Reinforcement Learning with Non-Conventional Value Function Approximation

Antreas Tsiakkas

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2021

Abstract

The use of a function approximation model to represent the value function has been a major contributor to the success of Reinforcement Learning methods. Despite its importance, the choice of model defaults to either a linear- or neural network-based approach mainly due to their good empirical performance, vast amount of research, and available resources. Still, these conventional approaches are not without limitations and alternative approaches could offer advantages under certain criteria. Yet, these remain under-utilised due to a lack of understanding of the problems or requirements where their use would be beneficial. This dissertation provides empirical results on the performance, reliability, sample efficiency, training time, and interpretability of non-conventional value function approximation methods, which are evaluated under a consistent evaluation framework and compared to the conventional approaches. This allows the identification of the relative strengths and weaknesses of each model that is highly dependent on the environment and task at hand. Results suggest that both the linear and neural network models suffer from sample inefficiencies, whilst alternative models –such as support vector regression– are significantly more sample efficient. Further, the neural network model is widely considered a black-box model whilst alternative models –such as decision trees– offer interpretable architectures. Both limitations have become increasingly important in the adaptation of Reinforcement Learning models in real-world applications where they are required to be efficient, transparent and accountable. Hence, this work can inform future research and promote the use of non-conventional approaches as a viable alternative to the conventional approaches.

Acknowledgements

I would like to thank my project supervisor, Stefano, and co-supervisor, Cillian, for their support and guidance throughout this project.

I would also like to express my gratitude to my family and my partner, for their continuous love and support during my master's and throughout this challenging year.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Antreas Tsiakkas*)

Table of Contents

Chapter 1

Introduction

Reinforcement Learning (RL), a class of methods for modelling and solving problems of sequential decision making in stochastic environments, has seen an increased interest in recent years due to the wide range of possible real-world applications [87, 43, 57, 76]. RL methods utilise the experience from environment interactions to derive an optimal acting policy for solving the given problem. This policy specifies the optimal action that should be selected at each environment state and thus demands a quantification of the long-term value of each action given that state. This numerical description is expressed for each state-action pair through the notion of a value function [88].

The estimation of the value function can either be done explicitly for each possible state or state/action pair, or implicitly through function approximation. The focus of this project is on the latter. Function Approximation (FA) in RL aims in establishing a generalisable relationship between the agent's simulated experience and the value of the state/action function. This allows the consideration of large or even infinite state/action spaces and, by extension, offers increased flexibility in problem modelling which has been a major contributor to the recent successes of RL methods [5, 96, 14].

FA is an extensively researched topic in the more general Machine Learning (ML) discipline with different approaches developed, each offering relative advantages and disadvantages [45, 67]. Despite this wealth of research, only a small number of approaches have been widely used in the RL context and are considered conventional, with all using either linear or (deep) neural network models. The domination of these approaches is attributed to their good performance but also to their parametric nature which allows their natural application in an online and incremental fashion, as is required within the RL framework. Yet, these methods are not without limitations, with most notable the lack of interpretability and sample inefficiencies that these exhibit.

These limitations have become increasingly important in recent years with the surge of real-world applications. Why, then, have alternative approaches –which do not suffer from these limitations– remained under-utilised? It is first noted that alternative approaches are not under-researched. There have been significant efforts in adapting non-conventional FA approaches in the RL context and demonstrating their applicability as viable alternatives. Hence, what is currently lacking from the literature and what restricts the wider application of alternative approaches is a lack of understanding of the types of situations where these offer comparable advantages.

This gap in the literature has motivated the central research hypothesis of this project, which is that under certain problems or environments, there exist non-conventional function approximation methods which offer advantages compared to the conventional methods, mainly in terms of interpretability and sample efficiency, and which perform comparably in terms of performance. To test this hypothesis, a systematic evaluation framework is created which enables their comparison under different problems.

The main contribution of this project is the implementation and empirical evaluation of the considered approaches. The analysis of the experimental results and their comparison with the conventional methods offers insights on the relative strengths and weaknesses of each approach. In particular, it is observed that these highly depend on the task at hand, on the environment state and action spaces, on the sparsity of the reward signal, and on the amount of exploration that each problem requires. As expected, the conventional methods have outperformed the non-conventional methods in almost all environments in terms of their performance on the given task. Yet, a notable difference in sample efficiency was observed, with most non-conventional models managing to reach certain levels of performance in significantly less time-steps, and thus using less data, than the conventional models. Finally, a number of important limitations and modelling considerations on the use of non-parametric models in the RL context is discussed based on empirical experimentation.

The remainder of this dissertation is structured as follows. Chapter 2 introduces the basics of the RL framework. Chapter 3 introduces each function approximation method that is explored as part of this project and surveys related research. Chapter 4 focuses on the adaptation of these methods and the implementational details. Chapter 5 covers the evaluation framework and explains the choice of environments and evaluation criteria. Chapter 6 presents and analyses the results for each of the environments and models considered. Finally, Chapter 7 summarises the key insights from the experiments, discusses future work and concludes the project.

Chapter 2

Reinforcement Learning Framework

2.1 Markov Decision Processes

RL methods are approaches to solving sequential decision problems which are modelled as Markov Decision Processes (MDPs) [9]. MDPs are characterised by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$ representing the state $(\mathcal{S})$, action $(\mathcal{A})$ and reward $(\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R})$ spaces in the environment, the state transition probabilities (P) and the discount factor $\gamma \in (0,1]$ [88]. Such problems can be described as the effort of an agent to optimise its behaviour in an environment by interpreting the resulting reward signal. This process is naturally split into time-steps, where at each time-step, t, the agent observes the current environment state, S_t, takes an action, A_t, and observes a reward, R_t, along with the next state, S_{t+1} [48].

2.2 Optimal Behaviour

The actions taken by the agent are dictated by a policy function, π, which maps states to actions, $\pi : S \mapsto A$ [48]. The objective, then, is to derive a policy which specifies the optimal action at each state, such that the agent's expected return is maximised. The return is defined as the discounted cumulative reward: $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. To allow comparison between actions, an action-value function (or q-function) is used to quantify the long-term value of each state-action pair, defined as: $q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$ [88]. This represents the return the agent expects to receive when action a is taken at state s, and assuming the policy π is followed thereafter. Based on this formulation, the action-value function of an optimal policy is maximal, and, hence, the optimisation of the action-value function leads to deriving an optimal policy.

2.3 Solution Approaches

A distinction is made between tabular and approximate solution approaches for optimising the q-value function. Tabular methods explicitly estimate its value at each possible state-action pair, but are of limited use due to their inability to tackle problems with large or infinite state/action spaces [88]. Conversely, approximate solution methods establish a mapping between state-action pairs and an estimated q-value through function approximation: $\hat{q}(s,a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. These will be the focus of this project.

A further distinction between model-based and model-free approaches is made. Model-based approaches depend on a model which is used to predict how the environment will respond to the agent's actions whilst model-free approaches do not. Instead, they rely on real interactions with the environment to learn and derive an optimal policy. FA is used in both, but the scope of this project is limited to consider only model-free approaches.

2.4 Q-Learning

This section introduces the Q-learning algorithm [95], a model-free tabular solution method for deriving the optimal action-value function, which is the central idea behind the algorithms that are considered as part of this project. Q-Learning belongs to the more general class of Temporal Difference (TD) methods. TD methods can learn from raw experience without requiring knowledge of the environment's dynamics. They store estimates of the q-value for each state-action pair and update these sequentially at every time-step using the estimates of the previous time-step:

$$Q_{t+1}(S_t,A_t) = Q_t(S_t,A_t) + \alpha \left[\Delta_t - Q_t(S_t,A_t)\right] \tag{2.1}$$

Here, α represents the learning rate and Δ_t represents the TD target at time t. Different TD targets have been used, but for the purpose of this project the Q-learning TD target is used:

$$\Delta_t = R_t + \gamma \max_{a \in \mathcal{A}} Q_t(S_{t+1}, a) \tag{2.2}$$

This is defined in terms of the immediately observed reward, R_t, and the 1-step-discounted q-values of the subsequent state, S_{t+1}. Q-Learning has been particularly effective on a wide range of tasks but most importantly, it has been successfully combined with function approximation approaches [65, 14]. This has allowed the emergence of methods which hold impressive modelling capabilities and can handle large, or even infinite, state and action spaces.

Chapter 3

Function Approximation in Reinforcement Learning

FA has been an indispensable part of the RL framework [88, 14]. The action-value function approximation problem provides an instance of a supervised regression problem. Supervised regression refers to the task of learning a function mapping between inputs and outputs when the model has access to a training dataset consisting of (input, output) pairs and when the output is a continuous-valued variable. In the RL setting, the following set, $\mathcal{D} = \{(x^{(i)}, \hat{q}^{(i)})\}_{i=1}^{n}$, where $x^{(i)} = (s^{(i)}, a^{(i)})$ is a state-action pair and $\hat{q}^{(i)}$ its predicted q-value, is considered as the training dataset.

This project considers conventional the use of neural networks and linear models and non-conventional any approach which does not involve these two models. Even though the main reason for this distinction is the extensive use and breadth of research of these conventional models, their parametric nature is also highlighted as one of the major contributors for their wide adaptation in RL. A parametric model uses a fixed number of parameters a priori which are independent of the training data [14]. As such, they can be optimised in an incremental fashion through stochastic parameter updates which does not require access to the full training dataset from onset. This enables their natural adaptation in the RL context, where the training data is acquired as the agent interacts with the environment. Conversely, non-parametric models depend on the training dataset to form the model structure [14] and, thus, cannot be naturally used in a fully online way, even though there have been recent efforts in adapting such models in online algorithms (discussed later in the chapter). The remainder of this chapter, introduces both the parametric and non-parametric models which are explored as part of this project, and surveys related research.

3.1 Parametric Models

A parametric action-value function approximation model is considered, $\hat{q}(s,a,\mathbf{w})$, with parameters $\mathbf{w} = [w_1, ..., w_d]$. This parameterisation allows for an optimisation procedure which updates the model's parameters using an update rule based on gradient descent, a first-order iterative optimisation algorithm [79]. In the context of the RL framework, this can be applied at every time-step by using an update target, U_t, and a learning rate, denoted a, which controls the rate of change:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\ [U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)]\ \nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \tag{3.1}$$

For the purpose of this project, the Q-Learning TD target introduced in (2.2) is used as the update target and the following conventional parametric algorithms are considered, which act as baselines for the experiments:

3.1.1 Linear Models

Linear FA represents the action-value function as a linear combination of features or basis functions [72]: $\hat{q}(s,a,\mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s,a) = \sum_i^d w_i\, x_i(s,a)$. This produces a parametric function approximation model where $\mathbf{w} = [w_1, ..., w_d]$ represents the model's weights and $\mathbf{x}(s,a) = [x_1(s,a), ..., x_d(s,a)]$ represents any choice of feature representation defined in terms of the state (and action). Linear models demand a significant amount of feature engineering and model design which is in most cases problem-specific due to their parametric nature which defines the model structure (and representational capacity) a priori [14].

Research on linear FA in RL has mainly focused on the choice of feature representation which includes polynomial and radial basis functions [10, 91], tile coding [88, 82], kernel-based features [71, 7] and Fourier basis features [50]. Related work has shown that the success of these feature representations depend largely on the problem at hand [88, 72]. As such, for the problems considered as part of this project, a number of different approaches is implemented and evaluated.

3.1.2 Neural Networks

Neural networks (NNs) is a broad class of parametric non-linear FA models which consist of a number of interconnected and organised into layers artificial computational elements, called neurons [81]. NNs hold impressive representational capabilities due to their flexible model structure which may contain an arbitrary number of layers and neurons and the application of a non-linear transformation on the outputs of each layer. For the purpose of this project, we consider the traditional feed-forward fully-connected architecture but we note the development of an impressive number of different architectures. We refer the reader to Le Cun et al. [53] for a review.

NNs have been used as FA models in RL since early work on the field [8, 89]. More recently, the use of deep architectures has given rise to a class of Deep RL approaches [5, 66, 94], whose performance and generalisation abilities have made them the conventional choice in many tasks. Despite their wide adaptation and success, NNs have long been considered as ”black-box” models due to their non-interpretable inference processes [55]. This constitutes an important limitation, especially in the deployment of RL models in domains such as finance and healthcare [6, 3].

3.2 Non-Parametric Models

There have been various attempts at adapting non-parametric models as value function approximators in RL. A non-parametric model offers several advantages despite its unnatural application in the online setting of the RL framework. Firstly, it does not specify the model structure a priori, and thus does not require significant design effort and prior knowledge about the system [22]. Second, it allows increased flexibility by automatically selecting the most informative features from the data. Lastly, it involves a reduced numbers of hyperparameters to tune, thus often requiring less computational resources to train. The remainder of this section introduces the non-parametric models that have been considered and experimented with as part of this project.

3.2.1 Decision Trees

Decision Trees (DTs) are non-parametric, non-linear models with a flow-chart structure which consists of ordered binary conditions imposed on the feature space, as depicted in Figure 3.1. This set of conditions is derived from the training data and can be used for classification or regression [12]. DTs are specified by the split criterion (or loss function) which measures the quality of a split. For a given split, $P = \{P_1, P_2\}$, and a loss function, L, the quality of the split is defined as: $Q(P) = p_1 L(P_1) + p_2 L(P_2)$, where p_i denotes the proportion of samples in partition i. The tree is then constructed through the iterative selection of partitions which optimise this criterion. For regression problems, the mean squared function is usually used as the loss function.

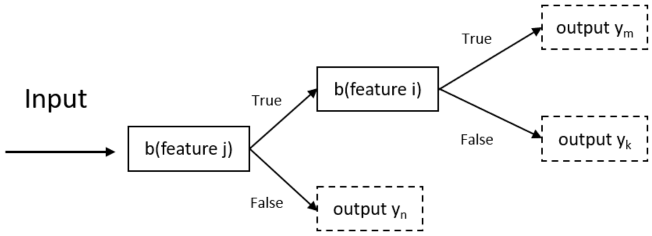


Figure 3.1: A simple decision tree, where $b()$ denotes a binary condition imposed on a particular feature of the input space.

DTs were one of the first non-parametric models which have been adapted as action-value function approximators [92, 42]. These early approaches focused on the discretisation of the state space which assumed the presence of less useful state regions. Later work [29, 15], used DTs and ensembles of trees –such as Random Forests [11], Extremely Randomised Tree [34] etc– along with the Fitted-Q Iteration algorithm (Section 3.3) achieving good performance on specific problems, but lacked an empirical comparison between these and alternative FA models. More recently, Silva et al [83] proposed the adaptation of Differentiable DTs (DDTs) [85] in an online action-value approximation algorithm which achieved performance comparable to a neural network on several tasks but the increased complexity resulted in a less interpretable model, a significant limitation given that one of the main advantages of using DTs is their interpretable inference processes. Nevertheless, the authors suggested and demonstrated the discretisation of these DDTs back to an interpretable form.

3.2.2 Support Vectors

Support Vector (SV) algorithms are non-linear supervised learning models for classification and regression that are based on the derivation of a high-dimensional hyperplane. For regression, this is constructed based on an underlying optimisation problem which tries to minimise the distance of all training examples, which lie outside a certain distance, ε, from the plane:

$$\min_{\alpha,\alpha^*} \frac{1}{2}(\alpha-\alpha^*)^\top K(\alpha-\alpha^*) + \varepsilon\, e^\top(\alpha+\alpha^*) - \hat{\mathbf{q}}^\top(\alpha-\alpha^*), \tag{3.2}$$

$$\text{subject to } e^\top(\alpha-\alpha^*) = 0,\ a_i, a_i^* \in [0, C]. \tag{3.3}$$

Here, $K_{ij} = k(x_i, x_j)$ is the kernel function which maps examples to a higher dimensional space, e is a vector of all ones, and C is a regularisation parameter. Given a new example, x_{n+1}, the algorithm uses the support vectors that were used to derive the hyper-plane for making predictions:

$$\hat{q}_{n+1} = \sum_{i \in SV} (a_i - a_i^*)\, k(x_i, x_{n+1}) + b \tag{3.4}$$

Early work on the adaptation of SV algorithms in the RL framework argued about their increased flexibility in modelling the action-value function, but lacked an empirical evaluation of their performance and a comparison with alternative approaches [24, 51]. Maclin et al. [58] adapted a knowledge-based SV regression algorithm which incorporated prior information provided by the user, thus requiring a good understanding of the problem and of the optimal behaviour in the environment from onset. Their algorithm outperformed a simple SV regression approach but was not compared to alternative models. Later work, adapted SV clustering and classification algorithms for discretising the state-space [37, 59], but lacked a comprehensive empirical evaluation. More recently, efforts have been focused on developing online algorithms. Lee et al. [54] proposed an online SV regression algorithm for approximating the action-value function. Their algorithm was tested on a 4-chain walk and the cart-pole balancing problems indicating promising results but was only compared against tabular solution methods. Esposito et al. [30] proposed an approximate policy iteration algorithm using SV regression to model the value function. They provided a theoretical analysis of their algorithm and empirical results on the mountain-car and balancing-bicycle problems for both batch and online implementations, but did not compare it with alternative approaches. Finally, An et al. [4] have adapted an SV classification model as the actor component of an actor-critic algorithm, but their analysis was limited to finite state-space problems.

3.2.3 k-Nearest Neighbours

K-Nearest Neighbours (kNN) is a class of supervised regression and classification methods which make predictions based on the location of the test example in the feature space in relation to the locations of the training examples. A key characteristic of the algorithm is that there is no training phase involved but, rather, all calculations are performed during prediction. Given a distance measure and an integer number, k, the kNN algorithm finds the k training examples which are nearest to the test example according to the specified distance measure, and uses their labels to derive its predicted value. In regression problems, the predicted value is the (distance-weighted) average of the k-nearest training examples' labels.

The use of a kNN perception field as an action-value function approximation in RL has been one of the first successful adaptations of the algorithm [63]. The initially proposed kNN-TD algorithm [61] used a set of uniformly-distributed-across-the-state-space points as the neighbours, which maintained a q-value estimate for each action and were updated incrementally. Given a new state, the algorithm predicted a weighted-average value according to the stored estimates of the k-nearest neighbours in the state-space. This was extended to use eligibility traces [62] and handle continuous action spaces [46]. Later work built upon these models to apply them successfully on real-world applications [78], and analyse their theoretical properties [75].

3.2.4 Gaussian Processes

Gaussian Processes (GPs) are non-parametric models which define distributions over functions and consist of a number of multivariate random variables, any finite number of these having a joint Gaussian distribution [77]. A GP, denoted $f \sim \mathcal{GP}(m,k)$, is fully characterised by its mean, $m(\cdot)$, and covariance, $k(\cdot,\cdot)$, functions.

GPs can be used for supervised regression through Bayesian inference. Given a training dataset, $\mathcal{D}$, the dependent variable can be modelled as $\hat{q} = f(x) + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ representing the noise that is present in the training observations, and $f(x)$ defined as a GP. Given a new state-action pair, x_{n+1}, we can compute the conditional distribution of the function at that test point as: $f_{n+1}|\mathcal{D}, x_{n+1} \sim \mathcal{N}(\mathbb{E}(f_{n+1}), \mathbb{V}(f_{n+1}))$, where:

$$\mathbb{E}(f_{n+1}) = \mathbf{k}_{n+1}^{\top}[K(\mathbf{x},\mathbf{x}) + \sigma_n^2\, I]^{-1}\mathbf{y}, \tag{3.5}$$

$$\mathbb{V}(f_{n+1}) = k(x_{n+1}, x_{n+1}) - \mathbf{k}_{n+1}^{\top}[K(\mathbf{x},\mathbf{x}) + \sigma_n^2\, I]^{-1}\mathbf{k}_{n+1} \tag{3.6}$$

Here, $\mathbf{k}_{n+1}$ denotes the vector containing the covariances between the new observation, x_{n+1}, and all training observations. Equations (3.5) and (3.6) constitute the key predictive equations for regression with GPs. Still, the inversion of the covariance matrix constitutes an expensive computation. To circumvent this limitation, there have been efforts in developing sparse online GP algorithms which iteratively update the model's mean and covariance functions [20].

The work of Engel et al [27, 28] has been one of the first attempts at adapting GPs as FA models in the RL context. This focused on policy evaluation through the development of a Gaussian Process TD (GPTD) algorithm. Later work focused on the use of GPs for both model-based [77, 23, 21, 22] and model-free [44, 2] RL, but did not take advantage of the variance estimation that GPs offer. More recent work attempted to utilise this property in an effort of developing sample efficient algorithms. Jung et al. [47] introduced the GP-RMAX algorithm, a model-based RL algorithm which incorporated an exploration strategy based on the GP uncertainty estimates. Chung et al. [18] incorporated an information gain measure in the action selection process to encourage early exploration. More recent work, have adapted GPs in online algorithms taking advantage of sparsification techniques developed outside of RL [17, 38, 44] achieving significant reductions in the amount of computational resources required. Xuan et al. [97] implemented a Bayesian Deep RL model which adopted neural networks as the kernel function of the GP, but the empirical evaluation of their algorithm showed great instability in terms of performance. Finally, the recent work of Ghavamzadeh et al [35] provided one of the first successful attempts at using GPs alongside policy-gradient and actor-critic approaches. Their work included both theoretical and empirical results on their proposed algorithms' performance and convergence properties.

3.3 Fitted-Q Iteration

Fitted-Q Iteration (FQI) is a framework for adapting general supervised regression models as action-value function approximators in RL introduced in [29], borrowing ideas from the interpolated value iteration method discussed by Gordon [36]. The high-level idea is to keep re-fitting a supervised regression model at each time-step. In essence, at every time-step, a training set is created, $TS = \{x^{(j)}, y^{(j)}\}_{j=1}^{b}$, where b is the batch size. The data are created using a set of 4-tuples, (s: state, a: action, s': next state, r: reward), and are defined as follows:

$$x^{(j)} = (s^{(j)}, a^{(j)}) \tag{3.7}$$

$$y^{(j)} = r^{(j)} + \gamma \max_{a'} \hat{Q}_{prev}(s^{(j)\prime}, a') \tag{3.8}$$

Here, $\hat{Q}_{prev}$ denotes the model fitted in the previous time-step and γ is the discount rate of the MDP. This dataset is then used to fit a new model. The FQI algorithm has been successfully applied on a wide range of applications [70, 15, 90, 33, 64, 80]. Yet, it contains no theoretical convergence guarantees due to the continual re-fitting of the model which is highly dependent on the sampled batch at each time-step [14]. This may result in unstable training performance or convergence to sub-optimal solutions.

3.4 Related Work

Surveying related work in the adaptation of non-conventional value FA approaches has revealed two notable observations. First, there is a significant amount of work in this area with early work focusing on the development of batch algorithms, whilst recent work focuses on the adaptation of incremental implementations to be used in the online data generation framework of RL. Second, despite this breadth of research, there is not, currently, a consistent evaluation of the proposed methods either in terms of the evaluation criteria used or in terms of the environments these were tested on. In addition, there is a distinct lack of comparison between the proposed approaches and any alternatives. Hence, the work of this project –i.e. the implementation, evaluation, and comparison of these different methods under a consistent and systematic evaluation framework– would be an important contribution towards understanding the relative strengths and weaknesses of each model and towards their wider use.

Related work to this project is limited. The work of Busoniu et al. [14] contains a comprehensive survey of approaches to FA in the RL context, covering both parametric and non-parametric models. They focus on the theoretical analysis of the considered algorithms, yet they include limited empirical results. Similarly, Lange et al. [52] also focus on theoretical analysis, though they narrow their scope around batch approaches. Their work provides insights as to the use of the Fitted-Q Iteration algorithm and kernel-based models. Despite lacking empirical results as well, Xu et al. [96] provide a valuable review of recent advances and applications in this area, including the use of tree-based and kernel-based approaches. Finally, Melo et al. [65] provide an analysis on the theoretical convergence guarantees of several FA models within RL, but their work does not include any non-parametric model.

Chapter 4

Methodology

4.1 Implementation of Parametric Models

Some general ideas from the Deep-Q Network framework as proposed in [66] are adapted. An experience replay buffer is used to store a set of 4-tuples, (s, a, s', r), which represent the state, action, next state and reward, as generated from the environment interactions, and then to uniformly sample batches of fixed size at every time-step for updating the model's parameters. Two model instances are maintained for this purpose. The primary model is updated at every-time step and is used in action selection and in the prediction of the q-value of the current state-action pair. The target model is updated every a fixed number of steps with the parameters of the primary model and is used in the prediction of the next state-action pairs, as used in the max-operator of the q-learning update rule. The loss function is then defined as follows, where θ denotes the parameters of the model:

$$L(\theta) = [r + \gamma \max_{a'} \hat{Q}^{\text{target}}(s', a') - \hat{Q}(s, a)]^2 \tag{4.1}$$

The use of experience replay and a target model reduce correlations in the data which arise due to their sequential generation, thus achieving more stable training performance [66]. The Pytorch library [73] is used for constructing the models. These are optimised through batch gradient-descent using the Adam optimisation algorithm with default parameters [49] and the loss function defined above.

For the neural network model, a simple feed-forward fully-connected architecture is used with an arbitrary number of layers and neurons (depending on the problem) and ReLU units applied at the output of each layer (apart from the output layer). The model takes as inputs the state description directly and outputs the predicted q-values for each action in the environment. The linear model consists of a single layer of linearly combined basis functions which are constructed from the state variables. Two kinds of feature representations are constructed: linear and polynomial [82]. These are implemented and tested on each of the problems that we consider. Table 4.1 shows the hyperparameters of these models which were optimised according to the procedure described in Section 4.4.

Parameter	**Description**
gamma	γ parameter of the MDP
batch_size	Batch size
learning_rate	Learning rate used in Adam optimisation
target_update_freq	Frequency of target network updates
lr_reduct	Rate of reduction of the learning rate
lr_reduct_freq	Frequency of reduction of the learning rate
hidden_size (for Neural Network)	Sizes of hidden layers
poly_degree (for Linear Model)	Degree of polynomial for polynomial features

Table 4.1: Description of parameters for the parametric models.

4.2 Implementation of Non-Parametric Models

4.2.1 Fitted Q-Iteration

As previously discussed, non-parametric models cannot be naturally applied in the online setting of RL. Hence, these are implemented using the Fitted-Q Iteration (FQI) framework introduced in the previous chapter. Similarly with the parametric models, an experience replay buffer is used to store environment observations and then uniformly sample a batch at each time-step. To stabilise the performance of the algorithm, experiments were conducted with storing the fitted models at intermediate checkpoints and using a weighted average of the outputs of the stored models for making predictions. This mitigated the risk of using a single model which was fitted with samples that do not cover the state-action space sufficiently. In addition, thresholding the addition of new observations in the replay buffer was tested as an alternative approach. This threshold specified the minimum distance (Euclidean distance) between a new observation and all stored observations that needs to be satisfied for the new observation to be added and could be helpful in environments with sparse reward signals. Of course, the use of thresholding presents two limitations. First, the relationships of data points in high-dimensions is not well-understood and metrics such as the Euclidean distance may be problematic [1]. Second, bias may be introduced in the sampling process since the proportions of observations in the replay buffer of particular state-action regions will have changed. Thence, when thresholding is applied, the models are trained with all observations in the replay buffer. Despite these limitations, it was observed that in environments where significant exploration is demanded, the use of thresholding significantly aids these models to explore more effectively.

The Scikit-learn library [74] was used to implement the various models that were considered within the FQI framework. These models were selected based on the good empirical results reported in related work as identified in the previous chapter. A number of parameters were kept fixed across all environments after their empirical evaluation and the consideration of results from related work. For the kernel function used by the SV Regression and GP models, the Radial Basis Function (RBF) was selected, defined as: $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_f^2 \, exp[-\frac{1}{2}\sum_d (x_d^{(i)} - x_d^{(j)})^2 / l_d^2]$. All tree-based algorithms used the Mean Squared Error as the loss function. The rest of the parameters (shown in Table 4.2) were tuned according to the procedure described in Section 4.4. In addition, the $replay\_threshold$, $model\_save\_freq$, $model\_save\_capacity$, $batch\_size$ and $gamma$ parameters are tuned for all FQI-based models.

Model	Parameter	Description
Decision Tree	max_depth $min\_samples\_split$ $min\_samples\_leaf$	Maximum tree depth Min samples required for a split Min samples required at a leaf
Random Forest Extra Trees	$n\_estimators$ max_depth $min\_samples\_split$ $min\_samples\_leaf$	Number of trees in the ensemble Maximum tree depth Min samples required for a split Min samples required at a leaf
Support Vectors	C	Regularisation parameter
K-Neighbors	$n\_neighbors$ $weights$	Number of neighbours Weight function used in prediction
Gaussian Process	$length\_scale$	Parameter of the RBF kernel

Table 4.2: Description of parameters for the models used within the FQI framework.

4.2.2 Online Gaussian Processes

Based on related work [44, 38, 17], the adaptation of an online algorithm based on Gaussian Processes is particularly attractive as it fits the online data generation of the RL framework, and its uncertainty estimates can be used as part of an informed exploration strategy. Similarly with the cited papers, the sparsification algorithm proposed in Csato and Opper [20] is used. The algorithm maintains a dictionary of basis points, $Z \in \mathbb{R}^{d \times d}$, which adequately represent the state-action space, and a vector, $\alpha \in \mathbb{R}^{d}$, along with a matrix, $C \in \mathbb{R}^{d \times d}$, which are updated incrementally and are used for prediction. Given a new example, x_{t+1}, its mean and variance are calculated as:

$$m(x_{t+1}) = \alpha_t^\top \, \mathbf{k}(Z, x_{t+1}) \tag{4.2}$$

$$\Sigma(x_{t+1}) = k(x_{t+1}, x_{t+1}) + \mathbf{k}(Z, x_{t+1})^\top \, C_t \, \mathbf{k}(Z, x_{t+1}) \tag{4.3}$$

Here, $k(\cdot,\cdot)$ represents the kernel function and $\mathbf{k}(Z, x_{t+1})$ a vector containing the covariances between the new example and all examples in the dictionary. Details of the algorithm along with the procedure for updating α and C given a training example and the construction of the dictionary can be found in Appendix A. In essence, each new example enters the basis only if it satisfies a certain criterion. Hence, the dictionary is constructed as data are generated and stores those which are deemed useful. A parameter, *epsilon_tol*, controls the threshold for adding a new example in the dictionary, which acts as a hyperparameter of the model. As with the FQI-GP algorithm, an RBF kernel is used with its length scale parameter defined a priori and act as a hyperparameter. Finally, it was observed that the initial values of α and C impact the model's performance significantly, and hence these are also treated as hyperparameters.

4.3 Exploration Strategy

The exploitation-exploration dilemma is a key consideration of the RL framework. It describes the fine balance between exploration in the environment as to discover optimal behaviour, and exploitation of the knowledge that has been already accumulated by the agent [39, 88]. All models considered utilise an ε-greedy exploration strategy. This is implemented in the action selection component of the RL agent, which, at every time-step during training, selects an action randomly out of those available with probability ε, or otherwise selects the greedy action, i.e. the action with the highest estimated q-value. The value of ε is reduced over-time. This is controlled using two hyperparameters: *eps_max_reduct*, which represents the maximum reduction, and *eps_decay*, which controls the rate of reduction.

In addition to ε-greedy, the use of the uncertainty estimates of the Gaussian Process models as an alternative exploration strategy is also explored. Intuitively, high uncertainty indicate regions in the state-action space which may be beneficial to explore. The experimentation is based on work from the area of Bayesian Optimisation –whereas the tradeoff between exploration and exploitation is also a significant challenge– and makes use of an acquisition function for determining which data point should be evaluated next [84]. The value of the acquisition function at a given observation can then be used instead of its predicted mean in the action selection process. For the purpose of this project, experiments were conducted using the Upper Confidence Bound (UCB) acquisition function, defined as follows:

$$\alpha_{UCB}(x_{t+1}; \mathcal{D}, \theta) = m(x_{t+1}) + \kappa\, \Sigma(x_{t+1}) \tag{4.4}$$

Here, κ is a hyperparameter which controls the balance between exploration and exploitation. However, the use of this alternative exploration strategy instead of the ε-greedy strategy that is conventionally used, has resulted in significant performance drops in all of the environments considered. Hence, it was not adopted for the final models which are presented in Chapter 6.

4.4 Hyperparameter Tuning

The hyperparameters of each model were tuned according to the following procedure. For each environment, a maximum number of steps was selected that each algorithm would train for. This was chosen based on empirical results which showed that all considered models appeared to converge at a stable policy by that time. Then, each parameter setting was used in 3 training runs (random seeds) and the policy at the end of each run was evaluated over 3 episodes where the policy was followed deterministically. The resulting average evaluation returns constituted the model's performance under the given parameter setting. The parameter setting which performed the best was then selected for each model and environment. A number of different values, chosen through grid-search, was tested for each parameter.

Chapter 5

Evaluation Framework

The primary objective of this project is the systematic evaluation of non-conventional value function approximation approaches and their comparison with the conventional ones. A number of environments and evaluations metrics were selected which would evaluate each method in terms of their performance on the given task, sample efficiency, reliability and training time. These are introduced and explained in the remainder of this chapter.

5.1 Environments

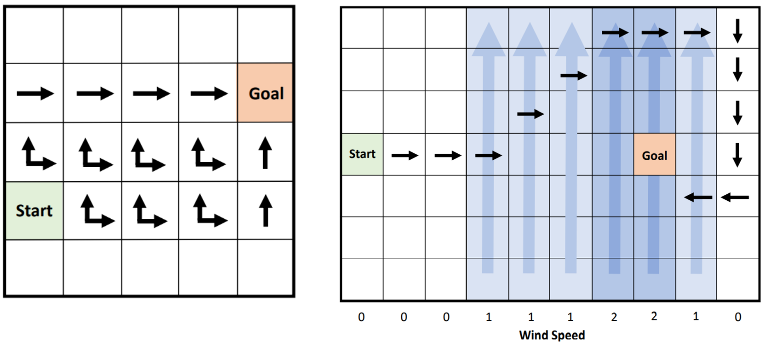
All environments that were used in this project were adapted or taken directly from the OpenAI gym library [13]. This is a library consisting of environment implementations of various problems from the RL literature, and its use is considered conventional. This offers the advantages of standardisation and comparability with related research. The environments were selected such that the strengths and weaknesses of the models can be assessed. These are introduced below and depicted in Figures 5.1 and 5.2:

> **SimpleGridworld**: This is a simple 5×5 gridworld where the objective is for the agent to reach a goal located in one of the grids. Both starting and goal positions are deterministic. Both state and actions spaces are discrete. The state is described in terms of a 25-dimensional one-hot vector, which describes in which of the 25 grids the agent is located. The agent can take four actions: move up, right, down, or left. The agent receives a reward of -1 on every transition. The episode ends either when the agent reaches the goal or when a maximum of 50 steps were taken. If the agent takes an action that would bring it outside of the grid, it remains on the same position.

WindyGridworld: This is an implementation of the WindyGridworld problem [88], a 7×10 gridworld. Both state and actions spaces are discrete. The state is described in terms of a 70-dimensional one-hot vector, which describes in which of the 70 grids the agent is located. The rest of its characteristics are the same as in simple gridworld, apart from the existence of a south-blown wind which passes through the middle grid columns. This causes any actions that would originally take the agent on one of these grids, to instead take it a number of grids up instead. The number of grids can either be 0, 1 or 2, depending on the wind speed as shown in Figure 5.1. The episode ends either when the agent reaches the goal or when a maximum of 500 steps were taken.

CartPole: This is an implementation of the Pole-Balancing problem [8]. The objective in the CartPole environment is to balance a pole which is standing on top of a cart by moving the cart left or right. The state space is continuous whilst the action space is discrete. The environment state is described in terms of four continuous-valued variables: $CartPosition \in [-4.8, 4.8]$, $CartVelocity \in (-\infty, \infty)$, $PoleAngularVelocity \in (-\infty, \infty)$ and $PoleAngle \in [-0.418, 0.418]$ (in rad). These are initialised by uniformly sampling in the interval $[-0.1, 0.1]$. The agent can take two actions: push cart to the right or left. The agent receives a reward of $+1$ at every time-step until the episode ends. The episode ends if either $|PoleAngle| > 0.21$, $|CartPosition| > 2.4$, or if 200 time-steps have passed.

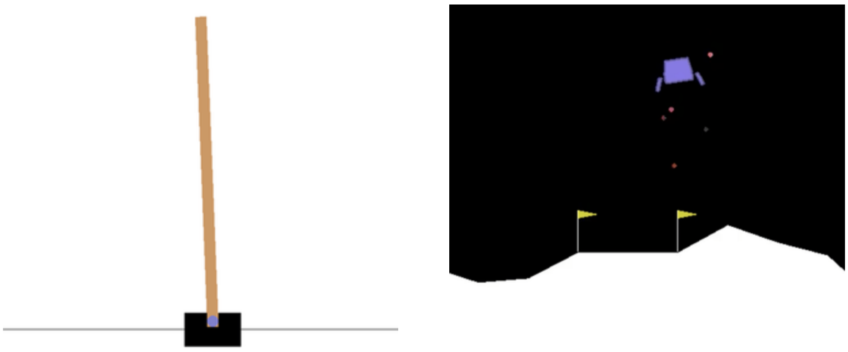
LunarLander: The objective in this environment is landing a spaceship on the moon surface. The state space is continuous whilst the action space is discrete. The environment state is described in terms of eight variables. Six of these are continuous-valued $\in (-\infty, \infty)$: $AngularSpeed$, $HorizontalCoordinate$, $VerticalCoordinate$, $HorizontalSpeed$, $VerticalSpeed$, and $Angle$, and two are binary: $FirstLegContact$, $SecondLegContact$. The agent can take four actions: do nothing, fire left engine, fire right engine, or fire main engine. The agent receives reward for successfully landing the spaceship and bringing it to rest. It receives a negative reward for crushing it. The episode ends either when the lander crashes, comes to rest or when 500 time-steps have passed.



(a) Simple Gridworld

(b) Windy Gridworld

Figure 5.1: Gridworld environments. Black arrows indicate optimal path to the goal.



(a) CartPole

(b) LunarLander

Figure 5.2: Classic control RL environments.

5.2 Evaluation Criteria

The models are evaluated in terms of their performance on the given tasks, reliability, sample efficiency and training times. For all algorithms considered, a maximum number of time-steps is defined for each task. This is chosen based on empirical results which show that all models appear to converge at a stable policy by that time. The terminology introduced in [16] which describes three axes of variability –during training: within runs, during training: across runs and after training– is used, and metrics which capture all three axes are adopted. A further distinction between (training) runs and (evaluation) rollouts is made. A run involves episodic environment interactions for a maximum number of time-steps, during which the agent learns and updates its policy. A rollout involves a single-episode interaction where the given policy is followed deterministically and there is no learning. Evaluating across multiple runs allows for the quantification of the stochasticity induced by the random seed and the initialisations of the environment and optimisers. Evaluating across multiple rollouts allows for the quantification of the stochasticity induced by the environment and the optimisation process. Finally, the evaluation metrics used are defined below. It is denoted as q_a the a-th percentile, and $Q_1 = q_{25}$ and $Q_3 = q_{75}$ as the first and third quartiles respectively. Importantly, the reported results were measured using the same model specifications across all metrics.

(1) **Average evaluation returns**: This is the average return the agent receives across ten evaluation rollouts. It is measured after certain number of time-steps and it therefore assesses after-training variability. To ascertain a reliable measure of this quantity, it is recorded for thirty runs, initialised with different seeds. The median, Q_1 and Q_3 across all runs are then reported, which provide a measure of centrality and dispersity.

(2) **Average training returns**: This is the average return the agent receives within a training run. To ascertain a reliable measure of this quantity, it is recorded for thirty runs, initialised with different seeds. The median, Q_1 and Q_3 across all runs are then reported for certain intermediate time-steps. Given the stochasticity induced from the exploration strategy which is active during training, the training return is expected to be noisier than the evaluation return. Nevertheless, the shape of the training curve should indicate that the agent is learning. This metric ascertains both during-training axes of variability. The reported quartiles measure the dispersity of the cumulative training returns across runs (random seeds) whilst the shape of the training curves provide an indication for the stability of training performance within runs.

(3) **Worst-case average evaluation returns**: This is a reliability measure, which uses the 5-th percentile, q_5, as the measure of the (1-in-20) worst-case average evaluation return the agent may receive. A q_5 value close to the median is a strong indication of the robustness of the given model. Conversely, a value significantly lower than the median indicates that the model is prone to occasionally converging at sub-optimal policies.

(4) **Number of time-steps required to reach certain performance**: This is a sample efficiency measure, quantifying the amount of training data required for the

model to reach certain performance levels. These are environment-specific and are selected to reflect optimal behaviour. This is measured by assessing the resulting policy after each episode within a training run. If the average returns it achieves across ten evaluation rollouts satisfies the required performance level, then the number of time-steps elapsed are recorded. This is repeated for thirty runs, after which the box-plot of the resulting sample is reported.

(5) **Training time**: The actual time taken for the program to run (wall-clock time), measured in seconds, is used to quantify the training time of each model.

5.3 Model Comparison

In addition to the evaluation of each model, statistical analyses are conducted to compare the models' average evaluation return achieved after the maximum number of training time-steps have elapsed. The result of a Welch's t-test is reported, as suggested by Colas et al. [19], for each pair of models. The Welch's t-test, in contrast with t-test, does not assume equal variances between the two random variables, a more realistic condition for the project's setting. For two given models, denote the true means of these quantities as μ_1 and μ_2 respectively. Then a statistical test is performed to assess whether their difference, $\mu_1 - \mu_2$, is greater than 0. Under this test, the null hypothesis is: $H_0 : \mu_1 - \mu_2 = 0$ and the alternative hypothesis is: $H_a : |\mu_1 - \mu_2| > 0$. The test uses a numerical quantity calculated from the sampled observations, called a test statistic, defined as follows:

$$t = \frac{|\bar{x}_1 - \bar{x}_2|}{\sqrt{\frac{s_1^2 + s_2^2}{n}}} \tag{5.1}$$

Here, $\bar{x}_i$ and s_i are the sample mean and sample standard deviation of model i respectively, and n is the sample size. Based on this test statistic, a p-value is calculated which denotes the probability under the null hypothesis of observing results which are at least as extreme as the ones observed already. Small p-values are an indication that the null hypothesis is wrong, but the confidence level under which one can confidently reject the null hypothesis is debatable [40, 86]. For the purpose of this project, a rather conservative confidence level of 0.01 is used.

5.4 Interpretability

Assessing the interpretability of a Machine Learning model is a challenging task due to the lack of a precise definition of the term and the ambiguity in its evaluation [26, 69]. Yet, it constitutes an essential requirement, especially for the deployment of the model in real-world applications, considering its correlation with accountability, robustness and trust [55, 26, 68, 32]. For the purpose of this project, a model is considered interpretable when its prediction mechanism is transparent and can be interpreted on a high level by a human [55, 26, 60]. To compare between the different models that are considered in this project, these are allocated into two classes; sufficiently interpretable and not sufficiently interpretable. A more comprehensive assessment of how interpretable each model is is considered outside this project's scope. For the allocation, results from related work are used. In essence, a model is considered sufficiently interpretable if there is a general consensus in the literature as to its good interpretability properties, and vice versa.

Out of the seven model architectures that are implemented, only the Linear and Decision Tree models are considered sufficiently interpretable. The Linear model has one of the simplest and most intuitive model structures, but this depends on the choice of feature representation [55]. Nevertheless, the two feature representations that are implemented (linear and polynomial) are all instances of relatively simple additive models and, hence, maintain the good interpretability properties of the model [56]. The Decision Tree model, an instance of a rule list architecture, is widely considered one of the most interpretable model structures due to its intuitive prediction process [25, 83, 32]. The rest of the models, are considered not sufficiently interpretable either due to their large number of parameters (Neural Networks) [32, 60, 69], ensemble status (Random Forests) [93, 41, 31], general-purpose (complex) architectures (Neural Networks, Support Vectors, Gaussian Processes) [32, 60], or use of kernels (k-Nearest Neighbours, Gaussian Processes) [32, 60, 1].

Chapter 6

Results and Discussion

6.1 Performance

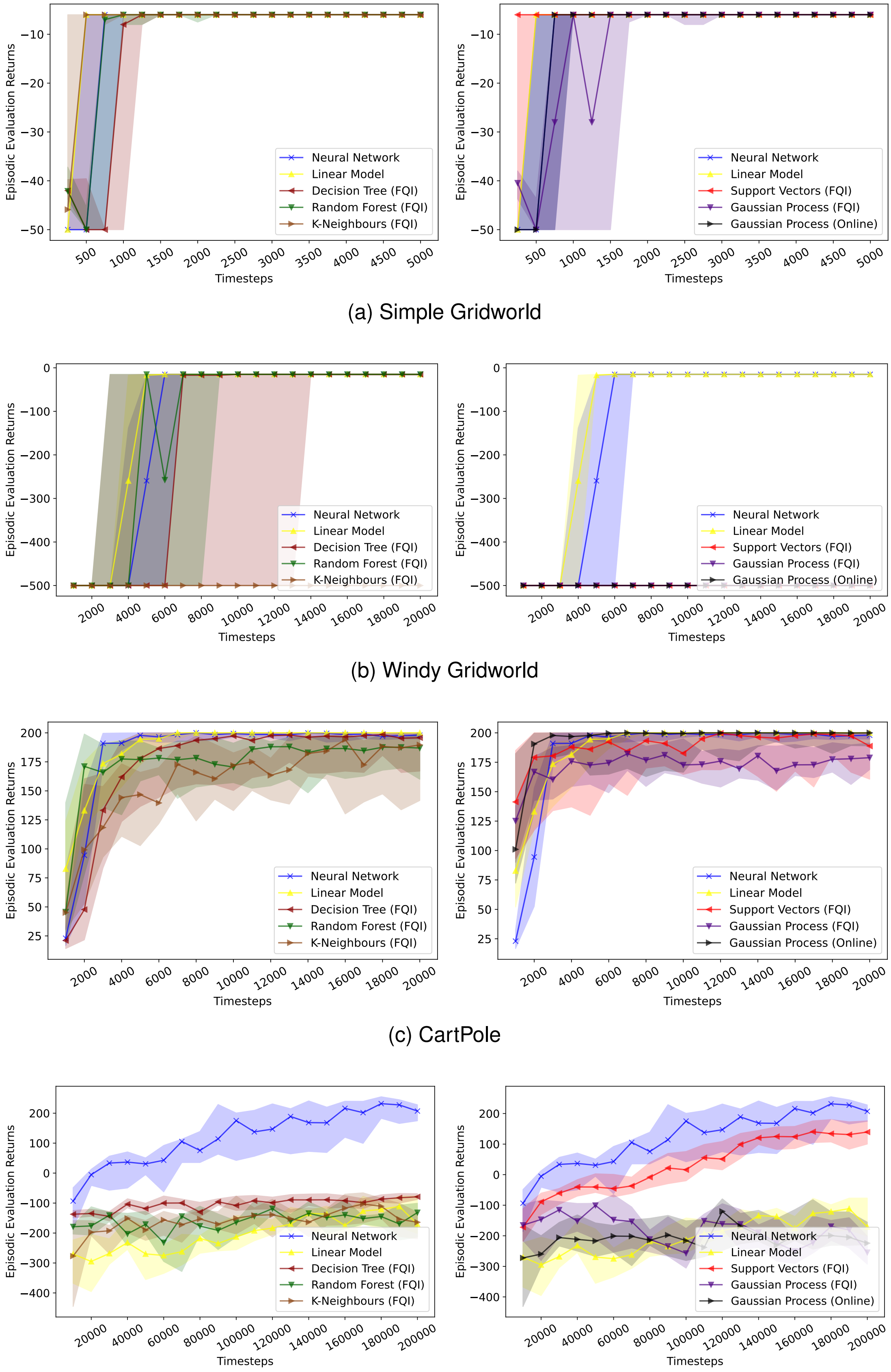
Figures 6.1 and 6.2 show the average return for each model and environment under evaluation and training modes respectively. As expected, both parametric models, –i.e. the Linear and Neural Network (NN)– were able to converge at an optimal policy consistently in all of the environments considered, apart from LunarLander where the Linear model was unable to do so. On the other hand, the non-parametric models' performance varied depending on the task.

In SimpleGridworld, all models were able to find the optimal path consistently by the end of the maximum 5,000 training steps. Notably, the Decision Tree (DT-FQI), Random Forest (RF-FQI), Support Vector Regression (SVR-FQI) and k-Nearest Neighbours (kNN-FQI) models were able to efficiently converge at an optimal policy and maintained that performance consistently throughout the training process. The Gaussian Process (GP-FQI) and Online Gaussian Process (GP-On) required a higher number of time-steps to learn a useful policy and were more unstable as indicated by their training curves.

Experiments on WindyGridworld, showed that none of the non-parametric models were able to find an optimal policy. This was due to two main reasons. First, the sparse reward signal of the environment demands a significant amount of exploration. Since the FQI-framework fits a new model at each time-step, if none of the random exploration steps results in reaching the goal, then the data which are constructed from these observations will not be useful in training the model. Second, the state-action space of this environment is discrete and there is only a limited number of unique observations. Hence, even if the goal is reached and those observations are added in the replay buffer, it is still difficult for these to be sampled and not constitute a small minority of the batch that is used as training data. These two limitations of the FQI-framework are addressed by applying a threshold of zero for adding new samples in the replay buffer, which enabled the tree-based methods to explore the environment more effectively and ultimate reach an optimal policy. However, the SVR-FQI, kNN-FQI, GP-FQI and GP-On models were still unable to formulate an appropriate input-output mapping, despite managing to explore the environment sufficiently, as seen from their training curves. This was due to their reliance on the use of kernels to define the similarity between two points using a distance metric, e.g. the Euclidean distance. In this environment this is problematic as the feature space is high-dimensional [1] and consists of binary variables whose position in the feature space is not representative of the actual distances in the grid since the Euclidean distance between any pair of one-hot vectors is the same.

In CartPole, the performances of the DT-FQI, SVR-FQI and GP-On models were comparable with the conventional models in terms of median returns. The rest of the non-conventional models achieved slightly lower median performance but with a significantly larger dispersion around this median and more unstable training curves. Nevertheless, all models appear to be able to solve the given problem, indicating that environments with a dense reward signal, no significant exploration required, and a state-space described by continuous-valued variables, fits into the strengths of these non-parametric models.

LunarLander is the most challenging of the considered environments due to the complex relationships between the state variables and actions. The agent needs to learn to avoid crushing the lander, land it efficiently, and bring it to rest after landing. These events would result in episodic returns of around 0, 100 and 200 respectively. The plot of the evaluation returns shows that the most complex models are able to formulate an appropriate relationship between the state-action pairs and q-values whilst the simpler models struggle. The SVR-FQI model converges at a policy that lands the spaceship reliably but sometimes fails to bring it to rest, as a result of insufficient exploration. The DT-FQI converges at a suboptimal policy and fails to explore the environment sufficiently. The RF-FQI, GP-FQI, GP-On and kNN-FQI models seem to be unable to represent an appropriate input-output relationship and thus fail the task.



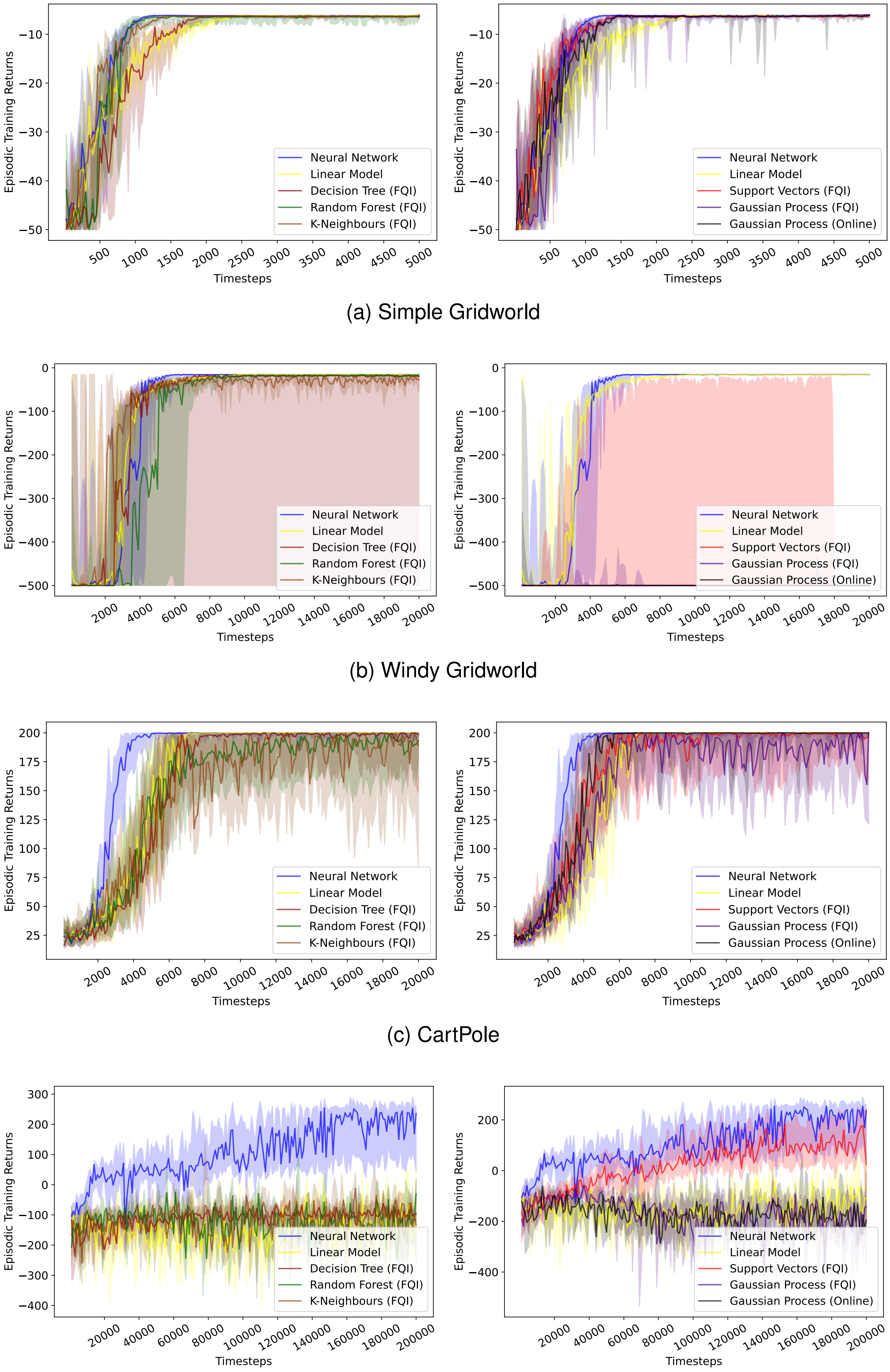
(a) Simple Gridworld

(b) Windy Gridworld

(c) CartPole

(d) LunarLander

Figure 6.1: Median (solid line) and IQR (shaded region) over thirty runs (seeds) of the average episodic evaluation return (over 10 rollouts) for each model and environment.



(a) Simple Gridworld

(b) Windy Gridworld

(c) CartPole

(d) LunarLander

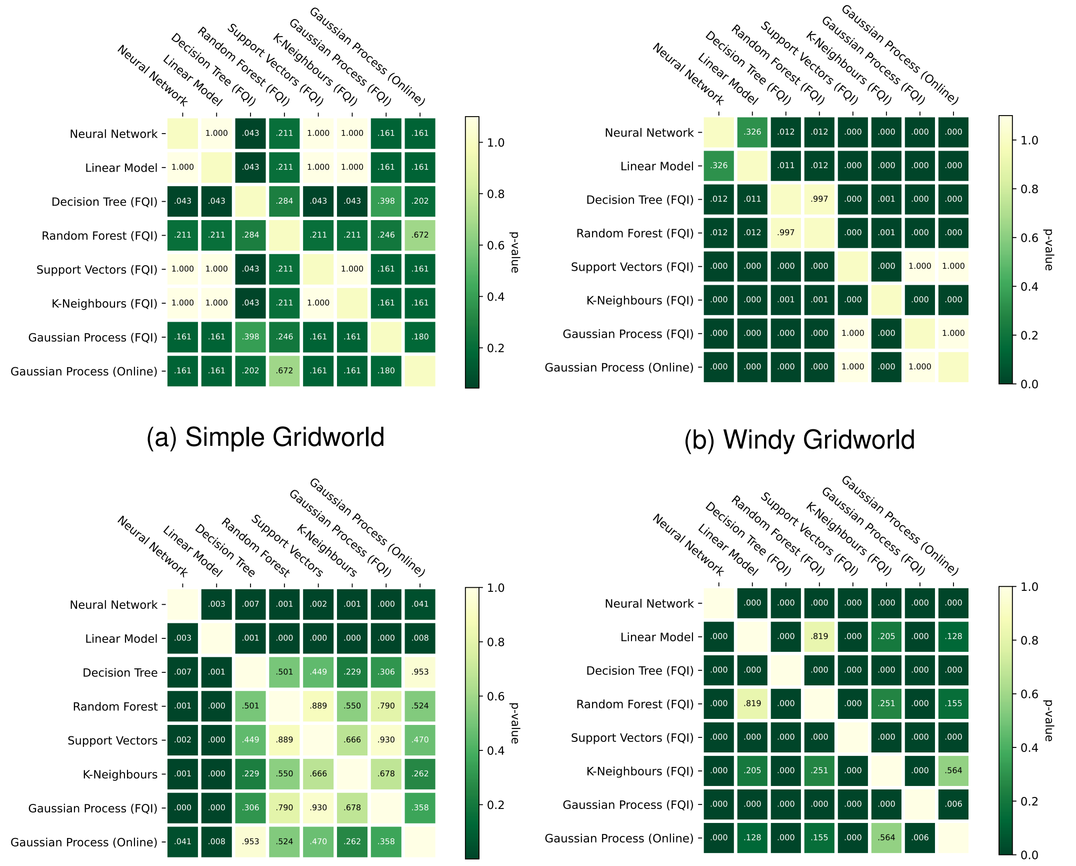
Figure 6.2: Median (solid line) and IQR (shaded region) over thirty runs (seeds) of the episodic training return for each model and environment.

6.2 Statistical Analysis

Figure 6.3 shows heatmaps which represent the p-values of Welch's t-tests between each pair of models and for each environment. In SimpleGridworld, there is no statistically significant difference between the evaluation returns after 5,000 time-steps of any model. In WindyGridworld, the difference between the evaluation returns after 20,000 time-steps of any of the Linear, NN, DT-FQI and RF-FQI models and any of the SVR-FQI, kNN-FQI, GP-FQI and GP-On is statistically significant as expected. In CartPole, the difference of the evaluation return after 20,000 time-steps between the Linear model and any other model, and the difference of the return between the NN model and any other non-parametric model are statistically significant. There are no statistically significant differences between the returns of any pairs of non-parametric models. Finally, in Lunarlander, the difference of the return between the NN or SVR-FQI models and any other model is statistically significant as expected. There is also statistically significant difference between the return of the two, with the NN model outperforming the SVR-FQI model.

6.3 Reliability

Figure 6.4 presents the 5-th percentile of the evaluation return over thirty runs (random seeds) averaged over ten evaluation rollouts measured at intermediate time-steps. Overall, it seems that the parametric models are more reliable considering this metric. It is observed that the FQI framework suffers from occasional convergence to suboptimal policies. This is because of the reliance of the framework on the construction of an appropriate training dataset at each time-step, since the model is fitted anew. This dataset is constructed from observations sampled from the replay buffer. If these happen to not sufficiently represent the state-action space, then the fitted model trained on these will provide unhelpful predictions resulting in a drop in performance. This limitation is addressed with the use of a replay buffer threshold and by storing the fitted models at intermediate time-steps and using them for predictions, as explained in Section 4.2.1 which stabilises training performance but may still result in insufficient exploration. A similar argument applies for the GP-On model which depends on sufficient exploration to add useful observations in its set of support points. The occasional convergence of the model at suboptimal policies is a result of under-exploration.



	Neural Network	Linear Model	Decision Tree (FQI)	Random Forest (FQI)	Support Vectors (FQI)	K-Neighbours (FQI)	Gaussian Process (FQI)	Gaussian Process (Online)
Neural Network		1.000	.043	.211	1.000	1.000	.161	.161
Linear Model	1.000		.043	.211	1.000	1.000	.161	.161
Decision Tree (FQI)	.043	.043		.284	.043	.043	.398	.202
Random Forest (FQI)	.211	.211	.284		.211	.211	.246	.672
Support Vectors (FQI)	1.000	1.000	.043	.211		1.000	.161	.161
K-Neighbours (FQI)	1.000	1.000	.043	.211	1.000		.161	.161
Gaussian Process (FQI)	.161	.161	.398	.246	.161	.161		.180
Gaussian Process (Online)	.161	.161	.202	.672	.161	.161	.180	

(a) Simple Gridworld

	Neural Network	Linear Model	Decision Tree (FQI)	Random Forest (FQI)	Support Vectors (FQI)	K-Neighbours (FQI)	Gaussian Process (FQI)	Gaussian Process (Online)
Neural Network		.326	.012	.012	.000	.000	.000	.000
Linear Model	.326		.011	.012	.000	.000	.000	.000
Decision Tree (FQI)	.012	.011		.997	.000	.001	.000	.000
Random Forest (FQI)	.012	.012	.997		.000	.001	.000	.000
Support Vectors (FQI)	.000	.000	.000	.000		.000	1.000	1.000
K-Neighbours (FQI)	.000	.000	.001	.001	.000		.000	.000
Gaussian Process (FQI)	.000	.000	.000	.000	1.000	.000		1.000
Gaussian Process (Online)	.000	.000	.000	.000	1.000	.000	1.000	

(b) Windy Gridworld

	Neural Network	Linear Model	Decision Tree	Random Forest	Support Vectors	K-Neighbours	Gaussian Process (FQI)	Gaussian Process (Online)
Neural Network		.003	.007	.001	.002	.001	.000	.041
Linear Model	.003		.001	.000	.000	.000	.000	.008
Decision Tree	.007	.001		.501	.449	.229	.306	.953
Random Forest	.001	.000	.501		.889	.550	.790	.524
Support Vectors	.002	.000	.449	.889		.666	.930	.470
K-Neighbours	.001	.000	.229	.550	.666		.678	.262
Gaussian Process (FQI)	.000	.000	.306	.790	.930	.678		.358
Gaussian Process (Online)	.041	.008	.953	.524	.470	.262	.358	

(c) CartPole

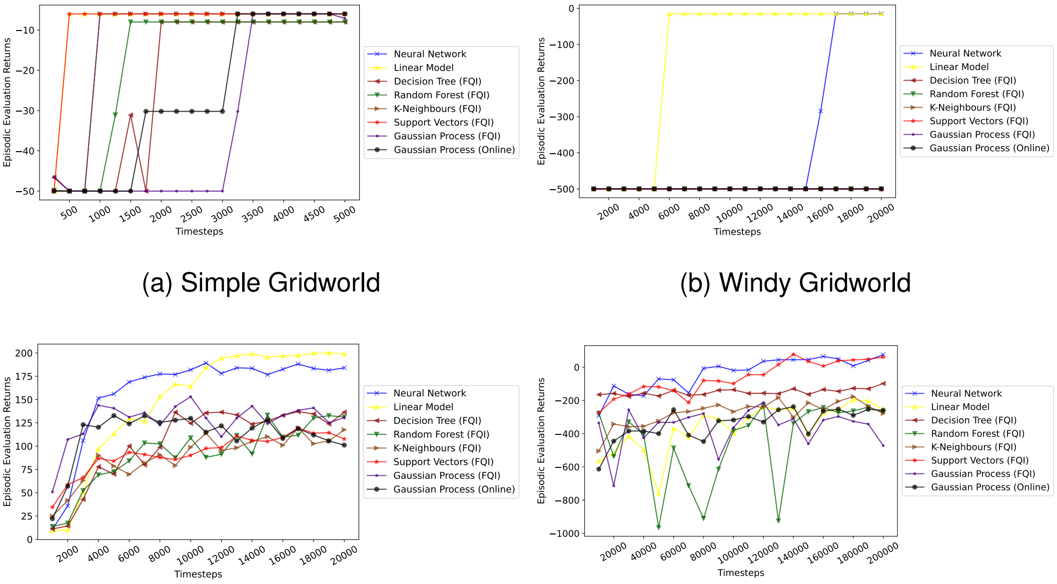
	Neural Network	Linear Model	Decision Tree (FQI)	Random Forest (FQI)	Support Vectors (FQI)	K-Neighbours (FQI)	Gaussian Process (FQI)	Gaussian Process (Online)
Neural Network		.000	.000	.000	.000	.000	.000	.000
Linear Model	.000		.000	.819	.000	.205	.000	.128
Decision Tree (FQI)	.000	.000		.000	.000	.000	.000	.000
Random Forest (FQI)	.000	.819	.000		.000	.251	.000	.155
Support Vectors (FQI)	.000	.000	.000	.000		.000	.000	.000
K-Neighbours (FQI)	.000	.205	.000	.251	.000		.000	.564
Gaussian Process (FQI)	.000	.000	.000	.000	.000	.000		.006
Gaussian Process (Online)	.000	.128	.000	.155	.000	.564	.006	

(d) LunarLander

Figure 6.3: Heatmap of Welch's t-test p-values for each pair of models on the difference of their average evaluation returns after the maximum number of training time-steps. A small p-value provides evidence for the rejection of the null hypothesis –that this difference is zero– at the given significance level.

6.4 Sample Efficiency

Figure 6.5 presents boxplots for the number of steps required for each model to reach a certain level of performance in each of the considered environments. The target performance is environment specific. For the SimpleGridworld and WindyGridworld environments, this was set to an episodic evaluation return of at least -6 and -15 respectively, reflecting the optimal path to the goal. In the CartPole environment, this was set to an episodic evaluation return of at least 195, again reflecting optimal behaviour. In the LunarLander environment, the target performance was set to an episodic evaluation return of at least 100. Even though this is not representative of solving optimally the given problem (which would require a return of at least 200), it was deemed a more in-



(a) Simple Gridworld

(b) Windy Gridworld

(c) CartPole

(d) LunarLander

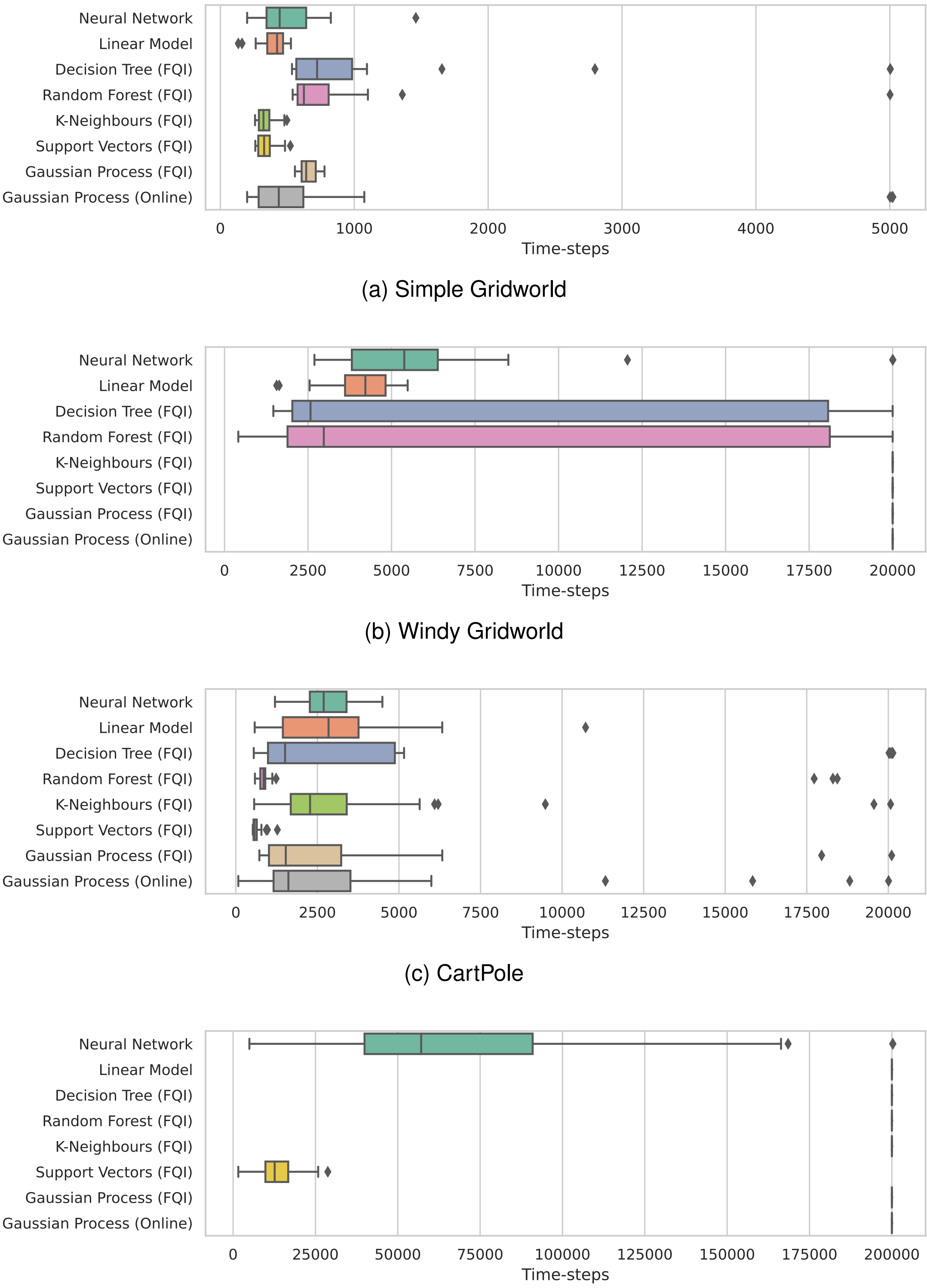
Figure 6.4: 5-th percentile (q_5) over thirty runs (seeds) of the average episodic evaluation return (over 10 rollouts) for each model and environment.

formative measurement given that most models, both parametric and non-parametric, struggled in this environment.

In SimpleGridworld, the Linear, kNN-FQI and SVR-FQI models were able to reach an optimal policy within less than 500 time-steps consistently whilst the rest of the models required slightly more than 500 time-steps according to the median and IQR of the collected samples. The DT-FQI, RF-FQI and GP-On models had also runs where the required performance was not reached at all. In WindyGridworld, the Linear Model seems to be the most reliable with the vast majority of its runs reaching the target within 5,000 time-steps. The kNN-FQI, SVR-FQI, GP-FQI and GP-On models did not reach the target performance in any of the runs. Interestingly, both DT-FQI and RF-FQI models outperform the parametric models in terms of median number of steps required, but the right tails of their distributions are significantly longer due to having runs where the required performance was not reached. Nevertheless, this shows that if the environment is explored sufficiently, these models are able to utilise the collected data more efficiently. In CartPole, all non-parametric models outperformed the parametric ones in terms of median number of steps. The DT-FQI, RF-FQI, GP-FQI and GP-On had a few runs where the required performance was not reached. On the other hand, the RF-FQI and SVR-FQI models not only achieved the required level of performance consistently, they were also able to reach it within 1,000 time-steps in the vast majority of their runs, vastly outperforming all other models. This is a strong indication of how sample efficient these models are, in environments where the reward signal is dense and no significant exploration is required. Finally, in LunarLander, none of the Linear, DT-FQI, RF-FQI, kNN-FQI, GP-FQI and GP-On models reached the required level of performance in any of their runs. The NN and SVR-FQI models on the other hand, were able to do so consistently. In addition, the SVR-FQI model seems to be significantly more sample efficient than the NN model when the median and IQR of the number of samples required to reach the target performance is taken into account. This is consistent with what was observed on CartPole, despite the LunarLander environment requiring considerable higher levels of exploration.

6.5 Training Time

Table 6.1 shows the wall-clock time required for each model to complete a training run for each of the environments considered. As expected, the conventional models' training times is significantly lower due to, first, the use of a highly optimised library for their implementation, second, the continual model fitting that is required for the non-conventional models under the FQI-framework and, third, the manual implementation of the calculations required by the GP-On model. The Linear model required more time than the NN model on *CartPole*, despite being a simpler model, due to the manual implementation of the polynomial feature representation that was used and the small number of hidden layers (2) and units (32 each) of the NN model. This was not the case on LunarLander, where the large number of hidden units of the NN model (256 and 128 for the first and second hidden layers respectively) required more training time than the Linear model, despite also using polynomial features. Within the non-conventional models, the DT-FQI model required less time than the more complex models of RF-FQI, SV-FQI and GP-FQI as expected. The kNN-FQI model, required the most training time due the large batch size, which has a large impact on the the prediction time of the model. The GP-On model required significantly less training time than most FQI-based models, indicating the effectiveness of the sparsification technique that was used. Further, comparing the two GP-based models, there was a considerable difference in training times, yet no significant performance differences. Finally, thresholding the addition of observations in the replay buffer on *WindyGridworld* and *LunarLander* for the FQI-based models, reduced significantly the training time of these models.



(a) Simple Gridworld

(b) Windy Gridworld

(c) CartPole

(d) LunarLander

Figure 6.5: Boxplots of the number of time-steps required by each model to reach certain levels of performance in the considered environments. The target levels of performance are episodic returns of at least 195, -6, -15 and 100 for the SimpleGridworld, WindyGridworld, CartPole and LunarLander environments respectively.

Model \ Environment	Simple Gridworld	Windy Gridworld	CartPole	Lunar Lander
Linear Model	4	15	96	361
Neural Network	7	31	26	615
Decision Tree (FQI)	37	34	106	1432
Random Forest (FQI)	122	79	464	8762
K-Neighbors (FQI)	572	134	632	12225
Support Vectors (FQI)	214	25	423	5253
Gaussian Process (FQI)	280	26	550	10400
Gaussian Process (Online)	56	260	356	4142

Table 6.1: Training time (wall-clock time) for each model in seconds.

6.6 Interpretability

As discussed in Section 5.4, only the Linear and Decision Tree models were deemed sufficiently interpretable. This section demonstrates how their prediction mechanisms can be interpreted, by considering the CartPole environment and extracting the learned models at the end of training. The Linear model is fully described in terms of two equations which are used for predicting the q-value of the two actions in the environment (push_left, push_right) for a given state $s = (s_{\text{pos}}, s_{\text{vel}}, s_{\text{ang}}, s_{\text{ang\_vel}})$:

$$\begin{aligned}\hat{q}(s, push\_left) = 18.81 &- 10.28\, s_{\text{pos}} + 1.29\, s_{\text{vel}} - 0.34\, s_{\text{ang}} + 0.48\, s_{\text{ang\_vel}} \\ &+ 9.42\, s_{\text{pos}}^2 + 4.58\, s_{\text{vel}}^2 + 0.32\, s_{\text{ang}}^2 - 2.64\, s_{\text{ang\_vel}}^2 \\ &- 5.16\, s_{\text{pos}}\, s_{\text{vel}} - 3.71\, s_{\text{pos}}\, s_{\text{ang}} + 3.57\, s_{\text{pos}}\, s_{\text{ang\_vel}} \\ &+ 11.30\, s_{\text{vel}}\, s_{\text{ang}} - 0.31\, s_{\text{vel}}\, s_{\text{ang\_vel}} - 10.62\, s_{\text{ang}}\, s_{\text{ang\_vel}}\end{aligned} \tag{6.1}$$

$$\begin{aligned}\hat{q}(s, push\_right) = 18.88 &- 14.22\, s_{\text{pos}} + 4.26\, s_{\text{vel}} + 16.59\, s_{\text{ang}} + 3.47\, s_{\text{ang\_vel}} \\ &+ 13.64\, s_{\text{pos}}^2 + 6.22\, s_{\text{vel}}^2 + 11.72\, s_{\text{ang}}^2 - 1.81\, s_{\text{ang\_vel}}^2 \\ &- 0.54\, s_{\text{pos}}\, s_{\text{vel}} - 5.39\, s_{\text{pos}}\, s_{\text{ang}} + 8.44\, s_{\text{pos}}\, s_{\text{ang\_vel}} \\ &+ 12.85\, s_{\text{vel}}\, s_{\text{ang}} + 1.06\, s_{\text{vel}}\, s_{\text{ang\_vel}} - 6.77\, s_{\text{ang}}\, s_{\text{ang\_vel}}\end{aligned} \tag{6.2}$$

The Decision Tree model is fully described in terms of the binary conditions which split the tree at each internal node. These are visualised in Figure 6.6. The tree has a maximum depth of ten and takes as inputs the four state variables of the environment along with the two actions for a total of 6 features.

To understand what these models predict for a given state, some illustrative state descriptions are chosen and their respective q-values under the Linear and Decision Tree models are calculated (Table 6.2). It is observed that the linear model is biased towards the action push_right when the pole is up-right and both cart and pole are stationary (as in the first and second examples) no matter where the cart is positioned. Intuitively, the action push_left should be preferred when the cart is positioned to the right of the centre. The rest of the examples show that the linear model behaves as expected. As for the Decision Tree model, it seems that it is indifferent to the position and velocity of the cart when both angle and angular velocity are zero (as in the second and third examples), which may result in erroneous actions. The rest of the examples indicate that the Decision Tree model behaves as expected. Nevertheless, this analysis was only possible due to the two models being sufficiently interpretable and demonstrates the importance of being able to test and observe the behaviour of a given model when its robustness and reliability are important, as is the case for many real-world applications.

State (s)	$\hat{q}_{linear}(s, left)$	$\hat{q}_{linear}(s, right)$	$\hat{q}_{dt}(s, left)$	$\hat{q}_{dt}(s, right)$
[0, 0, 0, 0]	18.81	**18.88**	52.58	52.58
[+1 (-1), 0, 0, 0]	17.95 (38.51)	**18.30** (**46.74**)	52.58 (52.32)	52.58 (52.32)
[0, +1 (-1), 0, 0]	24.68 (**22.10**)	**29.36** (20.84)	52.58 (52.01)	52.58 (52.01)
[0, 0, +1 (-1), 0]	18.79 (**19.47**)	**47.19** (14.01)	34.97 (35.87)	**48.00** (35.87)
[0, 0, 0, +1 (-1)]	16.65 (**15.69**)	**20.54** (13.60)	21.44 (**33.45**)	**42.95** (18.879)

Table 6.2: q-values for the actions push_left and push_right for different state descriptions on CartPole under the Linear and Decision Tree models. Largest values in bold.

6.7 General Observations

Fitted-Q Iteration: The use of the FQI framework alongside the non-parametric models which were considered as part of this project has demonstrated the viability of adapting these methods in the RL context. The main advantages of using the framework are the flexibility in using any supervised regression model and the small number

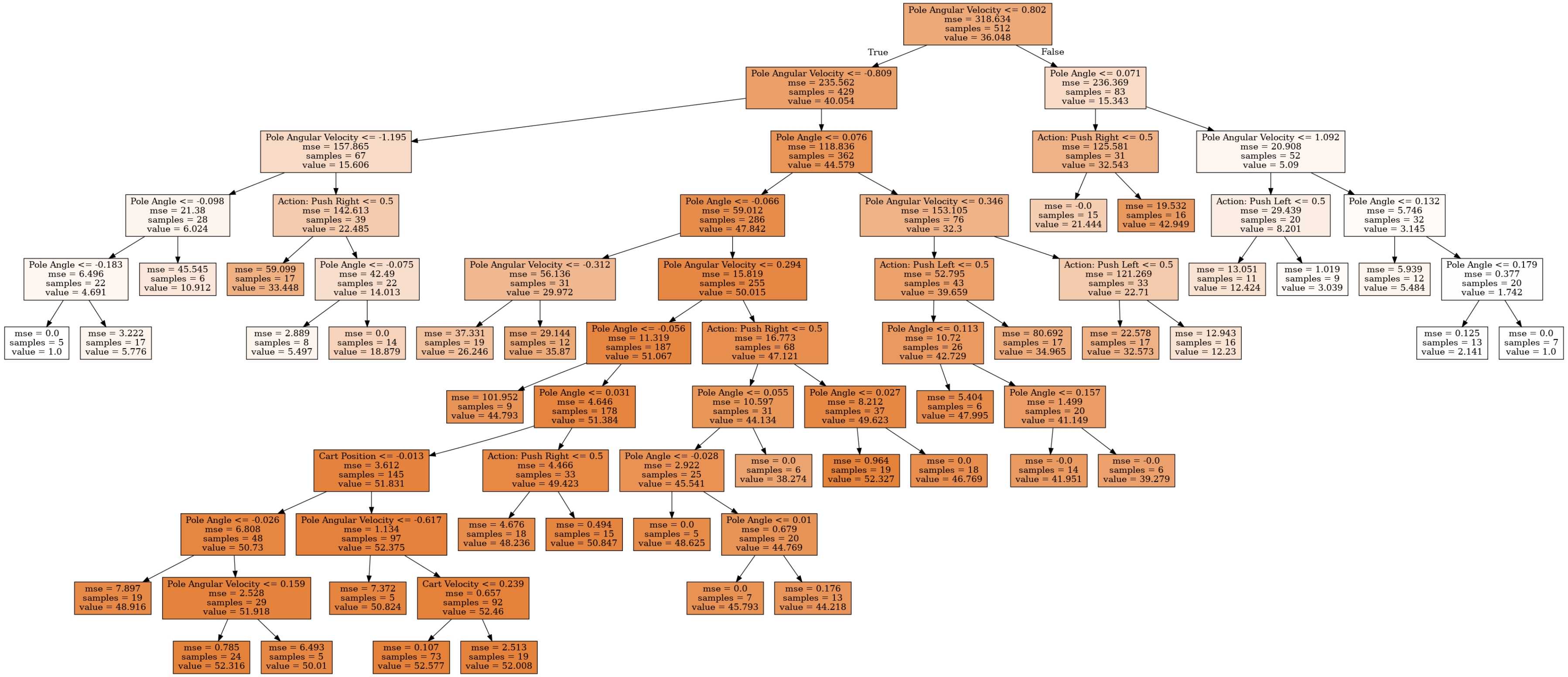

Pole Angular Velocity <= 0.802 mse = 318.634 samples = 512 value = 36.048
True
False
Pole Angular Velocity <= -0.809 mse = 235.562 samples = 429 value = 40.054
Pole Angle <= 0.071 mse = 236.369 samples = 83 value = 15.343
Pole Angular Velocity <= -1.195 mse = 157.865 samples = 67 value = 15.606
Pole Angle <= 0.076 mse = 118.836 samples = 362 value = 44.579
Action: Push Right <= 0.5 mse = 125.581 samples = 31 value = 32.543
Pole Angular Velocity <= 1.092 mse = 20.908 samples = 52 value = 5.09
Pole Angle <= -0.098 mse = 21.38 samples = 28 value = 6.024
Action: Push Right <= 0.5 mse = 142.613 samples = 39 value = 22.485
Pole Angle <= -0.066 mse = 59.012 samples = 286 value = 47.842
Pole Angular Velocity <= 0.346 mse = 153.105 samples = 76 value = 32.3
mse = -0.0 samples = 15 value = 21.444
mse = 19.532 samples = 16 value = 42.949
Action: Push Left <= 0.5 mse = 29.439 samples = 20 value = 8.201
Pole Angle <= 0.132 mse = 5.746 samples = 32 value = 3.145
Pole Angle <= -0.183 mse = 6.496 samples = 22 value = 4.691
mse = 45.545 samples = 6 value = 10.912
mse = 59.099 samples = 17 value = 33.448
Pole Angle <= -0.075 mse = 42.49 samples = 22 value = 14.013
Pole Angular Velocity <= -0.312 mse = 56.136 samples = 31 value = 29.972
Pole Angular Velocity <= 0.294 mse = 15.819 samples = 255 value = 50.015
Action: Push Left <= 0.5 mse = 52.795 samples = 43 value = 39.659
Action: Push Left <= 0.5 mse = 121.269 samples = 33 value = 22.71
mse = 13.051 samples = 11 value = 12.424
mse = 1.019 samples = 9 value = 3.039
mse = 5.939 samples = 12 value = 5.484
Pole Angle <= 0.179 mse = 0.377 samples = 20 value = 1.742
mse = 0.0 samples = 5 value = 1.0
mse = 3.222 samples = 17 value = 5.776
mse = 2.889 samples = 8 value = 5.497
mse = 0.0 samples = 14 value = 18.879
mse = 37.331 samples = 19 value = 26.246
mse = 29.144 samples = 12 value = 35.87
Pole Angle <= -0.056 mse = 11.319 samples = 187 value = 51.067
Action: Push Right <= 0.5 mse = 16.773 samples = 68 value = 47.121
Pole Angle <= 0.113 mse = 10.72 samples = 26 value = 42.729
mse = 80.692 samples = 17 value = 34.965
mse = 22.578 samples = 17 value = 32.573
mse = 12.943 samples = 16 value = 12.23
mse = 0.125 samples = 13 value = 2.141
mse = 0.0 samples = 7 value = 1.0
mse = 101.952 samples = 9 value = 44.793
Pole Angle <= 0.031 mse = 4.646 samples = 178 value = 51.384
Pole Angle <= 0.055 mse = 10.597 samples = 31 value = 44.134
Pole Angle <= 0.027 mse = 8.212 samples = 37 value = 49.623
mse = 5.404 samples = 6 value = 47.995
Pole Angle <= 0.157 mse = 1.499 samples = 20 value = 41.149
Cart Position <= -0.013 mse = 3.612 samples = 145 value = 51.831
Action: Push Right <= 0.5 mse = 4.466 samples = 33 value = 49.423
Pole Angle <= -0.028 mse = 2.922 samples = 25 value = 45.541
mse = 0.0 samples = 6 value = 38.274
mse = 0.964 samples = 19 value = 52.327
mse = 0.0 samples = 18 value = 46.769
mse = -0.0 samples = 14 value = 41.951
mse = -0.0 samples = 6 value = 39.279
Pole Angle <= -0.026 mse = 6.808 samples = 48 value = 50.73
Pole Angular Velocity <= -0.617 mse = 1.134 samples = 97 value = 52.375
mse = 4.676 samples = 18 value = 48.236
mse = 0.494 samples = 15 value = 50.847
mse = 0.0 samples = 5 value = 48.625
Pole Angle <= 0.01 mse = 0.679 samples = 20 value = 44.769
mse = 7.897 samples = 19 value = 48.916
Pole Angular Velocity <= 0.159 mse = 2.528 samples = 29 value = 51.918
mse = 7.372 samples = 5 value = 50.824
Cart Velocity <= 0.239 mse = 0.657 samples = 92 value = 52.46
mse = 0.0 samples = 7 value = 45.793
mse = 0.176 samples = 13 value = 44.218
mse = 0.785 samples = 24 value = 52.316
mse = 6.493 samples = 5 value = 50.01
mse = 0.107 samples = 73 value = 52.577
mse = 2.513 samples = 19 value = 52.008

Figure 6.6: Visualisation of a Decision Tree model (max_depth=10) which was trained under the Fitted-Q Iteration algorithm and achieved optimal behaviour on CartPole.

of hyperparameters that it involves. Its main limitations are its poor performance in environments where a significant amount of exploration is required, the added computational and memory costs that it demands as a result of fitting a model anew at each time-step, and the performance unreliability which results from its dependency on the good quality of the training data.

Tree-based methods: Tree-based methods achieved good results in most of the environments tested. It was observed that the maximum depth parameter has a significant effect on the model's performance. It is noted that the tuning process for this parameter can be aided by utilising prior information on the environment where available. For example, understanding the environment dynamics in CartPole and considering that the model takes as input a 6-dimensional variable, one can refine the range of parameter values considered to range between four and fifteen. Despite the ability, in theory, of DTs of all depths considered to derive an optimal action-value function, it was observed that shallower trees resulted in poor performance and large variance whilst deeper trees resulted in unstable performance within runs as a result of overfitting. Hence, tuning successfully this hyperparameter is essential. Further, it was observed that there is no significant improvement in performance when considering a tree-ensemble method –such as the RF model– instead of a simple DT architecture, but there is some evidence for improved sample efficiency based on CartPole results. Finally, it is noted that the DT model's main advantage lies in its interpretability status as discussed in Section 6.6.

k-Nearest Neighbours: Experimentation with the kNN-FQI model showed some promising results but also revealed some limitations. Its reliance on the appropriateness of a distance measure to quantify the relationships between two points meant that its use is inappropriate in some environments, e.g gridworlds. In addition, the model under-performed in LunarLander as it was unable to formulate an appropriate input-output mapping. Finally, its training time was the longest of all models considered, with no particular advantages in terms of performance to justify this computational burden.

Support Vectors: The SVR-FQI model was the most successful of the non-parametric models in terms of performance on the given tasks, as it performed comparably with the parametric models on CartPole and SimpleGridworld, and outperformed the Linear model and underperformed compared to the Neural Network on LunarLander. However, it was unable to solve WindyGridworld due to the use of kernels and was deemed unreliable on CartPole when its 1-in-20 worst case performance was considered, due to its implementation through the FQI framework. Still, its major strength seemed to be its sample efficiency, as it was able to consistently reach the required levels of performance on SimpleGridworld, CartPole and LunarLander within the least amount of time-steps compared to the rest of the models. It is also noteworthy to mention its flexibility, given that only a single hyperparameter was tuned, the regularisation parameter C, and yet the model was able to formulate an appropriate input-output mapping in all of the environments considered (apart from WindyGridworld).

Gaussian Processes: Two Gaussian Processes models were implemented, a batch-based method under the FQI framework, and an incremental method which took advantage of a sparsification algorithm. Both models struggled on the gridworld environments due to their reliance on kernels and on LunarLander due to their inability to explore the environment sufficiently. On CartPole, both models performed well with no statistical difference between their evaluation returns after the end of training, but their 1-in-20 worst case performance showed signs of unreliability. Further, their training times showed that the online algorithm required significantly less time than the FQI-based implementation as a result of the sparsification that was applied. Finally, it is noted that the use of the variance estimations of the GP model as part of an informative exploration strategy did not work well, despite the argument from related work that this constitutes the main advantage of using this model in the RL context [77, 21, 47, 18, 38].

Chapter 7

Conclusion

This research aimed to identify whether a non-conventional value function approximation approach is advantageous over the Neural Network or Linear model based methods which have been conventionally used in the Reinforcement Learning setting. The implementation of promising approaches as identified from surveying related work, and their empirical evaluation under a consistent framework has allowed the systematic comparison between different models. The main research hypothesis was that there exist non-conventional models which offer advantages under certain criteria. Indeed, even though the Linear and Neural Network models performed strongly in terms of average evaluation returns and reliability in almost all environments, they underperformed in terms of sample efficiency, with the Neural Network also suffering from a lack of interpretability. In particular, the Support Vector Regression model implemented under the Fitted-Q Iteration framework vastly outperformed all other models in terms of sample efficiency in most environments. Further, the use of Decision Trees allowed the interpretation of the prediction process through the extraction and visualisation of the fitted model, offering a strong advantage in terms of interpretability.

Nonetheless, Gaussian Processes and the utilisation of their uncertainty estimates in an informative exploration strategy achieved poor results, contrary to previous work suggesting otherwise [21, 47, 18, 17]. Still, the consideration of different exploration strategies was by no means exhaustive and it focused on the adaptation of the Upper Confidence Bound (UCB) acquisition function, in contrast with most related work, drawing insights from the Bayesian Optimisation domain [84]. Future work could additionally consider the use of different acquisition functions, such as Probability of Improvement (PoI) or Expected Improvement (EI) [84], or different exploration strategies such as the information gain-based method considered by Chung et al. [18], or the rmax-based method considered by Jung and Stone [47].

A major remark based on the experimental results of this work is that the non-parametric nature of the non-conventional methods is both their main advantage and limitation. Advantage because of their flexibility, small number of parameters to tune and expressiveness. Limitation since they cannot be naturally updated incrementally and, thus, they do not fit the online data generation framework of Reinforcement Learning. As a result of this limitation, it was deemed necessary to adapt them through the Fitted-Q Iteration framework. Despite its wide success, the framework under-performs in environments where significant exploration is required, and it is prone to occasional convergence at sub-optimal policies. Nevertheless, it allows the use of any supervised regression model taking advantage of the strengths of each method, as discussed in the previous chapter. Moreover, the choice of which supervised regression model to use is a major modelling choice which impacts significantly performance on the given task. For example, the use of kernel-based methods –such as k-Nearest Neighbours, Support Vectors and Gaussian Processes– requires a well-defined and meaningful distance measure between the state-action regions, and is, thus, inappropriate in environments where the state-space is described in terms of binary or one-hot variables.

The scope of this work was limited to single-agent Reinforcement Learning problems and model-free, action-value function approximation solution approaches. An interesting direction for future work could be the consideration of multi-agent problems, which is currently lacking from the literature. Further, the expansion to model-based solution methods seems promising, as seen from related work surveyed in Chapter 3 [77, 23, 47, 21, 22]. Additionally, the framework of policy-gradient or actor-critic methods may fit into the strengths of many of the non-parametric models that were considered in this project. For discrete action-spaces, the policy-gradient and actor-critic methods involve instances of supervised classification problems, a class of problems for which most of the considered models (such as tree-based methods, support vector machines, and k-nearest neighbours) were originally designed. It would be interesting to explore how these models perform on classification problems under such methods, and indeed, there has been a surge in interest over recent years in this area [4, 35].

One of the main motivations for undertaking this research, was to understand why these non-parametric models remain under-utilised –especially in applications outside academia– despite the significant amount of research on their development and evaluation. It was argued that the lack of a systematic evaluation and comparison of these different approaches on a range of problems and environments is one of the main hindrances to their wider adaptation. In fact, the work of this project has provided some indications on the relative strengths and weaknesses of each method and will, hopefully, allow their use where they are deemed useful, and promote the evaluation of future methods under a consistent framework which enables their systematic comparison with alternative approaches.

To conclude, the consideration of alternative to the conventional approaches has become increasingly important in recent years due to the demand for sample efficient, interpretable and accountable models, properties which the conventional models lack. The work of this research can inform future work on the development of approaches which are fit for purpose, do not suffer from the identified limitations and can be used as viable alternatives. Based on the empirical results provided in this work, this direction seems promising.

Bibliography

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1973, 2001.

[2] Alejandro Agostini and Enric Celaya. Reinforcement learning with a Gaussian mixture model. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, 7 2010.

[3] Alnour Alharin, Thanh Nam Doan, and Mina Sartipi. Reinforcement learning interpretation methods: A survey. *IEEE Access*, 8, 2020.

[4] Yuexuan An, Shifei Ding, Songhui Shi, and Jingcan Li. Discrete space reinforcement learning algorithm based on support vector machine classification. *Pattern Recognition Letters*, 111, 2018.

[5] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey, 2017.

[6] Hal Ashton. Causal Campbell-Goodhart's law and Reinforcement Learning, 2020.

[7] André M.S. Barreto, Doina Precup, and Joelle Pineau. Practical kernel-based reinforcement learning. *Journal of Machine Learning Research*, 17, 2016.

[8] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5), 1983.

[9] Richard Bellman. A Markovian Decision Process. *Indiana University Mathematics Journal*, 6(4), 1957.

[10] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[11] Leo Breiman. Random forests. *Machine Learning*, 45(1), 2001.

[12] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[14] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017.

[15] A. Castelletti, S. Galelli, M. Restelli, and R. Soncini-Sessa. Tree-based reinforcement learning for optimal water reservoir operation. *Water Resources Research*, 46(9), 2010.

[16] Stephanie C.Y. Chan, Samuel Fishman, John Canny, Anoop Korattikara, and Sergio Guadarrama. Measuring the reliability of reinforcement learning algorithms, 2019.

[17] Girish Chowdhary, Miao Liu, Robert Grande, Thomas Walsh, Jonathan How, and Lawrence Carin. Off-policy reinforcement learning with Gaussian processes. *IEEE/CAA Journal of Automatica Sinica*, 1(3), 2014.

[18] Jen Jen Chung, Nicholas R.J. Lawrance, and Salah Sukkarieh. Gaussian processes for informative exploration in reinforcement learning. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2013.

[19] Cédric Colas, Olivier Sigaud, and Pierre Yves Oudeyer. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms, 2019.

[20] Lehel Csató and Manfred Opper. Sparse on-line Gaussian processes. *Neural computation*, 14(3):641–668, 2002.

[21] Marc Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes*. 2010.

[22] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2), 2015.

[23] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9), 2009.

[24] Thomas G Dietterich and Xin Wang. Support vectors for reinforcement learning. In *European Conference on Machine Learning*, page 600. Springer, 2001.

[25] Zihan Ding, Pablo Hernandez-Leal, Gavin Weiguang Ding, Changjian Li, and Ruitong Huang. CDT: Cascading Decision Trees for Explainable Reinforcement Learning. *arXiv preprint arXiv:2011.07553*, 2020.

[26] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[27] Yaakov Engel, Shie Mannor, and Ron Meir. Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning. In *Proceedings, Twentieth International Conference on Machine Learning*, volume 1, 2003.

[28] Yaki Engel, Shie Mannor, and Ron Meir. Bayesian Reinforcement Learning with Gaussian Process Temporal Difference Methods. *Electrical Engineering*, 2008.

[29] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005.

[30] Gennaro Esposito and Mario Martin. Bellman residuals minimization using online support vector machines. *Applied Intelligence*, 47(3), 2017.

[31] C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. From ensemble methods to comprehensible models. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 2534, 2002.

[32] Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1), 2014.

[33] Matteo Gaeta, Francesco Orciuoli, Luigi Rarità, and Stefania Tomasiello. Fitted Q-iteration and functional networks for ubiquitous recommender systems. *Soft Computing*, 21(23), 2017.

[34] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[35] Mohammad Ghavamzadeh, Yaakov Engel, and Michal Valko. Bayesian policy gradient and actor-critic algorithms. *Journal of Machine Learning Research*, 17, 2016.

[36] Geoffrey J Gordon. *Approximate Solutions to*
}*Markov Decision Processes. Carnegie Mellon University, 1999.*

[37] Ryo Goto and Hiroshi Matsuo. State generalization method with support vector machines in reinforcement learning. Systems and Computers in Japan*, 37(9), 2006.*

[38] Robert C. Grande, Thomas J. Walsh, and Jonathan P. How. Sample efficient reinforcement learning with Gaussian processes. In 31st International Conference on Machine Learning, ICML 2014*, volume 4, 2014.*

[39] Anil K. Gupta, Ken G. Smith, and Christina E. Shalley. The interplay between exploration and exploitation. Academy of Management Journal*, 49(4), 2006.*

[40] Lewis G. Halsey, Douglas Curran-Everett, Sarah L. Vowler, and Gordon B. Drummond. The fickle P value generates irreproducible results, 2015.

[41] Satoshi Hara and Kohei Hayashi. Making tree ensembles interpretable: A Bayesian model selection approach. In International Conference on Artificial Intelligence and Statistics, AISTATS 2018*, 2018.*

[42] A E Howe and L D Pyeatt. Decision Tree Function Approximation in Reinforcement Learning. Technical report, 1998.

[43] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. Science*, 364(6443), 2019.*

[44] Hunor Jakab, Botond Bocsi, and Lehel Csato. Non-parametric value function approximation in robotics. In MACS2010: The 8th Joint Conference on Mathematics and Computer Science, volume Selected Papers*, pages 235–248, 2011.*

[45] *M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects, 2015.*

[46] *H. José Antonio Martín and Javier De Lope. Exa: An effective algorithm for continuous actions reinforcement learning problems. In* IECON Proceedings (Industrial Electronics Conference)*, 2009.*

[47] *Tobias Jung and Peter Stone. Gaussian processes for sample efficient reinforcement learning with RMAX-like exploration. In* Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6321 LNAI, 2010.*

[48] *L P Kaelbling, M L Littman, and A W Moore. Reinforcement Learning: A Survey.* Journal of Artificial Intelligence Research*, 4:237–285, 5 1996.*

[49] *Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In* 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.*

[50] *George Konidaris, Sarah Osentoski, and Philip Thomas. Value Function Approximation in Reinforcement Learning Using the Fourier Basis.* Proceedings of the AAAI Conference on Artificial Intelligence*, 25(1), 8 2011.*

[51] *Michail G Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In* Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 424–431, 2003.*

[52] *Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In* Reinforcement learning*, pages 45–73. Springer, 2012.*

[53] *Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning.* nature*, 521(7553):436–444, 2015.*

[54] *Dong Hyun Lee, Vo Van Quang, Sungho Jo, and Ju Jang Lee. Online support vector regression based value function approximation for reinforcement learning. In* IEEE International Symposium on Industrial Electronics*, 2009.*

[55] *Zachary C. Lipton. The mythos of model interpretability.* Communications of the ACM*, 61(10), 2018.*

[56] Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible Models for Classification and Regression. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 150–158, New York, NY, USA, 2012. Association for Computing Machinery.*

[57] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying Chang Liang, and Dong In Kim. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey, 2019.

[58] Richard Maclin, Jude Shavlik, Trevor Walker, and Lisa Torrey. Knowledge-based support-vector regression for reinforcement learning. Reasoning, Representation, and Learning in Computer Games*, page 61, 2005.*

[59] Hicham Mansouri and Theodore B. Trafalis. Reinforcement learning: An on-line framework using support vectors. In IEEE International Conference on Neural Networks - Conference Proceedings*, 2007.*

[60] David Martens, Jan Vanthienen, Wouter Verbeke, and Bart Baesens. Performance of classification models from a user perspective. Decision Support Systems*, 51(4), 2011.*

[61] José Antonio Martín H. and Javier De Lope. A k-NN based perception scheme for reinforcement learning. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4739 LNCS, 2007.*

[62] José Antonio Martín H., Javier De Lope, and Darío Maravall. The kNN-TD reinforcement learning algorithm. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5601 LNCS, 2009.*

[63] José Antonio Martín H, Javier de Lope, and Darío Maravall. Robust high performance reinforcement learning through weighted k-nearest neighbors. Neurocomputing*, 74(8), 2011.*

[64] Brida V. Mbuwir, Mahtab Kaffash, and Geert Deconinck. Battery Scheduling in a Residential Multi-Carrier Energy System Using Reinforcement Learning. In 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids, SmartGridComm 2018*, 2018.*

[65] Francisco S Melo, Sean P Meyn, and M Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 664–671, Helsinki, Finland, 2008. ACM Press.*

[66] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature*, 518(7540), 2015.*

[67] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning*. MIT press, 2018.*

[68] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Interpretable Machine Learning – A Brief History, State-of-the-Art and Challenges. In Communications in Computer and Information Science*, volume 1323, 2020.*

[69] W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. Proceedings of the National Academy of Sciences of the United States of America*, 116(44), 2019.*

[70] Gerhard Neumann and Jan Peters. Fitted Q-iteration by advantage weighted regression. In Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, 2009.*

[71] Dirk Ormoneit and Åśaunak Sen. Kernel-based reinforcement learning. Machine Learning*, 49(2-3), 2002.*

[72] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 752–759, Helsinki, Finland, 2008. ACM Press.*

[73] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, and Luca Antiga.

Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems*, 32:8026–8037, 2019.*

[74] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research*, 12(85):2825–2830, 2011.*

[75] Adolfo Perrusquía and Wen Yu. Robust control under worst-case uncertainty for unknown nonlinear systems using modified reinforcement learning. International Journal of Robust and Nonlinear Control*, 30(7), 2020.*

[76] Athanasios S. Polydoros and Lazaros Nalpantidis. Survey of Model-Based Reinforcement Learning: Applications on Robotics. Journal of Intelligent and Robotic Systems: Theory and Applications*, 86(2), 2017.*

[77] Carl Edward Rasmussen. Gaussian Processes in machine learning. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3176, 2004.*

[78] Kasra Rezaee, Baher Abdulhai, and Hossam Abdelgawad. Application of reinforcement learning with continuous state space to ramp metering in real-world conditions. In IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2012.*

[79] Sebastian Ruder. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747*, 2016.*

[80] F. Ruelens, B. J. Claessens, S. Quaiyum, B. De Schutter, R. Babuška, and R. Belmans. Reinforcement Learning Applied to an Electric Water Heater: From Theory to Practice. IEEE Transactions on Smart Grid*, 9(4), 2018.*

[81] Warren S Sarle. Neural Networks and Statistical Models. SAS USers Group International Conference*, 1994.*

[82] Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In Lecture Notes in Computer Science (including

subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3607 LNAI, 2005.*

[83] Andrew Silva, Taylor Killian, Ivan Rodriguez Jimenez, Sung Hyun Son, and Matthew Gombolay. Optimization methods for interpretable differentiable decision trees in reinforcement learning, 2019.

[84] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In Advances in Neural Information Processing Systems*, volume 4, 2012.*

[85] Alberto Suárez and James F Lutsko. Globally optimal fuzzy decision trees for classification and regression. IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, 1999.*

[86] Gail M. Sullivan and Richard Feinn. Using Effect Size—or Why the P Value Is Not Enough . Journal of Graduate Medical Education*, 4(3), 2012.*

[87] Haochen Sun, Zhumu Fu, Fazhan Tao, Longlong Zhu, and Pengju Si. Data-driven reinforcement-learning-based hierarchical energy management strategy for fuel cell/battery/ultracapacitor hybrid electric vehicles. Journal of Power Sources*, 455, 2020.*

[88] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. *MIT press, 2018.*

[89] Richard S. Sutton, Andrew G. Barto, and Ronald J. Williams. Reinforcement Learning is Direct Adaptive Optimal Control. IEEE Control Systems*, 12(2), 1992.*

[90] Samuele Tosatto, Matteo Pirotta, Carlo D'Eramo, and Marcello Restelli. Boosted fitted q-iteration. In 34th International Conference on Machine Learning, ICML 2017*, volume 7, 2017.*

[91] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. IEEE Transactions on Automatic Control*, 42(5), 1997.*

[92] William T.B. Uther and Manuela M. Veloso. Tree based discretization for continuous state space reinforcement learning. In Proceedings of the National Conference on Artificial Intelligence*, 1998.*

[93] Anneleen Van Assche and Hendrik Blocked. Seeing the forest through the trees: Learning a comprehensible model from an ensemble. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4701 LNAI, 2007.*

[94] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-Learning. In 30th AAAI Conference on Artificial Intelligence, AAAI 2016*, 2016.*

[95] Christopher J C H Watkins and Peter Dayan. Q-learning. Machine learning*, 8(3-4):279–292, 1992.*

[96] Xin Xu, Lei Zuo, and Zhenhua Huang. Reinforcement learning algorithms with function approximation: Recent advances and applications. Information Sciences*, 261:1–31, 2014.*

[97] Junyu Xuan, Jie Lu, Zheng Yan, and Guangquan Zhang. Bayesian deep reinforcement learning via deep kernel learning. International Journal of Computational Intelligence Systems*, 12(1), 2018.*

Appendix A

Sparse Online Gaussian Processes

The key equations of the Sparse Online Gaussian Process algorithm are provided here, whilst the reader is referred to Csato and Opper [20] for further reading. The algorithm maintains a dictionary of basis points, Z, which adequately represent the state-action space. Given a new example, x_{t+1}, its mean and variance are calculated as follows:

$$m(x_{t+1}) = \alpha_t^\top \, \mathbf{k}(Z, x_{t+1}) \tag{A.1}$$

$$\Sigma(x_{t+1}) = k(x_{t+1}, x_{t+1}) + \mathbf{k}(Z, x_{t+1})^\top \, C_t \, \mathbf{k}(Z, x_{t+1}) \tag{A.2}$$

$k(\cdot,\cdot)$ represents the kernel function and $\mathbf{k}(Z, x_{t+1})$ a vector containing the covariances between the new example and all examples in the dictionary. The vector α and matrix C are updated sequentially as follows:

$$\alpha_{t+1} = T_{t+1}(\alpha_t) + q^{(t+1)} \, \mathbf{s}_{t+1} \tag{A.3}$$

$$C_{t+1} = U_{t+1}(C_t) + r^{(t+1)} \, \mathbf{s}_{t+1} \, \mathbf{s}_{t+1}^\top \tag{A.4}$$

$$\mathbf{s}_{t+1} = T_{t+1}(C_t \, \mathbf{k}(Z, x_{t+1})) + \mathbf{e}_{t+1} \tag{A.5}$$

$\mathbf{e}_{t+1}$ denotes the (t+1)-th unit vector, T_{t+1} an operation which extends a t-dimensional vector to a $(t+1)$-dimensional one by appending zero at the end, and U_{t+1} an operation which extends a $(t \times t)$ matrix to a $(t+1) \times (t+1)$ one by appending a row and a column of zeros at the end. The following scalar quantities are used:

$$q^{(t+1)} = \frac{y_t - \alpha_t^\top \, \mathbf{k}(Z, x_t)}{\sigma_0^2 + \Sigma(x_{t+1})} \tag{A.6}$$

$$r^{(t+1)} = -\frac{1}{\sigma_0^2 + \Sigma(x_{t+1})} \tag{A.7}$$

Here, σ_0^2 is a parameter of the kernel function defined a priori and y can be defined as the Q-learning target in the RL framework: $y_t = r_t + \gamma \, \max_{a'} \hat{Q}_t(s', a')$. A new example, x_{t+1}, enters the dictionary if $\beta_{t+1} > \beta_{tol}$, where:

$$\beta_{t+1} = k(x_{t+1}, x_{t+1}) - \mathbf{k}(Z, x_{t+1})^{\top} \; K(Z,Z)^{-1} \; \mathbf{k}(Z, x_{t+1}) \tag{A.8}$$

Finally, the inverse of the matrix $K(Z,Z)$, denoted P, that is required for the calculation of β_{t+1} can be updated online through:

$$\hat{e}_{t+1} = P_t \; \mathbf{k}(Z, x_{t+1})^{\top} \tag{A.9}$$

$$P_{t+1} = U_{t+1}(P_t) + \beta_{t+1} \; (T_{t+1}(\hat{e}_{t+1}) - e_{t+1}) \; (T_{t+1}(\hat{e}_{t+1}) - e_{t+1})^{\top} \tag{A.10}$$

Csato and Opper [20] also describe how the dictionary can be maintained of a fixed size through the deletion of basis points. However, it was observed that, for the purpose of this project, this practice increased the computational cost of the algorithm with no improvements in terms of performance. Hence, the dictionary's size was not fixed and no deletion of basis points was implemented.

Appendix B

Final Parameter Values

The final parameter settings that were used for the results of the models presented in this project are summarised in the Appendix.

B.1 Linear Model

Environment Parameter	Simple Gridworld	Windy Gridworld	Cart Pole	Lunar Lander
gamma	0.99	0.99	0.99	0.99
batch_size	32	32	32	64
learning_rate	0.02	0.02	0.02	0.02
target_update_freq	20	50	50	50
eps_max_deduct	0.97	0.97	0.97	0.95
eps_decay	0.5	0.5	0.5	0.1
lr_reduct_freq	250	1000	1000	1000
lr_reduct	0.95	0.99	0.99	0.99
poly_degree	1	1	2	4

Table B.1: Parameter values for the Linear model on each environment.

B.2 Neural Network

Environment	Simple	Windy	Cart	Lunar
Parameter	**Gridworld**	**Gridworld**	**Pole**	**Lander**
gamma	0.99	0.99	0.99	0.99
batch_size	32	32	32	64
learning_rate	0.00075	0.0007	0.00075	0.0015
target_update_freq	50	200	200	100
eps_max_deduct	0.97	0.97	0.97	0.95
eps_decay	0.25	0.3	0.25	0.1
lr_reduct_freq	250	1000	1000	1000
lr_reduct	0.95	0.95	0.95	0.99
hidden_size	(32,32)	(64,64)	(32,32)	(256,128)

Table B.2: Parameter values for the Neural Network model on each environment.

B.3 Decision Tree (FQI)

Environment	Simple	Windy	Cart	Lunar
Parameter	**Gridworld**	**Gridworld**	**Pole**	**Lander**
replay_threshold	-1	0	-1	0.1
gamma	0.99	0.99	0.99	0.99
batch_size	512	max	512	max
model_save_freq	250	n/a	1000	n/a
model_save_capacity	20	n/a	20	n/a
max_depth	15	100	10	20
min_samples_split	20	2	20	20
min_samples_leaf	5	1	5	5
eps_max_deduct	0.95	0.9	0.95	0.9
eps_decay	0.4	0.4	0.4	0.4

Table B.3: Parameter values for the Decision Tree (FQI) model on each environment.

B.4 Random Forest (FQI)

Environment Parameter	Simple Gridworld	Windy Gridworld	Cart Pole	Lunar Lander
replay_threshold	-1	0	-1	0.1
gamma	0.99	0.99	0.99	0.99
batch_size	512	max	512	max
model_save_freq	250	n/a	1000	n/a
model_save_capacity	20	n/a	20	n/a
n_estimators	10	5	10	10
max_depth	15	100	10	20
min_samples_split	20	2	20	20
min_samples_leaf	5	1	5	5
eps_max_deduct	0.95	0.9	0.95	0.9
eps_decay	0.4	0.4	0.4	0.4

Table B.4: Parameter values for the Random Forest (FQI) model on each environment.

B.5 Support Vector Regression (FQI)

Environment Parameter	Simple Gridworld	Windy Gridworld	Cart Pole	Lunar Lander
replay_threshold	-1	0	-1	0.1
gamma	0.99	0.99	0.99	0.99
batch_size	256	max	512	max
model_save_freq	250	n/a	1000	n/a
model_save_capacity	20	n/a	20	n/a
C	2.4	1	2	1.2
eps_max_deduct	0.95	0.95	0.95	0.95
eps_decay	0.3	0.4	0.3	0.5

Table B.5: Parameter values for the Support Vector Regression (FQI) model on each environment.

B.6 k-Nearest Neighbours (FQI)

Environment Parameter	Simple Gridworld	Windy Gridworld	Cart Pole	Lunar Lander
replay_threshold	-1	0	-1	0.1
gamma	0.99	0.99	0.99	0.99
batch_size	256	max	256	max
model_save_freq	250	n/a	1000	n/a
model_save_capacity	20	n/a	20	n/a
n_neighbours	3	5	7	10
eps_max_deduct	0.95	0.9	0.95	0.93
eps_decay	0.3	0.4	0.3	0.4
weights	”distance”	”distance”	”distance”	”distance”

Table B.6: Parameter values for the k-Nearest Neighbours (FQI) model on each environment.

B.7 Gaussian Process (FQI)

Environment Parameter	Simple Gridworld	Windy Gridworld	Cart Pole	Lunar Lander
replay_threshold	-1	0	-1	0.1
gamma	0.99	0.99	0.99	0.99
batch_size	256	max	512	max
model_save_freq	250	n/a	1000	n/a
model_save_capacity	20	n/a	20	n/a
length_scale	0.5	0.5	0.08	0.3
eps_max_deduct	0.95	0.95	0.95	0.9
eps_decay	0.3	0.3	0.3	0.4

Table B.7: Parameter values for the Gaussian Process (FQI) model on each environment.

B.8 Gaussian Process (Online)

Environment **Parameter**	**Simple Gridworld**	**Windy Gridworld**	**Cart Pole**	**Lunar Lander**
gamma	0.99	0.99	0.99	0.99
epsilon_tol	0.085	0.045	0.05	0.075
length_scale	0.5	0.5	0.5	0.3
eps_max_deduct	0.95	0.9	0.95	0.93
eps_decay	0.3	0.4	0.3	0.4
init	-10	-100	0	0

Table B.8: Parameter values for the Online Gaussian Process model on each environment.