{\boldsymbol{v}})^{-1}\boldsymbol{\Sigma}_{io}^T \tag{2.68}$$

where $\mathbf{\Sigma}_{io}$ is the cross-covariance of interest. Thus we can solve this linear system to determine $\mathbf{\Sigma}_{io}$. Through considering this cross-covariance we are making sure that the agent can account for this uncertainty during the policy optimization.

Proposition 2. *Given a Gaussian distributed input $\boldsymbol{p} \sim \mathcal{N}(\boldsymbol{m}^p, diag(\boldsymbol{v}^p))$ and resulting output $\boldsymbol{q} \sim \mathcal{N}(\boldsymbol{m}^L, diag(\boldsymbol{v}^L))$ from a learned BNN using PBP we can determine $\mathbf{\Sigma}_{io}$ by solving Eq. 2.68 and enforcing that $\mathbf{\Sigma}_{io}$ is positive definite.*

Through this formulation Hernandez has provided a means of propagating an initial distribution $\boldsymbol{m}^0 = [\tilde{\boldsymbol{x}}_t; 1]$, $\boldsymbol{v}^0 = [\text{diag}\big(\mathbf{\Sigma}\big); 0]$ forward and applying backpropagation with respect to the marginal log-likelihood to approach the true posterior distribution. Given this formulation it is already naturally extended to consider propagating distributions sequentially. The only concern is that distributions are restricted to having a diagonal covariance. Hence in relation to the GP's propagation this model is sacrificing flexibility in expressing the model's uncertainty for scalability to higher dimensional data and larger sample sets. Thus we expect it to allow for more complex systems to be considered. Now we have provided two forms, parametric and non-parametric, for learning a model on which to apply Eq. 1.2.

2.5 Bayesian Networks Related Works

Bayesian Neural Networks provide a nice extension onto neural networks accounting for model uncertainty and providing stochastic outputs. However, due to the model architecture, computing the posterior of such models is intractable making how to approximate this distribution an open question. Furthermore, there is an equivalence relationship between NNs and GPs [5], [9] allowing for the development of explainability of a neural network in terms of GPs.

MacKay [39] argues for the case of the Bayesian approach in neural networks through approximating the Bayesian inference with Laplace approximations. Although it was applied to small and shallow networks, he demonstrates the potential benefits the Bayesian approach could have through accounting for model flexibility, easing model comparisons with heirachical parameters, calibrating predictive uncertainty, and having robustness to overfitting. Given this initial proposition, the two main directions proposed where variation inference and markov chain monte carlo sampling. Hinton & Camp [40] and MacKay [41] developed the variational inference approach through the perspective of minimum descriptive length compression (MDL) and variational free energy minimization respectively. While Neal [42] uses Hamiltonian Monte Carlo (HMC) sampling to approach the posterior distribution.

2.5.1 Variational Inference

Follwing the initial excitement, Barber & Bishop [43] extend MacKay's approach to allow for multivariate gaussian distributions. Wu et al. [44] follow Barber & Bishop's idea of approximating closed form solutions for the marginal likelihood and KL divergence but differ in their approximations for the non-linearities and use MAP estimates for the hyperparameters rather than using Bayesian inference. After a decade, Graves [45] introduces Monte Caro variational inference (MCVI) where the evidence lower bound (ELBO) expectation is approximated through sampling. Bundell et al. [46] improve upon MCVI by allowing for non-Gaussian posteriors and having unbiased gradient estimates for the mean and variance through the reparameterizing trick [47]. In addition to reducing the variance in MCVI finding expressive variational families for the posteriors such as Matrix Gaussian distributions [48], multiplicative posteriors [47], and hierarchical posteriors [49] is a key research area.

2.5.2 Markov Chain Monte Carlo

Neal's HMC approach [42] started a new class of stochastic gradient Markov Chain Monte Carlo (MCMC) algorithms that approximate posterior parameter inference with unbiased log-likelihood estimates. Chen et al. [50] revisit the stochastic HMC and considers addendums to the Hamiltonian dynamics. Welling & Teh [51] introduce Stochastic Gradient Langevin Dynamics which follows the gradient based on a Langevin dynamics formulation. Ahn et al. [52] increase the efficieny of this method for correlated posteriors by estimating the Fisher Information matrix. Despite the development of this methods a theoretical understanding of guarantees remain open [53]. A literature review regarding this class of methods is provided by [54].

2.5.3 Alternative Inference.

Aside from variational inference, Hernández & Adam [37] propose applying an assumed density filter approach to iteratively refine Gaussian posterior distribution. They assume a diagonal covariance during propagation and homoscedastic regression. This formulation has been extended by Ghosh et al. [55] to the classification case and by Sun et al. [56] to consider matrix variate gaussian posterior distributions. Gal & Ghahramani [57] propose using Dropout and Monte Carlo sampling to provide uncertainty estimates equivalent to variational inference in deep Gaussian processes. Osband [58] argues that this approach does not satisfy Bayesian approximation of uncertainty as it is uncorrelated with the amount of data and is actually assessing *risk* at the output rather than model uncertainty. Alternatively, bootstrapped posteriors [59], [60], are generated by learning parameters for sample datasets from a dataset perturb by noise and using the resulting distribution to estimate uncertainty.

Chapter 3

Optimal Control Policy

Now that we have formulated two methods for learning a model, $\mathcal{M}$, we are able to start learning a policy with respect to those models. More formally, given an environment's dynamics f and an initial parameterized policy $\pi_{\boldsymbol{\psi}}$ we can sample from the environment's approximate state difference dynamics g and generate a dataset $\mathcal{D}$. With this dataset we can perform model learning to determine $\mathcal{M}$. Using this differentiable model we can construct an approximate MDP and perform a gradient based policy search.

Algorithm 1 Sequential Model Policy Learning

```
**Inputs:** $\mathcal{M}$, $\pi_{\boldsymbol{\psi}}$, $f$, $p(\boldsymbol{x}_0)$, $c$, $T$
$J^{\pi-} \leftarrow 0$
$\mathcal{D} \sim g \mid \pi_{\boldsymbol{\psi}}$
**while not** Done **do**
    $\mathcal{M} \leftarrow UpdateModel(\mathcal{M}, \mathcal{D})$
    $\pi_{\boldsymbol{\psi}} \leftarrow PolicySearch(\pi_{\boldsymbol{\psi}}, \mathcal{M}, p(\boldsymbol{x}_0), c, T)$
    $T, J^{\pi}, \text{Done} \leftarrow Evaluate(\pi_{\boldsymbol{\psi}}, \mathcal{M}, p(\boldsymbol{x}_0), c, T, J^{\pi-})$
    $J^{\pi-} \leftarrow J^{\pi}$
    $\mathcal{D}_n \sim g \mid \pi_{\boldsymbol{\psi}}$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_n$
**return** $\pi_{\boldsymbol{\psi}}, \mathcal{M}, J^{\pi}, T$
```

At a high-level overview of the algorithm, we see that it consists of three main steps. The *Update-Model* function is model dependent and discussed for the two models presented in Ch. 2. During this chapter we will formulate the *PolicySearch* and *Evaluate* functions in consideration of each model. The policy search consists of state distribution propagation and gradient based optimization. In order to consider learning a policy we also have to interpret how to propagate the predictive distribution forward in time. This will allow us to compute the expected immediate cost of a state distribution and therefore the expected cumulative cost J^{π}. Obtaining this function value will allow us to employ a gradient based optimization to iteratively improve the estimates of the policy parameters. For the content of this paper will discuss the cases where the policy is deterministic and has a linear or radial basis functional form. We choose these functional forms as they allow for closed form gradients without approximations. We organize the chapter as follows; we start by consider the prelimary policy forms and how to determine the distribution over the control space. To be practical, we discuss bounding a policy's range space as real systems have control constraints. As we then know both the model and the policy types we present how to propagate a state distribution at time t to time $t+1$. Given the state distributions until the time horizon we can determine the expected immediate cost $\mathbb{E}_{\tilde{\boldsymbol{x}}_t}[c(\boldsymbol{x}_t, \boldsymbol{u}_t)]$ which encode the objective in our cost function. Thus we introduce applicable cost functions for determining J^{π}. Lastly, we discuss how to compute the gradient of the objective with respect to the policy parameters $\boldsymbol{\psi}$ considering the models. This encompasses the *PolicySearch* function. We introduce a way to implement the *Evaluate* function in consideration of increasing performance of the policy. We conclude the chapter with literature review regarding model-based reinforcement learning.

3.1 Policy Search

Algorithm 2 Policy Search Algorithm

```
**Inputs:** $\mathcal{M}$, $\pi_{\boldsymbol{\psi}}$, $p(\boldsymbol{x}_0)$, $c$, $T$
$J^{\pi -} \leftarrow 0$
for $t = 1, \ldots, T$ do
    $p(\boldsymbol{x}_t) \leftarrow Propagate(\mathcal{M}, \pi_{\boldsymbol{\psi}}, p(\boldsymbol{x}_{t-1}))$
    $\kappa_t \leftarrow ExpectedCost(c, p(\tilde{\boldsymbol{x}}_t))$
    $\nabla_{\boldsymbol{\psi}} J^{\pi}_t \leftarrow CostGradient(c, p(\tilde{\boldsymbol{x}}_t))$
$J^{\pi} \leftarrow \sum_{t=1}^{T} \kappa_t$
while $|J^{\pi} - J^{\pi -}| > \epsilon_{search}$ do
    $J^{\pi -} \leftarrow J^{\pi}$
    $\nabla_{\boldsymbol{\psi}} J^{\pi} \leftarrow \sum_{t=1}^{T} \nabla_{\boldsymbol{\psi}} J^{\pi}_t$
    $\boldsymbol{\psi} \leftarrow ParameterUpdate(\nabla_{\boldsymbol{\psi}} J^{\pi}, \boldsymbol{\psi})$
    for $t = 1, \ldots, T$ do
        $p(\boldsymbol{x}_t) \leftarrow Propagate(\mathcal{M}, \pi_{\boldsymbol{\psi}}, p(\boldsymbol{x}_{t-1}))$
        $\kappa_t \leftarrow ExpectedCost(c, p(\tilde{\boldsymbol{x}}_t))$
        $\nabla_{\boldsymbol{\psi}} J^{\pi}_t \leftarrow CostGradient(c, p(\tilde{\boldsymbol{x}}_t))$
    $J^{\pi} \leftarrow \sum_{t=1}^{T} \kappa_t$
return $\pi_{\boldsymbol{\psi}}$
```

The policy search consists of propagating distributions from t to $t+1$ and evaluating the gradient through a gradient optimization procedure like conjugate gradients. We will formalize how the *Propagate* function works through Gaussian approximations. To do this we need to discuss the different parameterized policies. Then we can consider different cost functions in order to determine the *ExpectedCost* function. Given the cost function we can then also specify the *CostGradient* function.

3.1.1 Policy Forms

A deterministic policy $\pi_{\boldsymbol{\psi}}$ presents a mapping from the state space $\mathcal{X}$ to the control space $\mathcal{U}$. Different functional forms of the policy provide various applications and should be treated on an application basis. For stabilization near an equilibrium a linear policy performs well. For non-linear tasks and environments a radial basis or neural network policy are more applicable. We shall consider these functional forms of the policy as the preliminary policy $\tilde{\pi}_{\boldsymbol{\psi}}$ as these forms have an infinite range which may not be practical. Since the input to the policy is a distribution the resulting output will also be a distribution over the control space,

$\mathcal{U}$. It is also necessary to determine the distribution over the control space in order to properly propagate the the uncertainty forward without under-estimating it.

Linear Policy

A linear policy $\tilde{\pi}_{\boldsymbol{\psi}}$ has has to form:

$$\tilde{\pi}_{\boldsymbol{\psi}}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{3.1}$$

where $\boldsymbol{\psi} = \{\boldsymbol{W}, \boldsymbol{b}\}$. Due to the linearity propagating Gaussians through the policy remain in closed form. If we are given a state $\boldsymbol{x} \sim \mathcal{N}(\mu, \Sigma)$ and the parameters $\boldsymbol{W} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^{m}$ corresponding to the affine transformation, $\mathcal{W} : \mathcal{X} \mapsto \mathcal{U}$. It follows that the respective control is distributed as a Gaussian with parameters, $\boldsymbol{u} \sim \mathcal{N}(\boldsymbol{W}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{W}^T)$. We can further construct the joint distribution as,

$$\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{pmatrix} \sim \mathcal{N}\Bigg(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{W}\boldsymbol{\mu} + \boldsymbol{b} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma} & \boldsymbol{\Sigma}\boldsymbol{W}^T \\ \boldsymbol{W}\boldsymbol{\Sigma} & \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{W}^T \end{pmatrix} \Bigg) \tag{3.2}$$

Radial Basis Policy

For the nonlinear case we can consider a policy comprised of Gaussian basis functions (i.e. squared exponential kernels).

$$\tilde{\pi}_{\boldsymbol{\psi}}(\boldsymbol{x}) = \sum_{i=1}^{N} \beta_i k_\pi(\boldsymbol{z}_i, \boldsymbol{x}) = \boldsymbol{\beta}_\pi^T k_\pi(\boldsymbol{Z}, \boldsymbol{x})$$

where $k_\pi(\boldsymbol{z}, \boldsymbol{z}')$ is equivalent to the kernel specified in the GP model in Eq. 2.3. The set $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]$, $\boldsymbol{z}_i \in \mathbb{R}^n$, $i = 1, \ldots, N$ are the means of the basis functions while we can consider $\boldsymbol{\beta}_\pi$ as the vector of weights. We can directly estimate $\boldsymbol{\beta}_\pi$ or define them as $\boldsymbol{\beta}_\pi \triangleq (\boldsymbol{K}_\pi + \sigma_\pi^2 \boldsymbol{I})^{-1}\boldsymbol{y}_\pi$ where we consider the entries of $(\boldsymbol{K}_\pi)_{ij}$ to be $k_\pi(\boldsymbol{z}_i, \boldsymbol{z}_j)$ and $\boldsymbol{y}_\pi = \tilde{\pi}(\boldsymbol{Z}) + \epsilon_\pi$ are the target values where $\epsilon_\pi \sim \mathcal{N}(\boldsymbol{0}, \sigma_\pi^2 \boldsymbol{I})$. Both formulations are equally expressive and we can consider the parameters to be either $\psi = \{\beta_\pi, \boldsymbol{Z}, \alpha_\pi, \boldsymbol{\Lambda}_\pi\}$ or $\psi = \{\boldsymbol{y}_\pi, \boldsymbol{Z}, \alpha_\pi, \boldsymbol{\Lambda}_\pi\}$. For fixed N of basis functions this controller is functionally equivalent to the mean function of a GP. Given that the input distribution is Gaussian $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we can apply the prediction formulations mentioned in Sec. 2.1.2. This formulation is also naturally extended to the multivariate control. If we consider the control outputs to be independent then we can consider a controller per dimension as with the GPs.

Constrained Policy

In most practical applications the controls are constrained in a closed interval, $\boldsymbol{u} \in [-\boldsymbol{u}_{max}, \boldsymbol{u}_{max}]$. The policies mentioned above have an unconstrained range allowing for potential erroneous outputs. In order to apply a gradient based search we have to consider a differentiable (preferrably twice continuously differentiable) means of limiting the control signal. Some viable differentiable squashing functional forms that can be considered are sinusoid, sigmoid, or cumulative Gaussian. In this paper we will consider the sinusoid as the squashing function as it attains the limits ± 1 for finite values of $\pi(\boldsymbol{x})$, i.e. $\pi(\boldsymbol{x}) = \frac{\pi}{2} + k\pi$ for $k \in \mathbb{Z}$, making it sufficient for the preliminary policy to describe the function values in the range of $\pm\pi$. Furthermore, given that the input distribution $\tilde{\pi}(\boldsymbol{x})$ is Gaussian the sinusoid provides analytical computation of the mean and covariance of $\pi(\boldsymbol{x})$. See Appendix A for the derivation of the univariate case.

$$\pi(\boldsymbol{x}) = \boldsymbol{u}_{max} \sin(\tilde{\pi}(\boldsymbol{x})) \in [-\boldsymbol{u}_{max}, \boldsymbol{u}_{max}] \tag{3.3}$$

We prefer the sinusoid over the sigmoid as the sigmoid attains ± 1 in the limits of the policy. This limits the expressibility of the control and makes it harder to differentiate controls. An alternative to squashing the control through a function is to apply control constraints in the cost function thereby implicitly limiting the possible controls.

3.1.2 State Distribution Propagation

In order to obtain $\mathbb{E}_{\tilde{\boldsymbol{x}}}[c(\tilde{\boldsymbol{x}})]$ we have to be able to propagate $p(\boldsymbol{x}_t)$ to $p(\boldsymbol{x}_{t+1})$. Since we have specified a model and a policy we see that the states are all functionally related. This means that we can determine $p(\boldsymbol{x}_{t+1})$ through a composition of function. In addition we can also consider intermediate functions to constrain or augment the control or state space. Give $\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ we want to approximate $p(\boldsymbol{x}_{t+1})$ as a Gaussian with mean $\boldsymbol{\mu}_{t+1}$ and covariance $\boldsymbol{\Sigma}_{t+1}$.

Given an initial distribution p_0 from which we draw out initial state $\boldsymbol{x}_0$ we are interested in determining the distribution the state $\boldsymbol{x}_T$ after T time steps through sequential predictions from our model and control choices from our policy. The predictive distribution, $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$, are given by

$$p(\boldsymbol{x}_t) = \int\int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) p(\boldsymbol{u}_{t-1} \mid \boldsymbol{x}_{t-1}) p(\boldsymbol{x}_{t-1}) d\boldsymbol{x}_{t-1} d\boldsymbol{u}_{t-1} \quad t = 1, \ldots, T \tag{3.4}$$

where the model for the one time-step dynamics f yields the transition probability $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$. Since we model g this computation yields $p(\Delta\boldsymbol{x}_t)$. Thus in order to determine $p(\boldsymbol{x}_t)$ we consider the moment update steps,

$$\begin{aligned}\boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t + \boldsymbol{\mu}_{\Delta t} \\ \boldsymbol{\Sigma}_{t+1} &= \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_{\Delta t} + \mathrm{Cov}[\boldsymbol{x}_t, \Delta \boldsymbol{x}_t] + \mathrm{Cov}[\Delta \boldsymbol{x}_t, \boldsymbol{x}_t]\end{aligned} \tag{3.5}$$

where $\boldsymbol{\mu}_{\Delta t}, \boldsymbol{\Sigma}_{\Delta t}$ are the parameters of the predictive distribution $p(\Delta \boldsymbol{x}_t)$. Specifically we follow the steps below to determine the parameters of each successive Gaussian.

1. The control distribution $p(\boldsymbol{u}_t)$ is computed in two steps.
   (a) Given a Gaussian distribution $p(\boldsymbol{x}_t)$ a Gaussian approximation of the distribution $p(\tilde{\pi}(\boldsymbol{x}_t))$ is computed analytically.
   (b) The preliminary policy is squashed through the sin function $\pi(\boldsymbol{x}_t) = \boldsymbol{u}_{max} \sin(\tilde{\pi}(\boldsymbol{x}_t)$. The moments of the approximate Gaussian through this are provided in Appendix A.
2. The joint distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t) = p(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t))$ is determined since the input of the model are the state-control pairs.
   (a) The joint preliminary policy distribution $p(\boldsymbol{x}_t, \tilde{\pi}(\boldsymbol{x}_t))$ is computed. If the policy is linear this is exactly Gaussian as shown in Eq. 3.2. If we consider a radial basis policy the parameters values are calculated as shown in Eq. 2.16, 2.22, 2.32.
   (b) We compute and approximate fully joint Gaussian distribution $p(\boldsymbol{x}_t, \tilde{\pi}(\boldsymbol{x}_t), \pi(\boldsymbol{x}_t))$ and marginalize out $\tilde{\pi}(\boldsymbol{x}_t)$. The cross-covariance information between $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ is computed through,

$$\mathrm{Cov}[\boldsymbol{x}_t, \boldsymbol{u}_t] = \mathrm{Cov}[\boldsymbol{x}_t, \tilde{\pi}(\boldsymbol{x_t})]\mathrm{Cov}[\tilde{\pi}(\boldsymbol{x_t}), \tilde{\pi}(\boldsymbol{x_t})]^{-1}\mathrm{Cov}[\tilde{\pi}(\boldsymbol{x_t}), \boldsymbol{u}_t]$$

3. Given the approximate joint Gaussian distribution $p(\boldsymbol{x}_t, \boldsymbol{u}_t) = p(\tilde{\boldsymbol{x}}_t)$ the gradient dynamic's distribution $p(\Delta \boldsymbol{x}_t)$ is computed by applying the predictive model computations from Section 2.1.2 or 2.4.1.
4. The successor state distribution $p(\boldsymbol{x}_{t+1})$ is computed following Eqs. 3.5

Given this process we have formalized the *Propagate* function and are able to compute the set of distributions $\{p(\boldsymbol{x}_t)\}_{t=1}^{T}$. Given this set of distributions we can now discuss how to compute $\mathbb{E}_{\tilde{bmx}}[c(\tilde{\boldsymbol{x}}_t)]$. To do this we must specify the cost functions.

3.2 Cost Functions

In order to be able to pose this optimization properly we have to consider an objective function which is differentiable. Within this paper we will discuss two potential cost functions that prove applicable, i.e. a saturating cost function and a quadratic cost function. We choose for the cost functions to encode only a geometric distance metric d rather than incorporating constraints on the control signal and higher state derivatives such as speed or acceleration, i.e. $c(\boldsymbol{x}, \boldsymbol{u}) = c(\boldsymbol{x})$. The argument being that an autonomous agent should learn that high speeds or aggressive controls leads to "overshooting" the goal state therefore causing high cumulative costs to incur. Therefore, rather than computing the expectation with respect to $\tilde{\boldsymbol{x}}$ we are only concerned with $\boldsymbol{x}$. Note though that these immediate cost functions can be easily extended to include other terms.

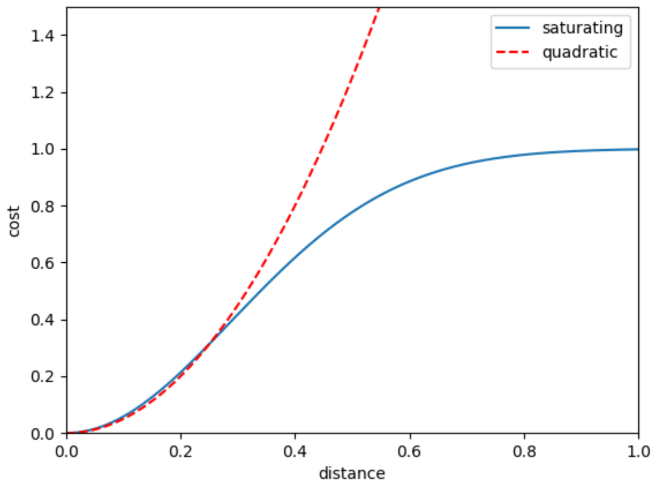

Figure 3.1: The quadratic and saturating cost functions over the interval $[0, 1]$ with scaling for the saturating cost function by $a^2 = 3$ and 5 for the quadratic cost function.

3.2.1 Saturating Cost

We propose a saturating cost function,

$$c(\boldsymbol{x}) \triangleq 1 - \exp\left(-\frac{1}{2a^2} d(\boldsymbol{x}, \boldsymbol{x}_{target})^2\right) \tag{3.6}$$

that is locally exponential and saturates at 1 for large deviations of d from the desired target $\boldsymbol{x}_{target}$. Give that a input distribution of a state is Gaussian, $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we can compute the expected cost,

$$\mathbb{E}_{\boldsymbol{x}}[c(\boldsymbol{x})] = 1 - \int c(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} = 1 - \int \exp\big(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_{target})^T \boldsymbol{T}^{-1}(\boldsymbol{x} - \boldsymbol{x}_{target})\big) p(\boldsymbol{x}) d\boldsymbol{x} \tag{3.7}$$

where $\boldsymbol{T}^{-1} = a^2\boldsymbol{C}^T\boldsymbol{C}$ for a suitable percision matrix $\boldsymbol{C}$. Thus we obtain the expected immediate cost,

$$\begin{gathered}\mathbb{E}_{\boldsymbol{x}}[c(\boldsymbol{x})] = 1 - \det\left(\boldsymbol{I} + \boldsymbol{\Sigma}\boldsymbol{T}^{-1}\right)^{-\frac{1}{2}} \exp(-\frac{1}{2}(\boldsymbol{\mu} - \boldsymbol{x}_{target})^T\tilde{\boldsymbol{S}}_1(\boldsymbol{\mu} - \boldsymbol{x}_{target})) \\ \tilde{\boldsymbol{S}}_1 = \boldsymbol{T}^{-1}(\boldsymbol{I} + \boldsymbol{\Sigma}\boldsymbol{T}^{-1})^{-1}\end{gathered} \tag{3.8}$$

In addition, to computing the expected cost we can also look at the variance of the cost. The second moment $\mathbb{E}_{\boldsymbol{x}}[c(\boldsymbol{x})]$ can be computed analytically and is given as,

$$\begin{gathered}\mathbb{E}_{\boldsymbol{x}}[c(\boldsymbol{x})] = |\boldsymbol{I} + 2\boldsymbol{\Sigma}\boldsymbol{T}^{-1}|^{-1/2} \exp\left(-(\boldsymbol{\mu} - \boldsymbol{x}_{target})^T\tilde{\boldsymbol{S}}_2(\boldsymbol{\mu} - \boldsymbol{x}_{target})\right) \\ \tilde{\boldsymbol{S}}_2 = \boldsymbol{T}^{-1}(\boldsymbol{I} + 2\boldsymbol{\Sigma}\boldsymbol{T}^{-1})^{-1}\end{gathered} \tag{3.9}$$

The variance is determined by subtracting the expected squared cost from the second moment.

Exploration vs. Exploitation

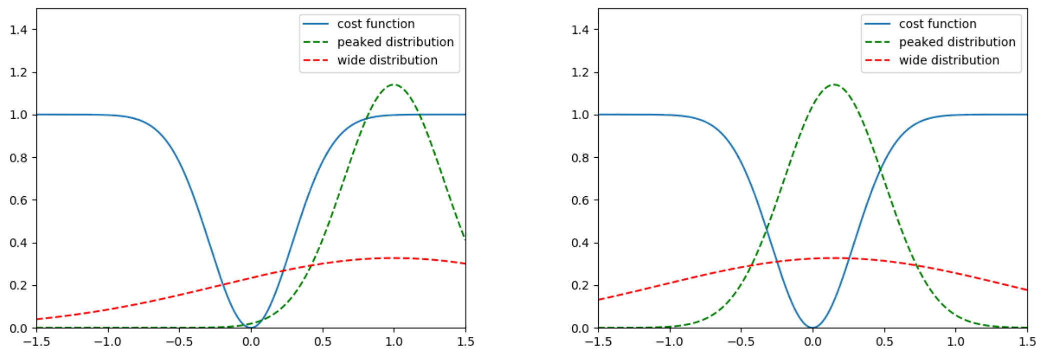

Figure 3.2: Given that the peaked distribution is far away from the target it will incur a high expected cost than the wide distribution as the wide distribution may still cover the target state. Whereas if both are located close to the target the wide distribution will incur a higher cost than the peaked distribution. Thus in the 1^{st} scenario exploration is encouraged while in the 2^{nd} scenario exploitation is encouraged.

An appeal for using this cost function is that it naturally encodes exploration vs. exploitation when a state distribution is Gaussian. If the mean of $p(\boldsymbol{x})$ is far from $\boldsymbol{x}_{target}$, a state distribution that has a large uncertainty to capture the target will be favored more leading to automatic *exploration*. Initially, the state's uncertainty will be due to the model uncertainty. Due to the saturation function making the algorithm be inclined to explore these uncertain regions in the subsequent iteration the model uncertainties will be reduced. It then follows that in the next policy search there would be tighter distributions around the previos uncertain regions. If the mean of $p(\boldsymbol{x})$ is close to $\boldsymbol{x}_{target}$ the algorithm will *exploit* the model knowledge as the majority of the distribution's mass is around low cost regions.

Alternatively, we can induce further exploration by altering the cost function to include a penalty term in regards to the costs' uncertainty. This encourages exploration by seeking to minimize the uncertainty of the cost which embeds some of the state's uncertainty. We consider the new objective function to be,

$$J^{\pi}(\boldsymbol{x}_0) = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] + b\sigma_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] \tag{3.10}$$

where $\sigma_{\boldsymbol{x}_t}$ is the standard deviation of the cost and b is a scaling factor such that $b > 0$ penalizes uncertainty in the cost. We consider the variance of the cost rather modifying the objective with the variance of the state as MacKay [41] has show that incorporating state uncertainty in the objective leads to extreme design choices. We want to approach regions of the state space that are encouraged as by J^{π} and c which leads to considering the variance of the predicted cost instead. Since the behavior of the predicted cost's distribution isn't one-to-one with the behavior of the states distribution this avoids extreme decision choices by the policy.

Partial Derivatives

For the policy search we must determine the partial derivatives of the expected cost with respect to the state distribution $p(\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ for solving Eq. **??**. These partial derivatives are given by

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\mu}} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] &= -\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)](\boldsymbol{\mu}_t - \boldsymbol{x}_{target})^T \tilde{\boldsymbol{S}}_1 \\ \frac{\partial}{\partial \boldsymbol{\Sigma}} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] &= \frac{1}{2} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)](\tilde{\boldsymbol{S}}_1(\boldsymbol{\mu}_t - \boldsymbol{x}_{target})(\boldsymbol{\mu}_t - \boldsymbol{x}_{target})^T - \boldsymbol{I})\tilde{\boldsymbol{S}}_1 \end{aligned} \tag{3.11}$$

where $\tilde{\boldsymbol{S}}_1$ is given by Eq. 3.8.

3.2.2 Quadratic Cost

A common cost function used in optimal control theory is that of quadratic cost,

$$c(\boldsymbol{x}_t) = a^2 d(\boldsymbol{x}_t, \boldsymbol{x}_{target})^2 \tag{3.12}$$

where a is the precision width. Considering the general form of the quadratic cost we can derive the expected immediate cost,

$$\begin{aligned} c(\boldsymbol{x}_t) &\triangleq (\boldsymbol{x} - \boldsymbol{x}_{target})^T \boldsymbol{T}^{-1} (\boldsymbol{x}_t - \boldsymbol{x}_{target}) \\ \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] &= \operatorname{tr}(\boldsymbol{\Sigma}\boldsymbol{T}^{-1}) + (\boldsymbol{\mu} - \boldsymbol{x}_{target})^T \boldsymbol{T}^{-1} (\boldsymbol{\mu} - \boldsymbol{x}_{target}) \\ \operatorname{var}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] &= \operatorname{tr}(2\boldsymbol{T}^{-1}\boldsymbol{\Sigma}\boldsymbol{T}^{-1}\boldsymbol{\Sigma}) + 4(\boldsymbol{\mu} - \boldsymbol{x}_{target})^T \boldsymbol{T}^{-1}\boldsymbol{\Sigma}\boldsymbol{T}^{-1} (\boldsymbol{\mu} - \boldsymbol{x}_{target}) \end{aligned} \tag{3.13}$$

where $\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{T}$ is the symmetric precision matrix that includes the scaling parameter a.

In contrast to the saturating cost function the quadratic cost does not embed exploration vs. exploitation naturally. This is noticed through the fact that the variance of the cost increases both due to the variance of the state distribution or if the mean of the state distribution is far away from the target state.

Partial Derivatives

Given that $\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we derive the partial derivatives of the cost function to be,

$$\begin{gathered}\frac{\partial}{\partial \boldsymbol{\mu}} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] = 2(\boldsymbol{\mu} - \boldsymbol{x}_{target})^T \boldsymbol{T}^{-1} \\ \frac{\partial}{\partial \boldsymbol{\Sigma}} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] = \boldsymbol{T}^{-1}\end{gathered} \tag{3.14}$$

Additionally, if we consider the altered objective function in Eq. **??** then the partials for the variance are provided as,

$$\begin{gathered}\frac{\partial}{\partial \boldsymbol{\mu}} \text{var}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] = -8\boldsymbol{T}^{-1}\boldsymbol{\Sigma}\boldsymbol{T}^{-1}(\boldsymbol{\mu} - \boldsymbol{x}_{target})^T \\ \frac{\partial}{\partial \boldsymbol{\Sigma}} \text{var}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)] = 4\boldsymbol{T}^{-1}\boldsymbol{\Sigma}\boldsymbol{T}^{-1} + 4\boldsymbol{T}^{-1}(\boldsymbol{\mu} - \boldsymbol{x}_{target})(\boldsymbol{T}^{-1}(\boldsymbol{\mu} - \boldsymbol{x}_{target}))^T\end{gathered} \tag{3.15}$$

We have now fully describe the forward propagation process of the *PolicySearch* function in consideration of a model $\mathcal{M}$, policy $\pi_{\boldsymbol{\psi}}$, and immediate cost $c(\boldsymbol{x})$. Specifically, we have specified the functions *ExpectedCost* and part of the *CostGradient* function. The final portion of this function is the objective evaluation and full gradient computation.

3.3 Policy Gradient Optimization

We can now propagate distributions to our time horizon T and estimate the expected cumulative cost. Since the horizon is finite the value function will not be equivalent to the underlying true value function. This leads to extensions where you can consider N trajectories starting from different initial state distributions $p(\boldsymbol{x}_0^{(i)})$ through using the sample mean,

$$\frac{1}{N} \sum_{i=1}^{N} J^{\pi}(\boldsymbol{x}_0^{(i)}) \tag{3.16}$$

For this paper, we will consider the single initial state distribution and use a gradient based policy search method. Specifically, we are interested in finding a parameterized policy $\pi^*_{\boldsymbol{\psi}}$ from a class of policies Π with,

$$\pi^*_{\boldsymbol{\psi}} \in \arg\min_{\pi \in \Pi} J^{\pi}(\boldsymbol{x}_0) \tag{3.17}$$

Here through parameterizing π we are constraining the searchable policy space. This generally leads to suboptimal policies, but depending on the expressiveness of Π the policies found can have similar expected cumulative costs $J^{\pi}(\boldsymbol{x}_0)$ as the globally optimal policy π^*. This optimization requires us to find the derivatives with respect to the policy parameters ψ. Given that $\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, $\boldsymbol{u}_t \sim \mathcal{N}(\boldsymbol{\mu}^u_t, \boldsymbol{\Sigma}^u_t)$, and $\tilde{\boldsymbol{x}}_t \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)$ and all the distributions are functionally related we can

$$\nabla_{\boldsymbol{\psi}} J^{\pi}(\boldsymbol{x}_0) = \sum_{t=0}^{T} \nabla_{\boldsymbol{\psi}} \mathbb{E}_{x_t,u_t}[c_t(\boldsymbol{x}_t, \boldsymbol{u}_t)] \tag{3.18}$$

$$\nabla_{\boldsymbol{\psi}} \mathbb{E}_{x_t}[c_t(\boldsymbol{x}_t, \boldsymbol{u}_t)] = \frac{\partial \mathbb{E}_{x_t}[c_t(\boldsymbol{x}_t, \boldsymbol{u}_t)]}{\partial \boldsymbol{\mu}_t} \nabla_{\boldsymbol{\psi}} \boldsymbol{\mu}_t + \frac{\partial \mathbb{E}_{x_t}[c_t(\boldsymbol{x}_t, \boldsymbol{u}_t)]}{\partial \boldsymbol{\Sigma}_t} \nabla_{\boldsymbol{\psi}} \boldsymbol{\Sigma}_t \tag{3.19}$$

$\frac{\partial \mathbb{E}_{x_t}[c_t(\boldsymbol{x}_t,\boldsymbol{u}_t)]}{\partial \boldsymbol{\mu}_t}$ and $\frac{\partial \mathbb{E}_{x_t}[c_t(\boldsymbol{x}_t,\boldsymbol{u}_t)]}{\partial \boldsymbol{\Sigma}_t}$ are dependent on the cost function definition and are shown in Eq. 3.11, 3.14. Then what remains to be determined are $\nabla_{\boldsymbol{\psi}} \boldsymbol{\mu}_t$ and $\nabla_{\boldsymbol{\psi}} \boldsymbol{\Sigma}_t$.

$$\nabla_{\boldsymbol{\psi}} \boldsymbol{\mu}_t = \frac{\partial \boldsymbol{\mu}_t}{\partial \tilde{\boldsymbol{\mu}}_{t-1}} \nabla_{\boldsymbol{\psi}} \tilde{\boldsymbol{\mu}}_{t-1} + \frac{\partial \boldsymbol{\mu}_t}{\partial \tilde{\boldsymbol{\Sigma}}_{t-1}} \nabla_{\boldsymbol{\psi}} \tilde{\boldsymbol{\Sigma}}_{t-1} + \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\psi}}, \quad \nabla_{\boldsymbol{\psi}} \boldsymbol{\Sigma}_t = \frac{\partial \boldsymbol{\Sigma}_t}{\partial \tilde{\boldsymbol{\mu}}_{t-1}} \nabla_{\boldsymbol{\psi}} \tilde{\boldsymbol{\mu}}_{t-1} + \frac{\partial \boldsymbol{\Sigma}_t}{\partial \tilde{\boldsymbol{\Sigma}}_{t-1}} \nabla_{\boldsymbol{\psi}} \tilde{\boldsymbol{\Sigma}}_{t-1} + \frac{\partial \boldsymbol{\Sigma}_t}{\partial \boldsymbol{\psi}} \tag{3.20}$$

Recall from Eqs. 3.5 that we can represent $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ in terms of of the models predictions and the previous state. We can assume $\nabla_{\boldsymbol{\psi}} \tilde{\boldsymbol{\mu}}_{t-1}, \nabla_{\boldsymbol{\psi}} \tilde{\boldsymbol{\Sigma}}_{t-1}$ as known since we start with a known state distribution independent of $\boldsymbol{\psi}$ and subsequentially can compute $\nabla_{\boldsymbol{\psi}} \boldsymbol{\mu}^u_t, \nabla_{\boldsymbol{\psi}} \boldsymbol{\Sigma}^u_t$ analytically.

Proposition 3. *Given that $\mathcal{M}$ is a BNN as formulated in Section 2.4.1 we can compute $\frac{\partial \boldsymbol{\mu}_t}{\partial \tilde{\boldsymbol{\mu}}_{t-1}}$, $\frac{\partial \boldsymbol{\mu}_t}{\partial \tilde{\boldsymbol{\Sigma}}_{t-1}}$, $\frac{\partial \boldsymbol{\Sigma}_t}{\partial \tilde{\boldsymbol{\mu}}_{t-1}}$, $\frac{\partial \boldsymbol{\Sigma}_t}{\partial \tilde{\boldsymbol{\Sigma}}_{t-1}}$ through standard backpropagation.*

Given that the distributions are functionally related it implies that the gradients can be computed through a repeated use of the chain rule. We expand the partials $\frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\psi}}$, $\frac{\partial \boldsymbol{\Sigma}_t}{\partial \boldsymbol{\psi}}$ once as shown in Eq. 3.21. Since we consider $\boldsymbol{u}_t$ to be bounded through the sin function we would have to expand $\frac{\partial \boldsymbol{\mu}^u_{t-1}}{\partial \boldsymbol{\psi}}$, $\frac{\partial \boldsymbol{\Sigma}^u_{t-1}}{\partial \boldsymbol{\psi}}$ once more based on the parameters of the preliminary policy distribution, $\boldsymbol{\mu}^{\tilde{u}}_{t-1}$ and $\boldsymbol{\Sigma}^{\tilde{u}}_{t-1}$.

$$\frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\psi}} = \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\mu}^u_{t-1}} \frac{\partial \boldsymbol{\mu}^u_{t-1}}{\partial \boldsymbol{\psi}} + \frac{\partial \boldsymbol{\mu}_t}{\partial \boldsymbol{\Sigma}^u_{t-1}} \frac{\partial \boldsymbol{\Sigma}^u_{t-1}}{\partial \boldsymbol{\psi}}, \quad \frac{\partial \boldsymbol{\Sigma}_t}{\partial \boldsymbol{\psi}} = \frac{\partial \boldsymbol{\Sigma}_t}{\partial \boldsymbol{\mu}^u_{t-1}} \frac{\partial \boldsymbol{\mu}^u_{t-1}}{\partial \boldsymbol{\psi}} + \frac{\partial \boldsymbol{\Sigma}_t}{\partial \boldsymbol{\Sigma}^u_{t-1}} \frac{\partial \boldsymbol{\Sigma}^u_{t-1}}{\partial \boldsymbol{\psi}} \tag{3.21}$$

3.4 Evaluation

We have covered how to perform the *PolicySearch* with regards to propagating the gradients of the model forward through time. The final concern for the algorithm is how to evaluate the policy and determine that continual improvements will have diminishing returns. There are several ways in which we can evaluate the policy after each update. Specifically, we can set a fixed amount of iterations over with we evaluate the policy or we can dynamically adjust the number of iterations of the algorithm through assessing the improvement relative to the last policy. We will formulate both scenarios in this section but only consider the fixed setting during our evaluation. We also formulated this problem as a finite horizon control problem which leads to the addition consideration of extending the time horizon to determine the capablities of the model.

3.4.1 Fixed Length Evaluation

Algorithm 3 Fixed Length Evaluation

```
Inputs: $\pi_{\boldsymbol{\psi}}, \mathcal{M}, p_0, c, T, P$
$\boldsymbol{x}_{target} \leftarrow Target(c)$
for $t = 1, \ldots T$ do)
    $p(\boldsymbol{x}_t) = Propagate(\mathcal{M}, \pi_{\boldsymbol{\psi}}, p(\boldsymbol{x}_{t-1}))$
    $\kappa_t \leftarrow ExpectedCost(c, p(\tilde{\boldsymbol{x}}_t))$
if $TargetReached(\boldsymbol{x}_{target}, \{p(\boldsymbol{x}_t)\}_{t=1}^{T})$ then
    $T \leftarrow T + \frac{1}{2}T$
$J^{\pi} = \sum_{t=1}^{T} \kappa_t$
Done $\leftarrow CachedIter(P)$
return $T, J^{\pi}$, Done
```

In this algorithm we see that the *Evaluation* function determines whether or not the agent is done developing a policy based on a cached integer that records the amount of iterations that have occurred. if this integer reaches the specified amount of iterations P then the agent is done. In addition, the *Evaluation* function updates the time horizon based on the agent's intermediate performance. Specifically, if the evaluation block has deem that the agent has reached the goal state prior to the termination of the current time horizon then the new time horizon will be increased by half of the current horizon. This allows for further improvements of the policy because it may be that the current policy with an extended horizon would over-shoot the goal state and incur a higher cumulative cost. The formulation of the general sequential reinforcement learning algorithm is written in regards to a dynamic evaluation and thus would need the slight modification of specifying P as an input. Alternatively the *TargetReached* function could specify reaching a goal region or distribution $p(\boldsymbol{x}_{target})$.

3.4.2 Dynamic Evaluation

Algorithm 4 Dynamic Evaluation

```
**Inputs:** $\pi_{\boldsymbol{\psi}}, \mathcal{M}, p_0, c, T, J^{\pi-}$
Done ← False
$\boldsymbol{x}_{target} \leftarrow Target(c)$
**for** $t = 1, \ldots T$ **do**
    $p(\boldsymbol{x}_t) = Propagate(\mathcal{M}, \pi_{\boldsymbol{\psi}}, p(\boldsymbol{x}_{t-1}))$
    $\kappa_t \leftarrow ExpectedCost(c, p(\tilde{\boldsymbol{x}}_t))$
**if** $TargetReached(\boldsymbol{x}_{target}, \{p(\boldsymbol{x}_t)\}_{t=1}^{T})$ **then**
    $T \leftarrow T + \frac{1}{2}T$
$J^{\pi} = \sum_{t=1}^{T} \kappa_t$
**if** $|J^{\pi} - J^{\pi-}| < \epsilon_{eval}$ **then**
    Done ← True
**return** $T, J^{\pi}$, Done
```

Here the dynamic evaluation considers the difference between cumulative cost with respect to a new policy *and* model. This differs from the policy search where the model is considered fixed. Thus the dynamic evaluation is equivalent to an agent who is trying to stabalize both their policy and model. The agent considers these to be stable once it detects a diminishing return for learning more regarding the model. Note that there are other ways to update the time horizon based on when the agent reached the goal. An alternative means of of evaluating whether the agent has learned a good policy and model is basd on the time horizon. If the time horizon reaches a value that is considered ”long” for the environment then the agent has successfully learned a policy that will perform well. With this we have fully specified the reinforcement learning algorithm that exploits the model derivatives to guide the policy search. We conclude the chapter with a literature review regarding model-based reinforcement learning.

3.5 Model-Based Reinforcement Learning Related Works

The appeal of being sample efficient with regard to the environment makes modeling the environment dynamics of interest. However, the concern for mitigating the effects of model bias leads to various approaches. Sutton [61] provides an initial framework for incorporating learning & planning together which spurred about a class of model-based plus model-free reinforcement learning algorithms that attempt to utilize the benefits of both approaches. Alternatively, algorithms that attempt to do a policy search through the model directly exploit the model gradients to accurately update the policy. The class of shooting algorithm approachs the model learning through approximately solving the receding horizon problem posed in model predictive control (MPC) [62].

Dyna-Styled Algorithms

Sutton [61] introduces the Dyna framework which interleaves planning and learning sequentially and extends it [63], [64] to consider a policy iteration, Q-learning, and iterative approximate dynamic programming version. Specifically this class of algorithms considers learning the model dynamics and using them to assist in learning policies through model-free based aglorithms. Depeweg et al. [65] use a BNN to learn a distribution over the dynamics model and perform gradient based policy optimization over a collection of models sampled from the distribution. Mishra et al. [66] learn a latent variable dynamic model over temporally extended segments of the trajectory, and perform gradient based policy optimization over the latent space. Their restriction being that the dataset must be generated prior to the initiation of the algorithm limiting them to environments where random exploration is sufficient. In an iterative procedure of model learning and policy optimization Kurutach et al. [67] employ an ensemble of models to combat model bias through which they generate fictious samples and use Trust Region Policy Optimization (TRPO) [68] to update the policy. In addition to that Clavera et al. [69] consider meta-learning a policy where each of the models is considered a different meta-task to train a deep network on. This makes the policy highly adaptable to varying dynamics. Rather than focusing on tackling model bias Luo et al. [70] focus on provide a framework with theoretical guarantees of monotone improvement of the policy. In contrast to the previous works they also rely only on a single network to model the dynamics.

Shooting Algorithms

These algorithms approximately solve the receeding horizon problem presented in MPC when concerned with non-linear dynamics and non-convex cost functions. Richards [71] presents variations of MPC that can handle state constraints through comparing a set of K candidate policies drawn from a uniform distribution. These candidates are evaluate through the learned dynamics and then the first control of the optimal policy is taken. Nagabandi et al. [72] extend this by learning the optimal candidate policy and distilling it into neural network that minimizes the KL divergence between the two policies. Finally, the policy is fine-tuned using TRPO. Chua et al. [73] consider an ensemble of probabilistic models to combat the bias through uncertainty incorporation and employ MPC as described by Richards.

Policy Search with Backpropagation in Time

Unnlike the Dyna-Styled Algorithms policy search algorithms that employ backpropagation through time (BPTT) rely on model derivatives to employ gradient based policy optimization. Mishra et. al [66] employ deep generative models over temporal segments of state-action pairs in an end-to-end fully differentiable policy optimization. Heess et al. [74] extend differentiable policy optimization to stochastic policies in which they employ the model only for gradient computations and not prediction. Diesenroth & Rasmussen [**?**] use GPs as the model and iterate between collecting data given the policy and policy optimization. Gal et al. [75] improve upon this by substituting the GPs with BNNs that employ Drop-Out as an approximate variational inference method. This requires sample from the network distribution to produce uncertainty estimates. Further results by Osband [58] challenge the efficacy of this approach. Sanket & Deisenroth [76] use GPs in conjunction with probabilistic model predictive control (MPC) to formulate a deterministic optimal control problem and exploit Pontryagin's minimum principle to handle state constraints. Levine & Abbeel [77] consider locally linear time-varying models with which they learn Gaussian-like controllers modeled as a neural network. The policy search is guided by augmenting the reward to reduce the deviation from an iterative Linear Quadratic Gaussian (iLQG) controller. Montgomery & Levine [78] improve upon the initial formulation of guided policy search (GPS) through the mirror descent perspective allowing them to formulate a simplified variant which corresponds to mirror descent under linear dynamics and convex policy spaces.

This concludes our formulation of a reinforcement learning algorithm that employs the model gradients for policy optimization. We presented how to encorporate the discussed models into the framework and specified various parameterizations of the policy. We conclude this paper with the final chapter presenting some preliminary results regarding the various formulations of the RL algorithm on some simple environments.

Chapter 4

Assessment

Given this propagation based reinforcement learning algorithm we can assess the performance of the different models in guiding the policy to a region of good parameters. There are several metrics with which we are concerned regarding the models. The main concern is determining the capacity of the model to guide the policy search in a principled and stable manner. This can be quantified as the accuracy of the model in representing the true unknown dynamics, how the model uncertainty changes over time given more data, and the models capability of producing estimates in regions of low information. To observe these various quantities analytically we first provide a policy free assessment of the models on synthetic data. Following this model assessment we introduce the policy optimization as a metric of model performance in the reinforcement learning framework. This chapter is structured as following; we introduce a dynamic system on which we learn our two models followed by a brief discussion on the model implementation details. Given the dynamics we measure the predictive accuracy and variance of each model based on different dataset sizes. We then present learning a policy and discuss a few implementation details regarding the models and policies. We observe the final policy results and use them as a metric to assess the model's capacity for capturing desired qualities of the true dynamics.

4.1 Algorithm Implementation

Here we will discuss the practical implementations of both of the models. The GP and BNN are learned using an Intel i7-6700HQ CPU and Nvidia GT960M GPU, respectively. The gradients of the models are computed using Automatic Differentiation from autograd and validated with the analytical derivations. The Gaussian Process is implemented in python using the Scikit-learn machine learning library [79]. For the GP we consider the noise variance σ_ϵ and scale α to be fixed and only learn the matrix Λ. Additionally, we consider Λ to be a diagonal matrix thereby implementing automatic relevance determination (ARD). The hyper-parameters are learned using the LBFGS algorithm. The Bayesian neural network using PBP is implemented in Tensorflow [80] while the hyper-parameters are learned using the PBP algorithm. We consider the structure of the network to be 4 layers with 50 hidden units each. We learn the parameters over 350 epochs and re-initialize the network after each iteration. We do not warm start the network with the previously learned weights as this degrades the performance of the model. The environment is implemented using the Sci-Py ODE solver with a zero-order controller. The policy is learned using the LFBGS optimizer from the Sci-Py library [81] given the gradients from autograd. We use the fixed horizon evaluation scheme with $P = 10$, a lookahead time of 4 seconds and a time discretization of $dt = .1$; leading to a planning horizon of $T = 40$. The intial distribution p_0 is considered to be Gaussian with zero mean and covariance matrix $\sigma_0^2 \boldsymbol{I}$ where $\sigma_0^2 = 1 \times 10^{-3}$. The radial basis policy has $N = 20$ basis functions where the targets $\boldsymbol{y}_\pi$ are initialized from a Gaussian with zero mean and σ_ϵ standard deviation.

4.1.1 Environment

We test the models on the pendulum environment. Suppose that the pendulum has mass m and length l and the angle ϕ is measured anti-clockwise from the hanging down position. A torque control u can be applied to the pendulum. We can derive the equations of motion using the system Lagrangian L being the difference between the kinetic energy and potential energy. Specifically, we determine the set of ordinary differential equations (ODE) to be,

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \frac{u - bx_1 - \frac{1}{2}mlg\sin(x_2)}{\frac{1}{4}ml^2 + I} \\ x_1 \end{bmatrix} \tag{4.1}$$

where $g = 9.81 \frac{m}{s^2}$ is gravity and I is the moment of inertia. Typically $m = 1$ kg and $l = 1$ m.

4.2 Model Evaluation

We first consider the models independent of the policy optimization procedure. Specifically, we provide some rollout data to evaluate the performance of the models. Since these are probabilistic models we are concerned with more than just the accuracy. Given that the model is in a region with low information we should see it's uncertainty in regards to the proper output react accordingly. While the models should not produce uniform uncertainty in such regions they should try to capture as much of the true uncertainty as possible. In our consideration we only consider the radial basis controller and compare the two models presented. We compare the model's accuracy results in relation the ground truth dynamics. Specifically, we observe the models accuracy in terms of approaching the ground truth phase portrait for fixed control. To generate these models we apply an optimal policy learned with the ground truth dynamics. In Fig. 4.1 we observe the phase portrait of the pendulum when no control is applied. We use this as our comparison metric as it demonstrates how well the model understands the intrinsic dynamics of the system without having it modulated by a control signal. In Fig. 4.2, 4.4 we can observe the models learned after 1 evaluation and 10 evaluations; while in Fig. 4.3, 4.5 we can observe the error between the ground truth and learned models.

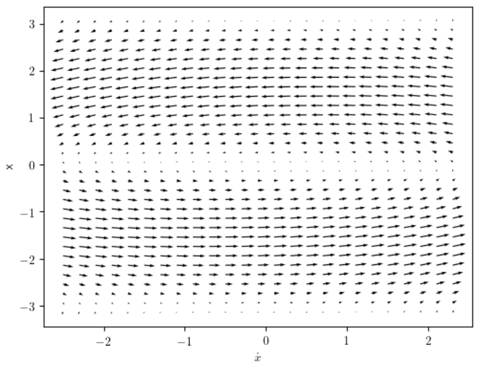

Figure 4.1: True Dynamics Phase Portrait with $u = 0$

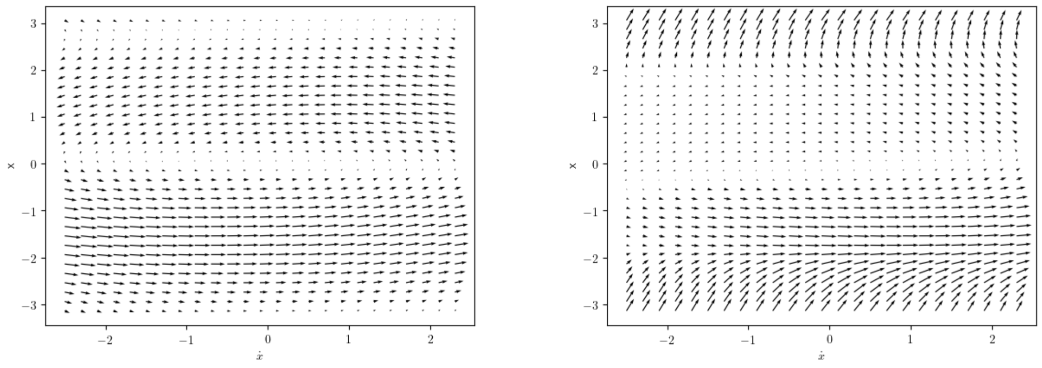

Figure 4.2: Sample GP & BNN Phase Portraits at $P = 1$

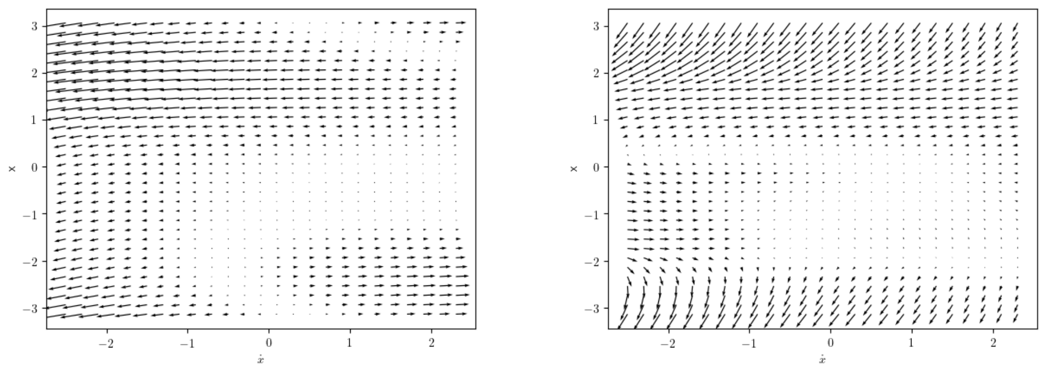

Figure 4.3: Sample GP & BNN Error Phase Portraits at $P = 1$

After one evaluation of the models the GP model demonstrates a good understanding of the dynamics. The BNN model shows a less clear understanding of the model. Both model shows the most error in regions of the state-space where the controller visits less likely. Although the BNN model shows more error over a larger subspace of the state-space.

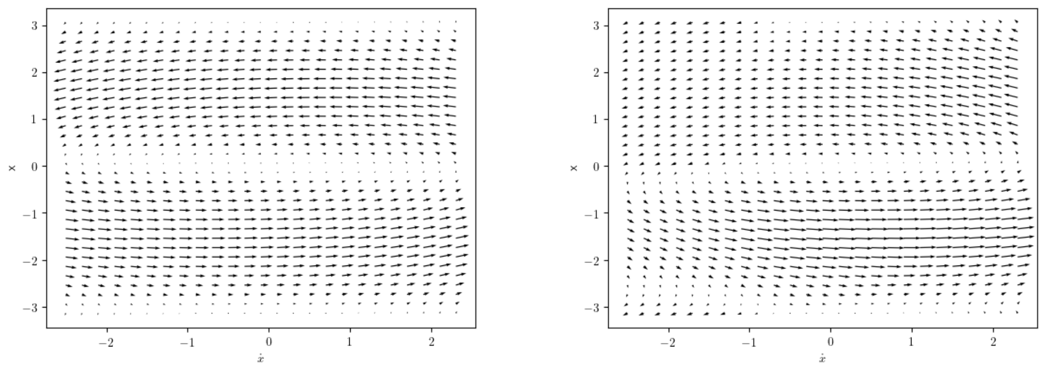

Figure 4.4: Sample GP & BNN Phase Portraits at $P = 10$

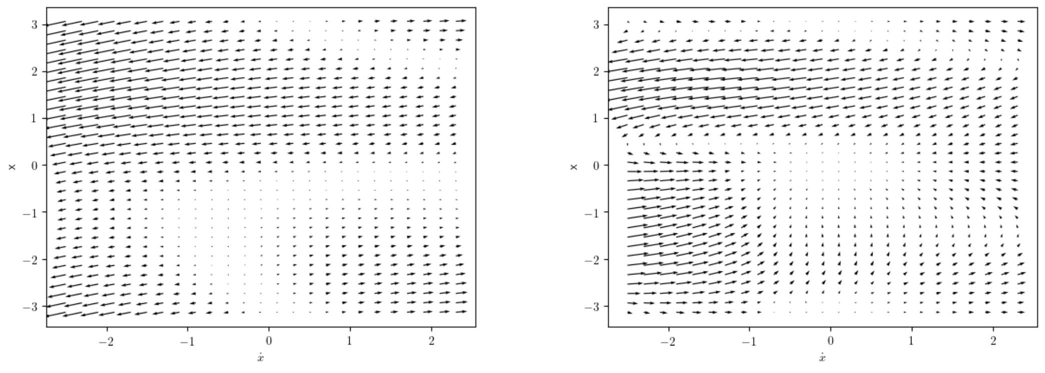

Figure 4.5: Sample GP & BNN Error Phase Portraits at $P = 10$

After 10 evaluations we see that the GP model has learned a model that is relatively accurate with regards to the true model dynamics. Again the errors are presented in regions of the state-space that are less explored. The BNN model reduces the error by trying to aligning it's estimates in the same direction of the ground truth gradients. We observe that over the whole state-space there is more error in the BNN model versus the GP model.

Table 4.1 shows the average error and standard deviation of the gradients in the x-axis and the y-axis over 100 trials. While the models are comparable in performance during the 1^{st} trial we see that at $P = 10$ the GP model has approached the true dynamics model more closely and reduced it's uncertainty much more than the BNN model. Note that the BNN model has a lot more variation than the GP model and even after 10 trials the model has difficulty decreasing it's uncertainty about the model behavior.

Table 4.1: Mean and standard deviation model gradient error averaged over 100 trials

	$\bar{x}_{error}$	$\bar{y}_{error}$	σ_x	σ_y
GP (P = 1)	-0.2175	0.0038	0.2386	0.0309
GP (P = 10)	-0.0219	-0.0035	0.0327	0.0057
BNN (P = 1)	0.1780	0.0115	0.7616	0.1064
BNN (P = 10)	-0.0828	-0.0116	0.6141	0.0935

Aside from the accuracy of the models we can observe their confidence. To do this we consider looking at the determinant of the covariance matrix. This metric can be considered as quantifying the scalar amount of uncertainty along each state. Given that the dynamics require a state-control pair as an input we consider fixed controls while observing the state space.

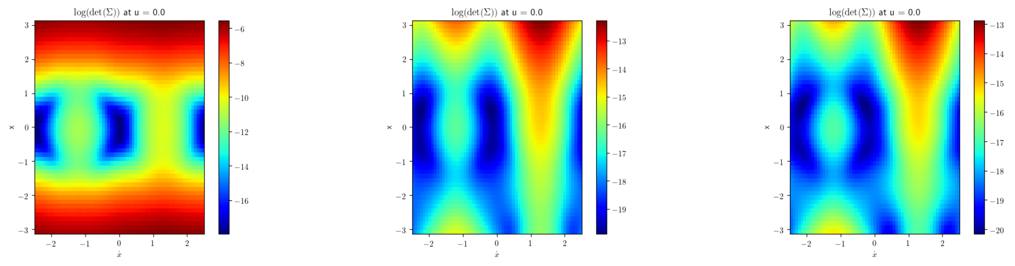

Figure 4.6: $\log(\det(\boldsymbol{\Sigma}))$ of GP model at $P = 1, 5, 10$ with $u = 0$

In Fig. 4.6 we observe the uncertainty of the model over iterations given no control input. Initially, the model is uncertain about all states except for those near the origin and at the extremes of the velocity. With each iteration it becomes more certain about more of the state space as it is exploring more of the states through the cost function. Even though the model is not fully certain about the dynamics of the model over the whole state space it is certain about regions where it has been taken by the policy. This is a good indication of how the model reduces it's local uncertainty over time. We

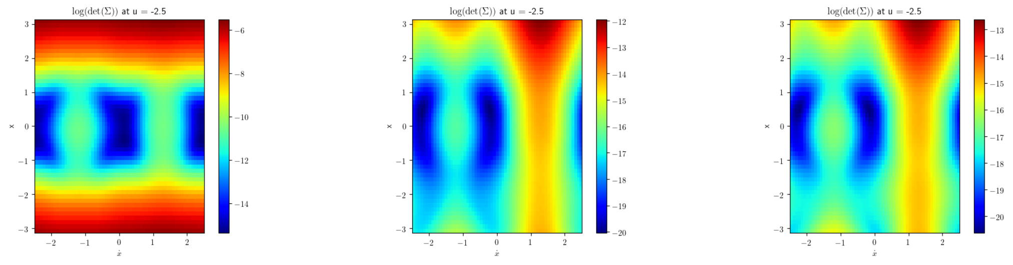

Figure 4.7: $\log(\det(\mathbf{\Sigma}))$ of GP model at $P = 1, 5, 10$ with $u = -2.5$

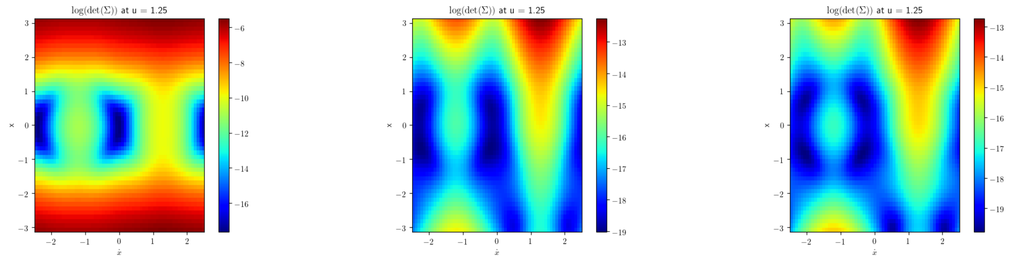

Figure 4.8: $\log(\det(\mathbf{\Sigma}))$ of GP model at $P = 1, 5, 10$ with $u = 1.25$

We present the uncertainty at different control inputs as well to show that the variation of the uncertainty is relatively agnostic of the control input and thereby learning the dynamics directly. Given that the model visits extreme regions of the state space less frequently it is understandable that the uncertainty in Fig. 4.7 is higher than the other control regions.

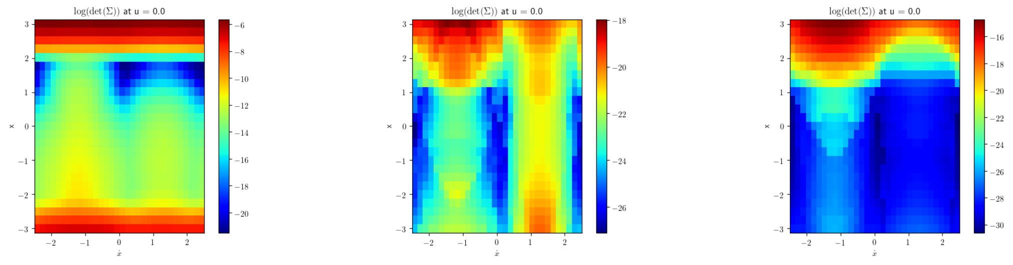

Figure 4.9: $\log(\det(\mathbf{\Sigma}))$ of BNN model at $P = 1, 5, 10$ with $u = 0$

In Fig. 4.9 we can observe the BNN's uncertainty over iterations with no control input. The uncertainty of the model fluctuates largely over iterations and the model becomes uncertaint in regions where it was previously certain. In general, the uncertainty of the model is significantly smaller than that of the GP model and indicates that there is relatively little uncertainty in the BNN model. Precisely, for no control input we can see that after the final iteration the BNN model's uncertainty over the state space is generally in the range of $[-26, -30]$. While for the GP model the uncertainty over the state space is generally in the range of $[-15, -20]$. The largest uncertainty for the BNN model is -15 which means that the model must be more certain than the GP model at all times. This is due to the independence assumption being made in the PBP formulation causing the model to consider there to be less inherent uncertainty in the environment. Therefore in order for the model to better capture the uncertainty within the environment less assumptions regarding the structure need to be made. Furthermore the difference in the models uncertainty indicates that the BNN model is not capturing all of the information provided.

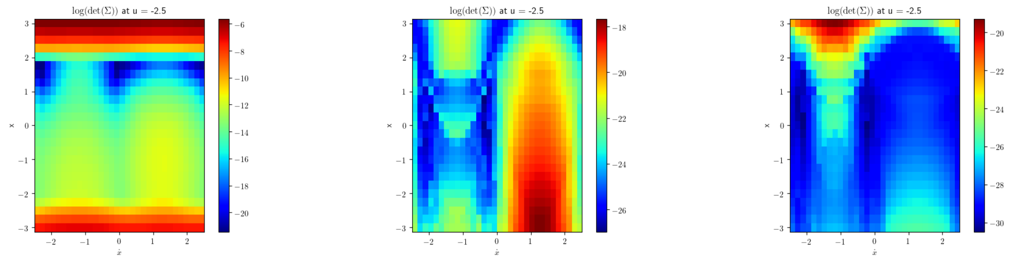

Figure 4.10: $\log(\det(\mathbf{\Sigma}))$ of BNN model at $P = 1, 5, 10$ with $u = -2.5$

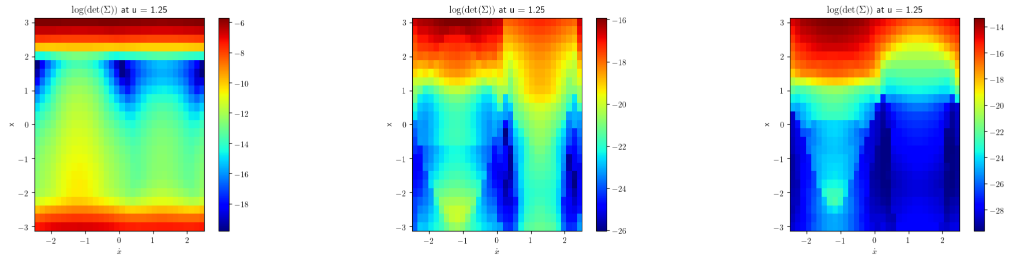

Figure 4.11: $\log(\det(\mathbf{\Sigma}))$ of BNN model at $P = 1, 5, 10$ with $u = 1.25$

We again present the uncertainty at different control inputs to show that the model behavior is generally consistent across the different controls. In contrast with the GP model we see that the BNN model displays a larger uncertainty with different control inputs.

4.3 Optimal Policy through Model

Given the two models presented we can also consider looking at how the policy performs with respect to each model. We evaluate the policy performance over 100 trials for each model and record the rate of success, minimum average time of task completion, and average cumulative cost. In Table 4.2 we see that the agent with a GP model successfully learns a policy each time and requires less that half the time compared to the agent with a BNN model. In addition the cost of the agent with the GP model is much less than the cost of the agent with the BNN model. This is further indicative of the BNN model not being expressive enough with it's uncertainty. Additionally, the BNN model is unable to successfully learn a policy consistently. As the model is generally overconfident in it's understanding of the state-space it causes the policy optimization to follow incorrect paths as gradients along *actually uncertain* regions may be larger than gradients along *certain* regions.

Table 4.2: Policy Performance per Model

	Success Rate	$\bar{T}$	$\bar{J}^{\pi}$
GP	1.0	8.19 s	15.02
BNN	0.74	24.21 s	23.76

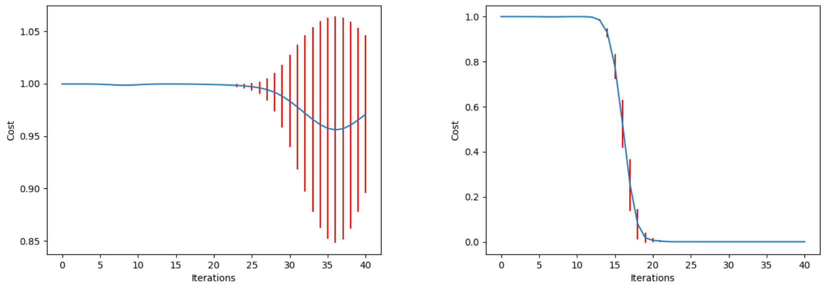

Figure 4.12: Sample Initial & Final Cost Trajectory with GP model

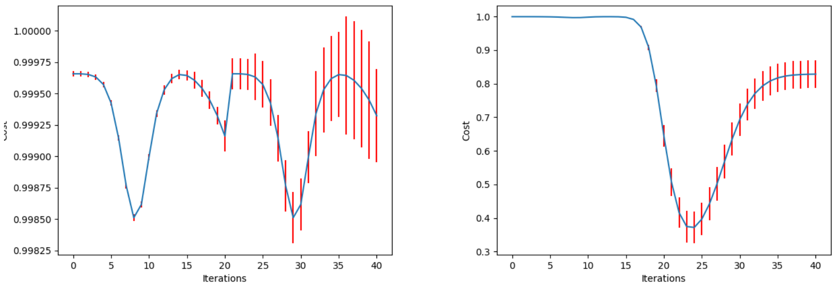

Figure 4.13: Sample Initial & Final Cost Trajectory with PBP model

We provide sample cost trajectories given the different models to highlight concerns for the model that need improvement. In Fig. 4.12 we observe the initial cost trajectory of the GP model and see that it is a flat line with the uncertainty compounding over time. This is natural as the model does not strongly reflect any of the environment's behaviors. In the final trajectory of the cost though we see that the policy perform's smoothly in decreasing the cost and also maintains a high certainty about the cost. This is indicative of the model understanding the environment's local behavior well. In comparison to the GP model, the BNN model also exhibits a flat behavior for the initial cost trajectory; although, the uncertainty throughout the cost is significantly less than with the GP model. This is again due to the expressiveness of the model in comparison to the GP model. We provide a sample final trajectory where the PBP model does not succeed in learning a successful policy. Similar to the agent with the GP model the policy starts the same and starts to decrease the cost but it underestimates the torque needed to reach the goal and losses momentum when approaching the goal causing it to fall back. We also see that the uncertainty of the cost is relatively constant throughout time which is inline with the almost neglible uncertainty that the model provides about the state-space.

4.4 Conclusion

We observe that the GP model performs better than an implementation of a BNN model using PBP. This is due to the strict independence assumption on the output and parameters of the BNN model. These assumptions reduce the expressiveness of the model in terms of capturing uncertainty about the model accuracy. Given that the outputs are independent of each other the BNN model further reduces the capabilities of the agent to learning a good policy.

4.5 Future Work

For future considerations we still believe that there are more expressive models that will allow a model-based reinforcement learning framework to scale to higher dimensions. Given that model-based reinforcement learning is more sample efficient this will allow for applications to real world scenarios and robots. Recent advances of hardware have reduced the complexity of models such as the GP or CP and allow for GP models without approximation to learn on datasets with millions of samples. This affords the use of more complex models such as the CP which removes assumptions regarding the outputs' independence and thereby captures more information regarding the test environments. The naive implementation of PBP that was used can be extended to consider row or column dependencies amount the parameters of the neural network capturing more structural information in the model. Thus rather than considering PBP an extension of this work could prove promising in performing better than a GP. Alternatively, new variational inference methods for determining the posterior of a BNN have been developped and can be used as well. Thus looking at variations of BNNs to determine where such models excel in the reinforcement learning framework is a good future direction. Additionally, due to the complexity of the CP model considering approximate variations through inducing points or variational inference can prove fruitful for high dimensional models such as a humanoid figure.

Appendix A

Sin Function Moments

This section provides the exact intregral equations for $\sin(x)$ which is required for squashing the control policy within a predetermined range. The expressions can be found in Gradsheteyn & Ryzhik [82], where $x \sim \mathcal{N}(\mu, \sigma^2)$.

$$\mathbb{E}_x[\sin(x)] = \int \sin(x)p(x)dx = \exp(-\frac{\sigma^2}{2})\sin(\mu) \tag{A.1}$$

$$\mathbb{E}_x[\cos(x)] = \int \cos(x)p(x)dx = \exp(-\frac{\sigma^2}{2})\cos(\mu) \tag{A.2}$$

$$\mathbb{E}_x[\sin(x)^2] = \int \sin(x)^2 p(x)dx = \frac{1}{2}\big(1 - \exp(-2\sigma^2)\cos(2\mu)\big) \tag{A.3}$$

$$\mathbb{E}_x[\cos(x)^2] = \int \cos(x)^2 p(x)dx = \frac{1}{2}\big(1 + \exp(-2\sigma^2)\cos(2\mu)\big) \tag{A.4}$$

Appendix B

Pilco Gradient Derivation

B.1 Partials w.r.t. $\tilde{\mu}_{t-1}$

B.1.1 Mean

$$\frac{\partial \mu_t}{\partial \tilde{\mu}_{t-1}} = \frac{\partial}{\partial \tilde{\mu}_{t-1}}(\mu_{t-1} + \mu_{\Delta_t}) = \frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_t]}{\partial \tilde{\mu}_{t-1}} = \frac{\partial}{\partial \tilde{\mu}_{t-1}}\Big[\beta_1^T \mathbf{q}_1, \ldots, \beta_D^T \mathbf{q}_D\Big]^T$$

$$\frac{\partial \mu_t^i}{\partial \tilde{\mu}_{t-1}} = \frac{\partial}{\partial \tilde{\mu}_{t-1}} \beta_i^T \mathbf{q}_i = \beta_i^T \left(\frac{\partial}{\partial \tilde{\mu}_{t-1}} \mathbf{q}_i\right)$$

Looking at the j^{th} index of $\mathbf{q}_i$:

$$\frac{\partial}{\partial \tilde{\mu}_{t-1}} q_{ij} = \frac{\partial}{\partial \tilde{\mu}_{t-1}} \frac{\alpha^2}{\sqrt(\det(\tilde{\Sigma}_{t-1}\Lambda_i^{-1} + \mathbf{I}))} \exp(-\frac{1}{2}(\mathbf{x}_j - \tilde{\mu}_{t-1})^T(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1}(\mathbf{x}_j - \tilde{\mu}_{t-1}))$$

$$\Rightarrow \frac{\partial q_{ij}}{\partial \tilde{\mu}_{t-1}} = q_{ij}\big((\mathbf{x}_j - \tilde{\mu}_{t-1})^T(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1}\big)$$

$$
\begin{aligned}
\frac{\partial \mu_t^i}{\partial \tilde{\mu}_{t-1}} &=, \beta_i^T \begin{bmatrix} q_{i1}\big((\mathbf{x}_1 - \tilde{\mu}_{t-1})^T(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1}\big) \\ q_{i2}\big((\mathbf{x}_2 - \tilde{\mu}_{t-1})^T(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1}\big) \\ \vdots \\ q_{in}\big((\mathbf{x}_n - \tilde{\mu}_{t-1})^T(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1}\big) \end{bmatrix} \\
&= (\beta_i \odot \mathbf{q}_i)^T \begin{bmatrix} (\mathbf{x}_1 - \tilde{\mu}_{t-1})^T \\ (\mathbf{x}_2 - \tilde{\mu}_{t-1})^T \\ \vdots \\ (\mathbf{x}_n - \tilde{\mu}_{t-1})^T \end{bmatrix} (\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1} \\
&= (\beta_i \odot \mathbf{q}_i)^T (\mathbf{X}^T - \tilde{\mu}_{t-1}^T \otimes \mathbf{1})(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1} \in \mathbb{R}^{1\times(D+F)}
\end{aligned}
$$

B.1.2 Covariance

$$
\begin{aligned}
\frac{\partial \Sigma_t}{\partial \tilde{\mu}_{t-1}} &= \frac{\partial}{\partial \tilde{\mu}_{t-1}} \big(\Sigma_{t-1} + \mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t] + \mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]^T + \Sigma_{\Delta_t}\big) \\
&= \frac{\partial}{\partial \tilde{\mu}_{t-1}} \big(\mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t] + \mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]^T + \Sigma_{\Delta_t}\big)
\end{aligned}
$$

Looking at the i^{th} index of $\mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]$:

$$
\begin{aligned}
\frac{\partial \mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_{t_i}]}{\partial \tilde{\mu}_{t-1}} &= \frac{\partial}{\partial \tilde{\mu}_{t-1}} \sum_{j=1}^{n} \beta_{ij} q_{ij} \tilde{\Sigma}_{t-1}\big(\tilde{\Sigma}_{t-1} + \Lambda_i\big)^{-1}(\mathbf{x}_j - \tilde{\mu}_{t-1}) \\
&= -\sum_{j=1}^{n} \beta_{ij} q_{ij} \tilde{\Sigma}_{t-1}\big(\tilde{\Sigma}_{t-1} + \Lambda_i\big)^{-1} \\
&= -\beta_i^T \mathbf{q}_i \tilde{\Sigma}_{t-1} \mathbf{R}_i^{-1}
\end{aligned}
$$

$$
\Rightarrow \frac{\partial \mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]}{\partial \tilde{\mu}_{t-1}} = -\begin{bmatrix} \beta_1^T \mathbf{q}_1 \tilde{\Sigma}_{t-1} \mathbf{R}_1^{-1} & \dots & \beta_d^T \mathbf{q}_d \tilde{\Sigma}_{t-1} \mathbf{R}_d^{-1} \end{bmatrix}^T
$$

For the predictive covariance Σ_{Δ_t} we look at the variance and cross-covariance elements.

$$\frac{\partial \big(\Sigma_{\Delta_t}\big)_{ab}}{\partial \tilde{\mu}_{t-1}} = \frac{\partial}{\partial \tilde{\mu}_{t-1}} \begin{cases} \beta_a^T \mathbf{Q} \beta_b - \mu_{\Delta_t}^a \mu_{\Delta_t}^b & a \neq b \\ \beta_a^T \mathbf{Q} \beta_a - (\mu_{\Delta_t}^a)^2 + \alpha_a^2 - \operatorname{tr}((\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1} \mathbf{Q}) & a = b \end{cases}$$

Considering the case $a \neq b$ first,

$$\frac{\partial}{\partial \tilde{\mu}_{t-1}} \beta_a^T \mathbf{Q} \beta_b - \tilde{\mu}_{t-1}^a \tilde{\mu}_{t-1}^b = \frac{\partial \beta_a^T \mathbf{Q} \beta_b}{\partial \tilde{\mu}_{t-1}} - \frac{\partial \tilde{\mu}_{t-1}^a \tilde{\mu}_{t-1}^b}{\partial \tilde{\mu}_{t-1}}$$

$$\begin{aligned}
\frac{\partial \beta_a^T \mathbf{Q} \beta_b}{\partial \tilde{\mu}_{t-1}} &= \operatorname{tr}\Big(\beta_b \beta_a^T \frac{\partial \mathbf{Q}}{\partial \tilde{\mu}_{t-1}}\Big) \\
&= \operatorname{tr}\left(\begin{bmatrix} \beta_{b1}\beta_{a1} & \dots & \beta_{b1}\beta_{an} \\ \vdots & \ddots & \vdots \\ \beta_{bn}\beta_{a1} & \dots & \beta_{bn}\beta_{an} \end{bmatrix} \begin{bmatrix} \frac{\partial Q_{11}}{\partial \tilde{\mu}_{t-1}} & \dots & \frac{\partial Q_{1n}}{\partial \tilde{\mu}_{t-1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial Q_{n1}}{\partial \tilde{\mu}_{t-1}} & \dots & \frac{\partial Q_{nn}}{\partial \tilde{\mu}_{t-1}} \end{bmatrix} \right) \\
&= \operatorname{tr}\left(\begin{bmatrix} \beta_{b1} \sum_{i=1}^n \beta_{ai} \frac{\partial Q_{i1}}{\partial \tilde{\mu}_{t-1}} & \dots & \beta_{b1} \sum_{i=1}^n \beta_{ai} \frac{\partial Q_{in}}{\partial \tilde{\mu}_{t-1}} \\ \vdots & \ddots & \vdots \\ \beta_{bn} \sum_{i=1}^n \beta_{ai} \frac{\partial Q_{i1}}{\partial \tilde{\mu}_{t-1}} & \dots & \beta_{bn} \sum_{i=1}^n \beta_{ai} \frac{\partial Q_{in}}{\partial \tilde{\mu}_{t-1}} \end{bmatrix} \right) \\
&= \sum_{j=1}^n \sum_{i=1}^n \beta_{ai} \beta_{bj} \frac{\partial Q_{ij}}{\partial \tilde{\mu}_{t-1}}
\end{aligned}$$

Looking at the ij^{th} element of $\mathbf{Q}$,

$$\frac{\partial Q_{ij}}{\partial \tilde{\mu}_{t-1}} = \frac{\partial Q_{ij}}{\partial \eta_{ij}^2} \frac{\partial \eta_{ij}^2}{\partial \tilde{\mu}_{t-1}}$$

$$\frac{\partial Q_{ij}}{\partial \eta_{ij}^2} = \frac{\partial}{\partial \eta_{ij}^2} \frac{\exp(\eta_{ij}^2)}{\sqrt{\det(\mathbf{S})}} = \frac{\exp(\eta_{ij}^2)}{\sqrt{\det(\mathbf{S})}} = Q_{ij}$$

where $\mathbf{S} = \tilde{\Sigma}_{t-1}(\Lambda_a^{-1} + \Lambda_b^{-1}) + \mathbf{I}$

$$\frac{\partial \eta_{ij}^2}{\partial \tilde{\mu}_{t-1}} = \frac{\partial}{\partial \tilde{\mu}_{t-1}} \Big(2(\log(\alpha_a) + \log(\alpha_b))) - \frac{\zeta_i^T \Lambda_a^{-1} \zeta_i + \zeta_j^T \Lambda_b^{-1} \zeta_j - z_{ij}^T \mathbf{S}^{-1} \tilde{\Sigma}_{t-1} z_{ij}}{2} \Big)$$

$$\frac{\partial \zeta_i^T \Lambda_a^{-1} \zeta_i}{\partial \tilde{\mu}_{t-1}} = -2(\mathbf{x}_i - \tilde{\mu}_{t-1})^T \Lambda_a^{-1} = -2 \zeta_i^T \Lambda_a^{-1}$$

$$\frac{\partial \zeta_i^T \Lambda_a^{-1} \zeta_j}{\partial \tilde{\mu}_{t-1}} = -(\mathbf{x}_i + \mathbf{x}_j - 2\tilde{\mu}_{t-1})^T \Lambda_a^{-1} = -(\zeta_i + \zeta_j)^T \Lambda_a^{-1}$$

$$\begin{aligned}
\frac{\partial z_{ij}^T \mathbf{S}^{-1} \tilde{\Sigma}_{t-1} z_{ij}}{\partial \tilde{\mu}_{t-1}} &= \frac{\partial}{\partial \tilde{\mu}_{t-1}} \left(\Lambda_a^{-1}\zeta_i + \Lambda_b^{-1}\zeta_j\right)^T \mathbf{S}^{-1} \tilde{\Sigma}_{t-1} \left(\Lambda_a^{-1}\zeta_i + \Lambda_b^{-1}\zeta_j\right) \\
&= \frac{\partial}{\partial \tilde{\mu}_{t-1}} \left(\zeta_i^T \mathbf{Y}_{aa} \zeta_i + \zeta_i^T (\mathbf{Y}_{ab} + \mathbf{Y}_{ba}) \zeta_j + \zeta_j^T \mathbf{Y}_{bb} \zeta_j\right) \\
&= -2\zeta_i^T \mathbf{Y}_{aa} - (\zeta_i + \zeta_j)^T (\mathbf{Y}_{ab} + \mathbf{Y}_{ba}) - 2\zeta_j^T \mathbf{Y}_{bb} \\
&= -\zeta_i^T \left(2\mathbf{Y}_{aa} + \mathbf{Y}_{ab} + \mathbf{Y}_{ba}\right) - \zeta_j^T \left(2\mathbf{Y}_{bb} + \mathbf{Y}_{ab} + \mathbf{Y}_{ba}\right)
\end{aligned}$$

where $\mathbf{Y}_{ab} := \Lambda_a^{-1} \mathbf{S}^{-1} \tilde{\Sigma}_{t-1} \Lambda_b^{-1}$

$$\Rightarrow \frac{\partial \eta_{ij}^2}{\partial \tilde{\mu}_{t-1}} = \frac{1}{2} \Big(\zeta_i^T \left(2\Lambda_a^{-1} - 2\mathbf{Y}_{aa} - \mathbf{Y}_{ab} - \mathbf{Y}_{ba}\right) + \zeta_j^T \left(2\Lambda_b^{-1} - 2\mathbf{Y}_{bb} - \mathbf{Y}_{ab} - \mathbf{Y}_{ba}\right) \Big)$$

$$\Rightarrow \frac{\partial Q_{ij}}{\partial \tilde{\mu}_{t-1}} = \frac{Q_{ij}}{2} \Big(\zeta_i^T \left(2\Lambda_a^{-1} - 2\mathbf{Y}_{aa} - \mathbf{Y}_{ab} - \mathbf{Y}_{ba}\right) + \zeta_j^T \left(2\Lambda_b^{-1} - 2\mathbf{Y}_{bb} - \mathbf{Y}_{ab} - \mathbf{Y}_{ba}\right) \Big)$$

$$\frac{\partial \mu_{\Delta_t}^a \mu_{\Delta_t}^b}{\partial \tilde{\mu}_{t-1}} = \mu_{\Delta_t}^b \frac{\partial \mu_{\Delta_t}^a}{\partial \tilde{\mu}_{t-1}} + \mu_{\Delta_t}^a \frac{\partial \mu_{\Delta_t}^b}{\partial \tilde{\mu}_{t-1}}$$

$\frac{\partial \mu_{\Delta_t}^b}{\partial \tilde{\mu}_{t-1}}$ is derived in B.1.1.

For the case $a = b$ we only need to consider the additional term $(\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}\mathbf{Q}$,

$$
\begin{aligned}
\frac{\partial \mathrm{tr}((\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}\mathbf{Q})}{\partial \tilde{\mu}_{t-1}} &= \mathrm{tr}\left(\frac{(\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}\partial \mathbf{Q}}{\partial \tilde{\mu}_{t-1}} \right) \\
&= \mathrm{tr}\left(\begin{bmatrix} (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{11} & \dots & (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{1n} \\ \vdots & \ddots & \vdots \\ (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{n1} & \dots & (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{nn} \end{bmatrix} \begin{bmatrix} \frac{\partial Q_{11}}{\partial \tilde{\mu}_{t-1}} & \dots & \frac{\partial Q_{1n}}{\partial \tilde{\mu}_{t-1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial Q_{n1}}{\partial \tilde{\mu}_{t-1}} & \dots & \frac{\partial Q_{nn}}{\partial \tilde{\mu}_{t-1}} \end{bmatrix} \right) \\
&= \mathrm{tr}\left(\begin{bmatrix} \sum_{i=1}^n (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{1i} \frac{\partial Q_{i1}}{\partial \tilde{\mu}_{t-1}} & \dots & \sum_{i=1}^n (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{1i} \frac{\partial Q_{in}}{\partial \tilde{\mu}_{t-1}} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{ni} \frac{\partial Q_{i1}}{\partial \tilde{\mu}_{t-1}} & \dots & \sum_{i=1}^n (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{ni} \frac{\partial Q_{in}}{\partial \tilde{\mu}_{t-1}} \end{bmatrix} \right) \\
&= \sum_{j=1}^n \sum_{i=1}^n (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{ij} \frac{\partial Q_{ij}}{\partial \tilde{\mu}_{t-1}}
\end{aligned}
$$

$$
\frac{\partial (\Sigma_{\Delta_t})_{ab}}{\partial \tilde{\mu}_{t-1}} = \begin{cases} \sum_{j=1}^n \sum_{i=1}^n \beta_{ai}\beta_{bj} \frac{\partial Q_{ij}}{\partial \tilde{\mu}_{t-1}} - (\mu^b_{\Delta_t} \frac{\partial \mu^a_{\Delta_t}}{\partial \tilde{\mu}_{t-1}} + \mu^a_{\Delta_t} \frac{\partial \mu^b_{\Delta_t}}{\partial \tilde{\mu}_{t-1}}) & a \neq b \\ \\ \sum_{j=1}^n \sum_{i=1}^n (\beta_{ai}\beta_{bj} - (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}_{ij}) \frac{\partial Q_{ij}}{\partial \tilde{\mu}_{t-1}} - 2\mu^a_{\Delta_t} \frac{\partial \mu^a_{\Delta_t}}{\partial \tilde{\mu}_{t-1}} & a = b \end{cases}
$$

B.2 Partials w.r.t. $\tilde{\Sigma}_{t-1}$

B.2.1 Mean

$$
\frac{\partial \mu_t}{\partial \tilde{\Sigma}_{t-1}} = \frac{\mathbb{E}_{x_{t-1}, u_{t-1}, f}[\Delta_t]}{\partial \tilde{\Sigma}_{t-1}}
$$

$$
\frac{\partial \mu_t^i}{\partial \tilde{\Sigma}_{t-1}} = \beta_i^T \left(\frac{\partial}{\partial \tilde{\Sigma}_{t-1}} \mathbf{q}_i \right)
$$

Again looking at the j^{th} index of $\log \mathbf{q}_i$:

$$\begin{aligned}
\frac{\partial}{\partial \tilde{\Sigma}_{t-1}} \log q_{ij} &= -\frac{1}{2}\frac{\partial}{\partial \tilde{\Sigma}_{t-1}}\left[\log\det\left(\tilde{\Sigma}_{t-1}\Lambda_i^{-1}+I\right)+(\mathbf{x}_j-\tilde{\mu}_{t-1})^T(\tilde{\Sigma}_{t-1}+\Lambda_i)^{-1}(\mathbf{x}_j-\mu_{t-1})\right] \\
&= -\frac{1}{2}\left(\Lambda_i^{-1}(\tilde{\Sigma}_{t-1}\Lambda_i^{-1}+I)^{-1}-(\tilde{\Sigma}_{t-1}+\Lambda_i)^{-1}(\mathbf{x}_j-\tilde{\mu}_{t-1})(\mathbf{x}_j-\tilde{\mu}_{t-1})^T(\tilde{\Sigma}_{t-1}+\Lambda_i)^{-1}\right) \\
&= -\frac{1}{2}\left((\tilde{\Sigma}_{t-1}+\Lambda_i)^{-1}-(\tilde{\Sigma}_{t-1}+\Lambda_i)^{-1}(\mathbf{x}_j-\tilde{\mu}_{t-1})(\mathbf{x}_j-\tilde{\mu}_{t-1})^T(\tilde{\Sigma}_{t-1}+\Lambda_i)^{-1}\right) \\
&= \frac{1}{2}\mathbf{R}_i^{-1}((\mathbf{x}_j-\tilde{\mu}_{t-1})(\mathbf{x}_j-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\mathbf{I})
\end{aligned}$$

$$\Rightarrow \frac{\partial}{\partial \tilde{\Sigma}_{t-1}}\log \mathbf{q}_i = \frac{1}{2}\begin{bmatrix} \mathbf{R}_i^{-1}((\mathbf{x}_1-\tilde{\mu}_{t-1})(\mathbf{x}_1-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\mathbf{I}) \\ \vdots \\ \mathbf{R}_i^{-1}((\mathbf{x}_n-\tilde{\mu}_{t-1})(\mathbf{x}_n-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\mathbf{I}) \end{bmatrix}$$

$$\Rightarrow \frac{\partial}{\partial \tilde{\Sigma}_{t-1}}\mathbf{q}_i = \frac{1}{2}\begin{bmatrix} q_{i1}\mathbf{R}_i^{-1}((\mathbf{x}_1-\tilde{\mu}_{t-1})(\mathbf{x}_1-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\mathbf{I}) \\ \vdots \\ q_{in}\mathbf{R}_i^{-1}((\mathbf{x}_n-\tilde{\mu}_{t-1})(\mathbf{x}_n-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\mathbf{I}) \end{bmatrix}$$

$$\begin{aligned}
\Rightarrow \frac{\partial \mu_t^i}{\partial \tilde{\Sigma}_{t-1}} &= \frac{1}{2}\beta_i^T\begin{bmatrix} q_{i1}\mathbf{R}_i^{-1}((\mathbf{x}_1-\tilde{\mu}_{t-1})(\mathbf{x}_1-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\mathbf{I}) \\ \vdots \\ q_{in}\mathbf{R}_i^{-1}((\mathbf{x}_n-\tilde{\mu}_{t-1})(\mathbf{x}_n-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\mathbf{I}) \end{bmatrix} \\
&= \frac{1}{2}\sum_{j=1}^{n}\beta_{ij}q_{ij}\mathbf{R}_i^{-1}((\mathbf{x}_j-\tilde{\mu}_{t-1})(\mathbf{x}_j-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\mathbf{I}) \\
&= \frac{1}{2}\sum_{j=1}^{n}\beta_{ij}q_{ij}\mathbf{R}_i^{-1}(\mathbf{x}_j-\tilde{\mu}_{t-1})(\mathbf{x}_j-\tilde{\mu}_{t-1})^T\mathbf{R}_i^{-1}-\beta_i^T\mathbf{q}_i\mathbf{R}_i^{-1} \\
&= \frac{1}{2}\left(\mathbf{T}^T\left(\mathbf{T}\odot[(\beta_i\odot\mathbf{q}_i)\otimes\mathbf{1}]\right)-\beta_i^T\mathbf{q}_i\mathbf{R}^{-1}\right)\in\mathbb{R}^{(D+F)\times(D+F)}
\end{aligned}$$

where $\mathbf{T} := (\mathbf{X}^T-\tilde{\mu}_{t-1}^T\otimes\mathbf{1})\mathbf{R}^{-1}$ and $\mathbf{R} := \tilde{\Sigma}_{t-1}+\Lambda_i$

B.2.2 Covariance

$$
\begin{aligned}
\frac{\partial \Sigma_t}{\partial \tilde{\Sigma}_{t-1}} &= \frac{\partial}{\partial \tilde{\Sigma}_{t-1}}(\Sigma_{t-1} + \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t] + \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]^T + \Sigma_{\Delta_t}) \\
&= \frac{\partial}{\partial \tilde{\Sigma}_{t-1}}(\text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t] + \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]^T + \Sigma_{\Delta_t})
\end{aligned}
$$

Look at i^{th} index of Δ_t:

$$
\begin{aligned}
\frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_{t_i}]}{\partial \tilde{\Sigma}_{t-1}} &= \frac{\partial}{\partial \tilde{\Sigma}_{t-1}} \sum_{j=1}^{n} \beta_{ij} q_{ij} \tilde{\Sigma}_{t-1}(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1}(\mathbf{x}_j - \tilde{\mu}_{t-1}) \\
&= \sum_{j=1}^{n} \beta_{ij} q_{ij} \frac{\partial}{\partial \tilde{\Sigma}_{t-1}} (\mathbf{I} + \Lambda_i \tilde{\Sigma}_{t-1}^{-1})^{-1}(\mathbf{x}_j - \tilde{\mu}_{t-1})
\end{aligned}
$$

Let $\mathbf{P}_i := \Lambda_i \tilde{\Sigma}_{t-1}^{-1}$, $\mathbf{M}_i := (\mathbf{I} + \mathbf{P}_i)^{-1}$, and $\mathbf{f} := \mathbf{M}_i \zeta_j$.

$$
\frac{\partial \mathbf{f}}{\partial \tilde{\Sigma}_{t-1}} = \frac{\partial \mathbf{f}}{\partial \mathbf{M}_i} \frac{\partial \mathbf{M}_i}{\partial \mathbf{P}_i} \frac{\partial \mathbf{P}_i}{\partial \tilde{\Sigma}_{t-1}}
$$

$$
\frac{\partial \mathbf{f}}{\partial \text{vec}(\mathbf{M}_i)} = (\zeta_j^T \otimes \mathbf{I})
$$

$$
\frac{\partial \text{vec}(\mathbf{M}_i)}{\partial \text{vec}(\mathbf{P}_i)} = -(\mathbf{M}_i^T \otimes \mathbf{M}_i)
$$

$$
\frac{\partial \text{vec}(\mathbf{P}_i)}{\partial \text{vec}(\tilde{\Sigma}_{t-1})} = -(\tilde{\Sigma}_{t-1}^{-T} \otimes \mathbf{P}_i)
$$

$$
\Rightarrow \frac{\partial \mathbf{f}}{\partial \text{vec}(\tilde{\Sigma}_{t-1})} = (\zeta_j^T \otimes \mathbf{I})(\mathbf{M}_i^T \otimes \mathbf{M}_i)(\tilde{\Sigma}_{t-1}^{-T} \otimes \mathbf{P}_i)
$$

$$
\Rightarrow \frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_{t_i}]}{\partial \text{vec}(\tilde{\Sigma}_{t-1})} = \sum_{j=1}^{n} \beta_{ij} q_{ij} (\zeta_j^T \otimes \mathbf{I})(\mathbf{M}_i^T \otimes \mathbf{M}_i)(\tilde{\Sigma}_{t-1}^{-T} \otimes \mathbf{P}_i)
$$

Another way of deriving this without the use of vectorization. Lets look at the k^{th} element of $\mathbf{f}$.

$$
\begin{aligned}
\frac{\partial \mathbf{e}_k^T \mathbf{f}}{\partial \tilde{\Sigma}_{t-1}} &= \frac{\partial \mathrm{tr}(\zeta_j \mathbf{e}_k^T \mathbf{M}_i)}{\partial \tilde{\Sigma}_{t-1}} \\
&= \frac{\mathrm{tr}(\zeta_j \mathbf{e}_k^T \partial \mathbf{M}_i)}{\partial \tilde{\Sigma}_{t-1}} \\
&= -\frac{\mathrm{tr}(\zeta_j \mathbf{e}_k^T \mathbf{M}_i \partial(\mathbf{I} + \mathbf{P}_i)\mathbf{M}_i)}{\partial \tilde{\Sigma}_{t-1}} \\
&= -\frac{\mathrm{tr}(\mathbf{M}_i \zeta_j \mathbf{e}_k^T \mathbf{M}_i \partial \mathbf{P}_i)}{\partial \tilde{\Sigma}_{t-1}} \\
&= \frac{\mathrm{tr}(\mathbf{M}_i \zeta_j \mathbf{e}_k^T \mathbf{M}_i \Lambda_i \tilde{\Sigma}_{t-1}^{-1} \partial \tilde{\Sigma}_{t-1} \tilde{\Sigma}_{t-1}^{-1})}{\partial \tilde{\Sigma}_{t-1}} \\
&= \tilde{\Sigma}_{t-1}^{-1} \mathbf{M}_i \zeta_j \mathbf{e}_k^T \mathbf{M}_i \mathbf{P}_i
\end{aligned}
$$

$$
\Rightarrow \frac{\partial \mathbf{f}}{\partial \tilde{\Sigma}_{t-1}} = \begin{bmatrix} \tilde{\Sigma}_{t-1}^{-1} \mathbf{M}_i \zeta_j \mathbf{e}_1^T \mathbf{M}_i \mathbf{P}_i & \ldots & \tilde{\Sigma}_{t-1}^{-1} \mathbf{M}_i \zeta_j \mathbf{e}_{d+f}^T \mathbf{M}_i \mathbf{P}_i \end{bmatrix}
$$

$$
\Rightarrow \frac{\partial \mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_{t_i}]}{\partial \tilde{\Sigma}_{t-1}} = \sum_{j=1}^{n} \beta_{ij} q_{ij} \begin{bmatrix} \tilde{\Sigma}_{t-1}^{-1} \mathbf{M}_i \zeta_j \mathbf{e}_1^T \mathbf{M}_i \mathbf{P}_i & \ldots & \tilde{\Sigma}_{t-1}^{-1} \mathbf{M}_i \zeta_j \mathbf{e}_{d+f}^T \mathbf{M}_i \mathbf{P}_i \end{bmatrix}
$$

Alternatively, if we do not simplify $\tilde{\Sigma}_{t-1}(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1}$ to $(\mathbf{I} + \Lambda_i \tilde{\Sigma}_{t-1}^{-1})^{-1}$, we can look at the k^{th} element again,

$$
\begin{aligned}
\frac{\partial \mathbf{e}_k^T \mathbf{f}}{\partial \tilde{\Sigma}_{t-1}} &= \frac{\partial \mathrm{tr}(\zeta_j \mathbf{e}_k^T \mathbf{M}_i)}{\partial \tilde{\Sigma}_{t-1}} \\
&= \frac{\partial \mathrm{tr}((\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1} \zeta_j \mathbf{e}_k^T \tilde{\Sigma}_{t-1})}{\partial \tilde{\Sigma}_{t-1}} \\
&= \frac{\mathrm{tr}(\partial(\tilde{\Sigma}_{t-1} + \Lambda_i)^{-1} \zeta_j \mathbf{e}_k^T \tilde{\Sigma}_{t-1})}{\partial \tilde{\Sigma}_{t-1}} \\
&= \left.\frac{\mathrm{tr}(\partial g_i(\mathbf{C})^{-1} \mathbf{A}_{jk} \tilde{\Sigma}_{t-1})}{\partial \mathbf{C}}\right|_{\mathbf{C}=\tilde{\Sigma}_{t-1}} + \left.\frac{\mathrm{tr}(g_i(\tilde{\Sigma}_{t-1})^{-1} \mathbf{A}_{jk} \partial \mathbf{D})}{\partial \mathbf{D}}\right|_{\mathbf{D}=\tilde{\Sigma}_{t-1}} \\
&= -\left.\frac{\mathrm{tr}(\mathbf{A}_{jk} \tilde{\Sigma}_{t-1} g_i(\mathbf{C})^{-1} \partial \mathbf{C} g_i(\mathbf{C})^{-1})}{\partial \mathbf{C}}\right|_{\mathbf{C}=\tilde{\Sigma}_{t-1}} + g_i(\tilde{\Sigma}_{t-1})^{-1} \mathbf{A}_{jk} \\
&= g_i(\tilde{\Sigma}_{t-1})^{-1} \mathbf{A}_{jk} (\mathbf{I} - \tilde{\Sigma}_{t-1} g_i(\tilde{\Sigma}_{t-1})^{-1})
\end{aligned}
$$

where $\mathbf{A}_{jk} := \zeta_j \mathbf{e}_k^T$ and $g_i(\mathbf{C}) := (\mathbf{C} + \Lambda_i)$

$$\Rightarrow \frac{\partial \mathbf{f}}{\partial \tilde{\Sigma}_{t-1}} = \begin{bmatrix} g_i(\tilde{\Sigma}_{t-1})^{-1}\mathbf{A}_{j1} & \dots & g_i(\tilde{\Sigma}_{t-1})^{-1}\mathbf{A}_{j(d+f)} \end{bmatrix}^T (\mathbf{I} - \tilde{\Sigma}_{t-1} g_i(\tilde{\Sigma}_{t-1})^{-1})$$

$$\Rightarrow \frac{\partial \mathrm{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_{t_i}]}{\partial \tilde{\Sigma}_{t-1}} = \sum_{j=1}^{n} \beta_{ij} q_{ij} \begin{bmatrix} g_i(\tilde{\Sigma}_{t-1})^{-1}\mathbf{A}_{j1} & \dots & g_i(\tilde{\Sigma}_{t-1})^{-1}\mathbf{A}_{j(d+f)} \end{bmatrix}^T (\mathbf{I} - \tilde{\Sigma}_{t-1} g_i(\tilde{\Sigma}_{t-1})^{-1})$$

For the predictive covariance we look at the variance and cross-covariance elements.

$$\frac{\partial (\Sigma_{\Delta_t})_{ab}}{\partial \tilde{\Sigma}_{t-1}} = \frac{\partial}{\tilde{\Sigma}_{t-1}} \begin{cases} \beta_a^T \mathbf{Q} \beta_b - \mu_{\Delta_t}^a \mu_{\Delta_t}^b & a \neq b \\ \beta_a^T \mathbf{Q} \beta_a - (\mu_{\Delta_t}^a)^2 + \alpha_a^2 - \mathrm{tr}((\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1}\mathbf{Q}) & a = b \end{cases}$$

Looking at the ij^{th} element of $\mathbf{Q}$:

$$\frac{\partial \log Q_{ij}}{\partial \tilde{\Sigma}_{t-1}} = \frac{\partial}{\partial \tilde{\Sigma}_{t-1}} (\eta_{ij}^2 - \frac{1}{2} \log \det(\mathbf{S}))$$

$$\begin{aligned}
\frac{\partial \eta_{ij}^2}{\partial \tilde{\Sigma}_{t-1}} &= \frac{\partial}{\partial \tilde{\Sigma}_{t-1}} \Big(2(\log(\alpha_a) + \log(\alpha_b))) - \frac{\zeta_i^T \Lambda_a^{-1} \zeta_i + \zeta_j^T \Lambda_b^{-1} \zeta_j - z_{ij}^T \mathbf{S}^{-1} \tilde{\Sigma}_{t-1} z_{ij}}{2} \Big) \\
&= \frac{1}{2} \frac{\partial}{\partial \tilde{\Sigma}_{t-1}} \left(z_{ij}^T \mathbf{S}^{-1} \tilde{\Sigma}_{t-1} z_{ij} \right) \\
&= \frac{1}{2} \mathrm{tr} \left(\frac{\partial}{\partial \tilde{\Sigma}_{t-1}} z_{ij} z_{ij}^T (\tilde{\Sigma}_{t-1}(\Lambda_a^{-1} + \Lambda_b^{-1}) + \mathbf{I})^{-1} \tilde{\Sigma}_{t-1} \right) \\
&= -\frac{1}{2} \mathrm{tr} \left((\Lambda_a^{-1} + \Lambda_b^{-1} + \tilde{\Sigma}_{t-1}^{-1})^{-1} z_{ij} z_{ij}^T (\Lambda_a^{-1} + \Lambda_b^{-1} + \tilde{\Sigma}_{t-1}^{-1})^{-1} \frac{\partial \tilde{\Sigma}_{t-1}^{-1}}{\partial \tilde{\Sigma}_{t-1}} \right) \\
&= \frac{1}{2} \mathrm{tr} \left(\tilde{\Sigma}_{t-1}^{-1} (\Lambda_a^{-1} + \Lambda_b^{-1} + \tilde{\Sigma}_{t-1}^{-1})^{-1} z_{ij} z_{ij}^T (\Lambda_a^{-1} + \Lambda_b^{-1} + \tilde{\Sigma}_{t-1}^{-1})^{-1} \tilde{\Sigma}_{t-1}^{-1} \frac{\partial \tilde{\Sigma}_{t-1}}{\partial \tilde{\Sigma}_{t-1}} \right) \\
&= \frac{1}{2} \left(\tilde{\Sigma}_{t-1}^{-1} (\Lambda_a^{-1} + \Lambda_b^{-1} + \tilde{\Sigma}_{t-1}^{-1})^{-1} z_{ij} z_{ij}^T (\Lambda_a^{-1} + \Lambda_b^{-1} + \tilde{\Sigma}_{t-1}^{-1})^{-1} \tilde{\Sigma}_{t-1}^{-1} \right) \\
&= \frac{1}{2} (((\Lambda_a^{-1} + \Lambda_b^{-1}) \tilde{\Sigma}_{t-1} + \mathbf{I})^{-1} z_{ij} z_{ij}^T (\tilde{\Sigma}_{t-1}(\Lambda_a^{-1} + \Lambda_b^{-1}) + \mathbf{I})^{-1} \\
&= \frac{1}{2} (((\Lambda_a^{-1} + \Lambda_b^{-1}) \tilde{\Sigma}_{t-1} + \mathbf{I})^{-1} z_{ij} z_{ij}^T \mathbf{S}^{-1}
\end{aligned}$$

$$\frac{\partial \log \det(\mathbf{S})}{\partial \tilde{\Sigma}_{t-1}} = (\Lambda_a^{-1} + \Lambda_b^{-1}) \mathbf{S}^{-1}$$

$$\Rightarrow \frac{\partial \log Q_{ij}}{\partial \tilde{\Sigma}_{t-1}} = \frac{1}{2}(((\Lambda_a^{-1} + \Lambda_b^{-1})\tilde{\Sigma}_{t-1} + \mathbf{I})^{-1} z_{ij} z_{ij}^T \mathbf{S}^{-1} - (\Lambda_a^{-1} + \Lambda_b^{-1})\mathbf{S}^{-1})$$
$$= \frac{1}{2}(((\Lambda_a^{-1} + \Lambda_b^{-1})\tilde{\Sigma}_{t-1} + \mathbf{I})^{-1} z_{ij} z_{ij}^T - \Lambda_a^{-1} - \Lambda_b^{-1})\mathbf{S}^{-1}$$

$$\Rightarrow \frac{\partial Q_{ij}}{\partial \tilde{\Sigma}_{t-1}} = \frac{Q_{ij}}{2}(((\Lambda_a^{-1} + \Lambda_b^{-1})\tilde{\Sigma}_{t-1} + \mathbf{I})^{-1} z_{ij} z_{ij}^T - \Lambda_a^{-1} - \Lambda_b^{-1})\mathbf{S}^{-1}$$

$$\frac{\partial \mu_{\Delta_t}^a \mu_{\Delta_t}^b}{\partial \tilde{\Sigma}_{t-1}} = \mu_{\Delta_t}^b \frac{\partial \mu_{\Delta_t}^a}{\partial \tilde{\Sigma}_{t-1}} + \mu_{\Delta_t}^a \frac{\partial \mu_{\Delta_t}^b}{\partial \tilde{\Sigma}_{t-1}}$$

$\frac{\partial \mu_{\Delta_t}^b}{\partial \tilde{\Sigma}_{t-1}}$ is derived in B.1.2.

$$\frac{\partial (\Sigma_{\Delta_t})_{ab}}{\partial \tilde{\Sigma}_{t-1}} = \begin{cases} \sum_{j=1}^n \sum_{i=1}^n \beta_{ai}\beta_{bj} \frac{\partial Q_{ij}}{\partial \tilde{\Sigma}_{t-1}} - (\mu_{\Delta_t}^b \frac{\partial \mu_{\Delta_t}^a}{\partial \tilde{\Sigma}_{t-1}} + \mu_{\Delta_t}^a \frac{\partial \mu_{\Delta_t}^b}{\partial \tilde{\Sigma}_{t-1}}) & a \neq b \\ \\ \sum_{j=1}^n \sum_{i=1}^n (\beta_{ai}\beta_{bj} - (\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})_{ij}^{-1}) \frac{\partial Q_{ij}}{\partial \tilde{\Sigma}_{t-1}} - 2\mu_{\Delta_t}^a \frac{\partial \mu_{\Delta_t}^a}{\partial \tilde{\Sigma}_{t-1}} & a = b \end{cases}$$

B.3 Derivatives w.r.t. ψ

B.3.1 Mean

$$\frac{d\tilde{\mu}_{t-1}}{d\psi} = \frac{d}{d\psi}\left[\mu_{t-1}, \mu_{t-1}^u\right]^T$$

We are given $\frac{d}{d\psi}\mu_{t-1}$

$$\frac{d\mu_{t-1}^u}{d\psi} = \frac{\partial \mu_{t-1}^u}{\partial \mu_{t-1}^{\tilde{u}}} \frac{d\mu_{t-1}^{\tilde{u}}}{d\psi} + \frac{\partial \mu_{t-1}^u}{\partial \Sigma_{t-1}^{\tilde{u}}} \frac{d\Sigma_{t-1}^{\tilde{u}}}{d\psi}$$

$$\frac{d\mu_{t-1}^{\tilde{u}}}{d\psi} = \frac{\partial \mu_{t-1}^{\tilde{u}}}{\partial \mu_{t-1}} \frac{d\mu_{t-1}}{d\psi} + \frac{\partial \mu_{t-1}^{\tilde{u}}}{\partial \Sigma_{t-1}} \frac{d\Sigma_{t-1}}{d\psi} + \frac{\partial \mu_{t-1}^{\tilde{u}}}{\partial \psi}$$

$$\frac{d\Sigma_{t-1}^{\tilde{u}}}{d\psi} = \frac{\partial \Sigma_{t-1}^{\tilde{u}}}{\partial \mu_{t-1}} \frac{d\mu_{t-1}}{d\psi} + \frac{\partial \Sigma_{t-1}^{\tilde{u}}}{\partial \Sigma_{t-1}} \frac{d\Sigma_{t-1}}{d\psi} + \frac{\partial \Sigma_{t-1}^{\tilde{u}}}{\partial \psi}$$

$$\begin{aligned}
\frac{\partial \mu_t}{\partial \psi} &= \frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \psi} \\
&= \frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \mu^u_{t-1}} \frac{\partial \mu^u_{t-1}}{\partial \psi} + \frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \Sigma^u_{t-1}} \frac{\partial \Sigma^u_{t-1}}{\partial \psi} \\
&= \frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \mu^u_{t-1}} \frac{\partial \mu^u_{t-1}}{\partial p(\tilde{\mathbf{u}};\psi)} \frac{\partial p(\tilde{\mathbf{u}};\psi)}{\partial \psi} + \frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \Sigma^u_{t-1}} \frac{\partial \Sigma^u_{t-1}}{\partial p(\tilde{\mathbf{u}};\psi)} \frac{\partial p(\tilde{\mathbf{u}};\psi)}{\partial \psi} \\
&= \frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \mu^u_{t-1}} \left(\frac{\partial \mu^u_{t-1}}{\partial \mu^{\tilde{u}}_{t-1}} \frac{\partial \mu^{\tilde{u}}_{t-1}}{\partial \psi} + \frac{\partial \mu^u_{t-1}}{\partial \Sigma^{\tilde{u}}_{t-1}} \frac{\partial \Sigma^{\tilde{u}}_{t-1}}{\partial \psi} \right) + \frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \Sigma^u_{t-1}} \left(\frac{\partial \Sigma^u_{t-1}}{\partial \mu^{\tilde{u}}_{t-1}} \frac{\partial \mu^{\tilde{u}}_{t-1}}{\partial \psi} + \frac{\partial \Sigma^u_{t-1}}{\partial \Sigma^{\tilde{u}}_{t-1}} \frac{\partial \Sigma^{\tilde{u}}_{t-1}}{\partial \psi} \right)
\end{aligned}$$

$\frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \mu^u_{t-1}}$ and $\frac{\partial \mathbb{E}_{x_{t-1},u_{t-1},f}[\Delta_{t-1}]}{\partial \Sigma^u_{t-1}}$ are known from the previous derivations in **??**, **??**. Whereas the partials within the parenthesis are determined by the controller and type of squashing function in **??**, **??**.

B.3.2 Covariance

$$\frac{d\tilde{\Sigma}_{t-1}}{d\psi} = \frac{d}{d\psi} \begin{bmatrix} \Sigma_{t-1} & \Sigma^{xu}_{t-1} \\ (\Sigma^{xu}_{t-1})^T & \Sigma^u_{t-1} \end{bmatrix}$$

We are given $\frac{d}{d\psi}\Sigma_{t-1}$.

$$\frac{d\Sigma^u_{t-1}}{d\psi} = \frac{\partial \Sigma^u_{t-1}}{\partial \mu^{\tilde{u}}_{t-1}} \frac{d\mu^{\tilde{u}}_{t-1}}{d\psi} + \frac{\partial \Sigma^u_{t-1}}{\partial \Sigma^{\tilde{u}}_{t-1}} \frac{d\Sigma^{\tilde{u}}_{t-1}}{d\psi}$$

$$\frac{d\Sigma^{xu}_{t-1}}{d\psi} = \frac{\partial \Sigma^{xu}_{t-1}}{\partial \mu^{\tilde{u}}_{t-1}} \frac{d\mu^{\tilde{u}}_{t-1}}{d\psi} + \frac{\partial \Sigma^{xu}_{t-1}}{\partial \Sigma^{\tilde{u}}_{t-1}} \frac{d\Sigma^{\tilde{u}}_{t-1}}{d\psi}$$

$$\frac{\partial \Sigma_t}{\partial \psi} = \frac{\partial}{\partial \psi}(\text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t] + \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]^T + \Sigma_{\Delta_t})$$

$$\begin{aligned}
\frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]}{\partial \psi} &= \frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]}{\mu^u_{t-1}} \frac{\partial \mu^u_{t-1}}{\partial \psi} + \frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]}{\Sigma^u_{t-1}} \frac{\partial \Sigma^u_{t-1}}{\partial \psi} \\
&= \frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]}{\mu^u_{t-1}} \left(\frac{\partial \mu^u_{t-1}}{\partial \mu^{\tilde{u}}_{t-1}} \frac{\partial \mu^{\tilde{u}}_{t-1}}{\partial \psi} + \frac{\partial \mu^u_{t-1}}{\partial \Sigma^{\tilde{u}}_{t-1}} \frac{\partial \Sigma^{\tilde{u}}_{t-1}}{\partial \psi} \right) + \frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1}, \Delta_t]}{\Sigma^u_{t-1}} \left(\frac{\partial \Sigma^u_{t-1}}{\partial \mu^{\tilde{u}}_{t-1}} \frac{\partial \mu^{\tilde{u}}_{t-1}}{\partial \psi} + \frac{\partial \Sigma^u_{t-1}}{\partial \Sigma^{\tilde{u}}_{t-1}} \frac{\partial \Sigma^{\tilde{u}}_{t-1}}{\partial \psi} \right)
\end{aligned}$$

$\frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1},\Delta_t]}{\partial \mu^u_{t-1}}$ and $\frac{\partial \text{cov}[\tilde{\mathbf{x}}_{t-1},\Delta_t]}{\partial \Sigma^u_{t-1}}$ are known from the previous derivations in B.2.1, B.2.2. Whereas the partials within the parenthesis are determined by the controller and type of squashing function.

For the predictive covariance we look at the variance and cross-covariance elements.

$$\frac{\partial(\Sigma_{\Delta_t})_{ab}}{\partial\psi} = \frac{\partial}{\psi}\begin{cases}\beta_a^T\mathbf{Q}\beta_b - \mu_{\Delta_t}^a\mu_{\Delta_t}^b & a \neq b \\ \beta_a^T\mathbf{Q}\beta_a - (\mu_{\Delta_t}^a)^2 + \alpha_a^2 + \mathrm{tr}((\mathbf{K}_a + \sigma_\epsilon^2\mathbf{I})^{-1}\mathbf{Q}) & a = b\end{cases}$$

$$= \begin{cases}\beta_a\beta_b^T(\frac{\partial\mathbf{Q}}{\partial\mu_{t-1}^u}\frac{\partial\mu_{t-1}^u}{\partial\psi} + \frac{\partial\mathbf{Q}}{\partial\Sigma_{t-1}^u}\frac{\partial\Sigma_{t-1}^u}{\partial\psi}) - (\mu_{\Delta_t}^b\frac{\partial\mu_{\Delta_t}^a}{\partial\psi} + \mu_{\Delta_t}^a\frac{\partial\mu_{\Delta_t}^b}{\partial\psi}) & a \neq b \\ \beta_a\beta_b^T(\frac{\partial\mathbf{Q}}{\partial\mu_{t-1}^u}\frac{\partial\mu_{t-1}^u}{\partial\psi} + \frac{\partial\mathbf{Q}}{\partial\Sigma_{t-1}^u}\frac{\partial\Sigma_{t-1}^u}{\partial\psi}) - 2\mu_{\Delta_t}^a\frac{\partial\mu_{\Delta_t}^a}{\partial\psi} + (\mathbf{K}_a + \sigma_\epsilon^2\mathbf{I})^{-1}\frac{\partial\mathbf{Q}}{\partial\psi} & a = b\end{cases}$$

$\frac{\partial\mathbf{Q}}{\partial\mu_{t-1}^u}$, $\frac{\partial\mathbf{Q}}{\partial\Sigma_{t-1}^u}$ are derived in B.1.2, B.2.2 respectively. $\frac{\partial\mu_{t-1}^u}{\partial\psi}$, $\frac{\partial\Sigma_{t-1}^u}{\partial\psi}$ are derived from the controller and squashing function. $\frac{\partial\mu_{\Delta_t}^a}{\partial\psi}$ is derived in B.3.1.

Appendix C

Convolution Kernel

Here we provide the derivation for the convolution kernel of a multi-output GP as specified in Section 2.2. Assume that the smoothing kernels are Gaussian kernels while the latent kernels are squared exponential kernels.

Give the formulation of the covariance between function outputs and/or latent functions we would like to derive the closed form solution given that the smooth kernels and latent function kernels are of exponential form.

Let their be one latent function, $R = 1$,

$$k_i(\boldsymbol{x}-\boldsymbol{z}) = \frac{1}{\sqrt{(2\pi)^n|\det(\boldsymbol{L}_i)|}} \exp\big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{z})^T\boldsymbol{L}_i^{-1}(\boldsymbol{x}-\boldsymbol{z})\big) \quad i = 1,\dots,m$$

$$k_u(\boldsymbol{z}',\boldsymbol{z}) = \alpha^2 \exp\big(-\frac{1}{2}(\boldsymbol{z}'-\boldsymbol{z})^T\boldsymbol{L}_u^{-1}(\boldsymbol{z}'-\boldsymbol{z})\big)$$

where $\boldsymbol{L}_i, \boldsymbol{L}_u \in \mathcal{S}_{++}^n$.

$$\operatorname{Cov}[f_{pj}(\boldsymbol{x}), f_{pi}(\boldsymbol{x}')] = \int k_j(\boldsymbol{x}'-\boldsymbol{z}')\int k_i(\boldsymbol{x}-\boldsymbol{z})k_u(\boldsymbol{z}',\boldsymbol{z})d\boldsymbol{z}d\boldsymbol{z}'$$

$$= \int k_j(\boldsymbol{x}'-\boldsymbol{z}')\int \frac{\alpha^2}{\sqrt{(2\pi)^n|\det(\boldsymbol{L}_i)|}} \exp\big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{z})^T\boldsymbol{L}_i^{-1}(\boldsymbol{x}-\boldsymbol{z})\big)\exp\big(-\frac{1}{2}(\boldsymbol{z}'-\boldsymbol{z})^T\boldsymbol{L}_u^{-1}(\boldsymbol{z}'-\boldsymbol{z})\big)d\boldsymbol{z}d\boldsymbol{z}'$$

Looking at the inner integral,

$$g(\boldsymbol{x},\boldsymbol{z}') \triangleq \int \frac{\alpha^2}{\sqrt{(2\pi)^n|\det(\boldsymbol{L}_i)|}} \exp\big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{z})^T\boldsymbol{L}_i^{-1}(\boldsymbol{x}-\boldsymbol{z})\big)\exp\big(-\frac{1}{2}(\boldsymbol{z}'-\boldsymbol{z})^T\boldsymbol{L}_u^{-1}(\boldsymbol{z}'-\boldsymbol{z})\big)d\boldsymbol{z}$$

$$
\begin{aligned}
&= \frac{\alpha^2}{\sqrt{(2\pi)^n |\det(\boldsymbol{L}_i)|}} \int \exp\Big(-\frac{1}{2}\big((\boldsymbol{x}-\boldsymbol{z})^T \boldsymbol{L}_i^{-1}(\boldsymbol{x}-\boldsymbol{z}) + (\boldsymbol{z}'-\boldsymbol{z})^T \boldsymbol{L}_u^{-1}(\boldsymbol{z}'-\boldsymbol{z})\big)\Big) d\boldsymbol{z} \\
&= \frac{\alpha^2}{\sqrt{(2\pi)^n |\det(\boldsymbol{L}_i)|}} \int \exp\Big(-\frac{1}{2}\big(\boldsymbol{z}^T(\boldsymbol{L}_i^{-1}+\boldsymbol{L}_u^{-1})\boldsymbol{z}^T + \boldsymbol{z}^T(-2(\boldsymbol{L}_i^{-1}\boldsymbol{x}+\boldsymbol{L}_u^{-1}\boldsymbol{z}')) + \boldsymbol{x}^T\boldsymbol{L}_i^{-1}\boldsymbol{x} + \boldsymbol{z}'^T\boldsymbol{L}_u^{-1}\boldsymbol{z}'\big)\Big) d\boldsymbol{z} \\
&= \frac{\alpha^2}{\sqrt{(2\pi)^n |\det(\boldsymbol{L}_i)|}} \int \exp\Big(-\frac{1}{2}\big((\boldsymbol{z}+\boldsymbol{h})^T \boldsymbol{A}(\boldsymbol{z}+\boldsymbol{h}) + k\big)\Big) d\boldsymbol{z} \\
&= \frac{\alpha^2\sqrt{(2\pi)^n|\det(\boldsymbol{A}^{-1})|}}{\sqrt{(2\pi)^n |\det(\boldsymbol{L}_i)|}} \exp\big(-\frac{1}{2}k\big) \int \frac{1}{\sqrt{(2\pi)^n|\det(\boldsymbol{A}^{-1})|}} \exp\Big(-\frac{1}{2}\big((\boldsymbol{z}+\boldsymbol{h})^T \boldsymbol{A}(\boldsymbol{z}+\boldsymbol{h})\big)\Big) d\boldsymbol{z} \\
&= \frac{\alpha^2\sqrt{(2\pi)^n|\det(\boldsymbol{A}^{-1})|}}{\sqrt{(2\pi)^n |\det(\boldsymbol{L}_i)|}} \exp\big(-\frac{1}{2}k\big) \int \mathcal{N}(\boldsymbol{z} \mid -\boldsymbol{h}, \boldsymbol{A}^{-1}) d\boldsymbol{z} \\
&= \frac{\alpha^2}{\sqrt{|\det(\boldsymbol{A})||\det(\boldsymbol{L}_i)|}} \exp\big(-\frac{1}{2}k\big) \\
&= \frac{\alpha^2}{\sqrt{|\det(\boldsymbol{I}+\boldsymbol{L}_u^{-1}\boldsymbol{L}_i)|}} \exp\big(-\frac{1}{2}k\big)
\end{aligned}
$$

where $\boldsymbol{A} = (\boldsymbol{L}_i^{-1}+\boldsymbol{L}_u^{-1})$, $\boldsymbol{b} = -2(\boldsymbol{L}_i^{-1}\boldsymbol{x}+\boldsymbol{L}_u^{-1}\boldsymbol{z}')$, $c = \boldsymbol{x}^T\boldsymbol{L}_i^{-1}\boldsymbol{x} + \boldsymbol{z}'^T\boldsymbol{L}_u^{-1}\boldsymbol{z}'$. Since $\boldsymbol{A} \in \mathcal{S}_{++}^n \Rightarrow \boldsymbol{h} = \frac{1}{2}\boldsymbol{A}^{-1}\boldsymbol{b}$ and $k = c - \frac{1}{4}\boldsymbol{b}^T\boldsymbol{A}^{-1}\boldsymbol{b}$.

Simplifying k,

$$
\begin{aligned}
k &= \boldsymbol{x}^T\boldsymbol{L}_i^{-1}\boldsymbol{x} + \boldsymbol{z}'^T\boldsymbol{L}_u^{-1}\boldsymbol{z}' - \frac{1}{4}\Big(-2(\boldsymbol{L}_i^{-1}\boldsymbol{x}+\boldsymbol{L}_u^{-1}\boldsymbol{z}')^T(\boldsymbol{L}_i^{-1}+\boldsymbol{L}_u^{-1})^{-1}(-2(\boldsymbol{L}_i^{-1}\boldsymbol{x}+\boldsymbol{L}_u^{-1}\boldsymbol{z}'))\Big) \\
&= \boldsymbol{x}^T\boldsymbol{L}_i^{-1}\boldsymbol{x} + \boldsymbol{z}'^T\boldsymbol{L}_u^{-1}\boldsymbol{z}' - (\boldsymbol{L}_i^{-1}\boldsymbol{x}+\boldsymbol{L}_u^{-1}\boldsymbol{z}')^T(\boldsymbol{L}_i^{-1}+\boldsymbol{L}_u^{-1})^{-1}(\boldsymbol{L}_i^{-1}\boldsymbol{x}+\boldsymbol{L}_u^{-1}\boldsymbol{z}') \\
&= \boldsymbol{x}^T(\boldsymbol{L}_i^{-1} - \boldsymbol{L}_i^{-1}(\boldsymbol{L}_i^{-1}+\boldsymbol{L}_u^{-1})^{-1}\boldsymbol{L}_i^{-1})\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{L}_i^{-1}(\boldsymbol{L}_i^{-1}+\boldsymbol{L}_u^{-1})^{-1}\boldsymbol{L}_u^{-1}\boldsymbol{z}' + \boldsymbol{z}'^T(\boldsymbol{L}_u^{-1} - \boldsymbol{L}_u^{-1}(\boldsymbol{L}_u^{-1}+\boldsymbol{L}_u^{-1})^{-1}\boldsymbol{L}_u^{-1})\boldsymbol{z}' \\
&= \boldsymbol{x}^T(\boldsymbol{L}_i+\boldsymbol{L}_u)^{-1}\boldsymbol{x} - 2\boldsymbol{x}^T(\boldsymbol{L}_i+\boldsymbol{L}_u)^{-1}\boldsymbol{z}' + \boldsymbol{z}'^T(\boldsymbol{L}_i+\boldsymbol{L}\_u)^{-1}\boldsymbol{z}' \\
&= (\boldsymbol{x}-\boldsymbol{z}')^T(\boldsymbol{L}_i+\boldsymbol{L}_u)^{-1}(\boldsymbol{x}-\boldsymbol{z}')
\end{aligned}
$$

The second to last equality follows from the Woodbury Matrix Identity. Thus the inner integral has the closed form,

$$
g(\boldsymbol{x},\boldsymbol{z}') = \frac{\alpha^2\sqrt{\det(\boldsymbol{L}_u)}}{\sqrt{\det(\boldsymbol{L}_u+\boldsymbol{L}_i)}} \exp\big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{z}')^T(\boldsymbol{L}_i+\boldsymbol{L}_u)^{-1}(\boldsymbol{x}-\boldsymbol{z}')\big)
$$

Looking at the outer integral we can follow the exact same steps to arrive at the closed form solution for the convolution process kernel function,

$$\begin{aligned}
\operatorname{Cov}[f_{pj}(\boldsymbol{x}), f_{pi}(\boldsymbol{x}')] &= \int k_j(\boldsymbol{x}' - \boldsymbol{z}')g(\boldsymbol{x}, \boldsymbol{z}')d\boldsymbol{z}' \\
&= \alpha^2 \sqrt{(2\pi)^n |\det(\boldsymbol{L}_u)|} \int \mathcal{N}(\boldsymbol{x}' \mid \boldsymbol{z}', \boldsymbol{L}_j)\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{z}', \boldsymbol{L}_i + \boldsymbol{L}_u)d\boldsymbol{z}' \\
&= \alpha^2 \sqrt{(2\pi)^n |\det(\boldsymbol{L}_u)|} \int \mathcal{N}(\boldsymbol{z}' \mid \boldsymbol{x}', \boldsymbol{L}_j)\mathcal{N}(\boldsymbol{z}' \mid \boldsymbol{x}, \boldsymbol{L}_i + \boldsymbol{L}_u)d\boldsymbol{z}' \\
&= \alpha^2 \sqrt{(2\pi)^n |\det(\boldsymbol{L}_u)|} \mathcal{N}(\boldsymbol{x}' \mid \boldsymbol{x}, \boldsymbol{L}_i + \boldsymbol{L}_u + \boldsymbol{L}_j) \int \mathcal{N}(\boldsymbol{z}' \mid \boldsymbol{\alpha}, \boldsymbol{\beta})d\boldsymbol{z}' \\
&= \alpha^2 \frac{\sqrt{|\det(\boldsymbol{L}_u)|}}{\sqrt{|\det(\boldsymbol{L}_i + \boldsymbol{L}_u + \boldsymbol{L}_j)|}} \exp\Big(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T(\boldsymbol{L}_i + \boldsymbol{L}_u + \boldsymbol{L}_j)^{-1}(\boldsymbol{x}' - \boldsymbol{x})\Big)
\end{aligned}$$

Considering $R > 1$ then the covariance has the form of,

$$\operatorname{Cov}[f_{pj}(\boldsymbol{x}), f_{pi}(\boldsymbol{x}')] = \sum_{r=1}^{R} \frac{\alpha_r^2 \sqrt{|\det(\boldsymbol{L}_{u_r})|}}{\sqrt{|\det(\boldsymbol{L}_{ir} + \boldsymbol{L}_{u_r} + \boldsymbol{L}_{jr})|}} \exp\Big(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T(\boldsymbol{L}_{ir} + \boldsymbol{L}_{u_r} + \boldsymbol{L}_{jr})^{-1}(\boldsymbol{x}' - \boldsymbol{x})\Big)$$

We can apply the same methodology for the covariance between the output and latent function,

$$\operatorname{Cov}[f_{pi}(\boldsymbol{x}), u_r(\boldsymbol{z})] = g(\boldsymbol{x}, \boldsymbol{z})$$

Bibliography

[1] M. P. Deisenroth, "Efficient reinforcement learning using gaussian processes," Ph.D. dissertation, 2009.

[2] M. Kuss and C. Rasmussen, "Assessing approximations for gaussian process classification," in *Advances in neural information processing systems 18*, Max-Planck-Gesellschaft. Cambridge, MA, USA: MIT Press, May 2006, pp. 699–706.

[3] J. Quiñonero-Candela, A. Girard, and C. E. Rasmussen, "Prediction at an uncertain input for gaussian processes and relevance vector machines application to multiple-step ahead time-series forecasting," Tech. Rep., 2002.

[4] J. Quiñonero-Candela, A. Girard, J. Larsen, and C. E. Rasmussen, "Propagation of uncertainty in bayesian kernel models - application to multiple-step ahead forecasting," in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, vol. 2, April 2003, pp. II–701.

[5] R. M. Neal, *Bayesian Learning for Neural Networks.* Berlin, Heidelberg: Springer-Verlag, 1996.

[6] C. K. I. Williams and C. E. Rasmussen, "Gaussian processes for regression," in *Advances in Neural Information Processing Systems 8.* MIT press, 1996, pp. 514–520.

[7] D. J. C. MacKay, "Introduction to gaussian processes," 1998.

[8] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2005.

[9] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, "Deep Neural Networks as Gaussian Processes," *arXiv e-prints*, Oct 2017.

[10] E. Snelson and Z. Ghahramani, "Local and global sparse gaussian process approximations," in *AISTATS*, 2007.

[11] ——, "Sparse gaussian processes using pseudo-inputs," in *NIPS*, 2005.

[12] M. K. Titsias, "Variational learning of inducing variables in sparse gaussian processes," in *AISTATS*, 2009.

[13] D. R. Burt, C. E. Rasmussen, and M. van der Wilk, "Rates of convergence for sparse variational gaussian process regression," *ArXiv*, vol. abs/1903.03571, 2019.

[14] Y. Saatchi, "Scalable inference for structured gaussian process models," Ph.D. dissertation, 11 2011.

[15] J. P. Cunningham, K. V. Shenoy, and M. Sahani, "Fast gaussian process methods for point process intensity estimation," in *ICML*, 2008.

[16] A. G. Wilson and H. Nickisch, "Kernel interpolation for scalable structured gaussian processes (kiss-gp)," *ArXiv*, vol. abs/1503.01057, 2015.

[17] S. Sun, G. Zhang, C. Wang, W. Zeng, J. Li, and R. B. Grosse, "Differentiable compositional kernel learning for gaussian processes," in *ICML*, 2018.

[18] F. R. Bach, “High-dimensional non-linear variable selection through hierarchical kernel learning,” *ArXiv*, vol. abs/0909.0844, 2009.

[19] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen, “Additive gaussian processes,” in *NIPS*, 2011.

[20] D. K. Duvenaud, J. R. Lloyd, R. B. Grosse, J. B. Tenenbaum, and Z. Ghahramani, “Structure discovery in nonparametric regression through compositional kernel search,” in *ICML*, 2013.

[21] J. R. Lloyd, D. K. Duvenaud, R. B. Grosse, J. B. Tenenbaum, and Z. Ghahramani, “Automatic construction and natural-language description of nonparametric regression models,” *ArXiv*, vol. abs/1402.4304, 2014.

[22] S. Bochner, M. Functions, S. Integrals, H. Analysis, M. Tenenbaum, and H. Pollard, *Lectures on Fourier Integrals. (AM-42).* Princeton University Press, 1959.

[23] M. Lázaro-Gredilla, J. Q. Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, “Sparse spectrum gaussian process regression,” *J. Mach. Learn. Res.*, vol. 11, pp. 1865–1881, 2010.

[24] A. G. Wilson and R. P. Adams, “Gaussian process kernels for pattern discovery and extrapolation,” in *ICML*, 2013.

[25] Y.-L. K. Samo and S. J. Roberts, “Generalized spectral kernels,” 2015.

[26] G. Parra and F. Tobar, “Spectral mixture kernels for multi-output gaussian processes,” in *NIPS*, 2017.

[27] P. Boyle and M. R. Frean, “Dependent gaussian processes,” in *NIPS*, 2004.

[28] M. A. Álvarez and N. D. Lawrence, “Sparse convolved gaussian processes for multi-output regression,” in *NIPS*, 2008.

[29] A. Naish and S. B. Holden, “The generalized fitc approximation,” in *NIPS*, 2007.

[30] M. A. Álvarez, D. Luengo, M. K. Titsias, and N. D. Lawrence, “Efficient multioutput gaussian processes through variational inducing kernels,” in *AISTATS*, 2010.

[31] P. Moreno-Muñoz, A. Artés-Rodríguez, and M. A. Álvarez, “Heterogeneous multi-output gaussian process prediction,” in *NeurIPS*, 2018.

[32] A. C. Damianou and N. D. Lawrence, “Deep gaussian processes,” *ArXiv*, vol. abs/1211.0358, 2012.

[33] A. C. Damianou, M. K. Titsias, and N. D. Lawrence, “Variational gaussian process dynamical systems,” in *NIPS*, 2011.

[34] D. K. Duvenaud, O. Rippel, R. P. Adams, and Z. Ghahramani, “Avoiding pathologies in very deep networks,” in *AISTATS*, 2014.

[35] H. Salimbeni and M. P. Deisenroth, “Doubly stochastic variational inference for deep gaussian processes,” in *NIPS*, 2017.

[36] R. E. Turner and M. Sahani, “Two problems with variational expectation maximisation for time-series models,” 2011.

[37] J. M. Hernández-Lobato and R. P. Adams, “Probabilistic backpropagation for scalable learning of bayesian neural networks,” in *ICML*, 2015.

[38] T. P. Minka, “A family of algorithms for approximate bayesian inference,” Ph.D. dissertation, Cambridge, MA, USA, 2001, aAI0803033.

[39] D. J. C. MacKay, “A practical bayesian framework for backpropagation networks,” *Neural Computation*, vol. 4,

pp. 448–472, 1992.

[40] G. E. Hinton and D. van Camp, “Keeping neural networks simple,” 1993.

[41] D. J. C. MacKay, “Bayesian neural networks and density networks,” 1995.

[42] R. M. Neal, “Bayesian learning via stochastic dynamics,” in *NIPS*, 1992.

[43] D. G. Barber and C. M. Bishop, “Ensemble learning in bayesian neural networks,” 1998.

[44] A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernández-Lobato, and A. L. Gaunt, “Deterministic variational inference for robust bayesian neural networks,” in *ICLR*, 2019.

[45] A. Graves, “Practical variational inference for neural networks,” in *NIPS*, 2011.

[46] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural network,” in *ICML*, 2015.

[47] D. P. Kingma, T. Salimans, and M. Welling, “Variational dropout and the local reparameterization trick,” *ArXiv*, vol. abs/1506.02557, 2015.

[48] C. Louizos and M. Welling, “Structured and efficient variational deep learning with matrix gaussian posteriors,” in *ICML*, 2016.

[49] ——, “Multiplicative normalizing flows for variational bayesian neural networks,” in *ICML*, 2017.

[50] T. Chen, E. B. Fox, and C. Guestrin, “Stochastic gradient hamiltonian monte carlo,” in *ICML*, 2014.

[51] M. Welling and Y. W. Teh, “Bayesian learning via stochastic gradient langevin dynamics,” in *ICML*, 2011.

[52] S. Ahn, A. K. Balan, and M. Welling, “Bayesian posterior sampling via stochastic gradient fisher scoring,” in *ICML*, 2012.

[53] T. Nagapetyan, A. B. Duncan, L. Hasenclever, S. J. Vollmer, L. Szpruch, and K. Zygalakis, “The true cost of stochastic gradient langevin dynamics,” 2017.

[54] Y.-A. Ma, T. Chen, and E. B. Fox, “A complete recipe for stochastic gradient mcmc,” in *NIPS*, 2015.

[55] S. Ghosh, F. M. D. Fave, and J. Yedidia, “Assumed density filtering methods for learning bayesian neural networks,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI’16. AAAI Press, 2016, pp. 1589–1595.

[56] S. Sun, C. Chen, and L. Carin, “Learning structured weight uncertainty in bayesian neural networks,” in *AISTATS*, 2017.

[57] Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” in *ICML*, 2015.

[58] I. Osband, “Risk versus uncertainty in deep learning : Bayes , bootstrap and the dangers of dropout,” 2016.

[59] T. Fushiki, F. Komaki, and K. Aihara, “Nonparametric bootstrap prediction,” 2005.

[60] B. Lakshminarayanan, A. Pritzel, and C. Blundell, “Simple and scalable predictive uncertainty estimation using deep ensembles,” *ArXiv*, vol. abs/1612.01474, 2016.

[61] R. S. Sutton, “Dyna, an integrated architecture for learning, planning, and reacting,” *SIGART Bulletin*, vol. 2, pp. 160–163, 1990.

[62] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," *ArXiv*, vol. abs/1907.02057, 2019.

[63] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *ML*, 1990.

[64] ——, "Planning by incremental dynamic programming," in *ML*, 1991.

[65] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, "Learning and policy search in stochastic dynamical systems with bayesian neural networks," *ArXiv*, vol. abs/1605.07127, 2016.

[66] N. Mishra, P. Abbeel, and I. Mordatch, "Prediction and control with temporal segment models," in *ICML*, 2017.

[67] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *ArXiv*, vol. abs/1802.10592, 2018.

[68] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *ICML*, 2015.

[69] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *CoRL*, 2018.

[70] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darell, and T. Ma, "Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees," *ArXiv*, vol. abs/1807.03858, 2018.

[71] A. G. Richards, "Robust constrained model predictive control," 2005.

[72] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *ICLR*, 2018.

[73] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *NeurIPS*, 2018.

[74] N. M. O. Heess, G. Wayne, D. Silver, T. P. Lillicrap, Y. Tassa, and T. Erez, "Learning continuous control policies by stochastic value gradients," *ArXiv*, vol. abs/1510.09142, 2015.

[75] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving PILCO with Bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, ICML*, Apr. 2016.

[76] S. Kamthe and M. P. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control," in *AISTATS*, 2017.

[77] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *NIPS*, 2014.

[78] W. Montgomery and S. Levine, "Guided policy search as approximate mirror descent," *CoRR*, vol. abs/1607.04614, 2016.

[79] F. Pedregosa, 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, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[80] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke,

Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org.

[81] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001–.

[82] I. S. Gradshteyn and I. M. Ryzhik, *Table of integrals, series, and products*, 7th ed. Elsevier/Academic Press, Amsterdam, 2007, translated from the Russian, Translation edited and with a preface by Alan Jeffrey and Daniel Zwillinger, With one CD-ROM (Windows, Macintosh and UNIX).

[83] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014.

[84] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *ICML*, 2012.

[85] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, “When gaussian process meets big data: A review of scalable gps,” 2018.

[86] D. Nguyen-Tuong, M. W. Seeger, and J. Peters, “Local gaussian process regression for real time online model learning,” in *NIPS*, 2008.

[87] Y. W. Teh, M. W. Seeger, and M. I. Jordan, “Semiparametric latent factor models,” in *AISTATS*, 2005.

[88] M. P. Deisenroth and S. Mohamed, “Expectation propagation in gaussian process dynamical systems,” in *NIPS*, 2012.

[89] J. Ko and D. Fox, “Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models,” in *IROS*, 2008.

[90] V. Feinberg, L.-F. Cheng, K. Li, and B. E. Engelhardt, “Large linear multi-output gaussian process learning,” 2017.

[91] E. V. Bonilla, K. M. A. Chai, and C. K. I. Williams, “Multi-task gaussian process prediction,” in *NIPS*, 2007.

[92] A. G. de G. Matthews, J. Hensman, R. E. Turner, and Z. Ghahramani, “On sparse variational methods and the kullback-leibler divergence between stochastic processes,” in *AISTATS*, 2015.

[93] Y. Gal, Y. Chen, and Z. Ghahramani, “Latent gaussian processes for distribution estimation of multivariate categorical data,” in *ICML*, 2015.

[94] E. V. Bonilla, K. Krauth, and A. Dezfouli, “Generic inference in latent gaussian process models,” 2016.

[95] C. L. C. Mattos, Z. Dai, A. C. Damianou, J. Forth, G. D. A. Barreto, and N. D. Lawrence, “Recurrent gaussian processes,” *CoRR*, vol. abs/1511.06644, 2015.

[96] M. K. Titsias and N. D. Lawrence, “Bayesian gaussian process latent variable model,” in *AISTATS*, 2010.

[97] J. Knoblauch, J. Jewson, and T. Damoulas, “Generalized variational inference,” *ArXiv*, vol. abs/1904.02063, 2019.

[98] Z. Dai, A. C. Damianou, J. M. H. Gonzalez, and N. D. Lawrence, “Variational auto-encoded deep gaussian processes,” *CoRR*, vol. abs/1511.06455, 2015.

[99] J. Hensman and N. D. Lawrence, “Nested variational compression in deep g aussian processes,” 2014.

[100] J. Knoblauch, “Robust deep gaussian processes,” *ArXiv*, vol. abs/1904.02303, 2019.

[101] N. D. Lawrence, “Probabilistic non-linear principal component analysis with gaussian process latent variable models,” *J. Mach. Learn. Res.*, vol. 6, pp. 1783–1816, 2005.

[102] ——, “Gaussian process latent variable models for visualisation of high dimensional data,” in *NIPS*, 2003.

[103] M. A. Álvarez, L. Rosasco, and N. D. Lawrence, “Kernels for vector-valued functions: A review,” *ArXiv*, vol. abs/1106.6251, 2011.

[104] M. A. Álvarez, D. Luengo, and N. D. Lawrence, “Latent force models,” in *AISTATS*, 2009.

[105] A. C. Damianou, C. H. Ek, M. K. Titsias, and N. D. Lawrence, “Manifold relevance determination,” *ArXiv*, vol. abs/1206.4610, 2012.

[106] J. Zhao and S. Sun, “Variational dependent multi-output gaussian process dynamical systems,” *J. Mach. Learn. Res.*, vol. 17, pp. 121:1–121:36, 2014.

[107] K. Chen, P. Groot, J. Chen, and E. Marchiori, “Generalized spectral mixture kernels for multi-task gaussian processes,” *ArXiv*, vol. abs/1808.01132, 2018.

[108] S. Remes, M. Heinonen, and S. Kaski, “Non-stationary spectral kernels,” in *NIPS*, 2017.

[109] C. Archambeau and F. Bach, “Multiple gaussian process models,” 2011.

[110] C. Walder, K. I. Kim, and B. Schölkopf, “Sparse multiscale gaussian process regression,” in *ICML*, 2008.

[111] R. Salakhutdinov and G. E. Hinton, “Using deep belief nets to learn covariance kernels for gaussian processes,” in *NIPS*, 2007.

[112] F. A. Tobar, T. D. Bui, and R. E. Turner, “Learning stationary time series using gaussian processes with nonparametric kernels,” in *NIPS*, 2015.

[113] M. Gönen and E. Alpaydin, “Multiple kernel learning algorithms,” *J. Mach. Learn. Res.*, vol. 12, pp. 2211–2268, 2011.

[114] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, “When gaussian process meets big data: A review of scalable gps,” *ArXiv*, vol. abs/1807.01065, 2018.

[115] C.-A. Cheng and B. Boots, “Variational inference for gaussian process models with linear complexity,” in *NIPS*, 2017.

[116] W. Herlands, A. G. Wilson, H. Nickisch, S. Flaxman, D. B. Neill, W. V. Panhuis, and E. P. Xing, “Scalable gaussian processes for characterizing multidimensional change surfaces,” in *AISTATS*, 2015.

[117] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing, “Deep kernel learning,” in *AISTATS*, 2015.

[118] G. Pleiss, J. R. Gardner, K. Q. Weinberger, and A. G. Wilson, “Constant-time predictive distributions for gaussian processes,” in *ICML*, 2018.

[119] J. Hensman, N. Fusi, and N. D. Lawrence, “Gaussian processes for big data,” *ArXiv*, vol. abs/1309.6835, 2013.

[120] J. Q. Candela and C. E. Rasmussen, “A unifying view of sparse approximate gaussian process regression,” *J. Mach. Learn. Res.*, vol. 6, pp. 1939–1959, 2005.

[121] A. G. Wilson, E. Gilboa, J. P. Cunningham, and A. Nehorai, “Fast kernel learning for multidimensional pattern extrapolation,” in *NIPS*, 2014.

[122] A. G. de G. Matthews, M. Rowland, J. Hron, R. E. Turner, and Z. Ghahramani, “Gaussian process behaviour in wide deep neural networks,” *ArXiv*, vol. abs/1804.11271, 2018.

[123] W. Gong, Y. Li, and J. M. Hernández-Lobato, “Meta-learning for stochastic gradient mcmc,” *ArXiv*, vol. abs/1806.04522, 2018.

[124] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. A. Riedmiller, R. Hadsell, and P. W. Battaglia, "Graph networks as learnable physics engines for inference and control," *CoRR*, vol. abs/1806.01242, 2018.

[125] M. Yip and D. B Camarillo, "Model-less feedback control of continuum manipulators in constrained environments," *IEEE Transactions on Robotics*, vol. 30, pp. 880–888, 08 2014.

[126] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," 2017.

[127] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," *CoRR*, vol. abs/1509.06791, 2015.

[128] A. V. Rao, "A survey of numerical methods for optimal control," 2009.

[129] T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. M. O. Heess, Y. Li, R. Pascanu, P. W. Battaglia, D. Silver, and D. Wierstra, "Imagination-augmented agents for deep reinforcement learning," *ArXiv*, vol. abs/1707.06203, 2017.

[130] J. A. Bagnell and J. G. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 2, pp. 1615–1620 vol.2, 2001.

[131] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *Robotics: Science and Systems*, 2015.

[132] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," *ArXiv*, vol. abs/1603.00748, 2016.

[133] J. Fu, S. Levine, and P. Abbeel, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4019–4026, 2015.

[134] N. Wahlstrom, F. Lindsten, and M. P. Deisenroth, "From pixels to torques: Policy learning with deep dynamical models," *ArXiv*, vol. abs/1502.02251, 2015.

[135] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3223–3230, 2015.

[136] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, "Epopt: Learning robust neural network policies using model ensembles," *ArXiv*, vol. abs/1610.01283, 2016.

[137] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *ICML*, 2006.

[138] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[139] K. Asadi, D. Misra, and M. L. Littman, "Lipschitz continuity in model-based reinforcement learning," in *ICML*, 2018.

[140] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, pp. 1–142, 2013.

[141] A. M. Farahmand, A. Barreto, and D. Nikovski, "Value-aware loss function for model-based reinforcement learning," in *AISTATS*, 2017.

[142] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566, 2017.

[143] J. Peters and S. Schaal, "Policy gradient methods for robotics," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2219–2225, 2006.

[144] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *NIPS*, 2003.

[145] J. Ko, D. J. Klein, D. Fox, and D. Hähnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 742–747, 2007.

[146] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," *ArXiv*, vol. abs/1604.06778, 2016.

[147] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, pp. 39:1–39:40, 2015.

[148] M. Watter, J. T. Springenberg, J. Boedecker, and M. A. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *NIPS*, 2015.

[149] R. McAllister and C. E. Rasmussen, "Improving pilco with bayesian neural network dynamics models," 2016.

[150] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 378–383, 2016.

[151] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[152] H. van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 272–279, 2007.

[153] K. Lee, S.-A. Kim, J. Choi, and S.-W. Lee, "Deep reinforcement learning in continuous action spaces: a case study in the game of simulated curling," in *ICML*, 2018.

[154] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, pp. 408–423, 2015.

[155] M. P. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *ICML*, 2011.