university of groningen / faculty of science and engineering / bernoulli institute

Interpretable Function Approximation with Gaussian Processes in Value-Based Model-Free Reinforcement Learning

Bachelor's Project Thesis

Matthijs van der Lende, s4325621, m.r.van.der.lende@student.rug.nl
Supervisor: J.D. Cardenas Cartagena, M.Sc., j.d.cardenas.cartagena@rug.nl

Abstract: Estimating a value function for reinforcement learning (RL) in continuous spaces is a challenging task. To address this, the field of RL employs various function approximators, including linear models and deep neural networks. Linear models are interpretable but can only model simple functions, while deep neural networks can model complex functions but tend to be black-box models. Gaussian process (GP) models aim to offer the best of both worlds by being able to model complex nonlinear functions while providing interpretable uncertainty estimates. This includes extensions such as the sparse variational GP (SVGP) and deep GP (DGP). This thesis presents a Bayesian nonparametric framework for off-policy and on-policy learning using GPs for action-value function modeling. Results on the CartPole and Lunar Lander environments show that SVGPs/DGPs significantly outperform linear function approximation, but do not yet match the speed of convergence or performance of deep RL algorithms using neural networks. These findings highlight the potential of GPs in RL as function approximators in tasks where uncertainty and interpretability is mandatory.

1 Introduction

In reinforcement learning (RL), the goal is for an agent to interact with an environment and learn behaviors, known as policies, by maximizing a reward signal. Achieving this goal requires estimating complex functions, such as value functions and policies, that guide the agent's decisions and learning.

Initially, RL relied on tabular methods, which involve storing state-(action) values in a table. The classical Q-learning algorithm (R. Sutton & Barto, 2018) is an example of the tabular approach. This method is sufficient for problems where the number of states and actions are small, such as simple gridworlds or mazes (R. Sutton & Barto, 2018). However, the problem is that tabular methods do not scale to more advanced RL problems. A critical challenge in RL is the approximation of value functions and policies, especially in environments with continuous state and action spaces.

Model-free RL, which assumes no model of the environment, includes two subfields focused on function approximation. Value-based methods primarily use parametric models to approximate the (action)-value function, from which a policy is derived. Policy-gradient methods directly approximate the policy. Both approaches can also be combined in what are called actor-critic methods, where an approximate value function (the critic) is learned to aid in approximating the policy (the actor).

Linear methods have been used to address the limitations of tabular methods. R. S. Sutton (1988) tackled this by introducing a linear gradient-based temporal difference (TD) algorithm called TD(λ). Traditional linear models, such as linear regression, while simple and interpretable, often fall short in capturing the dynamics of complex environments, particularly when there are nonlinear relationships in the state-action spaces.

To address this limitation, kernel-based methods transform the state-action space into a higher-dimensional space where relationships become more linear (Taylor & Parr, 2009). That said, kernel-based methods are often limited by the chosen kernel function and the computational complexity involved in storing a kernel matrix over the training samples (Bishop, 2006).

The sub-field of deep reinforcement learning (DRL) seeks to address this problem by employing neural networks as function approximators. State-of-the-art RL algorithms, such as deep Q-network (DQN) (Mnih et al., 2013) and proximal policy optimization (PPO), (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017), are DRL algorithms, with the former being value-based and the latter actor-critic based.

Deep learning models are black-box models

(Rudin, 2019); however, unlike a linear model, where each weight has an interpretable meaning. The reason deep learning models are preferred is that they are capable of learning a larger class of functions and extracting features from data, as demonstrated in tasks such as learning directly from Atari image data (Mnih et al., 2013).

Despite their opacity, ongoing research aims to enhance the interpretability and trustworthiness of deep learning models, striving to make their decision-making processes more transparent. The field of explainable AI attempts to address this by attempting to produce human-interpretable explanations of machine learning model outputs. Uncertainty quantification addresses this by allowing a model to provide probabilistic outputs that reflect its level of confidence. For example, outputting a mean and standard deviation allows humans to consider uncertainty when interpreting results.

In RL, it is useful for learned policies or value functions to estimate their epistemic uncertainty, which represents uncertainty in the agent's knowledge about the value function. High epistemic uncertainty in a particular subset of the state space corresponds to under-exploration of that subset. This can help RL agents make more informed decisions about the exploration-exploitation trade-off (Lockwood & Si, 2022), reducing the amount of environment interactions needed to learn policies. This is in contrast to value-based methods like DQN, which rely on random action selection for sufficient exploration. Additionally, Safe RL utilizes uncertainty quantification to adhere to safety constraints during the learning process (Berkenkamp, 2019), particularly vital in robotics applications where exploration actions could lead to physical harm.

A popular choice for interpretable function approximation is a Gaussian process (GP), a nonparametric kernel-based model for regression or classification. GPs provide powerful epistemic uncertainty quantification by capturing a distribution over functions. Unlike linear models, they are capable of modeling complex nonlinear functions (Rasmussen & Williams, 2004; Murphy, 2023) and their nonparametric nature allows them to scale in complexity with the dataset size. GPs operate in a Bayesian framework, which, combined with their nonparametric nature, makes them inherently more robust against overfitting compared to frequentist approaches that obtain point-wise parameter estimates by optimizing a loss function (Bishop & Bishop, 2024).

Originally, GPs were designed for spatial data analysis, where it was known as Kriging (Cressie, 1990). Over the years, different variations of the GP model have been developed to address limitations of the base GP model. These include sparse variational GPs (SVGPs) (Cormen, Leiserson, Rivest, & Stein, 2009), which enable GPs to handle the computational complexity of the covariance matrix, and deep GPs (DGPs) (Damianou & Lawrence, 2013), which can be conceptualized as a neural network-type model where each unit is a GP. DGPs are shown to be strictly more general than traditional GPs, as the latter are ultimately constrained by the choice of kernel function (Murphy, 2023).

1.1 Related Work

Early work on GPs and RL used GPs for modeling the transition function and value function (Rasmussen & Williams, 2004).

Engel, Mannor, & Meir (2005) employed GPs to approximate the value and action-value function, resulting in a variant of the SARSA algorithm. This approach was further extended to the off-policy case by Chowdhary et al. (2014), who proposed a variant of Q-learning using GPs alongside a proof of convergence. Kameda & Tanaka (2023) applied variation inference methods to reduce computational complexity in GP Q-learning.

This prior research on GPs in value function approximation did not explore different exploration strategies beyond upper confidence bound and ϵ-greedy, as discussed in Section 2.4. By not exploring alternative exploration strategies, such as Thompson sampling, these studies potentially limit the effectiveness of GP-based RL algorithms.

Extensions to the basic GP model, such as the DGP model, have not been considered. DGPs can potentially capture more complex relationships in the data due to their hierarchical structure. Ignoring these advanced models might restrict the applicability and performance of GP-based methods in more complex RL tasks.

Moreover, previous research has not directly compared GP-based methods with linear function approximation nor neural network function approximation in DRL. Without such comparisons, it is challenging to position GP-based methods within the broader landscape of RL algorithms and to identify scenarios where they might offer unique advantages.

1.2 Contributions

This thesis focuses on the advantages and limitations of GPs as function approximators for model-free RL. With a particular focus on GP regression for the action-value function and how it performs compared to traditional TD learning approaches such as DQN. Specifically, this thesis presents an off-policy and on-policy Bayesian nonparametric framework, based on Q-learning and SARSA, respectively, which employ the GP model for the

action-value function. Different exploration strategies in the framework of Bayesian optimization will be considered and extension of the GP model to DGPs.

We expect that GP-based function approximation for the action-value function will outperform linear function approximation but perform worse than deep neural network (DNN)-based function approximation.

2 Theoretical Framework

2.1 Mathematical Notation

The notation in this thesis is mainly based on R. Sutton & Barto (2018), with adjustments for consistency with Murphy (2023).

Sets are denoted by majuscule and curly characters, e.g., $\mathcal{X}$, except for well-known sets like the real numbers $\mathbb{R}$, the complex numbers $\mathbb{C}$, the natural numbers $\mathbb{N}$, and the non-negative reals $\mathbb{R}^+$.

Random variables (RVs) are denoted by capital letters, e.g., X; latent variables in GPs are indexed and written as $f_{\text{GP}}(\mathbf{x}_i)$ or f_i for short to indicate a RV at index $\mathbf{x}_i$. Realizations of RVs or scalars use lower case letters, e.g., $x \in \mathbb{R}$.

Vectors are lower case bold letters, e.g., $\mathbf{x} \in \mathbb{R}^n, n \in \mathbb{N}$, except for states s, actions a, and rewards r in RL contexts, which remain lowercase. Vectors are assumed to be column vectors by default. The transpose operation is denoted $\top$, i.e., $\mathbf{x}^\top \in \mathbb{R}^{1\times n}$.

Matrices are bold capitals, e.g., $\mathbf{X}$, with the identity matrix denoted $\mathbf{I}$. $|\mathbf{X}|$ denotes the determinant for matrices, and $|\mathcal{X}|$ denotes the cardinality for sets.

Random vectors are italicized boldface, e.g., $\boldsymbol{X} = [X_1, X_2, \ldots, X_n]^\top$. A random vector $\boldsymbol{X}$ with a Gaussian distribution of mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is denoted $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. A sample $\mathbf{x}$ from this distribution is similarly denoted $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Function composition is denoted with $\circ$, e.g., $f(g(x)) = f \circ g(x)$.

For asymptotic analysis, we focus on upper bounds for time and space complexity. Given functions $f, g : \mathbb{N} \to \mathbb{R}^+$, $f(n) = O(g(n))$ if there are positive integers c and n_0 such that for all $n \geq n_0$, $f(n) \leq cg(n)$ (Sipser, 2018). Formally,

$$O(g(n)) = \{f(n) : \exists c, n_0 > 0 \; \forall n \geq n_0 \; f(n) \leq cg(n)\}.$$

2.2 Function Approximation in Reinforcement Learning

The versatility of RL stems from its ability to address complex challenges requiring sequential decision-making. If a problem can be described as an agent interacting with an environment, formalized as a Markov decision process (MDP), then RL can be applied. The end product is a policy, defining what actions to take in each state such that the cumulative reward received is maximized.

A Markov decision process is a 4-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathrm{p}, \mathrm{r})$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathrm{p} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition function, and $\mathrm{r} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function. Our RL agent is captured by a policy, which is a conditional probability distribution over actions given the state:

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s) \quad \forall s, a \in \mathcal{S} \times \mathcal{A}, \tag{2.1}$$

where S_t, A_t are RVs for the state and action at time t. The goal is to learn an optimal policy π_* that maximizes the expected discounted sum of rewards:

$$\pi_* = \operatorname*{argmax}_\pi \mathbb{E}_\pi[G_t] = \operatorname*{argmax}_\pi \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right], \tag{2.2}$$

where G is the return, R is the reward and $\gamma \in (0, 1]$ is a discount factor. The hypothesis in RL is that by interacting with the environment the agent is able to learn intelligent behavior.

In value-based methods this is achieved by learning a value function:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \tag{2.3}$$

or an action-value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a], \tag{2.4}$$

and then deriving a policy from it. As it turns out, every optimal policy π_* has the same optimal value and action-value function (R. Sutton & Barto, 2018):

$$\begin{aligned} v_*(s) &= \max_\pi v_\pi(s), \\ q_*(s, a) &= \max_\pi q_\pi(s, a). \end{aligned} \tag{2.5}$$

A deterministic optimal policy could then be defined as $\pi_*(s) = \operatorname{argmax}_{a\in\mathcal{A}} q_*(s, a)$.

If $\mathcal{S}$ and $\mathcal{A}$ are small finite sets, then a value function could simply be implemented as a table, this requires $O(|\mathcal{S}||\mathcal{A}|)$ storage. This approach becomes infeasible when we have continuous state and/or action spaces (e.g., $\mathcal{S}, \mathcal{A} \subseteq \mathbb{R}^m$).

What we want then is an approximation $\hat{v}$ or $\hat{q}$ that approaches the optimal value function and generalizes well. In case of a parametric model, we learn parameters $\mathbf{w} \in \mathbb{R}^d$. This can be a linear model or a neural network.

2.3 Gaussian Processes

A GP is a collection of RVs $\{f_{\text{GP}}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}$, any finite number of which has a joint Gaussian

distribution (Rasmussen & Williams, 2004). Note that $f_{\mathrm{GP}}(\mathbf{x})$ is the notation used for an indexed RV in GPs instead of the more typical X_t for a RV at index t. In the context of regression, the index set $\mathcal{X}$ is not related to time, but the input of some function $f : \mathcal{X} \to \mathbb{R}$ that we want to approximate (e.g., the action-value function). It can be interpreted as: At each point $\mathbf{x} \in \mathcal{X}$, the output of the GP regression model is a RV denoted $f_{\mathrm{GP}}(\mathbf{x})$.

A GP, denoted $f_{\mathrm{GP}}(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$, is fully specified by a mean function $m : \mathcal{X} \to \mathbb{R}$ and covariance or kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which are defined as:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f_{\mathrm{GP}}(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f_{\mathrm{GP}}(\mathbf{x}) - m(\mathbf{x}))(f_{\mathrm{GP}}(\mathbf{x}') - m(\mathbf{x}'))]. \end{aligned} \tag{2.6}$$

A GP offers a Bayesian approach to nonparametric regression. Without any data, the kernel function represents our prior belief about the function we are trying to model, by abuse of notation denoted $p(f)$, as it encodes similarity between data points, with closer points having higher covariance. A GP can then be conditioned on a dataset, $\mathcal{D}$, to get a posterior GP $f_{\mathrm{GP}}(\mathbf{x}) \sim \mathcal{GP}(m_{\mathrm{post}}(\mathbf{x}), k_{\mathrm{post}}(\mathbf{x}, \mathbf{x}'))$, which is our posterior belief about the function, $p(f|\mathcal{D})$.

Given a train set of noisy observations $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1,\dots,N}$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i = f(\mathbf{x}_i) + \epsilon_i$ with $\epsilon_i \sim \mathcal{N}(0, \sigma_y^2)$. Suppose we are interested in getting predictions $\mathbf{f}_* = [f_{\mathrm{GP}}(\mathbf{x}_1^*), \dots, f_{\mathrm{GP}}(\mathbf{x}_{N_*}^*)]^\top$* for test inputs $\mathbf{X}_* = (\mathbf{x}_1^*, \dots, \mathbf{x}_{N_*}^*)$. By the definitions of a GP it follows that the prior joint distribution $p(\mathbf{y}, \mathbf{f}_*|\mathbf{X}, \mathbf{X}_*)$ has the following form:

$$p(\mathbf{y}, \mathbf{f}_*|\mathbf{X}, \mathbf{X}_*) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_X \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \mathbf{K}_\sigma & \mathbf{K}_{X,*} \\ \mathbf{K}_{X,*}^\top & \mathbf{K}_{*,*} \end{bmatrix} \right), \tag{2.7}$$

where $\boldsymbol{\mu}_X = [(m(\mathbf{x}_1), ..., m(\mathbf{x}_N)]^\top$, $\boldsymbol{\mu}_* = [m(\mathbf{x}_1^*), \dots, m(\mathbf{x}_{N_*}^*)]^\top$, $\mathbf{K}_{X,X} = k(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N}$, $\mathbf{K}_\sigma = \mathbf{K}_{X,X} + \sigma_y^2 \mathbf{I}$, $\mathbf{K}_{X,*} = k(\mathbf{X}, , \mathbf{X}_*) \in \mathbb{R}^{N \times N_*}$, and $\mathbf{K}_{*,*} = k(\mathbf{X}_*, \mathbf{X}_*) \in \mathbb{R}^{N_* \times N_*}$ are matrices of all the covariances between relevant datapoints.

One can then condition on the observations to get the Bayesian predictive distribution for the test points. By Gaussian identities we can get this in closed form:

$$p(\mathbf{f}_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*) = \mathcal{N}(\boldsymbol{\mu}_{*|\mathcal{D},\mathbf{X}_*}, \boldsymbol{\Sigma}_{*|\mathcal{D},\mathbf{X}_*}), \tag{2.8}$$

where

$$\begin{aligned} \boldsymbol{\mu}_{*|\mathcal{D},\mathbf{X}_*} &= \boldsymbol{\mu}_* + \mathbf{K}_{X,*}^\top \mathbf{K}_\sigma^{-1}(\mathbf{y} - \boldsymbol{\mu}_X), \\ \boldsymbol{\Sigma}_{*|\mathcal{D},\mathbf{X}_*} &= \mathbf{K}_{*,*} - \mathbf{K}_{X,*}^\top \mathbf{K}_\sigma^{-1} \mathbf{K}_{X,*}. \end{aligned} \tag{2.9}$$

Assuming a zero mean function, this reduces for a single test point to:

$$\begin{aligned} \mathbb{E}[f_{\mathrm{GP}}(\mathbf{x}_*)] &= \mu(\mathbf{x}_*) = \mathbf{k}_*^\top \underbrace{\mathbf{K}_\sigma^{-1}\mathbf{y}}_{\boldsymbol{\alpha}} = \textstyle\sum_{i=1}^N \alpha_i k(\mathbf{x}_i, \mathbf{x}_*), \\ \mathbb{V}[f_{\mathrm{GP}}(\mathbf{x}_*)] &= \sigma^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top \mathbf{K}_\sigma^{-1} \mathbf{k}_*, \end{aligned} \tag{2.10}$$

where $\mathbf{k}_* = [k(\mathbf{x}_*, \mathbf{x}_1), \dots, k(\mathbf{x}_*, \mathbf{x}_N)]^\top$.

To get the posterior mean and covariance function we consider (2.9) and (2.10) over an infinite number of potential test points:

$$\begin{aligned} m_{\mathrm{post}}(\mathbf{x}_*) &= m(\mathbf{x}_*) + \mathbf{k}_*^\top \mathbf{K}_\sigma^{-1}(\mathbf{y} - \boldsymbol{\mu}_X) \\ k_{\mathrm{post}}(\mathbf{x}_*, \mathbf{x}_*') &= k(\mathbf{x}_*, \mathbf{x}_*') - \mathbf{k}_*^\top \mathbf{K}_\sigma^{-1} \mathbf{k}_{*'}, \end{aligned} \tag{2.11}$$

where $\mathbf{k}_{*'} = [k(\mathbf{x}_*', \mathbf{x}_1), \dots, k(\mathbf{x}_*', \mathbf{x}_N)]^\top$.

The generalization properties of GPs rely on the selection of the appropriate kernel (Rasmussen & Williams, 2004; Murphy, 2023). A common kernel is the Matern kernel (see Appendix A.4), given by

$$k_{\mathrm{matern}}(\mathbf{x}, \mathbf{x}'; l, \nu, \sigma_f) = \sigma_f \frac{2^{1-\nu}}{\Gamma(\nu)} \big(\sqrt{2\nu} d\big)^\nu K_\nu\big(\sqrt{2\nu} d\big), \tag{2.12}$$

with $d = (\mathbf{x} - \mathbf{x}')^\top l^{-2} (\mathbf{x} - \mathbf{x}')$, lengthscale parameter l, outputscale parameter σ_f, smoothness parameter ν, Gamma function Γ and a modified Bessel function K_ν. As $\nu \to \infty$, the Matern kernel approaches the Radial Basis Function (RBF) kernel:

$$k_{\mathrm{RBF}}(\mathbf{x}, \mathbf{x}'; l, \sigma_f) = \sigma_f \exp\left(- \frac{\| \mathbf{x} - \mathbf{x}' \|^2}{2l^2} \right), \tag{2.13}$$

which when used results in smooth infinitely differentiable functions being sampled.

For generalization it is also important to optimize the GP hyperparameters, such as kernel parameters and noise variance, alongside computing the predictive distribution as seen in (2.9). This can be done by performing type-II maximum likelihood estimation (MLE) through maximizing the marginal log likelihood (MLL) $\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$, for hyperparameters $\boldsymbol{\theta}$:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \int_{\mathbb{R}^N} p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta}) \, d\mathbf{f}. \tag{2.14}$$

As $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_\sigma)$ this integral can be computed as:

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2} \mathbf{y}^\top \mathbf{K}_\sigma^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_\sigma| - \frac{N}{2} \log(2\pi), \tag{2.15}$$

where the first term is a data fit term, the second term a model fit term and the last a constant. The negative MLL is differentiable with respect to $\boldsymbol{\theta}$ (A.3), so stochastic gradient descent can be used.

Using blackbox matrix-matrix multiplication (BBMM) inference (Gardner, Pleiss, Weinberger, Bindel, & Wilson, 2018), which leverages GPU acceleration, the overall time and space complexity of GP regression is $O(N^2)$. See Appendix A.2 for details.

*$\mathbf{f}_*$ is implicitly assumed to be a realization of the random vector.

2.4 Bayesian Optimization and Multi-Armed Bandits

Bayesian optimization (BayesOpt) concerns itself with global optimization of black-box functions $f : \mathcal{X} \to \mathbb{R}$ (Murphy, 2023). Commonly a GP is used as a regressor or surrogate for f based on the data collected so far.

The BayesOpt algorithm as shown in 2.1 proceeds as follows: At each iteration n, a dataset $\mathcal{D}_n = (\mathbf{x}_i, y_i)_{i=1,\dots,n}$ is maintained where the target outputs $y_i = f(\mathbf{x}_i) + \epsilon_i$ are assumed to be noisy outputs of the function f we want to optimize. A GP can then be used to estimate $p(f|\mathcal{D})$, a distribution over f. An acquisition function $\alpha(\mathbf{x}; \mathcal{D}_n)$ is then used to select a new candidate $\mathbf{x}$ based on its expected utility. Once $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_{n+1}$ has been observed, the GP is updated by computing $p(f|\mathcal{D}_{n+1})$.

There is an inherent tradeoff between selecting points $\mathbf{x} \in \mathcal{X}$ for which f is large (exploitation) and points with high uncertainty, represented by the variance, where one might be able to improve the GP by finding higher values of f (exploration). For this reason, next to its use in hyperparameter tuning (Snoek, Larochelle, & Adams, 2012), the technique has also been applied to the Bandit problem. In particular, Srinivas, Krause, Kakade, & Seeger (2012) provided sublinear regret bounds using their GP optimization algorithm with $y_n = R_n + \epsilon_n$ as targets, implying that the algorithm's action selection becomes optimal over time. However, this algorithm is constrained to the bandits problem, which learns a policy in a single-state, discrete-actions environment.

Algorithm 2.1 Bayesian Optimization

1: Collect initial dataset $\mathcal{D}_0 = (\mathbf{x}_i, y_i)_{i=1,\dots,n_0}$ from random queries $\mathbf{x}_i$ or a space-filling design
2: Initialize model (e.g., a GP) by computing $p(f|\mathcal{D}_0)$
3: **for** $n = 1, 2, \dots$ until convergence **do**
4: Choose next query point $\mathbf{x}_{n+1} = \text{argmax}_{\mathbf{x}\in\mathcal{X}}\, \alpha(\mathbf{x}; \mathcal{D}_n)$
5: Measure function value, $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_n$
6: Augment dataset, $\mathcal{D}_{n+1} = \mathcal{D}_n \cup \{(\mathbf{x}_{n+1}, y_{n+1})\}$
7: Update model by computing $p(f|\mathcal{D}_{n+1})$
8: **end for**

2.4.1 Acquisition Functions

The acquisition function regulates exploration in the input space, resembling behavioral policies in RL, designed to favor inputs $\mathbf{x}$ with high uncertainty in $f(\mathbf{x})$ while minimizing selections of already explored points. This approach results in more confident estimates for $f(\mathbf{x})$. Various acquisition functions exist (Murphy, 2023):

Upper confidence bound (UCB) is an acquisition function defined as:

$$\alpha_n(\mathbf{x}; \mathcal{D}_n) = \mu_n(\mathbf{x}) + \beta_n \sigma_n(\mathbf{x}), \tag{2.16}$$

where μ_n, σ_n are the mean and standard deviation outputs as is described in (2.10). β_n is an exploration parameter.

Another acquisition function is **Thompson sampling**. In the context of Bandits, this involves sampling an action-value function $\tilde{q}$ from the GP posterior predictive distribution, and greedily selecting an action according to the sample:

$$a_{n+1} = \underset{a\in\mathcal{A}}{\text{argmax}}\ \tilde{q}(a) \quad \tilde{q}(\cdot) \sim p(q|\mathcal{D}_n).$$

The intuition behind Thompson sampling is that exploration is encouraged by maximizing $\tilde{q}$ because the sampled function is within the credible interval (standard deviations around the mean) with high values around the areas with high uncertainty. Maximizing $\tilde{q}$ involves selecting actions where there is potentially high uncertainty on q. At the same time, actions with a high mean value are also likely to be sampled, which promotes exploitation.

2.5 Value-based RL using Gaussian Processes

Using GPs for value-based RL is done by capturing a distribution over possible q-functions, $q_\pi \sim \mathcal{GP}(m(z), k(z, z'))$, where the input domain is $z \in \mathcal{S} \times \mathcal{A}$ (Chowdhary et al., 2014). The core idea behind the GP-Q algorithm (see Algorithm 2.2) is to perform a type of Bayesian optimization on the TD error by setting the target values as described in Algorithm 2.1 to the TD(0) target. Using different acquisition functions, we now have more options in designing the behavioral policy other than just using ϵ-greedy. Chowdhary et al. (2014) for example used a variant of GP-Q using UCB.

The GP-Q algorithm is off-policy and uses

$$R_{t+1} + \gamma \max_{a\in\mathcal{A}} \bar{q}(S_{t+1}, a) \tag{2.17}$$

as the TD target, where $\bar{q}$ is the mean output of the GP (Chowdhary et al., 2014). An on-policy variant, GP-SARSA, can be created by setting the target to

$$R_{t+1} + \gamma \bar{q}(S_{t+1}, A_{t+1}). \tag{2.18}$$

Regarding the choice of kernel, Engel et al. (2005) suggested to define the kernel function $k : (\mathcal{S}\times\mathcal{A})\times(\mathcal{S}\times\mathcal{A}) \to \mathbb{R}$ as a combination of a state-kernel k_s and action-kernel k_a,

$$k((s, a), (s', a')) = k_s(s, s')k_a(a, a'), \tag{2.19}$$

as the states and actions are two different entities. By properties of kernels (Murphy, 2023), k is also a valid kernel function.

Algorithm 2.2 Online GP-Q for estimating $p(q_*|\mathcal{D})$

Collect initial dataset $\mathcal{D}_0 = (z_i, y_i)_{i=1,\dots,n_0}$ with $z_i \in \mathcal{S} \times \mathcal{A}$, $y_i \in \mathbb{R}$
Initialize GP by computing $p(q|\mathcal{D}_0)$
for each time step t **do**
  Choose A_t from S_t using behavioral policy (e.g., ϵ-greedy)
  Take action A_t, observe R_{t+1}, S_{t+1}
  let $Z_t = (S_t, A_t)^\top$ and $y_t = R_{t+1} + \gamma \max_{a'} \bar{q}(S_{t+1}, a')$, where $\bar{q}$ is the posterior mean function.
  Add state action pair to dataset, $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(Z_t, y_t)\}$
  if $|\mathcal{D}_{t+1}| >$ Budget **then**
    Delete some $z_i \in \mathcal{D}_{t+1}$
  end if
  Update model by computing $p(q|\mathcal{D}_{t+1})$
end for

2.6 Scaling Gaussian Process Inference to Large Datasets

To address the $O(N^2)$ time and space complexity of exact inference, where $N = |\mathcal{D}|$, different approximation approaches can be taken to allow GPs to scale to larger datasets. This is especially of concern in RL, where in theory we have a continually growing dataset. See Murphy (2023) for a full overview of the available techniques.

2.6.1 Sparse Variational Gaussian Processes

SVGPs (Titsias, 2009; Hensman, Matthews, & Ghahramani, 2015; Bauer, Van der Wilk, & Rasmussen, 2016; Jankowiak, Pleiss, & Gardner, 2020; Jakkala, 2021; Murphy, 2023) approximate the GP posterior predictive distribution through variational inference. The core idea is to use a set of inducing points $\mathbf{Z} = (\mathbf{z}_1, \dots, \mathbf{z}_M)$ where $M \ll N$, which serve as a sparse approximation of the full dataset. The associated inducing variables are denoted $\mathbf{u} = [f_{\text{GP}}(\mathbf{z}_1), \dots, f_{\text{GP}}(\mathbf{z}_M)]^\top$. The variational posterior is defined as:

$$\hat{p}(\mathbf{f}, \mathbf{u}) = p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z})\hat{p}(\mathbf{u}), \quad \hat{p}(\mathbf{u}) = \mathcal{N}(\mathbf{m}, \mathbf{S}), \tag{2.20}$$

where $p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z})$ is the conditional density of the function values $\mathbf{f} = f_{\text{GP}}(\mathbf{X})$ given train inputs $\mathbf{X}$, inducing points $\mathbf{Z}$ and inducing variables $\mathbf{u}$. $\hat{p}(\mathbf{u})$ is a Gaussian distribution with mean $\mathbf{m}$ and covariance matrix $\mathbf{S}$.

Variational inference aims to **minimize** the Kullback-Leibler (KL) divergence $D_{\mathbb{KL}}(\hat{p} \parallel p)$ between the variational posterior $\hat{p}$ and the true posterior p. Using the variational posterior in (2.20) and the lower bound on the marginal likelihood,

$$\mathbb{E}_{\hat{p}(\mathbf{f},\mathbf{u})}\left[\log \frac{p(\mathbf{y}, \mathbf{f}, \mathbf{u})}{\hat{p}(\mathbf{f}, \mathbf{u})}\right], \tag{2.21}$$

we obtain the evidence lower bound (ELBO):

$$\mathcal{L}_{\text{SVGP}} = \sum_{i=1}^{N} \mathbb{E}_{\hat{p}(f_i)}[\log p(y_i|f_i)] - D_{\mathbb{KL}}(\hat{p}(\mathbf{u}) \parallel p(\mathbf{u}|\mathbf{Z})), \tag{2.22}$$

where $p(y_i|f_i)$ is the Gaussian likelihood for the observations given latent function values. See Appendix A.7 for more details. Since the bound is a sum over the data, an unbiased estimator can be obtained using mini-batch subsampling. The variational parameters $\mathbf{Z}$, $\mathbf{m}$, and $\mathbf{S}$, are estimated by maximizing the lower bound $\mathcal{L}_{\text{SVGP}}$. This approach is guaranteed to converge because $\mathcal{L}_{\text{SVGP}}$ is a lower bound to the MLL, i.e., $\log p(\mathbf{y}|\mathbf{X}) \geq \mathcal{L}_{\text{SVGP}}$.

Posterior predictions for test points are now made by marginalizing over the inducing variables $p(\mathbf{f}_*|\mathcal{D}, \mathbf{X}_*) \approx \int_{\mathbb{R}^M} p(\mathbf{f}_*|\mathbf{u})\hat{p}(\mathbf{u})\, d\mathbf{u}$ which results in another multivariate Gaussian. The resulting mean and variance predictions for a test point $\mathbf{x}_*$ become:

$$\begin{aligned} \mu(\mathbf{x}_*) &= m(\mathbf{x}_*) + \alpha(\mathbf{x}_*)^\top(\mathbf{m} - m(\mathbf{Z})), \\ \sigma^2(\mathbf{x}_*) &= k(\mathbf{x}_*, \mathbf{x}_*) - \alpha(\mathbf{x}_*)^\top(\mathbf{K}_{Z,Z} - \mathbf{S})\alpha(\mathbf{x}_*), \\ \alpha(\mathbf{x}_*) &= \mathbf{K}_{Z,Z}^{-1} k(\mathbf{Z}, \mathbf{x}_*), \end{aligned} \tag{2.23}$$

where $\mathbf{K}_{Z,Z} = k(\mathbf{Z}, \mathbf{Z})$ and $k(\mathbf{Z}, \mathbf{x}_*) = [k(\mathbf{x}_*, \mathbf{z}_1), \dots, k(\mathbf{x}_*, \mathbf{z}_M)]^\top$.

The time complexity for SVGPs is $O(NM^2)$, as it can be shown that the likelihood term in (2.22) can be computed in $O(NM^2)$ time (Murphy, 2023). In terms of storage, we have an $N \times M$ and $M \times M$ covariance matrix, which is in $O(NM + M^2)$.

SVGPs do not overfit with an increasing number of inducing points, and as M increases, the approximation quality of exact inference is recovered. Too few inducing points may make the GP behave as if it was underfitting (Bauer et al., 2016).

2.7 Deep Gaussian Processes

Another drawback of GPs is the inability for their kernel functions to handle structured data where the similarity between two data points require hierarchical feature extraction, which occurs in image data and also some vector datasets (Jakkala, 2021). DGPs seek to address this issue, while still staying in a Bayesian nonparametric framework.

A DGP is a composition of GPs (Damianou &

Lawrence, 2013; Murphy, 2023):

$$\begin{aligned}\mathcal{DGP}(\mathbf{x}) &= \boldsymbol{f}_L \circ \cdots \circ \boldsymbol{f}_1(\mathbf{x}),\\ &\boldsymbol{f}_i(\cdot) = [f^{(1)}_{\mathrm{GP},i}(\cdot), \ldots, f^{(H_i)}_{\mathrm{GP},i}(\cdot)]^\top,\\ &f^{(j)}_{\mathrm{GP},i} \sim \mathcal{GP}(m_i(\cdot), k_i(\cdot,\cdot)).\end{aligned} \tag{2.24}$$

DGPs have a neural network-like structure with L layers, each containing H GPs. Empirical results suggest that DGPs do not overfit as the number of layers increases, even with limited data, and additional layers generally improve performance on large datasets (Salimbeni & Deisenroth, 2017). Salimbeni & Deisenroth (2017) also showed that for the same computational budget, increasing the number of layers can be more effective than increasing the number of inducing points in an SVGP.

One can show that a DGP is strictly more general than a GP (Murphy, 2023), as in a DGP is not just another GP. That said, posterior inference in DGP is quite expensive, as it requires marginalizing over a large number of RVs, corresponding to the hidden function values at each layer. Additionally, the posterior predictive distribution needs to be approximated using Monte Carlo samples, i.e., a finite mixture of Gaussian distributions.

Salimbeni & Deisenroth (2017) addressed the former by using a variational approach similar to the SVGP method in Section 2.6 to allow DGP to scale to larger datasets. This method is called doubly stochastic variational inference, which is the technique used for DGP modeling in this thesis.

The time and space complexity for DGPs using doubly stochastic variational inference is anologous to SVGPs. The ELBO also has a similar form, shown in (A.20), and takes $O(NM^2(D^1 + \ldots + D^L))$ time to compute for N train samples, M inducing points and where D^i is the number of GPs in layer i. Similarly, the space complexity is $O((NM + M^2)(D^1 + \ldots + D^L))$.

3 Methodology

3.1 Training Environments

Since we are comparing value-based algorithms that involve taking an (arg)max over the action space, we are constrained to environments with discrete action spaces. The state spaces are required to be continuous as we aim to compare function approximation methods for the action-value function. Thus, for any environment we can assume $s \in \mathbb{R}^n$ and $|\mathcal{A}| = m$. The `Gymnasium` library (Towers et al., 2023) is used to ensure all algorithms interact with the environments using the same interface.

3.1.1 CartPole

First we have the CartPole environment, as shown in Figure 3.1. The goal is to balance a pole on a cart by applying forces to move the cart left or right. The agent must keep the pole upright for as long as possible. The state space is $\mathcal{S} = \{[x, \dot{x}, \theta, \dot{\theta}] \mid -4.8 \leq x \leq 4.8, -\infty < \dot{x}, \dot{\theta} < \infty, -0.418 \leq \theta \leq 0.418\}$ where x is the position, $\dot{x}$ the velocity, θ the pole angle and $\dot{\theta}$ the pole angle velocity. The action space consists of two actions $\mathcal{A} = \{0, 1\}$, pushing the cart to the left (0) or right (1). The reward function provides a reward of +1 for every time step the pole remains balanced:

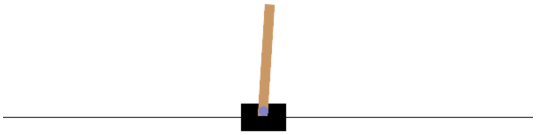

Figure 3.1: Visualization of the CartPole environment.

$$\mathrm{r}(s, a) = \begin{cases} 1, & \text{if episode is not terminated} \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

The episode ends when the cart position $x \notin [-2.4, 2.4]$ and the pole angle $\theta \notin [-0.2095, 0.2095]$, i.e., the cart goes out of bounds or the pole is no longer balanced, or the episode length exceeds 500 time steps. This implies that the maximum return each episode is 500.

3.1.2 Lunar Lander

The goal of the Lunar Lander environment is to successfully land a lunar module on a designated landing pad, visually shown in Figure 3.2. The agent must control the lunar module's thrusters to navigate and stabilize the lander, minimizing its speed and angle to achieve a safe landing. The state space is similar to Cartpole:

$$\mathcal{S} = \left\{ \left(\begin{array}{c} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ c_{\text{left}} \\ c_{\text{right}} \end{array} \right) \left| \begin{array}{l} x \in [-1.5, 1.5], \\ y \in [-1.5, 1.5], \\ \dot{x} \in [-5, 5], \\ \dot{y} \in [-5, 5], \\ \theta \in [-\pi, \pi], \\ \dot{\theta} \in [-5, 5], \\ c_{\text{left}} \in \{0, 1\}, \\ c_{\text{right}} \in \{0, 1\} \end{array} \right. \right\} \tag{3.2}$$

where x, y is the horizontal and vertical position, $\dot{x}, \dot{y}$ the horizontal and vertical velocity, θ is the angle of the lander relative to the vertical axis and $\dot{\theta}$ is the angular velocity. $c_{\text{left}}, c_{\text{right}}$ are 1 if the left/right leg of the lunar module are in contact with the ground, else 0. The action space consists of 4 discrete actions $\mathcal{A} = \{0, 1, 2, 3\}$ with

- 0: Do nothing: No thrusters are fired.
- 1: Fire left orientation engine: Applies a force to the left.

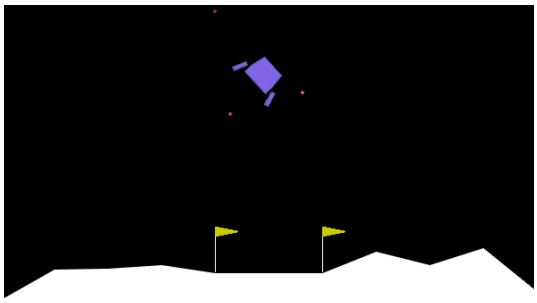

Figure 3.2: Visualization of the Lunar Lander environment.

- 2: Fire main engine: Applies a force upwards.
- 3: Fire right orientation engine: Applies a force to the right.

The reward function is based on proximity to the landing pad, speed, angle, and leg contact. It includes penalties for using engines and rewards for successful landing or penalties for crashing:

$$\mathrm{r}(s,a) = \begin{cases} 100 & \text{successful landing,} \\ -100 & \text{if lander crashes,} \\ -0.3 & \text{for each time step,} \\ -0.03(\dot{x}^2 + \dot{y}^2) & \text{penalty for speed,} \\ -0.3|\theta| & \text{penalty for angle.} \end{cases} \tag{3.3}$$

An episode is considered a solution if the obtained reward is at least 200.

The episode ends if the lander is motionless or not awake, crashes, moves out of bounds, or lands successfully.

3.2 Preprocessing

The type of normalization that was performed is standardization, which means scaling the data such that it has zero mean and unit variance:

$$\mathbf{x}_{\text{norm}} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{\sigma_{\mathbf{x}}^2 + \eta}}, \tag{3.4}$$

where $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}^2$ are the mean and variance of $\mathbf{x}$ and η is a small constant (by default `1e-08`) to prevent division by zero errors.

One exception is that for the Lunar Lander environment we only normalized the first 6 dimensions. As c_{left} and c_{right} are booleans.

3.3 Algorithm Design

The GP-Q and GP-SARSA algorithm were adopted for the off-policy and on-policy case respectively with the following modifications: The updates were done in batches, instead of per timestep. Meaning, that for a batch size B, we wait B timesteps before updating the GP. This was done to reduce computational complexity and improve sampling efficiency. Furthermore, this aimed to prevent the inaccurate estimate of the q-function from changing too much per timestep.

Algorithm 3.1 Adjusted GP-Q/GP-SARSA for estimating $p(q_*|\mathcal{D})$

Initialize:
  Collect initial dataset $\mathcal{D}_0 = (z_i, y_i)_{i=1,\dots,n_0}$ with $z_i \in \mathcal{S} \times \mathcal{A}$, $y_i \in \mathbb{R}$
  Initialize GP with $p(q|\mathcal{D}_0)$, noise variance σ_y^2, and hyperparameters $\boldsymbol{\theta}$
  Set $\boldsymbol{\theta}'_n = [\boldsymbol{\theta}, \sigma_y^2]^\top$ and batch counter $b = 0$
for each time step t **do**
  Choose action A_t from state S_t using a behavioral policy (e.g., ϵ-greedy, UCB, Thompson sampling)
  Take action A_t, observe reward R_{t+1} and next state S_{t+1}
  Increment batch counter: $b = b + 1$
  if Off-Policy **then**
    $y_t = R_{t+1} + \gamma \max_{a'} \bar{q}(S_{t+1}, a')$
  else {On-Policy}
    $A_{t+1} = \pi(S_{t+1})$
    $y_t = R_{t+1} + \gamma \bar{q}(S_{t+1}, A_{t+1})$
  end if
  Update dataset:
    Form state-action pair $Z_t = [S_t, A_t]^\top$
    Add (Z_t, y_t) to dataset: $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(Z_t, y_t)\}$
  if $|\mathcal{D}_{t+1}| >$ Budget **then**
    Delete oldest (z_i, y_i) from $\mathcal{D}_{t+1}$
  end if
  if $b ==$ batch size **then**
    Update GP model, $p(q|\mathcal{D}_{n+1})$**:**
      Minimize negative MLL or negative ELBO using stochastic gradient descent w.r.t. $\mathcal{D}_{n+1}$
      Update hyperparameters to $\boldsymbol{\theta}'_{n+1}$
      Reset batch counter: $b = 0$
  end if
end for

Initially, $\mathcal{D}_0 = \emptyset$. Updating the GP, i.e., computing $p(q|\mathcal{D}_{n+1})$, should be interpreted as performing type-II MLE of the hyperparameters and observation noise variance[†] $\boldsymbol{\theta}' = [\boldsymbol{\theta}, \sigma_y^2]^\top$, and then computing the predictive posterior distribution where the test inputs are the state-action pairs of the current batch. In the case of SVGP/DPGs we also have the ELBO with variational parameters $\mathbf{Z}, \mathbf{m}, \mathbf{S}$ as hyperparameters.

To estimate $\boldsymbol{\theta}'_{\text{opt}}$, which minimizes the negative MLL (2.15) or the negative ELBO (2.22) (A.20)

[†]In DGPs we also have noise variance between layers.

we utilized the Adam optimizer (Kingma & Ba, 2017) for stochastic gradient-based optimization. It was implicitly assumed that any hyperparameter values after optimization carry over to the next GP update.

As explained in Section 2.6 SVGPs allow you to perform mini-batching to compute the ELBO. This helps keep memory usage under control as SVGPs permit much greater dataset sizes. There are two approaches to mini-batching here that are reasonable in terms of time complexity. Either you mini-batch over the entire dataset while assuming the dataset is relatively small, in the hope that the variational parameters retain information from older thrown away data points. Or you keep a bigger dataset and optimize over a random subset of mini-batches every time. In the case of the former the SVGP objective becomes:

$$\mathcal{L}_{\text{SVGP}} = \left[\frac{N}{B}\sum_{b=1}^{B}\frac{1}{|\mathcal{B}_b|}\sum_{n\in\mathcal{B}_b}\mathbb{E}_{\hat{p}(f_n)}[\log p(y_n|f_n)]\right] - D_{\mathbb{KL}}(\hat{p}(\mathbf{u}) \parallel p(\mathbf{u}|\mathbf{Z})), \tag{3.5}$$

where $\mathcal{B}_b$ is the b'th batch, and B is the number of batches.

Regarding the removal and addition of data points; data points were added unconditionally in a first-in-first-out (FIFO) manner. If the maximum dataset size (budget) is exceeded then the oldest data points were removed. It is worth mentioning that for SVGPs/DPGs it is feasible to have a large dataset size of over $100,000$ data points while for exact GPs there are actual VRAM limitations for dataset sizes greater than 1000.

The following behavioral policies were considered: ϵ-greedy, UCB and Thompson sampling. UCB was implemented by taking (2.16) with $\mathbf{x} \in \mathcal{S} \times \mathcal{A}$. Thompson sampling was extended from the Bandit case by selecting the highest value action from the sample q-function for a fixed state $s \in \mathcal{S}$. This was done by considering the points $\{(s, a) \mid a \in \mathcal{A}\}$ sampling from the GP latent RVs, $\{f_{\text{GP}}(s, a) \mid a \in \mathcal{A}\}$, and selecting the action a for for which the sample q-value is highest.

All the modifications made to the base GP-Q/GP-SARSA algorithm are summarized in algorithm 3.1.

To answer the hypothesis, we took as baseline the random policy, and compared the GP based RL algorithms to two variants of Stable-Baselines3's Deep Q-network (DQN) algorithm (Raffin et al., 2021). One variant which used a linear function approximator, and the other which used a multilayer perceptron (MLP) whose shape is as described in Section 3.4.2.

3.3.1 Complexity Analysis of the GP-Q and GP-SARSA Algorithm Variants

The time and space complexity of GP-Q and GP-SARSA mainly depend on the GP model and how the GP is updated with the dataset. As they only differ in computing the TD target, their complexities are the same. These are summarized in Table 3.1.

3.4 Model Architectures

The computational backend for all the models is `PyTorch` (Paszke et al., 2019). For the GP models, we used `GPytorch` (Gardner et al., 2018), a high-performance GPU-accelerated library for GP modeling, in conjunction with `Botorch` (Balandat et al., 2020), a Bayesian optimization library that extends `GPytorch`. An NVIDIA RTX 4090 with 24 GB of VRAM was used for GPU acceleration.

We were also interested in measuring VRAM usage on the GPU, energy consumption, and execution time. For VRAM usage, we used `pynvml`, a Python interface for the NVIDIA Management Library. For energy consumption and execution time, we used the `Zeus` library (You, Chung, & Chowdhury, 2023).

3.4.1 Gaussian Processes and Kernel Selection

For the choice of GP model, we primarily considered DGPs, which reduces to an SVGP when using a single unit.

In Section 2.3, the derivations assumed an observation noise variance parameter σ_y^2. In the RL setting, it is hard to estimate the exact value of σ_y^2. To address this, instead of setting σ_y^2 a priori, we infered it alongside the hyperparameters by including σ_y^2 in the optimization process when minimizing the negative MLL in (2.15) or the negation of the ELBO in (2.22) and (A.20).

Regarding the choice of kernel for the states, existing literature on GPs and RL have used the RBF (2.13) or Matern (2.12) kernel (Chowdhary et al., 2014; Kameda & Tanaka, 2023). This implies that those authors had as prior belief that the action-value function is reasonably smooth. We assumed that this assumption was reasonable. Among the two, the RBF kernel was used.

Regarding Engel et al. (2005)'s suggestion to use a separate kernel for the states and action is sensible, but this causes issues with sparse variational methods when learning the inducing points $\mathbf{Z} \in \mathbb{R}^{Md}$ using gradient methods (Murphy, 2023). The actions are discrete, but the problem is that the optimizer that optimizes the inducing points $\mathbf{z}_i \in \mathcal{S} \times \mathcal{A}$ does not take this information into account. Using a kernel for categorical features

Table 3.1: Time and space complexity of the GP-Q and GP-SARSA algorithm's GP update for different GP models, assuming no mini-batching for uniformity. Here N is the size of the dataset and M are the number of inducing points. D^i is the number of GPs in layer i.

GP-Model	Time Complexity	Space Complexity
Exact GP (with BBMM)	$O(N^2)$	$O(N^2)$
Sparse Variation GP (SVGP)	$O(NM^2)$	$O(NM + M^2)$
Deep GP (DGP)	$O(NM^2(D^1 + \ldots + D^L))$	$O((NM + M^2)(D^1 + \ldots + D^L))$

introduces issues since the actions may not be exact integers anymore. For this reason, we used the approach by Chowdhary et al. (2014); Kameda & Tanaka (2023), and simply used one kernel for the states and actions.

Using a constant mean function in DGPs makes each GP mapping highly non-injective, leading to issues with the DGP prior (Duvenaud, Rippel, Adams, & Ghahramani, 2014). Following Salimbeni & Deisenroth (2017), we used a linear mean function, $m(\mathbf{X}) = \mathbf{XW}$, for all hidden layers. If input and output dimensions match, $\mathbf{W} = \mathbf{I}$; otherwise, $\mathbf{W}$ is set to the top D^l left eigenvectors from the data's singular value decomposition.

3.4.2 Neural Network Models

The linear model was treated as a single-layer feed-forward network with an identity activation function:

$$\mathbf{y} = \mathbf{W}s + \mathbf{b}, \tag{3.6}$$

with $\mathbf{W} \in \mathbb{R}^{n \times m}$ and $\mathbf{b} \in \mathbb{R}^m$.

Regarding the choice of function approximator for the DQN algorithm, we used the following MLP as shown in Table 3.2.

Note the difference with how the q-function is modeled compared to a GP. With a neural network we have as input the state and an output neuron for each action, containing $\hat{q}(s, a)$. This is different for a GP, where the input is a state-action pair $z \in \mathcal{S} \times \mathcal{A}$, and the output is a Gaussian with a mean and variance for the action-value.

3.5 Experimental Setup

3.5.1 Hyperparameter Tuning

Hyperparameter selection for DQN was based on pre-tuned settings from the `Stable-Baselines3 Zoo` GitHub repository (Raffin, 2020). Since the ϵ-greedy schedule in `Stable-Baselines3` is based on timesteps, we adjusted to compare policies over the same number of episodes by running DQN for the recommended timesteps, recording the episodes passed, and then running the remaining episodes without training.

To estimate expected resource consumption, time and space, we analyzed how regular GPs scale on a dataset $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1,\ldots,N}, \mathbf{x}_i \in \mathbb{R}^4, y_i \in \mathbb{R}$ (64-bit floating point) that simulated the samples collected for CartPole, as $N \in \{100, 500, 1000, 3000, 5000, 8000, 10000\}$ increases. Additionally, we analyzed resource consumption for an SVGP with a fixed $N = 10000$ dataset size but with increasing inducing points $M \in \{50, 200, 350, 500, 650, 800, 950\}$. The data collected from this procedure is averaged over 10 trials.

For GP-Q/GP-SARSA, due to the experimental nature of the algorithm and the computational complexity of certain hyperparameter settings, informal testing was performed on both CartPole and Lunar Lander to identify effective hyperparameter settings. This involved examining the reward curve on runs with different configurations. We tuned GP fitting settings, such as the initial learning rate of the Adam optimizer and how optimization over the dataset was performed at each GP update.

The hyperparameters used in the experimenal setup are summarized in Table D.1. A moderate dataset size of 10,000 or 20,000, with each GP update involving a random subset of approximately 3,500 samples, works well when optimized in mini-batches. Using a small learning rate (0.001 or 0.005) for the Adam optimizer ensures that hyperparameters are not too tightly fitted on early inaccurate q-value estimates. However, the learning rate should not be too low to allow the agent to learn a meaningful policy. The discount factor γ was set to 0.99, the same as with the DQN algorithm. The selected behavioral policy chosen was Thompson sampling, but a small comparison was made to ϵ-greedy and UCB in Lunar Lander.

What was tested more rigorously is the performance difference going from an SVGP to a DGP. According to Salimbeni & Deisenroth (2017), a DGP with a relatively small (~ 100) number of inducing points generally outperforms a single-layer DGP/SVGP with a larger number of inducing points (~ 500) on regression and classification.

Increasing the number of inducing points and units improves performance, but there is a point of diminishing returns. We adopted a similar approach to Salimbeni & Deisenroth (2017), validating performance using an SVGP with 512 inducing points and four-layer DGP with 128 inducing points per unit. For the number of units per layer in the DGP, we use $\min(30, D^0)$ for all inner layers, where

Table 3.2: Summary of neural network parameters. n is the input state dimensionality and m is the number of actions.

Layer (type)	Output Shape	Param #
Linear (layer1) + ReLU	[batch_size, 256]	$n \times 256$
Linear (layer2) + ReLU	[batch_size, 256]	256×256
Linear (layer3)	[batch_size, m]	$256 \times m$
Total Parameters		$n \times 256 + 65,536 + 256 \times m$

D^0 is the input dimensionality, and the same RBF kernel is used for all layers.

Learning was stopped for GP-Q/GP-SARSA if a close-to-optimal policy was found, defined as achieving a return of 500 for 5 consecutive episodes in CartPole and a return of ≥ 200 for 5 consecutive episodes in Lunar Lander.

3.5.2 Experiment Setup

We compared GP-Q with DQN using both an MLP and a linear model, as well as a random policy during both training and evaluation. In Lunar Lander we also compared with GP-SARSA. The expectation was that both GP-Q and GP-SARSA will behave similarly and so it is sufficient to mainly compare with GP-Q. All algorithms were trained for 1000 episodes on Cartpole and 3000 episodes on Lunar Lander. Evaluation was performed for 30 episodes. The episode counts were estimated such that all algorithms would converge.

This comparison involved examining the reward curve/return graph for policy convergence and performance, including the maximum return obtained, and measuring computational complexity and energy usage during training and evaluation[‡]. Time complexity was measured by considering execution time during training and evaluation. Space complexity was measured by examining VRAM usage during training. Energy usage was measured in kilojoules (kJ) during training and evaluation. Additionally, we measured the average negative ELBO for each environment over the number of GP updates.

At evaluation time, we also compared the means of returns. A two-sample one-tailed Wilcoxon rank-sum test was performed to compare the average return between a specific RL agent and a random policy after training. The null hypothesis was $H_0 : \mathbb{E}_{\pi_{\text{agent}}}[G] = \mathbb{E}_{\pi_{\text{random}}}[G]$ and the alternative was $H_A : \mathbb{E}_{\pi_{\text{agent}}}[G] > \mathbb{E}_{\pi_{\text{random}}}[G]$, where G is the undiscounted sum of rewards over an episode.

We also measured the stability of the GPQ with a DGP by examining how many times out of C_r runs of at least C_e episodes the algorithms performed better than random. For CartPole $C_r = 10$, $C_e = 200$ and Lunar Lander $C_r = 7, C_e = 1000$. If an algorithm is stable, we expect the same hyperparameter configuration and environment to produce consistent results w.r.t. random in most runs.

Finally, we recorded the agent environment interaction after training and described the general behavior in cases where it was insightful[§].

[‡]As GP-Q and GP-SARSA are the same, except for the computation of the TD target, such measurements were omitted from GP-SARSA for brevity.

4 Results

The following showcases the results from the experiment described in Sections 3.5.1 and 3.5.2. The results for GP scaling on toy datasets can be found in Appendix B.2. Comparison of exploration methods is found in Appendix B.1.

It is important to note that the hyperparameter configuration for GP-Q/GP-SARSA is *not* necessarily optimal. Meanwhile, the DQN hyperparameter settings are optimal according to empirical results from Raffin (2020). The results for each algorithm were based on a single run, and each run can vary slightly in terms of the learned policy and its convergence. This approach is acceptable if the deviations between runs are not significant, which also helps save on computational costs. It is also not uncommon in RL research to follow this practice (Kameda & Tanaka, 2023).

Based on the hyperparameter tuning that was performed, we can generally assume that GP-Q/GP-SARSA is reasonably consistent on Lunar Lander. However, this does not hold for CartPole. Here, the policies can converge too early, resulting in a policy that is, at best, as good as a random policy, and performance can degrade as training continues. The latter issue is mitigated by the early stop reward condition.

4.1 GP-Q vs GP-SARSA

GP-Q and GP-SARSA appear to have similar performance characteristics. There is at the very least no case where GP-SARSA clearly outperforms GP-Q.

[§]Select recordings are available on the GitHub page.

Algorithm	Train (1000 episodes)		Eval (30 episodes)		
	Avg. Return	Max Return	Avg. Return	Max Return	Sig. BTR
DQN (MLP)	266.43 ± 219.66	500	451.53 ± 147.89	500	True
GP-Q (DGP)	$\mathbf{326.34 \pm 191.95}$	**500**	$\mathbf{222.9 \pm 157.24}$	**500**	**True**
GP-Q (SVGP)	$\mathbf{270.48 \pm 208.33}$	**500**	$\mathbf{276.60 \pm 209.82}$	**500**	**True**
RANDOM	22.16 ± 11.83	96	22.23 ± 11.27	52	NA
DQN (Linear)	12.01 ± 5.19	55	9.33 ± 0.99	12	False

Table 4.1: Reward metrics for CartPole. Optimal reward per episode: 500. Sig. BTR: significantly better (avg. return) than random (Wilcoxon test, $p < 0.05$).

Algorithm	Train (1000 episodes)			Eval (30 episodes)	
	Time (s)	Memory (GB)	Energy (kJ)	Time (s)	Energy (kJ)
GP-Q (DGP)	**5995.15**	**1.80**	**252.07**	**121.11**	**4.29**
GP-Q (SVGP)	**2116.31**	**0.566**	**16.11**	**40.02**	**1.57**
DQN (MLP)	64.01	0.414	2.15	5.48	0.16
DQN (Linear)	12.01	0.414	0.49	0.29	0.0087

Table 4.2: Time, memory and energy usage on CartPole.

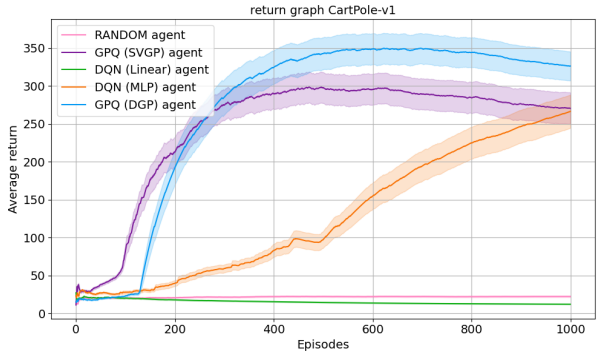

Figure 4.1: Comparison of cumulative average return ($\pm 0.1 \cdot$ std) in CartPole.

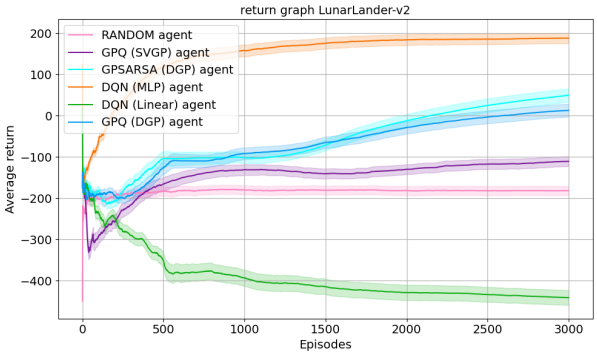

Figure 4.2: Comparison of cumulative average return ($\pm 0.1 \cdot$ std) in Lunar Lander.

4.2 CartPole

The reward curve is displayed in Figure 4.1, reward metrics in Table 4.1, and resource consumption in Table 4.2. GPQ (SVGP) stopped learning around episode 150, GPQ (DGP) around episode 170, and DQN (MLP) around episode 450.

The general behavior of GPQ and DQN (MLP) was similar; both were able to balance the pole for the duration of the episode most of the time, with DQN achieving this more often. They achieved balance by moving the cart left and right. Specifically, the cart was moved left, causing the pole to move slightly to the right. Before the pole lost its balance, the cart was moved right, causing the pole to move slightly to the left, and this process continues. One behavior observed in GPQ (SVGP) but not in GPQ (DGP) was that sometimes the agent stops balancing and moves off-screen to the right or left.

DQN (Linear) failed to learn any meaningful policy for balancing the pole, resulting in the pole losing its balance immediately.

4.3 Lunar Lander

The reward curve is displayed in Figure 4.2, reward metrics in Table 4.3, and resource consumption in Table 4.4. DQN (MLP) stopped learning around episode 600. For GPQ (DGP) and GP-SARSA (DGP), the reward stopping condition was reached around episode 1500. GPQ (SVGP) did not reach the stopping condition and converged to a sub-optimal policy.

DQN (MLP) typically landed the module in the designated landing spot reasonably quickly. Right before landing, an upward force was applied to slow down, and upon landing, the right or left engines were used for corrections. GP-Q (DGP) also landed the module reasonably well. GP-Q (DGP) kept the module upright and typically landed it successfully, though it applied more upward force, resulting in a more gradual and slow landing compared to DQN, which almost free-falled initially. GP-Q (SVGP) did not control the module as well as GP-Q (DGP) and as a result was not as successful in landing the module at the landing spot.

Algorithm	Train (3000 episodes)		Eval (30 episodes)		
	Avg. Return	Max Return	Avg. Return	Max Return	Sig. BTR
DQN (MLP)	188.17 ± 126.85	323.55	177.58 ± 120.07	312.30	True
GP-Q (DGP)	$\mathbf{13.09 \pm 156.93}$	**316.62**	$\mathbf{104.06 \pm 128.34}$	**271.29**	**True**
GP-SARSA (DGP)	$\mathbf{49.69 \pm 164.58}$	**315.95**	$\mathbf{-56.73 \pm 188.40}$	**273.29**	**True**
GP-Q (SVGP)	$\mathbf{-110.67 \pm 121.80}$	**269.63**	$\mathbf{-99.61 \pm 85.71}$	**142.78**	**True**
RANDOM	-181.95 ± 110.02	80.22	-191.07 ± 104.94	-71.85	NA
DQN (Linear)	-441.17 ± 178.65	33.8	-539.79 ± 156.03	-186.96	False

Table 4.3: Reward metrics for Lunar Lander. Optimal reward per episode: $\geq$ 200. Sig. BTR: significantly better (avg. return) than random (Wilcoxon test, $p < 0.05$).

Algorithm	Train (3000 episodes)			Eval (30 episodes)	
	Time (s)	Memory (GB)	Energy (kJ)	Time (s)	Energy (kJ)
GP-Q (DGP)	**27349.92($\approx 7h$)**	**2.35**	**2134.39**	**135.99**	**4.87**
GP-Q (SVGP)	**9568.53($\approx 3h$)**	**0.936**	**645.86**	**39.23**	**1.16**
DQN (MLP)	349.91	0.678	7.92	4.22	0.093
DQN (Linear)	124.44	0.02	2.79	1.95	0.042

Table 4.4: Time, memory and energy usage on Lunar Lander.

DQN (Linear) did not learn any meaningful policy. Typically, the lunar module flipped and crashed.

4.4 Algorithm Stability

For CartPole, out of 10 runs, the success rates were 100% for DQN (MLP), 40% for GP-Q (DGP) and 0% for DQN (Linear). For Lunar Lander out of 7 runs, the success rate were 100% for DQN (MLP), 100% for GP-Q (DGP) and 0% for DQN (Linear).

5 Discussion

The goal of this thesis was to compare a GP-based RL algorithm with linear and NN-based function approximations in two environments, using DQN with a linear model and an MLP. This aimed to highlight the advantages and limitations of GPs in RL and their position among function approximators in RL.

The findings from the simulations, as presented in Section 4, align with the hypothesis. The current GP-based algorithm is able to significantly outperform DQN using linear function approximation, but not DQN using an MLP.

In CartPole, GP-Q using at least an SVGP is able to approach a close-to-optimal policy. DQN with linear function approximation fails to learn any meaningful policy on any of the environments. In fact, learning often diverges, and the resulting policy is worse than random. Although GP-Q can perform quite well in CartPole, training can be unstable, occasionally converging to a sub-optimal policy. Worst case, an always go right or left policy, which is more common when using an SVGP. A possible explanation is that the uninformative reward function, which gives +1 reward for every timestep until the episode terminates, may bias the agent to one action, which is then propagated at each update. That said, GP-Q can converge to a close-to-optimal policy much faster than DQN, and this is without the need for any random action selection.

In contrast, Lunar Lander's results are more stable and outperform DQN with linear function approximation, though not DQN with an MLP. Here, DQN (Linear) also performs worse than random, suggesting that linear regression is unsuitable for both CartPole and Lunar Lander. This is unsurprising, as there is likely no close-to-linear relationship between state-action features and the associated action-value[¶].

The primary difference between GP-Q (DGP) and DQN (MLP) in Lunar Lander is that GP-Q takes more episodes to achieve a similar policy, typically with a return of ≥ 200 on most episodes, and this result is reasonably stable across runs, although the policy learned by GP-Q might be slightly less optimal. This stability could be attributed to the more informative reward function. It takes around 1000 more episodes for GP-Q/GP-SARSA to reach a close-to-optimal policy compared to CartPole, which may be due to the larger state space. This does bring into questions how well GP-Q/GP-SARSA converge for increasingly large state and action spaces. At the very least, benchmarks from Salimbeni & Deisenroth (2017) did suggest DGPs are able to handle reasonably high-dimensional data.

In any case, it is clear that DGPs generally produce better policies, yielding higher rewards, compared to SVGPs in both environments. The performance uplift going from an SVGP to a DGP

[¶]With random action selection, we mean uniform sampling on the action space.

could be explained by DGPs ability to handle less smooth/discontinuous functions much better, even when each unit uses an RBF kernel (see Appendix B.5). It is sensible to believe that the q-functions resulting from a space $\mathcal{S} \times \mathcal{A}$ where $\mathcal{A}$ is discrete may not be entirely smooth.

In both Lunar Lander and CartPole, the performance of the GP-Q/GP-SARSA algorithm may degrade as training continuous. Specifically, the close-to-optimal policy achieved at the reward threshold is often better than one trained for a longer number of episodes. This result might seem counterintuitive, as one would expect the TD(0) targets to converge to the true q-values. However, this decline can be attributed to a phenomenon known as forgetting: As the GP is updated with newer samples, earlier samples from previous episodes are increasingly excluded from the training process. A reasonable alternative to reward thresholding is to use a learning rate schedule for the optimizer that decays over time.

One interesting property of GP-Q/GP-SARSA is that Thompson sampling appears to automatically allow the agent to balance exploration and exploitation without the need for random action selection. It is possible that this introduces some stochasticity in the final policy, though as the uncertainty in the q-value estimates decreases, the policy would become more deterministic, as can be seen in Appendix B.4. It is possible that some areas with moderate uncertainty remain. This can be observed in the results for CartPole, where not every episode reaches the maximum possible reward. That being said, in more complex environments, this stochasticity could be more of an advantage.

The loss (negative ELBO) over time was also recorded as shown in Figure B.4. In general, SVGPs and DGPs exhibit similar convergence characteristics. For CartPole, the loss remains relatively constant and low at each update, while for Lunar Lander, the loss decreases more gradually. The lack of increases in loss as new datapoints are added suggests that the added datapoints are related, which makes sense given the fact we are computing TD(0) targets, but it may also indicate redundancy in the datapoints. This redundancy is partially alleviated by using inducing point methods, where the inducing points and associated inducing variables act as a summary of the data.

Regarding computational complexity, the results indicate the following ranking: DQN (Linear) < DQN (MLP) < GP-Q (SVGP) < GP-Q (DGP). The extended training times for the GP-based algorithms are due to two main factors: computing the posterior distribution and sampling q-values is more resource-intensive than performing a forward pass in an MLP. Additionally, calculating the ELBO at each update is more costly compared to the mean squared error objective of DQN (A.21).

GP-Q (DGP) especially has quite a significant time cost in Lunar Lander, which can be significantly reduced by reducing the number of units or number of inducing points. GP-Q training can also further be optimized by balancing dataset fitting frequency, considering the quick convergence of the loss it is possible that the GP update frequency can be decreased. However, updates that are too infrequent may hinder the agent's learning. It is important to note that GP-Q/GP-SARSA remains computationally more intensive even during inference even when using just a single SVGP, as evidenced by the evaluation results. Therefore, despite potential optimizations in the training phase, GP-Q/GP-SARSA still incurs higher computational time and energy costs. Nonetheless, ongoing advancements may lead to improvements in efficiency in the future.

There is also a clear correlation between computational complexity and energy usage, where higher computational demands typically lead to increased energy consumption. While this may not be surprising, overall energy usage provides a useful metric to better understand the effects of increased time and memory requirements.

One important point to discuss when it comes to this comparison of function approximation methods is that a large reason DRL algorithms like DQN perform so well is not just related to the choice of function approximator, but also numerous algorithmic "tricks" that help stabilize learning. For instance, DQN uses experience replay to break the correlation between consecutive experiences and target networks to provide stable learning targets. Additionally, techniques such as entropy regularization, adaptive learning rate schedules, gradient clipping, and batch normalization play crucial roles in ensuring stable, efficient, and generalizable learning. These "tricks", combined with function approximation methods, drive the success of modern DRL algorithms. In comparison, the proposed nonparametric algorithm using GPs is not quite as mature. DGPs are close to the state-of-the-art in GP modeling, but the base GP-Q/GP-SARSA algorithm is relatively simple, maintaining TD(0) targets, which are used to periodically update the GP.

6 Conclusions

6.1 Future Works

An important next step is extending the proposed algorithm to continuous action spaces. Currently, we are limited to discrete actions due to the need to compute an argmax over the action space. This could be achieved by integrating GPs into an actor-

critic framework. According to Lockwood & Si (2022), uncertainty quantification in RL has mainly focused on the critic (value functions) because it is directly affected by aleatoric (data) uncertainty and this uncertainty propagates to the actor through updates of the form $\nabla_{\boldsymbol{\theta}} J = \mathbb{E}[\nabla_{\boldsymbol{\theta}} \log(\pi_{\boldsymbol{\theta}})\delta]$, where δ depends on the critic.

There are also some unexplored extensions of the proposed algorithm. Currently, the algorithm uses the TD(0) target to estimate the action-value function. This approach can be generalized to the λ-return. This may help reduce bias in the q-value estimates at the cost of computational complexity (R. Sutton & Barto, 2018).

Another important point to consider is how to manage the dataset of state-action-q-value pairs in the GP-Q/GP-SARSA algorithm. It was demonstrated that for SVGPs/DGPs the dataset size can be set large enough such that a policy can be learned. Still, there may be redundant data points, so exploring information-theoretic methods could help reduce redundancy and improve the approximation quality of the inducing points.

Additionally, the GP-Q/GP-SARSA algorithm would also be well-suited for offline settings where a dataset of q-value estimates is available a priori. In terms of pre-processing, the primary focus was on normalization, but there is nothing stopping one from applying other techniques such as clustering or dimensionality reduction to optimize dataset construction.

Kernel selection is another important consideration. Existing literature using GPs in RL (Engel et al., 2005; Chowdhary et al., 2014; Kameda & Tanaka, 2023) has mainly used RBF or Matern kernels. However, a variety of kernels exists, such as the spectral mixture kernel, detailed in Appendix A.3, which can approximate any stationary kernel to arbitrary precision (Murphy, 2023).

Another approach is deep kernel learning (Wilson, Hu, Salakhutdinov, & Xing, 2016a,b), which combines neural networks with GPs by using a neural network as a feature extractor before applying the GP. However, a GP atop a deep network remains a GP, while deep GPs are strictly more general (Murphy, 2023). Additionally, deep kernel learning with highly parameterized neural networks introduce the well-known downsides of deep learning, such as the need for explicit regularization (Salimbeni & Deisenroth, 2017). In contrast, DGPs learn a representation hierarchy nonparametrically with a relatively small number of hyperparameters to optimize.

Regarding DGPs, Jankowiak et al. (2020) have proposed an alternative parametric model called deep sigma point process (DSPP). This model retains many properties of DGPs without needing to approximate the predictive posterior distribution. Empirical results suggest that DSPPs achieve better-calibrated predictive distributions and can outperform deep kernel learning and DGPs on univariate and multivariate regression tasks. For this reason, it is an important model to consider for future research in GPs and RL.

GPs are not the only models that offer uncertainty quantification; they have a close connection to Bayesian neural networks, a parametric model that learn a distribution over the weights. For future work in uncertainty quantification in RL, it is crucial to compare different methods of uncertainty quantification to GPs.

Lastly, steps need to be taken to reduce the overall time complexity, as there currently is a noticeable difference in training and inference speed compared to DQN. Ongoing research can tackle this from two angles: Make the algorithm itself more efficient, or work on developing a faster GP model, particularly in an online setting.

6.2 Conclusion

The thesis aimed to compare a GP-based RL algorithm with linear and neural network function approximators, specifically against DQN using linear and MLP models. The goal was to assess the strengths and limitations of GPs in RL.

Findings from simulations[‖] align with the hypothesis: GP-based algorithms (GP-Q/GP-SARSA), particularly using SVGPs and DGPs, outperform linear function approximation in CartPole and Lunar Lander. However, they do not match the stability of DQN with an MLP in CartPole or the overall performance in Lunar Lander. Additionally, GP-Q and GP-SARSA are more computationally expensive, even during inference.

Utilizing uncertainty quantification, GP-based agents via Thompson sampling automatically balance exploration and exploitation, unlike DQN which relies on random action selection.

These results underscore the potential of GPs, particularly DGPs, as function approximators in RL tasks requiring uncertainty quantification and interpretability, such as safe RL, where understanding the confidence in predictions can mitigate risks and ensure robust decision-making.

References

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2020). BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in neural information processing systems*

[‖] For the source code of the project, refer to `https://github.com/matthjs/BachelorProject`

33. Retrieved from http://arxiv.org/abs/1910.06403

Bauer, M., Van der Wilk, M., & Rasmussen, C. E. (2016). Understanding probabilistic sparse gaussian process approximations. *Advances in neural information processing systems*, *29*. Retrieved from https://arxiv.org/abs/1606.04820

Berkenkamp, F. (2019). *Safe exploration in reinforcement learning: Theory and applications in robotics* (Doctoral dissertation, ETH Zurich). Retrieved from https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/370833/root.pdf

Bishop, C. M. (2006). Kernel methods. In *Pattern recognition and machine learning* (pp. 291–325). Berlin, Heidelberg: Springer-Verlag.

Bishop, C. M., & Bishop, H. (2024). Bayesian machine learning. In S. Cham (Ed.), *Deep learning - foundations and concepts* (1st ed., pp. 54–57). Springer. doi: doi:https://doi.org/10.1007/978-3-031-45468-4

Chowdhary, G., Liu, M., Grande, R., Walsh, T., How, J., & Carin, L. (2014). Off-policy reinforcement learning with gaussian processes. *IEEE/CAA Journal of Automatica Sinica*, *1*(3), 227–238. doi: doi:10.1109/JAS.2014.7004680

Churchill, R. V., Bochner, S., Tenenbaum, M., & Pollard, H. (1960, 10). Lectures on Fourier Integrals. *The American mathematical monthly*, *67*(8), 819. doi: doi:10.2307/2308692

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms, third edition* (3rd ed.). The MIT Press.

Cressie, N. (1990). The origins of kriging. *Mathematical geology*, *22*, 239–252. doi: doi:10.1007/BF00889887

Damianou, A., & Lawrence, N. D. (2013, 29 Apr–01 May). Deep Gaussian processes. In C. M. Carvalho & P. Ravikumar (Eds.), *Proceedings of the sixteenth international conference on artificial intelligence and statistics* (Vol. 31, pp. 207–215). Scottsdale, Arizona, USA: PMLR. Retrieved from https://proceedings.mlr.press/v31/damianou13a.html

Duvenaud, D., Rippel, O., Adams, R., & Ghahramani, Z. (2014, 22–25 Apr). Avoiding pathologies in very deep networks. In S. Kaski & J. Corander (Eds.), *Proceedings of the seventeenth international conference on artificial intelligence and statistics* (Vol. 33, pp. 202–210). Reykjavik, Iceland: PMLR. Retrieved from https://proceedings.mlr.press/v33/duvenaud14.html

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on machine learning* (p. 201–208). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1102351.1102377 doi: doi:10.1145/1102351.1102377

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., & Wilson, A. G. (2018). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 31). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2018/file/27e8e17134dd7083b050476733207ea1-Paper.pdf

Hensman, J., Matthews, A., & Ghahramani, Z. (2015, 09–12 May). Scalable Variational Gaussian Process Classification. In G. Lebanon & S. V. N. Vishwanathan (Eds.), *Proceedings of the eighteenth international conference on artificial intelligence and statistics* (Vol. 38, pp. 351–360). San Diego, California, USA: PMLR. Retrieved from https://proceedings.mlr.press/v38/hensman15.html

Jakkala, K. (2021). Deep gaussian processes: A survey. *CoRR*, *abs/2106.12135*. Retrieved from https://arxiv.org/abs/2106.12135

Jankowiak, M., Pleiss, G., & Gardner, J. (2020, 03–06 Aug). Deep sigma point processes. In J. Peters & D. Sontag (Eds.), *Proceedings of the 36th conference on uncertainty in artificial intelligence (uai)* (Vol. 124, pp. 789–798). PMLR. Retrieved from https://proceedings.mlr.press/v124/jankowiak20a.html

Kameda, K., & Tanaka, F. (2023). Reinforcement learning with gaussian process regression using variational free energy. *Journal of Intelligent Systems*, *32*(1), 20220205. Retrieved 2024-07-22, from https://doi.org/10.1515/jisys-2022-0205 doi: doi:doi:10.1515/jisys-2022-0205

Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization.* Retrieved from https://arxiv.org/abs/1412.6980

Lockwood, O., & Si, M. (2022, Oct.). A review of uncertainty for deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *18*(1), 155-162. Retrieved from https://ojs.aaai.org/

index.php/AIIDE/article/view/21959 doi: doi:10.1609/aiide.v18i1.21959

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, *abs/1312.5602*. Retrieved from http://arxiv.org/abs/1312.5602

Murphy, K. P. (2023). *Probabilistic machine learning: Advanced topics*. MIT Press. Retrieved from http://probml.github.io/book2

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from https://arxiv.org/abs/1912.01703

Raffin, A. (2020). *Rl baselines3 zoo.* https://github.com/DLR-RM/rl-baselines3-zoo. GitHub.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1-8. Retrieved from http://jmlr.org/papers/v22/20-1364.html

Rasmussen, C. E., & Williams, C. K. (2004). *Gaussian Processes for Machine Learning*. Cambridge: The MIT Press. (OCLC: 1178958074)

Rudin, C. (2019, May). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, *1*(5), 206–215. Retrieved from https://doi.org/10.1038/s42256-019-0048-x doi: doi:10.1038/s42256-019-0048-x

Salimbeni, H., & Deisenroth, M. (2017). Doubly stochastic variational inference for deep gaussian processes. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/8208974663db80265e9bfe7b222dcb18-Paper.pdf

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, *abs/1707.06347*. Retrieved from http://arxiv.org/abs/1707.06347

Sipser, M. (2018). *Introduction to the Theory of Computation (third Edition)*. Cengage Learning.

Snoek, J., Larochelle, H., & Adams, R. P. (2012, August). *Practical Bayesian Optimization of Machine Learning Algorithms*. arXiv. Retrieved 2024-05-02, from http://arxiv.org/abs/1206.2944 (arXiv:1206.2944 [cs, stat]) doi: doi:10.48550/arXiv.1206.2944

Srinivas, N., Krause, A., Kakade, S. M., & Seeger, M. (2012, May). Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *IEEE Transactions on Information Theory*, *58*(5), 3250–3265. Retrieved 2024-05-02, from http://arxiv.org/abs/0912.3995 (arXiv:0912.3995 [cs]) doi: doi:10.1109/TIT.2011.2182033

Sutton, R., & Barto, A. (2018). *Reinforcement Learning, second edition*. Amsterdam, Nederland: Amsterdam University Press.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, *3*, 9–44.

Taylor, G., & Parr, R. (2009). Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th annual international conference on machine learning* (p. 1017–1024). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1553374.1553504 doi: doi:10.1145/1553374.1553504

Titsias, M. (2009, 16–18 Apr). Variational learning of inducing variables in sparse gaussian processes. In D. van Dyk & M. Welling (Eds.), *Proceedings of the twelfth international conference on artificial intelligence and statistics* (Vol. 5, pp. 567–574). Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR. Retrieved from https://proceedings.mlr.press/v5/titsias09a.html

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., ... Younis, O. G. (2023, March). *Gymnasium*. Zenodo. Retrieved 2023-07-08, from https://zenodo.org/record/8127025 doi: doi:10.5281/zenodo.8127026

Wilson, A. G., Hu, Z., Salakhutdinov, R., & Xing, E. P. (2016a, 09–11 May). Deep kernel learning. In A. Gretton & C. C. Robert (Eds.), *Proceedings of the 19th international conference on artificial intelligence and statistics* (Vol. 51, pp. 370–378). Cadiz, Spain: PMLR. Retrieved from https://proceedings.mlr.press/v51/wilson16.html

Wilson, A. G., Hu, Z., Salakhutdinov, R. R., & Xing, E. P. (2016b). Stochastic variational

deep kernel learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 29). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2016/file/bcc0d400288793e8bdcd7c19a8ac0c2b-Paper.pdf

You, J., Chung, J.-W., & Chowdhury, M. (2023, April). Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th usenix symposium on networked systems design and implementation (nsdi 23)* (pp. 119–139). Boston, MA: USENIX Association. Retrieved from https://www.usenix.org/conference/nsdi23/presentation/you

A Miscellaneous

A.1 The ϵ-greedy Policy

Given an action-value function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the ϵ-greedy policy is given by:

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}(s)|} + (1-\epsilon), & \text{if } a = \underset{a' \in \mathcal{A}(s)}{\operatorname{argmax}}\, q(s, a'), \\ \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{otherwise.} \end{cases} \tag{A.1}$$

Where $\mathcal{A}(s)$ is the set of actions available in state s and $\epsilon \in [0, 1]$ is the probability of selecting a random action.

A.2 Complexity Analysis of Exact Gaussian Process Inference

When it comes to computational complexity there are two points of interest: Computing the predictive posterior distribution (2.10) and computing the MLL (2.15).

To ensure numerical stability, the Cholesky decomposition of $\mathbf{K}_\sigma = \mathbf{L}_\sigma \mathbf{L}_\sigma^\top \in \mathbb{R}^{N \times N}$ is used, which is in $O(N^3)$. After computing $\boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1}\mathbf{y}$, predictions for each test point take $O(N)$ time for the mean and $O(N^2)$ time for the variance. Space complexity is $O(N^2)$ since an $N \times N$ covariance matrix must be stored.

BBMM inference (Gardner et al., 2018), used in the `GPytorch` library, allows for computing the GP MLL (2.15) and other expensive GP operations using only matrix multiplication, leveraging GPU acceleration. This reduces the time complexity for exact GP inference from $O(N^3)$ to $O(N^2)$. Note that this does not reduce the space complexity.

A.3 Derivative of the GP Marginal Log Likelihood

Let $\operatorname{Tr}(\cdot)$ denote the trace operation on matrices. The partial derivative of the MLL:

$$\mathcal{L}(\boldsymbol{\theta}|\mathbf{X}, y) = \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \propto -\mathbf{y}^\top \mathbf{K}_\sigma^{-1}\mathbf{y} - \log|\mathbf{K}_\sigma|, \tag{A.2}$$

w.r.t. a hyperparameter θ_j from a vector of hyperparameters $\boldsymbol{\theta}$, is given by:

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{1}{2}\mathbf{y}^\top \mathbf{K}_\sigma^{-1} \frac{\partial \mathbf{K}_\sigma}{\partial \theta_j} \mathbf{K}_\sigma^{-1}\mathbf{y} - \frac{1}{2}\operatorname{Tr}\left(\mathbf{K}_\sigma^{-1}\frac{\partial \mathbf{K}_\sigma}{\partial \theta_j}\right) \tag{A.3}$$

$$= \frac{1}{2}\operatorname{Tr}\left((\boldsymbol{\alpha}\boldsymbol{\alpha}^\top - \mathbf{K}_\sigma^{-1})\frac{\partial \mathbf{K}_\sigma}{\partial \theta_j}\right), \quad \boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1}\mathbf{y}. \tag{A.4}$$

If the dataset size is N, computing $\mathbf{K}_\sigma^{-1}$ using the standard Cholesky decomposition requires $O(N^3)$ time, followed by an additional $O(N^2)$ time per hyperparameter to compute the gradient (Rasmussen & Williams, 2004; Murphy, 2023). However, with BBMM, the overall time complexity is reduced to $O(N^2)$. Typically, instead of maximizing $\mathcal{L}$, we minimize $-\mathcal{L}$.

A.4 The Matern Kernel

The Matern kernel (Rasmussen & Williams, 2004) is given by:

$$k_{\text{matern}}(\mathbf{x}, \mathbf{x}'; l, \nu) = \frac{2^{1-\nu}}{\Gamma(\nu)}\big(\sqrt{2\nu}d\big)^\nu K_\nu\big(\sqrt{2\nu}d\big), \tag{A.5}$$

with $d = (\mathbf{x} - \mathbf{x}')^\top l^{-2}(\mathbf{x} - \mathbf{x}')$ and lengthscale parameter l, smoothness parameter ν, Gamma function Γ and a modified Bessel function K_ν.

The Gamma function is a function defined for a complex numbers $z \in \mathbb{C}$, except for the non-positive integers, and is given by the following improper integral:

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\,dt, \quad \Re(z) > 0. \tag{A.6}$$

The Gamma function is commonly used as an extension of the factorial function to complex numbers. That is $\forall n \in \mathbb{N}$, $\Gamma(n) = (n-1)!$.

$K_\nu(z)$ is a modified Bessel function. Specifically, a modified Bessel function of the second kind, which is a solution to the modified Bessel's differential equation:

$$z^2\frac{d^2w}{dz^2} + z\frac{dw}{dz} - (z^2+\nu^2)w = 0, \tag{A.7}$$

where $w : \mathbb{C} \to \mathbb{C}$ is the unknown function of the differential equation. $K_\nu(z)$ is defined for complex order ν and complex argument z. One of its integral representations is given by:

$$K_\nu(z) = \int_0^\infty e^{-z\cosh(t)}\cosh(\nu t)\,dt, \quad \Re(z) > 0. \tag{A.8}$$

For the Matern kernel, the Gamma function and modified Bessel function play key roles defining the smoothness and decay properties as the smoothness parameter ν changes. Loosely speaking, the Gamma function ensures proper normalization as $\nu \to \infty$ and the Bessel function determines the smoothness properties.

Evaluating the Matern kernel for arbitrary ν can be computationally expensive, due to needing to evaluate the modified Bessel function. Luckily there are special cases for ν, specifically for $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$ and $\nu \to \infty$ which are considerable cheaper to compute. For this reason, when performing type-II MLE on the kernel hyperparameters for the Matern kernel, the smoothness parameter ν is typically kept fixed and *not* optimized.

For $\nu = \frac{1}{2}$, the Matern kernel reduces to the absolute exponential kernel function:

$$k(\mathbf{x}, \mathbf{x}'; l) = \exp\left(-\sqrt{2}\frac{\| \mathbf{x} - \mathbf{x}' \|}{l}\right). \tag{A.9}$$

For $\nu = \frac{3}{2}$ we get once differentiable functions when used in a GP:

$$k(\mathbf{x}, \mathbf{x}'; l) = \left(1 + \sqrt{3}\frac{\| \mathbf{x} - \mathbf{x}' \|}{l}\right)\exp\left(-\sqrt{3}\frac{\| \mathbf{x} - \mathbf{x}' \|}{l}\right), \tag{A.10}$$

and similarly for $\nu = \frac{5}{2}$ we get twice differentiable functions:

$$k(\mathbf{x}, \mathbf{x}'; l) = \left(1 + \sqrt{5}\frac{\| \mathbf{x} - \mathbf{x}' \|}{l} + \frac{5}{3}\frac{\| \mathbf{x} - \mathbf{x}' \|^2}{l^2}\right)\exp\left(-\sqrt{5}\frac{\| \mathbf{x} - \mathbf{x}' \|}{l}\right). \tag{A.11}$$

As $\nu \to \infty$, the Matern kernel simplifies to:

$$k_{\text{RBF}}(\mathbf{x}, \mathbf{x}'; l) = \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}' \|^2}{2l^2}\right), \tag{A.12}$$

which when used in a GP gives infinitely differentiable functions. In general, functions sampled from a GP with a Matern kernel are k-times differentiable iff $\nu > k$.

Finally, it is common to introduce an outputscale parameter σ_f to the kernel function, resulting in the modified kernel $k'(\mathbf{x}, \mathbf{x}'; l, \sigma_f) = \sigma_f k(\mathbf{x}, \mathbf{x}'; l)$. The output scale parameter σ_f controls the amplitude of the kernel function, affecting the overall vertical variation. Meanwhile, the lengthscale parameter l determines how quickly the similarity between points decreases as their distance increases. A smaller l implies that points need to be relatively close to be considered similar, while a larger l allows for similarity over greater distances.

A.5 Spectral Mixture Learning

The spectral mixture kernel is defined for one-dimensional inputs $x, x' \in \mathbb{R}$ as:

$$k(x, x') = \sum_j w_j \cos((x - x')(2\pi\mu_j))\exp(-2\pi^2(x - x')^2 v_j), \tag{A.13}$$

where w_j are mixture weights and μ_j and v_j means and variances of the Gaussians in the spectral density. As it turns out, the spectral density is the dual form of any shift-invariant stationary kernel and is obtained by taking the Fourier transform (see Appendix A.6). The spectral mixture kernel is obtained by taking the inverse Fourier transform of a Gaussian mixture. Taking the Fourier transform of a RBF kernel gets a Gaussian spectral density centered at the origin and for the Matern kernel you get a Student spectral density centered at the origin (Murphy, 2023).

A.6 Random Fourier Features

For GPs it is possible to perform exact inference, but approximate the kernel with random features. The method, called random Fourier features (RFF), uses Bochner's theorem (Churchill, Bochner, Tenenbaum, & Pollard, 1960). Bochner's theorem states that a continuous and stationary kernel that satisfies $k(\mathbf{x}, \mathbf{x}') = k(\boldsymbol{\delta})$ with $\boldsymbol{\delta} = \mathbf{x} - \mathbf{x}'$ is positive definite if and only if it can be represented by its Fourier transform:

$$k(\boldsymbol{\delta}) = \int_{\mathbb{R}^d} p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^\top \boldsymbol{\delta}} d\boldsymbol{\omega}, \tag{A.14}$$

where $i^2 := -1$, $\boldsymbol{\omega}$ is the frequency, and $p(\boldsymbol{\omega})$ is known as the spectral density which is a normalized probability measure if $k(0) = 1$. One can then take a Monte Carlo approximation of (A.14) by constructing the random Fourier feature $\varphi(\mathbf{x}) \in \mathbb{R}^{2M}$:

$$\varphi(\mathbf{x}) = \sqrt{\frac{1}{M}} \begin{bmatrix} \sin(\boldsymbol{\omega}_1^\top \mathbf{x}) \\ \vdots \\ \sin(\boldsymbol{\omega}_M^\top \mathbf{x}) \\ \cos(\boldsymbol{\omega}^\top \mathbf{x}) \\ \vdots \\ \cos(\boldsymbol{\omega}_M^\top \mathbf{x}) \end{bmatrix}, \ \boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_M \sim p(\boldsymbol{\omega}), \tag{A.15}$$

and calculating $k(\mathbf{x}, \mathbf{x}') \approx \varphi(\mathbf{x})^\top \varphi(\mathbf{x}')$ which is equivalent to the sum:

$$k(\mathbf{x}, \mathbf{x}') \approx \frac{1}{M} \sum_{i=1}^{M} \cos(\boldsymbol{\omega}_i^\top (\mathbf{x} - \mathbf{x}')). \tag{A.16}$$

One can show that this approach reduces the time complexity to $O(NM + M^3)$ (Murphy, 2023). That said, the approximation of the kernel, results in reduced expressive power.

A.7 Evidence Lower Bound for Sparse Variational Gaussian Processes

Let $\mathbf{y}, \mathbf{f} \in \mathbb{R}^N, \mathbf{u} \in \mathbb{R}^M$ be (the realization of) the observation, GP output and inducing variables, where $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ are the input points and $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)$ the inducing points. The variational bound of the marginal likelihood is of the form (Salimbeni & Deisenroth, 2017):

$$\begin{aligned} \log p(\mathbf{y}|\mathbf{X}) &= \log \textstyle\int_{\mathbb{R}^N \times \mathbb{R}^M} p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) p(\mathbf{u}|\mathbf{Z}) \, d\mathbf{f} d\mathbf{u} \\ &\geq \mathbb{E}_{\hat{p}(\mathbf{f},\mathbf{u})} \left[\log \tfrac{p(\mathbf{y},\mathbf{f},\mathbf{u})}{\hat{p}(\mathbf{f},\mathbf{u})} \right] \\ &= \textstyle\int_{\mathbb{R}^N \times \mathbb{R}^M} \hat{p}(\mathbf{f}, \mathbf{u}) \log \frac{p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z}) p(\mathbf{u}|\mathbf{Z})}{\hat{p}(\mathbf{f},\, \mathbf{u})} \, d\mathbf{f} d\mathbf{u}. \end{aligned} \tag{A.17}$$

Where, assuming a zero mean function, $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{K}_{X,Z}\mathbf{K}_{Z,Z}^{-1}\mathbf{u}, \sigma_y^2 \mathbf{I})$ is the Gaussian likelihood for the observations given latent function values, $p(\mathbf{u}|\mathbf{Z}) = \mathcal{N}(\mathbf{0}, \mathbf{K}_{Z,Z})$ and $p(\mathbf{y}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) = \mathcal{N}(\mathbf{K}_{X,Z}\mathbf{K}_{Z,Z}^{-1}\mathbf{u}, \mathbf{K}_{X,X} - \mathbf{K}_{X,Z}\mathbf{K}_{Z,Z}^{-1}\mathbf{K}_{X,Z}^\top + \sigma_y^2 \mathbf{I})$. See also the correspondence to (2.9).

Plugging in (2.20) into equation A.17 to get the ELBO:

$$\begin{aligned} \mathcal{L}_{\text{SVGP}} &= \textstyle\int_{\mathbb{R}^N \times \mathbb{R}^M} p(\mathbf{f}|\mathbf{u}, \mathbf{X}, \mathbf{Z}) \hat{p}(\mathbf{u}) \log \frac{p(\mathbf{y}|\mathbf{f}) \cancel{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})} p(\mathbf{u}|\mathbf{Z})}{\cancel{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})} \hat{p}(\mathbf{u})} \, d\mathbf{f} d\mathbf{u} \\ &= \mathbb{E}_{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})\hat{p}(\mathbf{u})}[\log p(\mathbf{y}|\mathbf{f})] - D_{\mathbb{KL}}(\hat{p}(\mathbf{u}) \parallel p(\mathbf{u}|\mathbf{Z})). \end{aligned} \tag{A.18}$$

In `GPytorch` (Gardner et al., 2018) the ELBO is slightly adjusted and computed as follows:

$$\begin{aligned} \mathcal{L}_{\text{SVGP}} &= \mathbb{E}_{p_{\text{data}}(y,\mathbf{x})}[\mathbb{E}_{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})\hat{p}(\mathbf{u})}[\log p(\mathbf{y}|\mathbf{f})]] - \rho D_{\mathbb{KL}}(\hat{p}(\mathbf{u}) \parallel p(\mathbf{u}|\mathbf{Z})) \\ &\approx \textstyle\sum_{i=1}^{N} \mathbb{E}_{\hat{p}(f_i)}[\log p(y_i|f_i)] - \rho D_{\mathbb{KL}}(\hat{p}(\mathbf{u}) \parallel p(\mathbf{u}|\mathbf{Z})), \end{aligned} \tag{A.19}$$

where ρ, 1 by default, is a scaling constant that reduces the regularization effect of the KL divergence. Here $p_{\text{data}}(y, \mathbf{x})$ is the data distribution and N is the number of training samples.

A.8 Evidence Lower Bound for Deep Gaussian Processes

For DGPs, the evidence lower bound has a similar form to (2.22):

$$\mathcal{L}_{\text{DPG}} = \sum_{i=1}^{N} \mathbb{E}_{\hat{p}(\mathbf{f}_i^L)}[\log p(\mathbf{y}_i|\mathbf{f}_i^L)] - \sum_{l=1}^{L} D_{\mathbb{KL}}(\hat{p}(\mathbf{U}^l) \parallel p(\mathbf{U}^l; \mathbf{Z}^{l-1})), \tag{A.20}$$

where N is the number of training samples, $p(\mathbf{y}_i|\mathbf{f}_i^L)$ is the Gaussian likelihood for the observations given latent function values at the L-th (final) layer for the i-th data point. $\hat{p}(\mathbf{f}_i^L)$ is the variational distribution at the L-th layer for the i-th data point. $\mathbf{U}^l$ are the inducing variables for each dimension (depth) at layer l and $\mathbf{Z}^{l-1}$ are the inducing points at the previous layer $l-1$. $\hat{p}(\mathbf{U}^l)$ is the variational distribution for the inducing variables at layer l. $p(\mathbf{U}^l|\mathbf{Z}^{l-1})$ is the prior distribution of the inducing variables from the previous layer.

The full derivation of this lower bound can be found in Salimbeni & Deisenroth (2017).

A.9 Deep Q Network

DQN uses the TD(0) target in the following loss function, optimizing this loss means approaching the true action-value function:

$$\mathcal{L}(\theta) = \mathbb{E}_{(S_t, A_t, R_{t+1}, S_{t+1})\sim\mathcal{D}}[(R_{t+1} + \gamma \max_{a\in\mathcal{A}} \hat{q}(S_{t+1}, a, \boldsymbol{\theta}^-) - \hat{q}(S_t, A_t, \boldsymbol{\theta}))^2], \tag{A.21}$$

where $\mathcal{D}$ is an experience replay buffer and $\boldsymbol{\theta}$ are the parameters of the function approximator. Additionally, DQN uses a target model $\boldsymbol{\theta}^-$ containing frozen parameters periodically copied from $\boldsymbol{\theta}$. This improves performance and stability because supervised learning theory assumes a stationary distribution and independent and identically distributed samples.

The DQN update, which involves calculating the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$, has linear time complexity in the number of parameters due to the use of backpropagation.

B Additional Results

B.1 ϵ-greedy vs Upper Confidence Bound vs Thompson Sampling

The UCB and ϵ-greedy settings were not extensively tuned. UCB's β parameter (2.16) was set to 1.5, and the ϵ-greedy schedule was based on DQN for a specific environment.

Thompson sampling outperforms UCB and ϵ-greedy in Figure B.1, making it a preferred policy for the GP-Q/GP-SARSA algorithm. The performance gap in UCB may be due to its tendency for over-exploration, as noted by Chowdhary et al. (2014). Additionally, the ϵ-greedy schedule effective for DQN may not suit GP-Q. Thompson sampling's advantage is the lack of exploration parameters to tune.

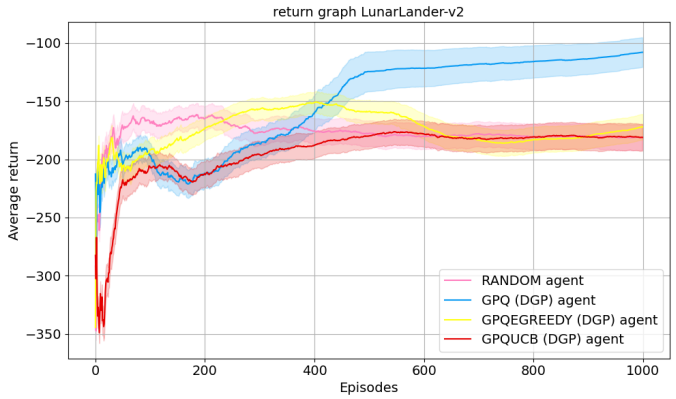

Figure B.1: Comparison of cumulative average return ($\pm 0.1 \cdot$std) in Lunar Lander between behavioral policies for the same DGP model.

B.2 GP scalability on toy dataset

Figure B.2 and B.3 display the results.

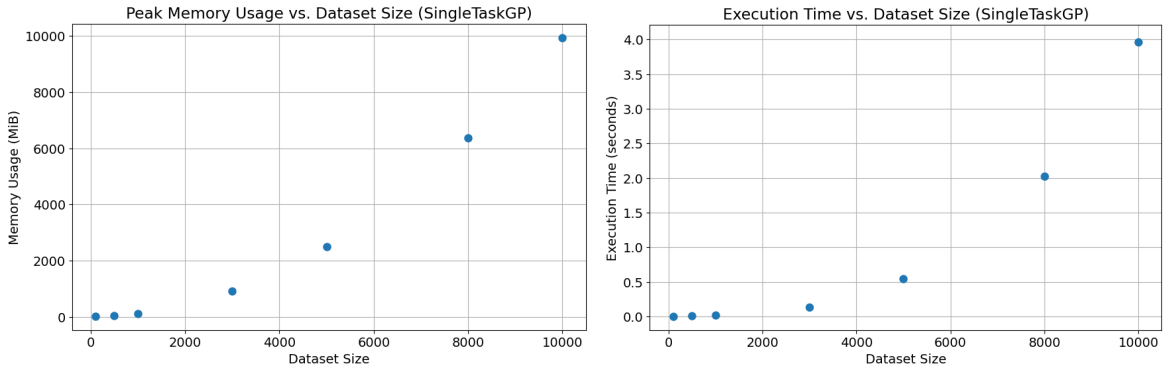

Figure B.2: Exact GP execution time and VRAM usage for $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1,\dots,N}, \mathbf{x}_i \in \mathbb{R}^4, y_i \in \mathbb{R}$ (64-bit floating point) as dataset size N increases. Averaged over 10 trials.

B.3 Loss Curve of Experiment

Figure B.4 showcases how the negative ELBO changes per update during training.

B.4 Visualizing the Posterior Predictive Distribution

For functions $f : \mathbb{R} \to \mathbb{R}$, we can visualize the GP mean and variance predictions as $\{(x, \mu(x) \pm \beta\sigma(x)) \mid x \in \mathbb{R}, \beta \in \mathbb{N}\}$. However, for CartPole or Lunar Lander, we deal with the action-value function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\mathcal{S} \subseteq \mathbb{R}^D$ with $D \geq 4$. This complicates visualizing the posterior predictive distributions. Still, we can visualize by fixing a state $s \in \mathcal{S}$ and plotting a Gaussian with mean $\mu(s, a)$ and variance $\sigma^2(s, a)$ for each $a \in \mathcal{A}$. Figure B.5 shows such Gaussians, indicating that as training progresses, GP-Q/GPSARSA

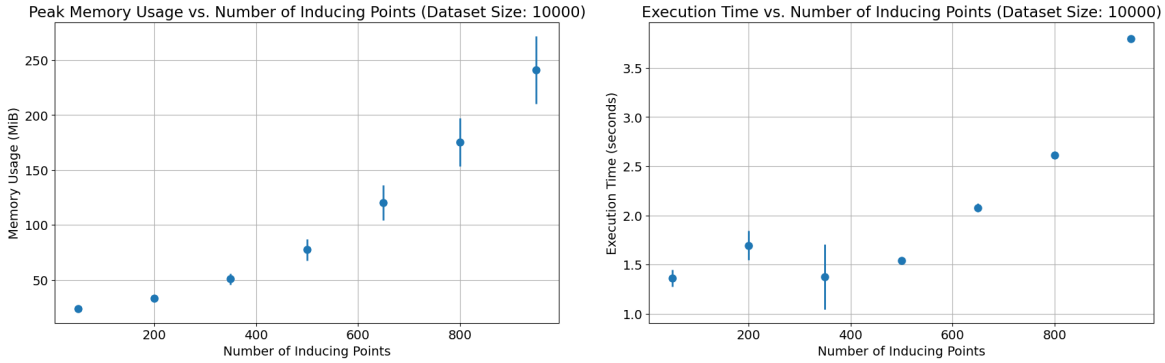

Figure B.3: SVGP execution time and VRAM usage for $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1,\ldots,N}, \mathbf{x}_i \in \mathbb{R}^4, y_i \in \mathbb{R}, N = 10000$ (64-bit floating point) as the number of inducing points increases. Batch size: 128, averaged over 10 trials.

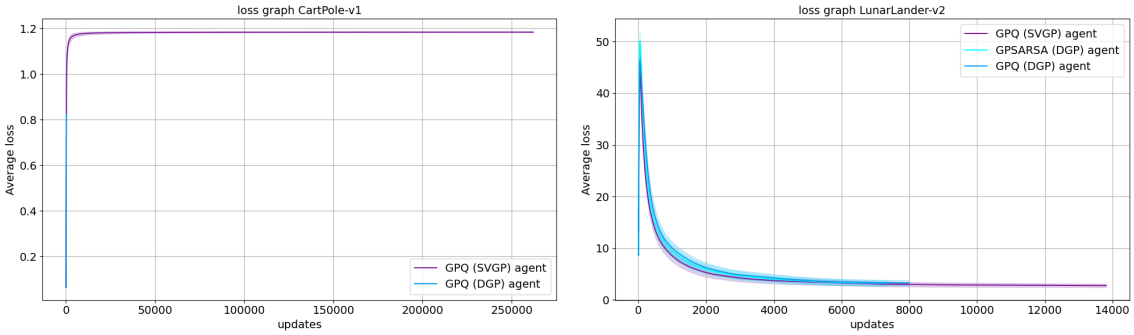

Figure B.4: Average loss (negative ELBO) $(\pm 0.1 \cdot \text{std})$ in CartPole and Lunar Lander.

agents with Thompson sampling become more greedy as uncertainty decreases and the mean q-values for each action shift.

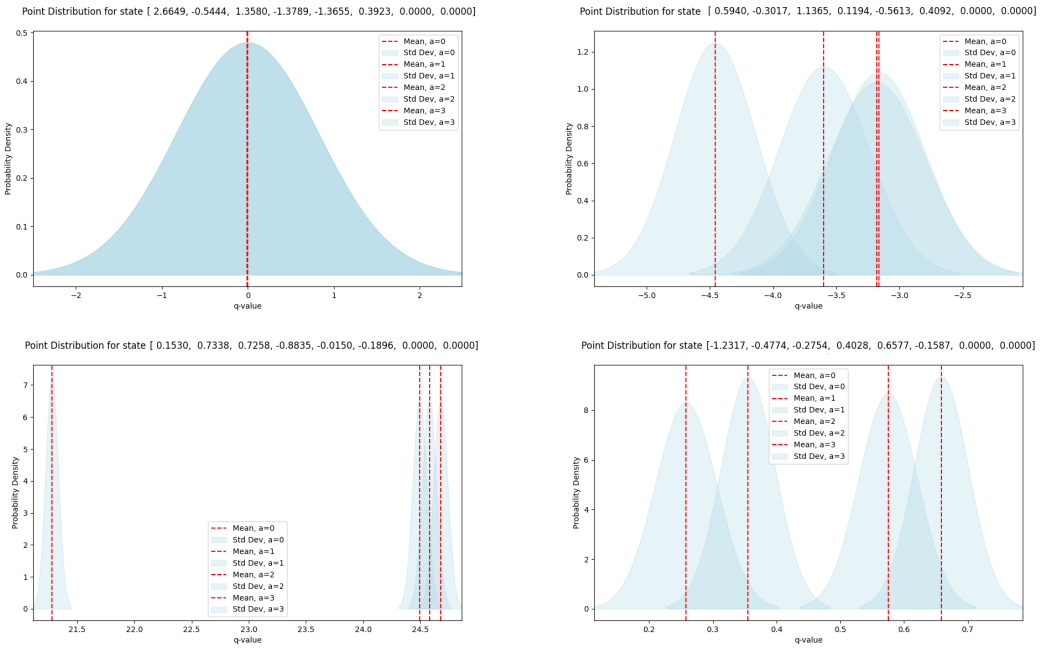

Figure B.5: Predictive action distribution for a given state for each action $a \in \mathcal{A}$. From top to bottom: The first two figures are taken from a GPQ (DGP) agent before training on Lunar Lander, and the last two are after training. As training continues, the action distributions shift and the standard deviations around the mean decreases.

B.5 Comparing GP Regression for a Step Function

Consider a GP $f_{\text{GP}}(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k_{\text{RBF}}(\mathbf{x}, \mathbf{x}'))$ with an RBF kernel and a zero mean function. Also, consider an SVGP with the same kernel, and a 4-layer, single unit per layer DGP with the same kernel.

We compare GP regression on a step function with Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.05)$:

$$s(x) = \begin{cases} 1.0 + \epsilon & \text{if } x < 0.5, \\ -1.0 + \epsilon & \text{if } x \geq 0.5. \end{cases} \tag{B.1}$$

Given the smoothness of functions from a GP with an RBF kernel, the fit on the step function for a standard GP will be poor. However, a DGP can overcome these limitations and learn to regress the discontinuity at $x = 0.5$, despite each unit using an RBF kernel.

We sample 150 points from this function with $x \in [0, 1]$ for training data. The test data contains 50 sample points where $x \in [-0.2, 0) \cup (1, 1.2]$. Using an Adam optimizer with a learning rate of 0.01, we fit each GP on the train data for 500 epochs without mini-batching.

Figures B.6 and B.7 show the predictions for each GP model. The regular GP has a poor fit, but surprisingly, the SVGP fits the training samples well. However, the SVGP's test predictions are poor, especially when examining the posterior mean function. In contrast, the DGP has the best overall fit among training and test samples.

Figure B.7 displays the DGP predictions per layer, which we can visualize as we have one GP per layer and one-dimensional input and outputs. The predictions from the first layer resemble those from a standard GP with an RBF kernel. However, as we go through each layer, the outputs transform, resembling a step function more each time.

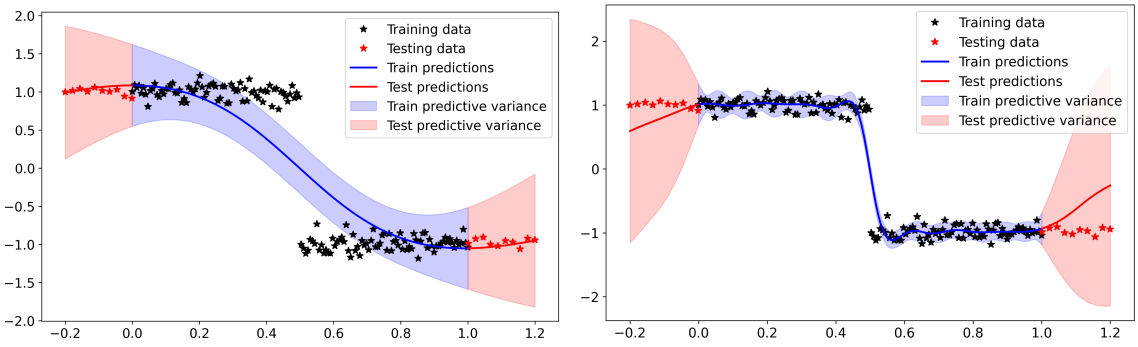

Figure B.6: GP (left) and SVGP (right) predictions with a $\pm 2\sigma$ credible interval.

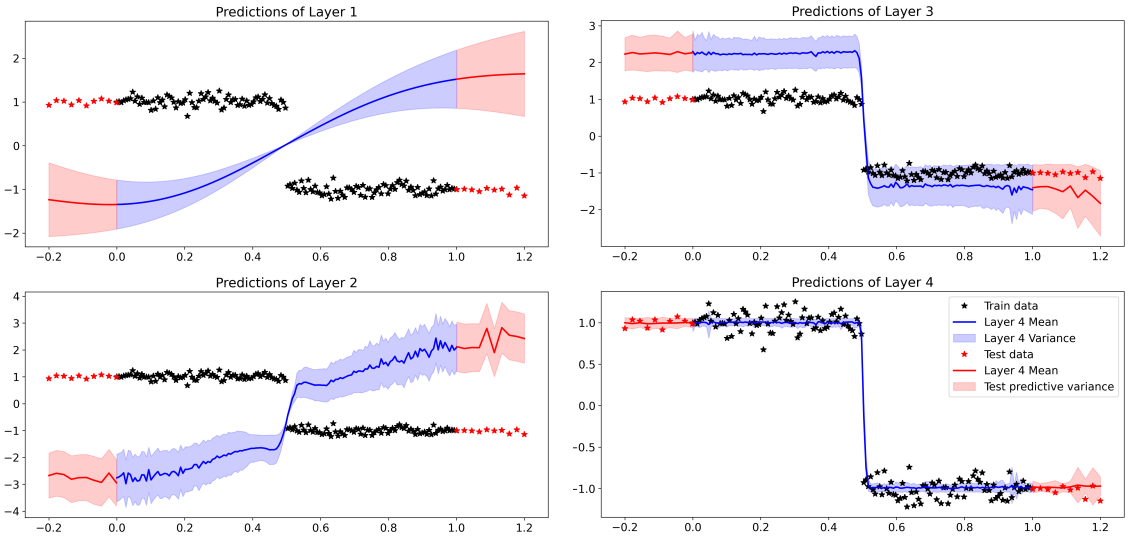

Figure B.7: DGP predictions with a $\pm 2\sigma$ credible interval. The final layer is the output of the DGP as a whole.

C Summary of Abbreviations

Table C.1: Summary of Abbreviations.

Abbreviation	Definition
BayesOpt	Bayesian Optimization
DGP	Deep Gaussian Process
DNN	Deep Neural Network
DQN	Deep Q-network
DRL	Deep Reinforcement Learning
DSPP	Deep Sigma Point Process
ELBO	Evidence Lower Bound
FIFO	First-In-First-Out
GP	Gaussian Process
GPU	Graphics Processing Unit
MDP	Markov Decision Process
MLE	Maximum Likelihood Estimation
MLL	Marginal Log-Likelihood
MLP	Multi-Layer Perceptron
NN	Neural Network
PPO	Proximal Policy Optimization
RBF	Radial Basis Function
RL	Reinforcement Learning
RV	Random Variable
SARSA	State-Action-Reward-State-Action
SVGP	Sparse Variational Gaussian Process
TD	Temporal Difference
UCB	Upper Confidence Bound
VRAM	Video Random Access Memory

D Hyperparameters

Hyperparameter	CartPole	Lunar Lander
Learning Rate	2.3e-3	6.3e-4
Batch Size	64	128
Buffer Size	100000	50000
Learning Starts	1000	0
Gamma	0.99	0.99
Target Update Interval	10	250
Train Frequency	256	4
Gradient Steps	128	-1
Exploration Fraction	0.16	0.12
Exploration Final Epsilon	0.04	0.1
Number of timesteps	5e4	1e5

Table D.1: Hyperparameters for DQN Algorithm in CartPole and Lunar Lander Environments.

Hyperparameter	CartPole	Lunar Lander
Fitting (with Adam optimizer)		
GP Fit Num Epochs	1	1
GP Fit Batch Size	128	512
GP Fit Num Batches	30	7
GP Fit Learning Rate	0.001	0.005
GP Fit Random Batching	True	True
Exploration		
UCB Beta	NA	1.5
GP E-Greedy Steps	NA	100,000
Model		
Discount Factor (γ)	0.99	0.99
Batch Size	32	128
Max Dataset Size/Budget	10,000	20,000
Kernel Type	RBF	RBF
Behavioral Policy	Thompson sampling	Thompson sampling
Posterior Observation Noise	False	False

Table D.2: Hyperparameters for GP Models in CartPole and Lunar Lander Environments.