Master's Thesis

Nonparametric Off-Policy Policy Gradient

João André Correia Carvalho

Examiners:	Prof. Dr. Joschka Boedecker
	Prof. Dr. Frank Hutter
Advisers:	MSc. Samuele Tosatto
	Prof. Dr. Jan Peters

Albert-Ludwigs-Universität Freiburg
Faculty of Engineering
Department of Computer Science
August 31th, 2019

Writing period
15. 04. 2019 – 31. 08. 2019

Examiners
Prof. Dr. Joschka Boedecker, Prof. Dr. Frank Hutter (Albert-Ludwigs-Universität Freiburg)

Advisers
MSc. Samuele Tosatto, Prof. Dr. Jan Peters (Technische Universität Darmstadt)

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

In the context of Reinforcement Learning, the Policy Gradient Theorem provides a principled way to estimate the gradient of an objective function with respect to the parameters of a differentiable policy. Computing this gradient includes an expectation over the state distribution induced by the current policy, which is hard to obtain because it is a function of the generally unknown environment's dynamics. Therefore, one way to estimate the policy gradient is by direct interaction with the environment. The need for constant interactions is one of the reasons for the high sample-complexity of policy gradient algorithms, and why it hinders their direct application in robotics. Off-policy Reinforcement Learning offers the promise to solve this problem by providing better exploration, higher sample efficiency, and the ability to learn with demonstrations from other agents or humans. However, current state-of-the-art approaches cannot cope with truly off-policy trajectories.
This work proposes a different path to improve the sample efficiency of off-policy algorithms by providing a full off-policy gradient estimate. For that we construct a Nonparametric Bellman Equation with explicit dependence on the policy parameters using kernel density estimation and regression to model the transition dynamics and the reward function, respectively. From this equation we are able to extract a value function and a gradient estimate computed in closed-form, leading to the Nonparametric Off-Policy Policy Gradient (NOPG) algorithm. We provide empirical results to show that NOPG achieves better sample-complexity than state-of-the-art techniques.

Zusammenfassung

Das *Policy Gradient Theorem* bietet eine Grundlage zur Lösung der Aufgabe des *Reinforcement Learning* durch die gradientenbasierte Optimierung einer parametrisierten und differenzierbaren *Policy* im Bezug auf eine Zielfunktion. Die Schätzung dieses Gradienten beinhaltet eine Erwartung über die durch die aktuelle *Policy* induzierte Zustandsverteilung, was ein anspruchsvolles Problem darstellt, da es eine Funktion der Dynamik der Umgebung ist. Im Allgemeinen ist diese nicht bekannt, weshalb eine Möglichkeit darin besteht, den *Policy Gradient* durch direkte Interaktion mit der Umwelt zu schätzen. Die Notwendigkeit ständiger Interaktionen ist einer der Gründe für die hohe Sample-Komplexität von *Policy Gradient*-Algorhythmen und warum sie ihre direkte Anwendung in der Robotik behindert. *Off-Policy Reinforcement Learning* adressiert dieses Problem, indem es bessere Exploration, höhere Dateneffizienz und die Fähigkeit, von Demonstrationen von anderen Agenten oder Menschen zu lernen. Diese Arbeit schlägt einen anderen Weg vor, um die Dateneffizienz von *Off-Policy*-Algorithmen zu verbessern, indem sie eine vollständige Gradientenschätzung liefert. Dazu konstruieren wir eine nichtparametrische Bellman Gleichung mit expliziter Abhängigkeit von den *Policy* Parametern mittels *Kernel Density Estimation* und *Regression*, um die Systemdynamik und die Belohnungsfunktion zu modellieren. Das Ergebnis ist ein *Nonparametric Off-Policy Policy Gradient* (NOPG) Algorithmus, mit dem die *Value Function* und *Policy Gradient* in geschlossener Form berechnet werden können. Wir liefern empirische Ergebnisse, die zeigen, dass NOPG eine bessere Dateneffizienz erreicht als der aktuelle Stand der Technik.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

1.1. Motivation

Designing controllers for robotic systems is a challenging task. Typically, one starts by describing with detail the physical laws of the systems' dynamics in the form of equations of motion, and after devising a controller to solve a particular task with the system. This approach poses some fundamental problems, such as not being able to encode a priori all the possible forces that will take place in real world environments, and the inability of controllers to adapt to different but somewhat similar tasks. For instance, in a grasping task it would be infeasible to model all the friction forces a robot encounters when interacting with different surfaces. An undesired solution would be to consider only a subset of objects that can be present in a scene and program the robot to grasp each distinct object. Ideally, robotic autonomous agents would have the ability to adjust to different scenarios and autonomously learn about their own dynamics and the surrounding environment with minimal human intervention and supervision.

The field of Reinforcement Learning (RL) is a subclass of Machine Learning (ML) that deals with learning controllers through direct experience collected by an agent when traversing an environment, and thus suited to solve the described robot learning problem. RL algorithms for discrete and small state and action spaces based on tabular representations are well studied and have guaranteed solutions [1]. In more complex environments however, the spaces may be countable but too large, or even continuous. A telling example is the game of chess where the number of possible states is in the order of 10^{120} [2]. Building a tabular representation of such an environment is just not possible. Therefore, methods that perform approximations are crucially sought after. However, contrary to tabular solutions, the convergence guarantees of approximate algorithms are only devised for some special cases, such as when a loss function is squared and the value function is a linear combination of features [3]. With the resurgence of Neural Networks (NNs), in particular of Deep Learning (DL), the field of RL has received in the recent years a great interest from the research community and the general public. Today, RL algorithms are capable of astonishing results by solving difficult tasks such as playing table games like Go [4] and Chess [5], or arcade and strategy computer video games like Atari [6] and Dota 2 [7]. Not less important and of extreme interest are the results obtained in robotic applications, some of which do not use recent DL techniques, including learning motor skills to play table tennis [8] or balancing an unicycle [9], and also the ones leveraging the NNs revival, such as learning locomotion polices to walk, run and jump from scratch in simulation [10], learning dexterity from simulation and transferring to a real platform [11], enabling a quadrupedal robot to learn to walk in the real word within hours [12], or even learning control policies from mapping images directly to actions using Convolution Neural Networks (CNNs) as feature extractors [13].

Although the success of recent methods using Deep Neural Networks (DNNs) are undeniable, these models may have millions (or even billions) of parameters and thus require massive amounts of data to be trained effectively. Simulated environments, such as games, are a good setup to test and benchmark new algorithms because data can be infinitely generated, but they are usually noise free and mostly with discrete action spaces. In contrast, human-like robots need to cope with measurement errors from noisy sensor readings and actions are typically continuous, for example the voltage applied to motors to control joints. Collecting experience in simulated environments can be admissible, but in real-world robotic applications there is the need for methods suited to work in low data regimes and with explainable models, especially to ensure safe behaviours when robots are deployed to collaborate with humans.

The major problem limiting the application of RL algorithms based on DL models, coined as Deep

Reinforcement Learning (DRL), in real robotic systems is their high sample complexity [14]. The source of this sample inefficiency problem comes from many algorithms using one of the foundational theorems in RL, the Policy Gradient Theorem. This theorem allows a RL agent to instead of finding a controller using value-based methods, optimizing the parameters of a parameterized differentiable policy via a gradient ascent technique, while computing the gradient estimate by directly sampling from the environment. To understand how to obtain better sample efficiency we need to focus on understanding the true source of the gradient estimation problem. The biggest issue comes from the Policy Gradient Theorem dependence on the state distribution induced by the current policy. Such a construct implies that each policy update changes the state visitation frequency, meaning that new rollouts are needed after every iteration. Instead of using on-policy methods, off-policy algorithms promise to be more sample efficient because they can reuse transitions collected from multiple policies, for instance stored in a memory replay buffer [6].
Computing off-policy gradients is typically done with Importance Sampling (IS) methods, which are unbiased but show large variance, or using the Off-Policy Policy Gradient Theorem introduced by Degris et. al [15]. Further well known algorithms such as the Deterministic Policy Gradient (DPG) [16] and the Deep Deterministic Policy Gradient (DDPG) are based on that work. These methods however are better regarded as semi-gradient methods, as they do not follow a true gradient but a biased approximation, and thus fail in trully off-policy settings. For instance, this inability is clear in the work of Fujimoto et. al [17], where the authors give as an example where DDPG cannot learn a policy if it only uses samples from a previously populated replay buffer.
We believe that given the advantages of off-policy methods are well fitted for robotics, it is of utmost importance to keep exploring this subfield of RL. Therefore, in this thesis we emphasize the need for off-policy algorithms and propose a novel model-based gradient estimate for deterministic and stochastic policies that addresses some of the issues found in current approaches. Our work was inspired and builds up on *A Non-Parametric Approach to Dynamic Programming* from Kroemer et. al [18]. Likewise, we propose a policy gradient using a model-based approach with transition dynamics modelled with kernel density estimation and reward function with kernel regression, which leads to a Nonparametric Bellman Equation that we can solve in closed-form with explicit dependence on a parameterized policy. This dependence allows the computation of a full gradient in closed-form, partially because the state distribution can be computed without the need of sampling, thus improving on the biased semi-gradient methods, and without resorting to importance sampling approaches. The advantages in comparison to other methods are its higher sample-efficiency, which is essential in robotic applications, and the capability of estimating the state distribution, which is of great interest in robotics, since it can predict beforehand if the system will move to dangerous areas of the state space.
To attest our approach we provide empirical qualitative results on the policy gradient direction in a simple 2-dimensional problem with linear dynamics and quadratic costs, and quantitative results in OpenAI Gym and Quanser environments [19, 20], showing that it can work with offline and off-policy datasets using carefully chosen or random behavioural policies.

1.2. Structure of the Thesis

In Chapter 2 we give an introduction on how to model Reinforcement Learning problems with Markov Decision Processes and the methods to solve them. We give particular emphasis to policy search methods, especially to the Policy Gradient Theorem. Additionally we give a brief explanation of kernel density estimation and regression. Then, we present in Chapter 3 a classification and overview of the current techniques to perform policy search using off-policy trajectories. In Chapter 4 we introduce the bulk of this thesis, a policy gradient estimator for offline and off-policy data. We empirically test our algorithm in basic control tasks and present the results obtained in Chapter 5. Finally, in Chapter 6 we summarize the main aspects found and conclude with possible future research paths.

2. Background

2.1. Reinforcement Learning

Reinforcement Learning (RL) is a field in Machine Learning (ML) that deals with solving decision making problems in an optimal sense [21, 1]. Typically an agent (e.g., a robot, a player in a game, ...) encounters itself in a state, from which it interacts with a (stochastic) environment by performing an action, collecting an immediate reward and transitioning to a next state. Through multiple interactions, RL algorithms aim to find a policy, a map from states to actions, whose goal is to maximize the sum of the future (discounted) rewards collected by the agent. In a certain way RL is very similar to Optimal Control, where often one wants to find a controller (a policy) that minimizes a given cost function (a return). The main difference lies in the fact that often in Optimal Control one is presented with a model of the environment [22], for instance the physics laws that govern a rigid body motion dynamics, with which one can compute an optimal controller. In particular, if the system is modelled in the Linear Quadratic Regulator (LQR) framework, the solution can even be found in closed-form [23]. On the contrary, in RL problems the environment laws are assumed to be unknown (expect in environments with human specified rules, such as games like Chess or Go) and finding a controller cannot be performed in closed-form. While having a systems' description can seem to be advantageous, modelling its intricacies is also very tedious and not straightforward. For example, it does not seem plausible to be able to model all kinds of friction forces a system is expected to be exposed to. In RL the agent can learn about the dynamics' details while it interacts with the environment, hence possibly capturing a better description of the true dynamics.

2.2. Markov Decision Processes

Decision making problems such as the ones faced in RL are commonly modelled as Markov Decision Processes (MDPs). Let a MDP [24] be defined for the general case as a tuple $(\mathcal{S}, \mathcal{A}, R, P, \mu_0, \gamma)$, where $\mathcal{S}$ is a state space $\boldsymbol{s} \in \mathcal{S}$, $\mathcal{A}$ is an action space $\boldsymbol{a} \in \mathcal{A}$, R is a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, P governs transition probabilities $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$, μ_0 is the initial state distribution $\mu_0 : \mathcal{S} \to [0, \infty)$ and $\gamma \in [0, 1)$ is a discount factor. In the particular case of discrete state and action spaces, the distributions change from probability densities to probability mass functions, and so the transition probabilities and initial state distribution are bounded to 1.

MDPs share the Markov property, i.e., the current state $\boldsymbol{s}_t$ and action $\boldsymbol{a}_t$ contain all the information necessary to determine the probability of transitioning to the next state $\boldsymbol{s}_{t+1}$, $P(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t) = P(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \ldots, \boldsymbol{s}_0, \boldsymbol{a}_0)$. The reward function R defines the problem at hand, since it guides the agent towards a desired behaviour [25]. For instance, if an agent's task is to reach a final position $\boldsymbol{s}_f$, a reasonable reward function could measure the euclidean distance from the current state, e.g., $R(\boldsymbol{s}, \boldsymbol{a}) = \|\boldsymbol{s} - \boldsymbol{s}_f\|_2$. An agent navigates in a MDP governed by a policy $\pi : \mathcal{S} \to \mathcal{A}$ that determines which action to take in the current state. The mapping from states to actions can be of two types: deterministic, in which case the action is picked deterministically, $\boldsymbol{a} = \pi(\boldsymbol{s})$; or stochastic, where the action is chosen from a distribution conditioned on the current state $\boldsymbol{a} \sim \pi(\cdot \mid \boldsymbol{s})$, such that $\int_{\mathcal{A}} \pi(\boldsymbol{a} \mid \boldsymbol{s}) \, \mathrm{d}\boldsymbol{a} = 1$. For large state spaces it is common to use a policy parameterized by a set of parameters $\boldsymbol{\theta}$.

By starting in an initial state $\boldsymbol{s}_0$ sampled from the initial state distribution and traversing a MDP with

a policy π, the agent computes a return as the expected sum of discounted rewards

$$J_\pi = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(\boldsymbol{s}_t, \boldsymbol{a}_t) \,\middle|\, \boldsymbol{s}_0 \sim \mu_0(\boldsymbol{s}), \boldsymbol{a}_t \sim \pi(\boldsymbol{a} \mid \boldsymbol{s}_t), \boldsymbol{s}_{t+1} \sim P(\boldsymbol{s}'|\boldsymbol{s}_t, \boldsymbol{a}_t)\right]. \tag{1}$$

The goal of a RL algorithm is to compute an optimal policy, i.e., a policy that maximizes the return. In Equation (1), a finite T encodes episodic problems, while $T = \infty$ represents infinite horizon problems. The discount factor γ controls the view of the agent towards the rewards. With $\gamma \to 1$ the agent regards immediate and late rewards as equally good. Whereas $\gamma \to 0$ provides a more myopic view, since the agent values more immediate rewards. For the infinite horizon setting we require $\gamma < 1$ to prevent the return from growing to infinity.

2.3. Value Function and State Distribution

There are two interesting quantities one can extract from a MDP, the value function and the state distribution. The state value function (or just value function) $V_\pi(\boldsymbol{s})$ encodes the expected return from a state $\boldsymbol{s}$ when following the policy π, and can be expressed in a recursive fashion known as the Bellman equation

$$V_\pi(\boldsymbol{s}) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(\boldsymbol{s}_i, \boldsymbol{a}_i) \,\middle|\, \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_t \sim \pi(\boldsymbol{a} \mid \boldsymbol{s}_t), \boldsymbol{s}_{t+1} \sim P(\boldsymbol{s}'|\boldsymbol{s}_t, \boldsymbol{a}_t)\right] \tag{2}$$

$$= \int_{\mathcal{A}} \pi(\boldsymbol{a} \mid \boldsymbol{s}) R(\boldsymbol{s}, \boldsymbol{a}) \,\mathrm{d}\boldsymbol{a} + \gamma \int_{\mathcal{A}} \int_{\mathcal{S}} \pi(\boldsymbol{a} \mid \boldsymbol{s}) P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) V_\pi(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \,\mathrm{d}\boldsymbol{a} \quad \forall \boldsymbol{s} \in \mathcal{S}, \tag{3}$$

which admits a unique fixed-point solution [1]. Notice that a value function is always defined assuming an underlying policy.

A state-action value function (or just Q-function) $Q_\pi(\boldsymbol{s}, \boldsymbol{a})$ represents the expected return of being in state $\boldsymbol{s}$, taking action $\boldsymbol{a}$ and following the policy thereafter. V_π and Q_π are related by $V_\pi(\boldsymbol{s}) = \mathbb{E}_{a \sim \pi(\cdot|\boldsymbol{s})}\left[Q_\pi(\boldsymbol{s}, \boldsymbol{a})\right]$. A nice property of the Q-function is that an optimal policy can be extracted by acting greedily in every state, i.e., by choosing the action as $\arg\max_{\boldsymbol{a}} Q_\pi(\boldsymbol{s}, \boldsymbol{a}) \;\; \forall \boldsymbol{s} \in \mathcal{S}$. The value function for an optimal policy can be written as $V_\pi(\boldsymbol{s}) = \max_{\boldsymbol{a}} Q_\pi(\boldsymbol{s}, \boldsymbol{a}) \;\; \forall \boldsymbol{s} \in \mathcal{S}$.

The discounted expected state visitation is also dependent on the policy and can be seen as the expected number of times a state is visited after infinitely transitioning the MDP following the policy π

$$\mu_\pi(\boldsymbol{s}) = \mu_0(\boldsymbol{s}) + \gamma \int_{\mathcal{S}} \int_{\mathcal{A}} \mu_\pi(\boldsymbol{s}') \pi(\boldsymbol{a} \mid \boldsymbol{s}') P(\boldsymbol{s} \mid \boldsymbol{s}', \boldsymbol{a}) \,\mathrm{d}\boldsymbol{a} \,\mathrm{d}\boldsymbol{s}' \quad \forall \boldsymbol{s} \in \mathcal{S}. \tag{4}$$

Notice that $\mu_\pi(\boldsymbol{s})$ is not a proper distribution because it is not normalized. Although, we refer to it as the discounted state distribution, because to get a valid distribution one can simply normalize it for every state with $\mu_\pi(\boldsymbol{s}) \leftarrow \mu_\pi(\boldsymbol{s}) / \int_{\mathcal{S}} \mu_\pi(\boldsymbol{z}) \,\mathrm{d}\boldsymbol{z}$, or more simply as $\mu_\pi(\boldsymbol{s}) \leftarrow (1-\gamma)\mu_\pi(\boldsymbol{s})$. A proof for the latter is given in Appendix A.

Comparing Equation (3) and Equation (4) one can see the similarities between the value function the and state distribution. In fact, when expressing the RL problem as a Linear Programming (LP) problem, the value function does not appear in the dual formulation, but instead the state distribution does. Therefore, the state distribution is often coined as the dual of the value function [26]. There is however a gigantic difference between the two. The second term of the sum on right hand side of Equation (3) is an expectation of the value function over the next state distribution. Hence, the value function can be computed by transitioning the MDP according to $\boldsymbol{s}' \sim P(\cdot \mid \boldsymbol{s}, \boldsymbol{a})$. On the contrary, computing the state distribution is not as simple. One cannot solve it just by sampling, because sampling $\boldsymbol{s}'$ from $P(\boldsymbol{s} \mid \boldsymbol{s}', \boldsymbol{a})$ is not possible, since it is not a valid probability distribution. For this reason computing the value function is often preferred to estimating the state distribution.

For MDPs with finite sets of state and action spaces, known transition dynamics, and for simplicity using a deterministic policy, the value function and state distribution can be seen as a finite set of a linear system of equations and thus computed in closed-form

$$\begin{aligned} & \boldsymbol{V}_\pi = \boldsymbol{R}_\pi + \gamma \mathbf{P}_\pi \boldsymbol{V}_\pi \\ \Leftrightarrow \; & \boldsymbol{V}_\pi = \left(\mathbf{I} - \gamma \mathbf{P}_\pi\right)^{-1} \boldsymbol{R}_\pi \end{aligned} \tag{5}$$

$$\begin{aligned} & \boldsymbol{\mu}_\pi = \boldsymbol{\mu}_0 + \gamma \mathbf{P}_\pi^\top \boldsymbol{\mu}_\pi \\ \Leftrightarrow \; & \boldsymbol{\mu}_\pi = \left(\mathbf{I} - \gamma \mathbf{P}_\pi^\top\right)^{-1} \boldsymbol{\mu}_0, \end{aligned} \tag{6}$$

where $\boldsymbol{V}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ is a vector with the value function of each state, $\boldsymbol{R}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ is the vector of reward per state, $\mathbf{P}_\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ is the transition probability matrix, $\boldsymbol{\mu}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ is a vector with the (unnormalized) state distribution and $\boldsymbol{\mu}_0 \in \mathbb{R}^{|\mathcal{S}|}$ the vector with the initial distribution per state. Computing $\boldsymbol{V}_\pi$ and $\boldsymbol{\mu}_\pi$ with Equation (5) and Equation (6) involves the inversion of a $|\mathcal{S}| \times |\mathcal{S}|$ matrix. Hence, environments with small state spaces are trivially solved, but for larger ones this method is definitely prohibitive. For instance, a lower bound for the game of Chess places the number of states around 10^{120} [2]. Even if we could solve such large problems through matrix inversion, most problems in the real world do not appear in discrete but rather in continuous domains. For instance, in robotics one encounters more often continuous state and action spaces, for which we can no longer solve using discrete methods, without performing some kind of discretization of the space.

2.4. Value Methods and Policy Search

To find optimal policies there are two main approaches: methods based on value functions; and methods based on policy search [27].

Value function methods rely on estimating the value function (or the state-action value function) for each state. As already mentioned in Section 2.3, if a Q-function is available, an optimal policy can be extracted at each state by greedily picking the action that maximizes Q. For large but discrete state spaces, if the transition probability matrix is known, a practical way to estimate V_π (or Q_π) is to recognize that the Bellman equation (Equation (3)) can be solved by Dynamic Programming (DP) [28], thus avoiding the matrix inversion as in Equation (5). When the transition dynamics are unknown there are two main methods to estimate the value function, namely Monte Carlo (MC) and Temporal Difference (TD) methods. MC methods rely on computing the return for each state by sampling and averaging over full trajectory runs on the environment. Albeit unbiased, they suffer from high variance estimates. TD learning methods use bootstrapping to learn the value function, i.e., instead of updating states' values only after the end of an episode, like in MC, states' values are continuously updated with the Bellman error provided by the current best estimates of all states. In comparison to MC, TD algorithms have less variance and seem to work better in practice, but their convergence to the true value function is only guaranteed in certain cases, such as the tabular setting or with linear function approximation [21].

RL methods that keep a value function entry for all states are coined as tabular methods. For large state and action spaces one needs to represent the value function using a function approximator, because recording all states' values is physically impossible. In the case of parametric methods, it is common to approximate the value function with a linear combination of features [21] or a Neural Network [6]. Typically, one formulates the problem as a supervised learning task and minimizes the mean squared error between the prediction and the true value, for instance using a gradient descent technique. Choosing which loss function to optimize is already a source of approximation error, which does not happen in DP methods. Besides parametric methods, value functions can also be approximated using nonparametric models such as kernel density estimation and regression [18] or Gaussian Processes [29]. The drawback of approximate methods in contrast to DP or MC methods is that they are only

sure to converge under certain conditions and for certain function approximators. For instance, TD learning with a linear function approximation has convergence guarantees [21], but not with nonlinear approximators [30]. However, even without theoretical guarantees, nonlinear function approximators have shown surprising positive empirical results. One example is the application of Deep Neural Networks to the traditional Q-learning algorithm to learn the Q-function [31], giving rise to the Deep Q-Network (DQN) [6].

Policy search (PS) methods extract a policy differently from value methods. They assume a policy is parameterized by a set of parameters $\boldsymbol{\theta} \in \Theta$ and search directly in the parameter space, thus not necessarily needing to keep an explicit value function. Parameterized policies allow to shrink the search space to a subset of possible configurations, which is of major importance for RL algorithms to deal with large action spaces [32].

PS methods start by defining an objective to guide the parameters search, for instance the average stationary reward

$$\begin{aligned} J_{\pi_\theta} &= \mathbb{E}_{\boldsymbol{s}\sim\mu_{\pi_\theta}(\cdot),\, \boldsymbol{a}\sim\pi_\theta(\cdot|\boldsymbol{s})}\left[R(\boldsymbol{s},\boldsymbol{a})\right] \\ &= \int_{\mathcal{S}}\int_{\mathcal{A}} \mu_{\pi_\theta}(\boldsymbol{s})\pi_\theta(\boldsymbol{a} \mid \boldsymbol{s})R(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a}\,\mathrm{d}\boldsymbol{s}, \end{aligned} \tag{7}$$

where μ_{π_θ} is the stationary state distribution induced by π_θ, or the starting state objective

$$J_{\pi_\theta} = \mathbb{E}_{\boldsymbol{s}\sim\mu_0(\cdot)}\left[V_{\pi_\theta}(\boldsymbol{s})\right] = \int_{\mathcal{S}} \mu_0(\boldsymbol{s})V_{\pi_\theta}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}, \tag{8}$$

where μ_0 is the initial state distribution. There are two big approaches to find the policy parameters that maximize J_{π_θ}: gradient-free and gradient-based methods. Gradient-free optimizers can deal with non differentiable policies and are typically simpler and have low computational complexity, but on the other hand can be very slow to achieve convergence. An example of such a methods evolutionary algorithms [33]. Gradient-based methods are more common, and named in the literature as Policy Gradient (PG) methods. In PG algorithms the policy needs to be differentiable. With an estimate for the gradient of the objective function the policy parameters can be updated with a gradient ascent technique

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\nabla_{\boldsymbol{\theta}} J_{\pi_\theta}, \tag{9}$$

where α is a learning rate, and $\nabla_{\boldsymbol{\theta}} J_{\pi_\theta}$ is commonly referred to as the policy gradient.

PG methods offer a couple of advantages in relation to value function approximation methods [21]: following the gradient moves the policy in the direction of a better objective; it works well in high-dimensional state-action spaces; the policy representation is often more compact than the value function representation; the ability to learn stochastic policies. There are of course some disadvantages such as slower convergence or that convergence to a local optimum is not always guaranteed.

A natural join of value function and policy search methods are Actor-Critic algorithms, which still search for the best policy parameterization but using learned value functions from full returns or TD errors, providing faster convergence with lower gradient variance estimates [34].

2.5. Model-based and Model-free Reinforcement Learning

In RL problems the transition probability density function is generally unknown. Even if the state and actions spaces were discrete and small, a closed-form solution for the value function as in Equation (5) could not be found, because $\mathbf{P}_\pi$ is unknown. However, the agent can still learn a value function from collecting experience from the environment in the form of state–action–reward–next-state tuples $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$. With the collected experience there are two major paths to learn a policy: with model-free RL methods; or with model-based RL methods.

In model-free RL there is no concept of a transition dynamics model. The agent uses the experience to directly learn the value function of a state or a policy. To compute the value function for a specific state, the RL agent could estimate the expected return by averaging the sum of discounted rewards of successive MC rollouts starting from that state. Alternatively it could estimate the Q-function also by MC rollouts as suggested by the Policy Gradient Theorem [35]. Sampling trajectories from simulators poses no problem apart from time. However, sampling from real systems is not always feasible as it is time consuming and can lead to material wear and tear. Examples of model-free algorithms are REINFORCE [36], Relative Entropy Policy Search (REPS) [37], Trust Region Policy Optimization (TRPO) [38] and Deep Deterministic Policy Gradient (DDPG) [39].

Model-based RL methods seek to improve the sample efficiency over model-free approaches by using the collected experience to learn a model of the transition dynamics. Agents can then sample from this model to generate trajectories to compute the value function or the policy gradient as if the samples came directly from the environment. Alternatively, the model might even allow a closed-form computation of the value function [18] or the policy gradient [9]. These methods have the advantage of requiring less interactions with the real system and therefore promise to be more sample efficient. However, when choosing a model class we are directly restricting the type of environments we can represent. The downside is that if the model poorly represents the real dynamics, the learning algorithm may explore the model errors to generate a control policy that may lead to a totally incorrect behaviour. It is not difficult to construct an example where model-based methods would fail. Consider for instance that in the real environment the agent can transition to two different states with almost equal probability. If $P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$ is modelled with a Gaussian distribution, which is very common due to its analytical properties [9, 29], we are directly making a mistake by assuming the transition to be unimodal. Examples of model-based algorithms are PILCO [9] and Guided Policy Search (GPS) [40].

Although model-free algorithms seem to be more sample inefficient, because they cannot generate synthetic samples, they offer advantages with respect to model-based ones. They do not suffer from model-bias, which is an issue of RL algorithms related to optimizing a policy based on samples from the model and not the true environment. Even though multiple interactions are required, learning a policy is often easier than learning precise dynamics models. Thus, model-free RL algorithms are more common in practice and recent successful algorithms are all model-free [37, 39, 38, 41]. Whether model-free works better than model-based is still an open research question. In model-free methods the model is somewhat encoded in the value function, while in model-based explicitly learning the model is difficult and one incurs in inductive-bias.

For the particular reason that in robotics applications it is extremely important to have algorithms with low sample complexity and safe exploration, in this thesis we will focus on using model-based algorithms.

2.6. On-Policy and Off-Policy Reinforcement Learning

A further class of RL algorithms regards the use of on- or off-policy data. It is common to refer to the policy being optimized as the *target* policy and the policy used to interact with the environment as the *behavioural* policy [21].

On-policy algorithms learn the target policy based on samples collected by interacting with the environment using the same target policy (i.e., the target and behavioural policies match). They are generally unbiased and require continuous interactions with the environment after every value or policy improvement, which constitute the main source of sample inefficiency of on-policy methods. Examples of algorithms belonging to this class include SARSA [42], REINFORCE [36] or TRPO [38].

Off-policy algorithms offer the promise of learning the target policy using samples from a different behavioural policy. In comparison to on-policy, they allow for: better exploration, since multiple behavioural policies can be used to provide data; better sample efficiency by reusing samples from other policies; learning from agents or human demonstrations; and safety, because in on-policy algorithms the initial target policy (used also as behavioural policy) can be arbitrarily bad. These points make

them more appealing than on-policy methods. However, estimating off-policy value functions or policy gradients is known for being a hard problem, because of the mismatch between the state-action distribution induced by the current target policy and the distribution of the off-policy data [17]. Examples of off-policy algorithms include Q-learning [31], REINFORCE with Importance Sampling [43], Off-Policy Actor-Critic (Off-PAC) [15] or DDPG [39]. In Chapter 3 we give a detailed overview of off-policy policy gradient methods.

2.7. Policy Gradient Theorem

As explained in Section 2.4 one way to compute the parameters of a parameterized policy is to guide the search with gradient ascent. For that we define an objective function, such as in Equation (7) or Equation (8), compute the gradient with respect to the parameters and update them using Equation (9). The main concern is on how to obtain a good estimator for $\nabla_\theta J_{\pi_\theta}$.

The Policy Gradient theorem introduced by Sutton et. al [35] provides a clear derivation on how to estimate the policy gradient from experience. Here we provided a simple and intuitive derivation of this theorem. We start by defining as objective the starting state formulation and the corresponding gradient

$$\begin{aligned} J_{\pi_\theta} &= \mathbb{E}_{\boldsymbol{s}\sim\mu_0(\cdot)}\left[V_{\pi_\theta}(\boldsymbol{s})\right] = \int_S \mu_0(\boldsymbol{s}) V_{\pi_\theta}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s} \\ \nabla_\theta J_{\pi_\theta} &= \nabla_\theta \mathbb{E}_{\boldsymbol{s}\sim\mu_0(\cdot)}\left[V_{\pi_\theta}(\boldsymbol{s})\right] = \int_S \mu_0(\boldsymbol{s}) \nabla_\theta V_{\pi_\theta}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}. \end{aligned} \tag{10}$$

Our goal is to find an expression for the gradient of the value function, such that we can estimate it by transitioning in the MDP. Using the relation between the value function and the Q-function we have

$$\begin{aligned} \nabla_\theta V_{\pi_\theta}(\boldsymbol{s}) &= \nabla_\theta \left(\int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a} \right) \\ &= \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{s})\right) Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a} + \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) \nabla_\theta Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a} \\ &= \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{s})\right) Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a} + \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) \nabla_\theta \left(R(\boldsymbol{s},\boldsymbol{a}) + \gamma \int_{\mathcal{S}} P\left(\boldsymbol{s}' \mid \boldsymbol{s},\boldsymbol{a}\right) V_{\pi_\theta}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' \right) \mathrm{d}\boldsymbol{a} \\ &= \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{s})\right) Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a} + \gamma \int_{\mathcal{A}} \int_{\mathcal{S}} \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) P\left(\boldsymbol{s}' \mid \boldsymbol{s},\boldsymbol{a}\right) \nabla_\theta V_{\pi_\theta}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}'\,\mathrm{d}\boldsymbol{a}. \end{aligned} \tag{11}$$

Before expanding the recursive term in Equation (11), let us introduce some useful quantities. We denote by $P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{x}, k)$ the probability of transitioning from state $\boldsymbol{s}$ to state $\boldsymbol{x}$ after k steps, while following the policy π_θ. The following identities are straightforward to obtain: $P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{s}, 0) = 1$ is the probability of staying in the same state without taking any step; $P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{s}', 1) = \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) P(\boldsymbol{s}' \mid \boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a}$ is the probability of moving to a next state in one step; and $P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{x}, k+1) = \int_{\mathcal{S}} P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{s}', k) P_{\pi_\theta}(\boldsymbol{s}' \to \boldsymbol{x}, 1)\,\mathrm{d}\boldsymbol{s}'$ is the recursive form of the probability of reaching $\boldsymbol{x}$ from $\boldsymbol{s}$ after $k+1$ by transitioning first to $\boldsymbol{s}'$ after k steps and finally from $\boldsymbol{s}'$ to $\boldsymbol{x}$ in one last step. We also introduce for notation purposes $f_{\pi_\theta}(s) = \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{s})\right) Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a}$. Expanding Equation (11) and introducing the defined quantities

we get

$$
\begin{aligned}
\nabla_\theta V_{\pi_\theta}(\boldsymbol{s}) &= f_{\pi_\theta}(\boldsymbol{s}) + \gamma \int_{\mathcal{A}} \int_{\mathcal{S}} \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) P\left(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}\right) \nabla_\theta V_{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' \, \mathrm{d}\boldsymbol{a} \\
&= f_{\pi_\theta}(\boldsymbol{s}) + \gamma \int_{\mathcal{S}} P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{s}', 1) \nabla_\theta V_{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' \\
&= f_{\pi_\theta}(\boldsymbol{s}) + \gamma \int_{\mathcal{S}} P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{s}', 1) \left(f_{\pi_\theta}(\boldsymbol{s}') + \gamma \int_{\mathcal{S}} P_{\pi_\theta}(\boldsymbol{s}' \to \boldsymbol{s}'', 1) \nabla_\theta V_{\pi_\theta}(\boldsymbol{s}'') \, \mathrm{d}\boldsymbol{s}'' \right) \mathrm{d}\boldsymbol{s}' \\
&= f_{\pi_\theta}(\boldsymbol{s}) + \gamma \int_{\mathcal{S}} P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{s}', 1) f_{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' + \gamma^2 \int_{\mathcal{S}} P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{s}'', 2) \nabla_\theta V_{\pi_\theta}(\boldsymbol{s}'') \, \mathrm{d}\boldsymbol{s}'' \\
&= \dots \\
&= \sum_{k=0}^{\infty} \gamma^k \int_{\mathcal{S}} P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{x}, k) f_{\pi_\theta}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \\
&= \int_{\mathcal{S}} \underbrace{\sum_{k=0}^{\infty} \gamma^k P_{\pi_\theta}(\boldsymbol{s} \to \boldsymbol{x}, k)}_{\eta_{\pi_\theta}(\boldsymbol{x})} f_{\pi_\theta}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \\
&= \int_{\mathcal{S}} \eta_{\pi_\theta}(\boldsymbol{x}) \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{x})\right) Q_{\pi_\theta}(\boldsymbol{x}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{x} \\
&= \int_{\mathcal{S}} \eta_{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' \int_{\mathcal{S}} \frac{\eta_{\pi_\theta}(\boldsymbol{x})}{\int_{\mathcal{S}} \eta_{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}'} \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{x})\right) Q_{\pi_\theta}(\boldsymbol{x}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{x} \\
&= \int_{\mathcal{S}} \eta_{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' \int_{\mathcal{S}} \mu_{\pi_\theta}(\boldsymbol{x}) \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{x})\right) Q_{\pi_\theta}(\boldsymbol{x}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{x} \\
&\propto \int_{\mathcal{S}} \mu_{\pi_\theta}(\boldsymbol{x}) \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{x})\right) Q_{\pi_\theta}(\boldsymbol{x}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{x}, \qquad (12)
\end{aligned}
$$

where η_{π_θ} is the discounted state visitation and $\mu_{\pi_\theta}(\boldsymbol{x})$ the discounted state distribution when starting at state $\boldsymbol{s}$ and following the policy π_θ.

By considering a single initial state, the policy gradient is the same as the value function gradient, and using the log-ratio trick in Equation (10), we get

$$
\begin{aligned}
\nabla_\theta J_{\pi_\theta} &\propto \int_{\mathcal{S}} \mu_{\pi_\theta}(\boldsymbol{s}) \int_{\mathcal{A}} \nabla_\theta \left(\pi_\theta(\boldsymbol{a}|\boldsymbol{s})\right) Q_{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{s} \\
&= \int_{\mathcal{S}} \mu_{\pi_\theta}(\boldsymbol{s}) \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) \nabla_\theta \left(\log \pi_\theta(\boldsymbol{a}|\boldsymbol{s})\right) Q_{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{s} \\
&= \mathbb{E}_{\boldsymbol{s} \sim \mu_{\pi_\theta}(\cdot), \boldsymbol{a} \sim \pi_\theta(\cdot|\boldsymbol{s})} \left[Q_{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a}) \nabla_\theta \log \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) \right]. \qquad (13)
\end{aligned}
$$

Equation (13) is referred to as the (on-policy) Policy Gradient Theorem [35]. There are a couple of important observations to make about this gradient estimator. The direct one is that an unbiased estimate of the gradient can be computed by sampling directly from the environment. Hence, although there is a dependence on the state distribution, its expression does not need to be known. Estimating by sampling is particular important in this case, because as stated by Sutton et. al [21]: "...the effect of the policy on the state distribution is a function of the environment and is typically unknown.". The other important observation is that there is no gradient over the state distribution or the Q-function. The main downside of the policy gradient theorem is that everytime the policy changes the state distribution changes accordingly, so after each policy update a new rollout to estimate the gradient is needed.

Although computing the policy gradient with online trajectories is easy, in many scenarios consecutive interactions with the environment are expensive and even not feasible. As detailed in Section 2.6, a better way to work in robotic applications is by using off-policy data. Therefore, a more desirable estimator is one that provides a gradient when presented with off-policy samples. Unfortunately there

is currently no unifying theorem for off-policy gradient estimation as there is for the on-policy case. In this thesis we will focus on off-policy gradient estimation and in Chapter 3 we will present the current different state-of-the-art approaches for its computation.

2.8. Policy Representations

When the state or action spaces are too large it is infeasible to store a tabular representation of the policy. Hence, they are better represented compactly using a set of parameters. Here we consider the class of differentiable policies that can be parameterized in different ways, with linear and nonlinear function approximators, such as linear weighted combinations of features or neural networks, respectively.
A linearly parameterized policy is of the form $\boldsymbol{\theta}^\top \phi(\boldsymbol{s}, \boldsymbol{a})$, where $\boldsymbol{\theta} \in \mathbb{R}^D$ are the parameters and $\phi(\boldsymbol{s}, \boldsymbol{a}) \in \mathbb{R}^D$ a state-action feature vector. In the case of a discrete set of actions, the softmax function can be used to build a normalized probability mass function

$$\pi_\theta(\boldsymbol{a}|\boldsymbol{s}) = \frac{\exp\left(\boldsymbol{\theta}^\top \phi(\boldsymbol{s}, \boldsymbol{a})\right)}{\sum_{\boldsymbol{z}} \exp\left(\boldsymbol{\theta}^\top \phi(\boldsymbol{s}, \boldsymbol{z})\right)}. \tag{14}$$

For continuous and stochastic action spaces the policy can be represented using a Gaussian distribution

$$\pi_\theta(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}\left(\boldsymbol{a} \,\middle|\, \boldsymbol{\mu} = f_\theta(\boldsymbol{s}), \boldsymbol{\Sigma} = g_\theta(\boldsymbol{s})\right),$$

with a mean parameterized by f_θ and covariance by g_θ. For complex representations, the linear policy can be replaced with other parameterizations. For instance, f_θ and g_θ can be the outputs of a multi-head neural network. The Gaussian is often a good choice because it easy to sample from and gradients can be backpropagated using the reparameterization trick [44].

2.9. Kernel Density Estimation

Density estimation is the problem of finding the underlying probability distribution of a random variable (or a set of random variables) given a dataset sampled from that distribution. It is of great importance in Machine Learning, since many algorithms assume the distributions are known, for instance in any Bayesian methods. There are generally two types of density models, parametric and nonparametric. As the name suggests, parametric models describe a probability distribution solely using a number of fixed parameters. For instance, the multivariate Gaussian distribution is fully described by its mean vector and covariance matrix. On the contrary, nonparametric methods do not have a fixed set of parameters, but rather describe the distribution using the set of data points. Naturally, parametric methods offer the advantage of having a low memory footprint and fast inference complexity, but unfortunately if the model is not complex enough we might not be able to capture all the details present in the data. A telling example is if the true data distribution is multimodal but we model the density with a single Gaussian, which is by definition unimodal. Nonparametric methods do not have these drawbacks, since they do not assume a fixed model for the distribution, and additionally under infinite sample assumptions they are able to fully express the underlying distribution [45]. As many nonparametric models, the disadvantages are the large memory footprint and slow inference. Naively, one needs to store the total number of samples and thus computing the density of a new data point is linear in the number of samples [46].
Assume we wish to find the probability density $p(\boldsymbol{x})$ of a random variable $\boldsymbol{x} \in \mathbb{R}^D$, where the D-dimensional space is Euclidean. The probability mass in a region R can be expressed as

$$P = \int_R p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}.$$

Using N samples from the true density, $\{\boldsymbol{x}_i\}_{i=1}^N$, $\boldsymbol{x}_i \sim p(\boldsymbol{x})$, and assuming the region R is sufficiently

small such that the density is constant, we can show that $p(\boldsymbol{x})$ is approximately [46]

$$p(\boldsymbol{x}) \approx \frac{K}{NV}, \tag{15}$$

where K is the number of points falling in region R, V is the volume of region R and N the number of datapoints.

The two free parameters in Equation (15) are K and V, and their choice leads to two different approaches of nonparametric estimation. For the case when K is fixed and V determined from the data we get a k-nearest neighbours estimate. If V is fixed and K determined from the data we have a kernel density estimation model. The approximation in Equation (15) can be shown to converge to the true density for growing N, decreasing V and growing K [46].

In kernel density estimation the simplest way to determine K is to consider that R is a hypercube in $\mathbb{R}^D$ of side h centered in $\boldsymbol{x}$. We define a kernel function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ nonnegative and normalized, i.e., $k(\boldsymbol{z}) \geq 0$ and $\int k(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z} = 1 \ \forall \boldsymbol{z} \in \mathbb{R}^D$, as

$$k\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right) = \begin{cases} 1, & \text{if } \boldsymbol{x}_i \text{ is inside the hypercube centered at } \boldsymbol{x} \\ 0, & \text{otherwise} \end{cases},$$

from which it directly follows that the total number of points laying inside the hypercube is

$$K = \sum_{i=1}^{N} k\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right). \tag{16}$$

Substituting Equation (16) back into Equation (15) we get the simplest kernel density estimator based on an hypercube in $\mathbb{R}^D$ of side-length h

$$p(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h^D} k\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right), \tag{17}$$

where we used the fact that $V = h^D$.

Using a hypercube as a kernel function can lead to discontinuous density estimations, which can be problematic not only because the density is not smooth but also because if we need to compute gradients of the kernel function we might not be able to. Therefore, an alternative to the hypercube is to use a kernel that produces continuous and differentiable density functions, such as the multivariate Gaussian. The estimation for the density at each point turns out

$$p(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}_i)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{x}_i)\right),$$

which can be seen as a sum of N Gaussians (normalized by $1/N$), each centered in $\boldsymbol{x}_i$, where the covariance matrix $\boldsymbol{\Sigma}$ can be interpreted as the hyperparameter h.

In Equation (17) we derived the density estimate only for a single variable. Very often we are interested in modelling joint distributions. It is straightforward to extend Equation (17) by noting that by definition a product of kernels is also a valid kernel [46]. Therefore, the joint distribution of a set of M variables $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^M)$ is

$$p\left(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^M\right) = \frac{1}{N} \sum_{i=1}^{N} \prod_{l=1}^{M} \frac{1}{h_l^{D_l}} k_l\left(\frac{\boldsymbol{x}^l - \boldsymbol{x}_i^l}{h_l}\right), \tag{18}$$

where each random variable $\boldsymbol{x}^l \in \mathbb{R}^{D_l}$, and k_l is the kernel corresponding to the joint variable l, with corresponding bandwidth h_l.

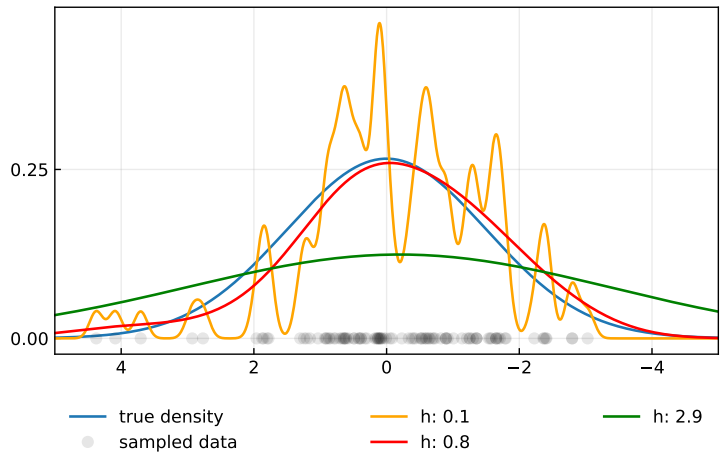


Figure 1.: Kernel density estimation with different bandwidths This figure illustrates the role of the bandwidth in kernel density estimation. Here we use Gaussian kernels to approximate the true density (in blue), which is a zero-mean and 1.5 standard deviation Gaussian, using 100 samples (in grey). In this setting, the bandwidth can be seen as the standard deviation of the Gaussian. With small bandwidths (orange) the estimate tends to overfit the data, while with large bandwidths (green) the model oversmooths the distribution. In red we show a bandwidth that brings us closer to the true density.

A crucial part of kernel density estimation relies on choosing a *good* value for the hyperparameter bandwidth h. As illustrated in Figure 1, a small value for h means that the estimation of $p(\boldsymbol{x})$ takes into account a very small amount of points around $\boldsymbol{x}$, which tends to overfit the data, meaning the kernel is very sensitive to noise. On the opposite, a large value of h leads to oversmooth estimations, meaning we underfit to the dataset.

There are multiple ways to estimate the bandwidth. Statistical methods such as Silverman's rule of thumb minimize the integrated mean squared error and derive a closed form solution, but assume the underlying distribution is known, for instance Gaussian [47]. Another approach is to assume the points are i.i.d. (independent, identically, distributed) and maximize the likelihood function $\mathcal{L}\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \,\middle|\, h\right)$ with respect to h [48]. For some kernel functions it results in maximum likelihood estimates in closed-form, while for others one can use a gradient approach. Finally, a third method commonly used in ML is to estimate h using k-fold Cross-Validation (CV) [49], by splitting the data into k sets (with the same number of points), compute h based of $k-1$ sets and validate the choice in the remaining set. The optimal bandwidth is chosen as the one having the lowest mean squared error in the validation set. Both the likelihood and the CV methods do not assume any type of distribution fixed a priori, and in our empirical evaluations both led to identical results.

Kernel methods can be powerful density approximators but along with their linear (in the dataset size) complexity for inference, they also suffer from the curse of dimensionality [50], since they measure Euclidean distances in high-dimensional spaces.

2.10. Kernel Regression

Let us assume we are faced with a dataset of input-output pairs $D \equiv \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{N}$ with $\boldsymbol{x}_i \in \mathbb{R}^D$ and $y_i = f(\boldsymbol{x_i}) + \varepsilon \in \mathbb{R}$, where $f(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$ is called the *regression function* and ε represents the noise in the measurement, often assumed to be zero-mean Gaussian distributed. In a regression problem the goal is to find $f(\boldsymbol{x})$ such that a loss function is minimized. One can show [46] that the regression

function that minimizes the expected squared error loss

$$\mathbb{E}_{(\boldsymbol{x},y)\sim p(\boldsymbol{x},y)}\left[\mathcal{L}(y, f(\boldsymbol{x}))\right] = \int_{\mathcal{Y}}\int_{\mathcal{X}} \left(y - f(\boldsymbol{x})\right)^2 p(\boldsymbol{x}, y)\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}y$$

is the conditional expectation of y given $\boldsymbol{x}$

$$f(\boldsymbol{x}) = \mathbb{E}\left[y \mid \boldsymbol{x}\right] = \int y p(y \mid \boldsymbol{x})\,\mathrm{d}y = \int y \frac{p(\boldsymbol{x}, y)}{p(y)}\,\mathrm{d}y. \tag{19}$$

Introducing the kernel density estimates for the single (Equation (17)) and joint distribution (Equation (18)) cases in Equation (19), and using the notation $k_h(z) = 1/h \cdot k(z/h)$, we get an approximate regression function $\hat{f}(\boldsymbol{x}) \approx f(\boldsymbol{x})$

$$\begin{aligned}
\hat{f}(\boldsymbol{x}) &= \int y \frac{\hat{p}(\boldsymbol{x}, y)}{\hat{p}(\boldsymbol{x})}\,\mathrm{d}y \\
&= \int \frac{y \frac{1}{N}\sum_{i=1}^{N} k_{h_{\boldsymbol{x}}}(\boldsymbol{x} - \boldsymbol{x}_i) k_{h_y}(y - y_i)}{\frac{1}{N}\sum_{j=1}^{N} k_{h_{\boldsymbol{x}}}(\boldsymbol{x} - \boldsymbol{x}_j)}\,\mathrm{d}y \\
&= \frac{\sum_{i=1}^{N} k_{h_{\boldsymbol{x}}}(\boldsymbol{x} - \boldsymbol{x}_i) \int y k_{h_y}(y - y_i)\,\mathrm{d}y}{\sum_{j=1}^{N} k_{h_{\boldsymbol{x}}}(\boldsymbol{x} - \boldsymbol{x}_j)} \\
&= \frac{\sum_{i=1}^{N} k_{h_{\boldsymbol{x}}}(\boldsymbol{x} - \boldsymbol{x}_i) y_i}{\sum_{j=1}^{N} k_{h_{\boldsymbol{x}}}(\boldsymbol{x} - \boldsymbol{x}_j)},
\end{aligned} \tag{20}$$

where $\int y k_{h_y}(y - y_i)\,\mathrm{d}y = y_i$ can be proved with a change of variables and assuming a zero mean kernel, i.e., with $\int k(z) z\,\mathrm{d}z = 0$.

Equation (20) is the Nadaraya-Watson regression [51, 52], or simply just kernel regression. It can be seen as a weighted average of $y_1, \ldots, y_N$, where the weights are related to how similar the desired point $\boldsymbol{x}$ is to each of the points in the dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. Therefore, it computes a local mean around $\boldsymbol{x}$. In Figure 2 we show how this local mean is affected by different values of bandwidth.

Kernel regression can in theory approximate any function locally, provided with infinite samples and decreasing bandwidth, in contrary to parametric methods, which are constrained to a model class and the number of parameters. Although evaluating a new data point is linear in the number of samples, by using kernels we can work directly in infinite dimensional spaces (e.g., using a Radial Basis Function kernel) and avoid the conversion to a fixed feature space.

2.11. Integral Equations of the Second Kind and the Galerkin Projection

A *Fredholm integral equation of the second kind* has the form

$$\lambda x(\boldsymbol{s}) - \int_{\mathcal{D}} K(\boldsymbol{s}, \boldsymbol{t}) x(\boldsymbol{t})\,\mathrm{d}\boldsymbol{t} = y(\boldsymbol{s}) \quad \forall \boldsymbol{s} \in \mathcal{D}, \tag{21}$$

where $\mathcal{D} \subset \mathbb{R}^m$ is a closed bounded set, $m \geq 1$, $\lambda \neq 0$, and $K(\boldsymbol{s}, \boldsymbol{t})$ is an absolutely integrable function [53]. We want to find the solution for the function x, given λ and $y \neq 0$ (if $y = 0$ we have an eigenvalue-eigenfunction problem). Solving this equation explicitly is a hard problem. Therefore, we find an approximate numerical solution $\tilde{x}$ of x, by projecting x to a space spanned by a finite family of functions.

Equation (21) can also be written as $(\lambda - \mathcal{K})x = y$, where $\mathcal{K}$ is a compact operator on a complete normed vector space (Banach space), for instance the 2-normed space $L^2(\mathcal{D})$. Let x be a function from the space of functions $\mathcal{X}$, $x \in \mathcal{X}$, and consider a subset $\mathcal{X}_n \subset \mathcal{X}$, as the set of functions spanned by n

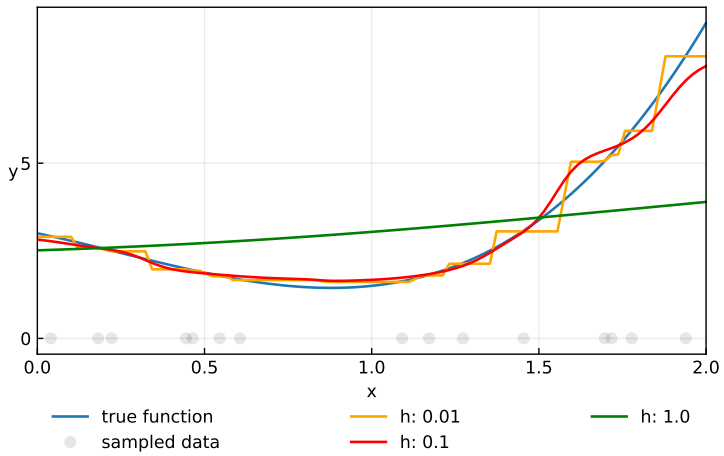


Figure 2.: Kernel regression with different bandwidths This figure illustrates the role of the bandwidth in kernel regression. Here we approximate the function $f(x) = 2x^3 - 1.5x^2 - 2x + 3$ (blue line) in the region $x \in [0, 2]$. We generate 15 pairs (x_i, y_i), where $y_i = f(x_i) + \varepsilon$, with $\epsilon \sim \mathcal{N}(\mu = 0, \sigma = 0.2)$, from which we estimate $f(x)$ with kernel regression, using a Gaussian kernel with varying bandwidths. With a large bandwidth (green) the estimate is very poor, leading to an almost constant value everywhere, since every data point receives very similar weights. Although in this case a very small bandwidth (yellow) approximates fairly well the desired function, it contains flat regions, because it only considers data points from a small neighbourhood. Flat and steep regions can be problematic if we need to differentiate the kernel function, since the gradient will be zero or infinite in many regions. A more plausible value for the bandwidth is shown in red, giving a smoother approximation of the true function.

basis functions $\{b_1, \ldots, b_n\}$. We can describe a function $\tilde{x}$ in the space $\mathcal{X}_n$ as a linear combination of the basis functions

$$\tilde{x}(\boldsymbol{s}) = \sum_{j=1}^{n} c_j b_j(\boldsymbol{s}) \quad \forall \boldsymbol{s} \in \mathcal{D}, \tag{22}$$

where the real coefficients c_j are to be found.

By introducing the approximation from Equation (22) in Equation (21) we can compute a residual when approximating x with $\tilde{x}$

$$\begin{aligned} \tilde{r}(\boldsymbol{s}) &= \lambda \tilde{x}(\boldsymbol{s}) - \int_{\mathcal{D}} K(\boldsymbol{s}, \boldsymbol{t}) \tilde{x}(\boldsymbol{t}) \, \mathrm{d}\boldsymbol{t} - y(\boldsymbol{s}) \\ &= \sum_{j=1}^{n} c_j \left(\lambda b_j(\boldsymbol{s}) - \int_{\mathcal{D}} K(\boldsymbol{s}, \boldsymbol{t}) b_j(\boldsymbol{t}) \, \mathrm{d}\boldsymbol{t} \right) - y(\boldsymbol{s}) \quad \forall \boldsymbol{s} \in \mathcal{D}, \end{aligned} \tag{23}$$

where c_j are determined such that $\tilde{r}$ is approximately zero.

The Galerkin projection method requires the inner product between the residuals and the basis functions to be zero

$$\langle \tilde{r}, b_i \rangle = 0 \quad \text{for } i = 1, \ldots, n, \tag{24}$$

where $\langle \cdot, \cdot \rangle$ is the inner product for the space $\mathcal{X}$, which in a real vector space of continuous functions of domain $\mathcal{D}$ can be written as $\langle f, g \rangle = \int_{\mathcal{D}} f(\boldsymbol{x}) g(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$.

Applying Equation (24) to Equation (23) we obtain

$$\sum_{j=1}^{n} c_j \left(\lambda \langle b_j(\boldsymbol{s}), b_i(\boldsymbol{s}) \rangle - \langle \int_{\mathcal{D}} K(\boldsymbol{s}, \boldsymbol{t}) b_j(\boldsymbol{t}) \, \mathrm{d}\boldsymbol{t}, b_i(\boldsymbol{s}) \rangle \right) - \langle y(\boldsymbol{s}), b_i(\boldsymbol{s}) \rangle = 0 \text{ for } i = 1, \ldots, n$$

$$\sum_{j=1}^{n} c_j \left(\lambda \int_{\mathcal{D}} b_j(\boldsymbol{s}) b_i(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} - \int_{\mathcal{D}} \left(\int_{\mathcal{D}} K(\boldsymbol{s}, \boldsymbol{t}) b_j(\boldsymbol{t}) \, \mathrm{d}\boldsymbol{t} \right) b_i(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \right) - \int_{\mathcal{D}} y(\boldsymbol{s}) b_i(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} = 0 \text{ for } i = 1, \ldots, n,$$

which can be viewed as a linear system of equations being solved for $\{c_1, \ldots, c_n\}$. Additionally, it can be shown that the solution for this problem exists and is unique [53].
Notice that if the basis functions are probability density functions, then Equation (24) can be seen as the expectation of the residuals under each of the n basis functions.

3. Related Work

The Policy Gradient Theorem [35] offers a powerful way to estimate the gradient of the objective function via an expectation. In an on-policy setting it is easily computed with Monte Carlo sampling by direct interaction with the environment, for instance using the REINFORCE algorithm [36]. However, every time the policy is updated the state distribution changes accordingly, meaning that new samples are needed to estimate the next gradient step. The successive interactions are the main cause of high sample complexity. One way to improve sample efficiency is to prevent the state-action distribution induced by the optimized policy begin far from the samples [37], or by sampling trajectories with policies close to the previous optimized one [38, 41], by specifying a threshold for the Kullback–Leibler (KL) divergence.
Off-policy algorithms promise better sample efficiency by optimizing the policy with data collected from different sources, for instance using human demonstrations or a suboptimal policy. However, devising a gradient estimate for off-policy data is not simple, because the state distributions under the behavioural and target policies do not exactly match. In the rest of this section we give an overview on algorithms for off-policy gradient estimation, separated into three classes: Pathwise Importance Sampling; Off-Policy Semi-Gradient; Model-based Policy Gradient.

3.1. Pathwise Importance Sampling

Importance Sampling (IS) methods allow to estimate an expectation of a quantity under a density function that is impractical to sample but easy to evaluate at a desired point [46]. In off-policy learning, this distribution is the policy being optimized π_θ. Using a different distribution β that we can easily sample from, IS computes the expectation and corrects the bias from sampling from β. To compute an expectation under π_θ the most straightforward way is to sample from β and correct the prediction just using the IS weights $\pi_\theta(z)/\beta(z)$, since $\mathbb{E}_{z\sim\pi_\theta}[f(z)] = \mathbb{E}_{z\sim\beta}[\pi_\theta(z)/\beta(z)f(z)]$. Although simple, IS methods have some flaws. If IS weights are low in regions where the behavioural policy is largely dense and high in the opposite case, it often produces high variance estimates and can lead to numerical issues. Therefore, IS methods are suited when both distributions are close.
We denote by Pathwise Importance Sampling (PWIS) the algorithms that weigh trajectories using IS. The first PWIS approaches to off-policy gradient estimation in MDPs and in Partially Observable MDPs (POMDPs) were presented in [43], where the authors apply IS and weighted IS on top of the REINFORCE algorithm [36]. Let us consider the probability of a trajectory τ sampled with π_θ and assuming a Markovian environment, $p_{\pi_\theta}(\tau) = p(\boldsymbol{s}_0)\prod_{t=0}^{T-1}\pi_\theta(\boldsymbol{a}_t \mid \boldsymbol{s}_t)p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$. REINFORCE's gradient estimation is an expectation over on-policy trajectories, $\nabla_\theta J_{\pi_\theta} = \mathbb{E}_{p_{\pi_\theta}(\tau)}\left[R(\tau)\nabla_\theta \log p_{\pi_\theta}(\tau)\right]$, where $R(\tau)$ is the (discounted) return of the sampled trajectory τ. If we can only sample from a behavioural policy β, we can still compute and expectation over p_{π_θ} by correcting the trajectories' probabilities using $\mathbb{E}_{p_{\pi_\theta}(\tau)}\left[f(\tau)\right] = \mathbb{E}_{p_\beta(\tau)}\left[\frac{p_{\pi_\theta}(\tau)}{p_{\beta(\tau)}}f(\tau)\right] = \mathbb{E}_{p_\beta(\tau)}\left[\prod_{t=0}^{T-1}\frac{\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\beta(\boldsymbol{a}_t|\boldsymbol{s}_t)}f(\tau)\right]$, where in the ratio between trajectories' probabilities the transition probabilities cancel out [21]. Hence, a simple modification to REINFORCE using IS leads to

$$\nabla_\theta J_{\pi_\theta} = \mathbb{E}_{p_\beta(\tau)}\left[\prod_{t=0}^{T-1}\frac{\pi_\theta(\boldsymbol{a}_t \mid \boldsymbol{s}_t)}{\beta(\boldsymbol{a}_t \mid \boldsymbol{s}_t)}R(\tau)\nabla_\theta \log p_{\pi_\theta}(\tau)\right].$$

Notice that the transition probabilities of the environment's dynamics do not need to be explicitly known, because $\nabla_\theta \log p_{\pi_\theta}(\tau) = \nabla_\theta \sum_{t=0}^{T-1}\log \pi_\theta(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$. This estimate can be further improved by

noticing that future actions do not influence past rewards [54].
Variations that deal with common IS problems have been developed along the years. For instance, in Guided Policy Search [40] the authors add a regularizer term to the straightforward IS formulation that ensures that some samples drawn from the behavioural policy are very probable under the target policy, and additionally controls how distant from the samples the target policy is allowed to differ. More recently Imani et. al [55] introduced an emphatically weighted policy gradient, which also uses importance sampling to correct for the off-policy state distribution. However the authors refer that to extend the algorithm to continuous action domains an alternative to importance sampling ratios is needed.
Although PWIS methods seem suited for off-policy learning, their main downsides are the possibly large variance estimates and that one needs to know the (stochastic) behavioural policy used to collect the off-policy samples. The latter limits the applicability in some scenarios where it is not easy to access the exact behavioural policy, such as in human demonstrations.

3.2. Off-Policy Semi-Gradient

The off-policy policy gradient theorem was first proposed by Degris et. al [15] as part of the first off-policy actor-critic (Off-PAC) algorithm. The authors consider a modified discounted infinite-horizon objective

$$\hat{J}_{\pi_\theta} = \mathbb{E}_{\boldsymbol{s}\sim\mu_\beta(\cdot)}\left[V_{\pi_\theta}(\boldsymbol{s})\right] = \int_{\mathcal{S}} \mu_\beta(\boldsymbol{s}) V_{\pi_\theta}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}, \tag{25}$$

where μ_β is the state distribution under the behavioural policy β. It is difficult to reason on this objective, because the value function is defined for π_θ but the expectation is taken under μ_β. Futhermore, the authors derive the gradient as

$$\begin{aligned}
\nabla_\theta \hat{J}_{\pi_\theta} &= \nabla_\theta \int_{\mathcal{S}} \mu_\beta(\boldsymbol{s}) \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a} \\
&= \int_{\mathcal{S}} \mu_\beta(\boldsymbol{s}) \int_{\mathcal{A}} Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a}) \nabla_\theta \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) + \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \nabla_\theta Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{a} \\
&\approx \int_{\mathcal{S}} \mu_\beta(\boldsymbol{s}) \int_{\mathcal{A}} Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a}) \nabla_\theta \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s})\,\mathrm{d}\boldsymbol{a},
\end{aligned}$$

where the term $\pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \nabla_\theta Q_{\pi_\theta}(\boldsymbol{s},\boldsymbol{a})$ is dropped because it is difficult to estimate with off-policy data [15]. The authors justify this approximation and provide a proof that guarantees improvement at every gradient step for discrete MDPs. By ignoring the referred term, Off-PAC and other algorithms built on top of the off-policy gradient theorem, such as Deterministic Policy Gradient (DPG) [16] and Deep DPG (DDPG) [39], are coined as semi-gradient algorithms.
We argue that the approximations made by semi-gradient algorithms introduce a bias in the gradient estimate, which is one of the reasons why they cannot deal with truly off-policy samples. A common way for RL algorithms to deal with this issue is by forcing the distributions induced by the target and behavioral policies to be close, following the ideas from Relative Entropy Policy Search (REPS) [37]. For instance, the replay buffer in DDPG is filled with samples from consecutive policies. If the policy parameters do not change abruptly, the samples in the buffer mimic an on-policy setting to a certain extent.
Fujimoto et. al [17] argued that learning algorithms based on sampling from a replay buffer fail in truly off-policy settings. The authors introduced the *extrapolation error* to denote the error made by learning algorithms that sample from a replay buffer with a data distribution that does not match the one induced by the current target policy. Their approach to reduce the extrapolation error is to generate for each state candidate actions that match the distribution in the batch and select the action with highest value from a learned Q-function, leading to the Batch-Constrained Deep Q-Learning (BCQ) algorithm.

3.3. Model-based Policy Gradient

A third way to perform policy gradients with off-policy data is by constructing a model of the underlying MDP. A dataset of transitions is used to model the transition probability of the dynamics and the reward function, from which we can compute the gradient. Since the transitions can come from any policy, most model-based algorithms can be classified as off-policy methods.
Wang et. al [56] solve the policy gradient with the online Model Based Policy Gradient (MBPG) algorithm. The agent explores the environment and learns tabular representations of small models for the transition probability matrix and reward function, which then can be used to compute the value function and the state distribution as in Equation (5) and Equation (6), allowing for a solution of the policy gradient similar to the Policy Gradient Theorem as

$$\nabla_\theta J_{\pi_\theta} = \sum_{\boldsymbol{s}} N_{\pi_\theta}(\boldsymbol{s}) \sum_{\boldsymbol{a}} \nabla_\theta \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \sum_{\boldsymbol{s}'} P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) \left(R(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) + V_{\pi_\theta}(\boldsymbol{s}') \right),$$

where $N_{\pi_\theta}(\boldsymbol{s})$ is the expected number of visits to state $\boldsymbol{s}$ when following π_θ. To update the parameters the authors use a line search algorithm to determine the step size. To keep the model and computations tractable, in successive iteration steps the state-action pairs that are not relevant to update π_θ are discarded. One of the drawbacks of the MBPG algorithm is that it needs to throw away information to be able to build small models of the underlying MDP.
A remarkable example in the class of model-based algorithms is PILCO [9], which learns a probabilistic dynamics model using Gaussian Processes (GPs) regression. By incorporating the modelling uncertainty expressed in the GP, the authors claim it reduces the model-bias. PILCO can learn from scratch with very few samples, making it one of the currently most sample efficient algorithms. To derive a policy gradient in closed-form, PILCO uses moment matching to approximate the distribution over the next state and propagates the gradient through time to update the policy parameters. One important drawback of PILCO is that by performing moment matching it is not able to learn multimodal transitions, which restricts the class of tasks it can solve.

4. Nonparametric Off-Policy Policy Gradient

4.1. Problem Definition

Off-policy sample efficient RL algorithms are crucial for real systems applications. Although recent algorithms perform well in simulated environments, such as the Mujoco physics engine [57], considering the amount of samples needed, sometimes in the order of millions, it would be infeasible to learn directly in the real world, not only due to the required time but also because of hardware wear and tear. A plausible approach is to learn the control policy in simulation and apply it directly in the real world with very little fine-tuning [58]. The problem with simulators is that it is difficult to model all the possible intricacies of interactions, such as friction or measurement noise. A path to obtain sample efficiency is by building a model using information collected directly from the environment, from which we can further sample from, thus reducing the direct interaction with the world. Naturally, building an imperfect model gives rise to other problems, such as the model-bias, which arises from optimizing the policy with data from a model and not from the true environment [59, 9].
Our approach is to derive a policy search gradient-based method to improve sample efficiency, by modelling the environment dynamics with nonparametric density estimation, using a dataset of pre-collected on- or off-policy samples from the environment. As is common in policy gradient methods, let us define the *discounted infinite-horizon starting state objective*

$$J_{\pi_\theta} = \mathbb{E}_{\boldsymbol{s}\sim\mu_0(\cdot)}\left[V_{\pi_\theta}(\boldsymbol{s})\right] = \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) V_{\pi_\theta}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}, \tag{26}$$

where $\mu_0(\boldsymbol{s})$ is the initial state distribution and $V_{\pi_\theta}(\boldsymbol{s})$ the value function at state $\boldsymbol{s}$.
If the policy being optimized is stochastic, the value function turns out

$$\begin{aligned} V_{\pi_\theta}(\boldsymbol{s}) &= \mathbb{E}_{\boldsymbol{a}\sim\pi_\theta(\cdot|\boldsymbol{s})}\left[R\left(\boldsymbol{s},\boldsymbol{a}\right) + \gamma \mathbb{E}_{\boldsymbol{s}'\sim P(\cdot|\boldsymbol{s},\boldsymbol{a})}\left[V_{\pi_\theta}(\boldsymbol{s}')\right]\right] \\ &= \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}\mid\boldsymbol{s}) \left(R\left(\boldsymbol{s},\boldsymbol{a}\right) + \gamma \int_{\mathcal{S}} P\left(\boldsymbol{s}'\mid\boldsymbol{s},\boldsymbol{a}\right) V_{\pi_\theta}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}'\right)\mathrm{d}\boldsymbol{a} \quad \forall \boldsymbol{s}\in\mathcal{S}, \end{aligned} \tag{27}$$

while if it is deterministic we drop the expectation over the actions

$$V_{\pi_\theta}(\boldsymbol{s}) = R(\boldsymbol{s},\pi_\theta(\boldsymbol{s})) + \gamma \int_{\mathcal{S}} P(\boldsymbol{s}'\mid\boldsymbol{s},\pi_\theta(\boldsymbol{s})) V_{\pi_\theta}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' \quad \forall \boldsymbol{s}\in\mathcal{S}. \tag{28}$$

In this chapter we will present our findings relative to the stochastic policy case, since it is more general than the deterministic one. Nevertheless, the derivations for the latter are straightforward to obtain.
The problem we want to solve is to maximize the objective in Equation (26), while complying with the value function constraints

$$\begin{aligned} \max_\theta \; & J_{\pi_\theta} = \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) V_{\pi_\theta}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s} \\ \text{s.t. } & V_{\pi_\theta}(\boldsymbol{s}) = \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}\mid\boldsymbol{s}) \left(R\left(\boldsymbol{s},\boldsymbol{a}\right) + \gamma \int_{\mathcal{S}} P\left(\boldsymbol{s}'\mid\boldsymbol{s},\boldsymbol{a}\right) V_{\pi_\theta}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}'\right)\mathrm{d}\boldsymbol{a} \quad \forall \boldsymbol{s}\in\mathcal{S}. \end{aligned} \tag{29}$$

Notice that it is not straightforward to extract a closed-form solution for the policy parameters (outside the Linear-Quadratic-Regulator assumptions), as the optimization problem presented above may include non-convex constraints over the whole continuous state space represented by the recursion in the value function. However, we can search for an approximate solution directly in the policy parameters space

by updating them using the gradient ascent from Equation (9).

Our goal is to compute an estimate for the policy gradient $\nabla_\theta J_{\pi_\theta}$, which is equivalent to finding $\nabla_\theta V_{\pi_\theta}$, when provided with trajectories collected offline and without explicitly knowledge of the behavioural policy. For that we will build a model of the transition density and reward functions. Of course when a model is present one could possibly use the Policy Gradient Theorem and compute the gradient estimate by running rollouts. However, we are interested in getting a closed-form solution without needing to sample.

4.2. Nonparametric Bellman Equation

As it is intractable to take into account the possibly infinitely many constraints of the optimization problem in Equation (29), following the ideas of Nonparametric Dynamic Programming from Kroemer et. al [18], we replace those constraints with a subset of linear constraints based on collected samples. However, whereas in [18] the authors derive a policy evaluation step without explicitly declaring the policy, we always keep the policy throughout the derivation, which allows to write the value function based on the current policy parameters.

For the probabilistic transitions of dynamical systems one is usually interested in the conditional probability of the next state given the current state-action pair $P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$. A more informative quantity is the joint probability distribution $P(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$, from which we can easily compute the desired conditional distribution using Bayes rule by marginalizing out the next state, $P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) = P(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \,/\, \int P(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}'$.

To represent the joint probability distribution we use kernel density estimation by assuming we only have a finite number of n available transition samples collected in a dataset $D \equiv \{\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'\}_{i=1}^n$. Here $\boldsymbol{s}_i \in \mathbb{R}^{d_S}$ can be any state, $\boldsymbol{a}_i \in \mathbb{R}^{d_A}$ is sampled from an unknown behavourial policy β, which can be independent of the current state, $\boldsymbol{s}'_i \in \mathbb{R}^{d_S}$ is sampled from an underlying probability distribution of the environment by taking action $\boldsymbol{a}_i$ in state $\boldsymbol{s}_i$, $\boldsymbol{s}'_i \sim P(\boldsymbol{s}' \mid \boldsymbol{s}_i, \boldsymbol{a}_i)$, and r_i is sampled from the unknown reward function $r_i \sim R(\boldsymbol{s}_i, \boldsymbol{a}_i)$.

Let us define three different kernel functions ψ, φ and ϕ, for the current state, action and next state, respectively, as

$$\psi : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^+ \tag{30}$$

$$\varphi : \mathcal{A} \times \mathcal{A} \to \mathbb{R}^+ \tag{31}$$

$$\phi : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^+, \tag{32}$$

where they need to be normalized, e.g., $\int \psi(\boldsymbol{x}, \boldsymbol{y}) \,\mathrm{d}\boldsymbol{x} = 1 \;\forall \boldsymbol{y}$, symmetric and positive definite [18]. In practice, the current state kernel ψ and the next state kernel ϕ are the same. Moreover, we define bandwidths for each kernel, $\boldsymbol{h}_\psi$, $\boldsymbol{h}_\varphi$ and $\boldsymbol{h}_\phi$. For notation purposes we rewrite the kernel functions of any input and a collected sample i from the dataset D with $\psi_i(\boldsymbol{s}) = \psi(\boldsymbol{s}, \boldsymbol{s}_i)$, $\varphi_i(\boldsymbol{a}) = \varphi(\boldsymbol{a}, \boldsymbol{a}_i)$ and $\phi_i(\boldsymbol{s}') = \phi(\boldsymbol{s}', \boldsymbol{s}'_i)$. This notation is slightly different from the one introduced in Section 2.9 and Section 2.10, but more readable in the following derivations, e.g., $\psi(\boldsymbol{x}, \boldsymbol{x}_i)$ would be equivalent to $\psi_{\boldsymbol{h}_\psi}(\boldsymbol{x} - \boldsymbol{x}_i)$.

If the next state conditional probability density is approximated with kernel density estimation it follows

$$P(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \approx \hat{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \coloneqq \frac{1}{n} \sum_{i=1}^{n} \psi_i(\boldsymbol{s}) \varphi_i(\boldsymbol{a}) \phi_i(\boldsymbol{s}')$$

$$P(\boldsymbol{s}, \boldsymbol{a}) \approx \hat{P}(\boldsymbol{s}, \boldsymbol{a}) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \psi_i(\boldsymbol{s}) \varphi_i(\boldsymbol{a})$$

$$P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) \approx \hat{P}(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) \coloneqq \frac{\sum_{i=1}^{n} \psi_i(\boldsymbol{s}) \varphi_i(\boldsymbol{a}) \phi_i(\boldsymbol{s}')}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s}) \varphi_j(\boldsymbol{a})}.$$

Similarly, the reward function is approximated with the Nadaraya-Watson (kernel) regression [51, 52]

$$R(\boldsymbol{s},\boldsymbol{a}) \approx \hat{R}(\boldsymbol{s},\boldsymbol{a}) \coloneqq \frac{\sum_{i=1}^{n} \psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a}) r_i}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})}.$$

We can now replace the approximate reward function and transition conditional probability back into the value function constraint of Equation (27) to obtain an approximation for the value function $\hat{V}_{\pi_\theta}(\boldsymbol{s}) \approx V_{\pi_\theta}(\boldsymbol{s})$, leading to the **Nonparametric Bellman Equation** (NPBE)

$$\begin{aligned}
\hat{V}_{\pi_\theta}(\boldsymbol{s}) &= \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \left(\hat{R}\left(\boldsymbol{s},\boldsymbol{a}\right) + \gamma \int_{\mathcal{S}} \hat{P}\left(\boldsymbol{s}' \mid \boldsymbol{s},\boldsymbol{a}\right) \hat{V}_{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \right) \mathrm{d}\boldsymbol{a} \\
&= \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \hat{R}\left(\boldsymbol{s},\boldsymbol{a}\right) \mathrm{d}\boldsymbol{a} + \gamma \int_{\mathcal{A}} \int_{\mathcal{S}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \hat{P}\left(\boldsymbol{s}' \mid \boldsymbol{s},\boldsymbol{a}\right) \hat{V}_{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \,\mathrm{d}\boldsymbol{a} \\
&= \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \frac{\sum_{i=1}^{n} \psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a}) r_i}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})} \,\mathrm{d}\boldsymbol{a} + \gamma \int_{\mathcal{A}} \int_{\mathcal{S}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \frac{\sum_{i=1}^{n} \psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})\phi_i(\boldsymbol{s}')}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})} \hat{V}_{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \,\mathrm{d}\boldsymbol{a} \\
&= \sum_{i=1}^{n} \underbrace{\left(\int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})} \,\mathrm{d}\boldsymbol{a} \right)}_{\varepsilon_i^{\pi_\theta}(\boldsymbol{s})} r_i \\
&\quad + \gamma \sum_{i=1}^{n} \underbrace{\left(\int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})} \,\mathrm{d}\boldsymbol{a} \right)}_{\varepsilon_i^{\pi_\theta}(\boldsymbol{s})} \int_{\mathcal{S}} \phi_i(\boldsymbol{s}') \hat{V}_{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \\
&= \boldsymbol{\varepsilon}_{\pi_\theta}^{\top}(\boldsymbol{s})\mathbf{r} + \gamma \int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}^{\top}(\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}') \hat{V}_{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \quad \forall \boldsymbol{s} \in \mathcal{S},
\end{aligned} \tag{33}$$

where we have defined $\boldsymbol{r} = [r_1, \ldots, r_n]^\top$, $\boldsymbol{\phi}(\boldsymbol{s}') = [\phi_1(\boldsymbol{s}'), \ldots, \phi_n(\boldsymbol{s}')]^\top$, and the basis functions

$$\varepsilon_i^{\pi_\theta}(\boldsymbol{s}) = \begin{cases} \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})} \,\mathrm{d}\boldsymbol{a} & \text{if } \pi_\theta \text{ is stochastic} \\ \frac{\psi_i(\boldsymbol{s})\varphi_i(\pi_\theta(\boldsymbol{s}))}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\pi_\theta(\boldsymbol{s}))} & \text{if } \pi_\theta \text{ is deterministic} \end{cases} \qquad \text{for } i = 1, \ldots, n. \tag{34}$$

Additionally, we make use of the special case of Fubini's Theorem to interchange the integral and summation order [60] (in various steps of the derivations we will use this theorem again without explicitly referring to it).

Notice that $\boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})$ is a valid distribution, because following the kernel and policy definitions we have $\varepsilon_i^{\pi_\theta}(\boldsymbol{s}) \geq 0$ and

$$\begin{aligned}
\sum_{i=1}^{n} \varepsilon_i^{\pi_\theta}(\boldsymbol{s}) &= \sum_{i=1}^{n} \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})} \,\mathrm{d}\boldsymbol{a} \\
&= \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \underbrace{\frac{\sum_{i=1}^{n} \psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^{n} \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})}}_{=1} \,\mathrm{d}\boldsymbol{a} \\
&= \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \,\mathrm{d}\boldsymbol{a} = 1 \quad \forall \boldsymbol{s} \in \mathcal{S}.
\end{aligned}$$

Equation (33) is similar to a Fredholm Integral Equation of the Second Kind (Equation (21)). When

comparing both equations we have

$$\underbrace{\hat{V}_{\pi_\theta}(\boldsymbol{s})}_{\lambda x(\boldsymbol{s})} - \int_{\mathcal{S}} \underbrace{\gamma \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')}_{K(\boldsymbol{s},\boldsymbol{t})} \underbrace{\hat{V}_{\pi_\theta}(\boldsymbol{s}')}_{x(\boldsymbol{t})} \,\mathrm{d}\boldsymbol{s}' = \underbrace{\boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s})\mathbf{r}}_{y(\boldsymbol{s})} \quad \forall \boldsymbol{s} \in \mathcal{S}, \tag{35}$$

with $\lambda = 1$ and $\boldsymbol{t} \equiv \boldsymbol{s}'$. Hence, as initially done in Kromer et. al [18], we can write an approximation for the value function in closed form as a linear combination of basis functions

$$\begin{aligned} \hat{V}_{\pi_\theta}(\boldsymbol{s}) &= \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \left(\mathbf{r} + \gamma \int_{\mathcal{S}} \boldsymbol{\phi}(\boldsymbol{s}') \hat{V}_{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \right) \\ &= \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{q}_{\pi_\theta} \\ &= \sum_i^n \varepsilon_i^{\pi_\theta}(\boldsymbol{s}) q_i^{\pi_\theta} \quad \forall \boldsymbol{s} \in \mathcal{S}, \end{aligned} \tag{36}$$

where $\boldsymbol{\varepsilon}_{\pi_\theta} = [\varepsilon_1^{\pi_\theta}, \ldots, \varepsilon_n^{\pi_\theta}]^\top$ are the n basis functions to project V_{π_θ} onto, and $\boldsymbol{q}_{\pi_\theta} = [q_1^{\pi_\theta}, \ldots, q_n^{\pi_\theta}]^\top$ can be interpreted as the value weights, with $q_i^{\pi_\theta} = \left(r_i + \gamma \int_{\mathcal{S}} \phi_i(\boldsymbol{s}') \hat{V}_{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \right)$ for $i = 1, \ldots, n$. Notice that differently from [18] we explicitly leave the dependency of the value function on the policy parameters. Since the basis are density functions, the value weights can be computed with the Galerking projection method by taking the expectation of Equation (33) with respect to each of the basis functions. Therefore, we project the value function defined over the whole state space onto the space spanned by the n basis functions, thus transforming an infinite set of constraints in a finite set of n constraints

$$\mathbb{E}_{\varepsilon_h^{\pi_\theta}(\boldsymbol{s})} \left[\hat{V}_{\pi_\theta}(\boldsymbol{s}) \right] = \mathbb{E}_{\varepsilon_h^{\pi_\theta}(\boldsymbol{s})} \left[\boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s})\mathbf{r} + \gamma \int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}') \hat{V}_{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}' \right] \quad \text{for } h = 1, \ldots, n.$$

Introducing the value function computed in Equation (36) we obtain

$$\mathbb{E}_{\varepsilon_h^{\pi_\theta}(\boldsymbol{s})} \left[\boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \boldsymbol{q}_{\pi_\theta} \right] = \mathbb{E}_{\varepsilon_h^{\pi_\theta}(\boldsymbol{s})} \left[\boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s})\mathbf{r} + \gamma \int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}') \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s}')^\top \boldsymbol{q}_{\pi_\theta} \,\mathrm{d}\boldsymbol{s}' \right] \quad \text{for } h = 1, \ldots, n$$

$$\begin{bmatrix} \int_{\mathcal{S}} \varepsilon_1^{\pi_\theta}(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \boldsymbol{q}_{\pi_\theta} \,\mathrm{d}\boldsymbol{s} \\ \vdots \\ \int_{\mathcal{S}} \varepsilon_n^{\pi_\theta}(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \boldsymbol{q}_{\pi_\theta} \,\mathrm{d}\boldsymbol{s} \end{bmatrix} = \begin{bmatrix} \int_{\mathcal{S}} \varepsilon_1^{\pi_\theta}(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \boldsymbol{r} \,\mathrm{d}\boldsymbol{s} + \gamma \int_{\mathcal{S}} \varepsilon_1^{\pi_\theta}(\boldsymbol{s}) \int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}') \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s}')^\top \boldsymbol{q}_{\pi_\theta} \,\mathrm{d}\boldsymbol{s}' \,\mathrm{d}\boldsymbol{s} \\ \vdots \\ \int_{\mathcal{S}} \varepsilon_n^{\pi_\theta}(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \boldsymbol{r} \,\mathrm{d}\boldsymbol{s} + \gamma \int_{\mathcal{S}} \varepsilon_n^{\pi_\theta}(\boldsymbol{s}) \int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}') \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s}')^\top \boldsymbol{q}_{\pi_\theta} \,\mathrm{d}\boldsymbol{s}' \,\mathrm{d}\boldsymbol{s} \end{bmatrix}$$

$$\underbrace{\int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \,\mathrm{d}\boldsymbol{s}}_{\mathbf{C}_{\pi_\theta} \in \mathbb{R}^{n \times n}} \boldsymbol{q}_{\pi_\theta} = \int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \,\mathrm{d}\boldsymbol{s}\, \boldsymbol{r} + \gamma \int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \,\mathrm{d}\boldsymbol{s} \underbrace{\int_{\mathcal{S}} \boldsymbol{\phi}(\boldsymbol{s}') \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s}')^\top \,\mathrm{d}\boldsymbol{s}'}_{\hat{\mathbf{P}}_{\pi_\theta} \in \mathbb{R}^{n \times n}} \boldsymbol{q}_{\pi_\theta}$$

$$\mathbf{C}_{\pi_\theta} \boldsymbol{q}_{\pi_\theta} = \mathbf{C}_{\pi_\theta} \boldsymbol{r} + \gamma \mathbf{C}_{\pi_\theta} \hat{\mathbf{P}}_{\pi_\theta} \boldsymbol{q}_{\pi_\theta}, \tag{37}$$

where we defined $\hat{P}_{ij}^{\pi_\theta} = \mathbb{E}_{\boldsymbol{s}' \sim \phi_i(\cdot)} \left[\varepsilon_j^{\pi_\theta}(\boldsymbol{s}') \right] = \int_{\mathcal{S}} \phi_i(\boldsymbol{s}') \varepsilon_j^{\pi_\theta}(\boldsymbol{s}') \,\mathrm{d}\boldsymbol{s}'$ and $\mathbf{C}_{\pi_\theta} = \int_{\mathcal{S}} \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \,\mathrm{d}\boldsymbol{s}$.

Additionally, we state that $\hat{\mathbf{P}}_{\pi_\theta}$ is a (right) stochastic matrix, since each row sums up to 1

$$\begin{aligned}\sum_{j=1}^{n} \hat{P}_{ij}^{\pi_\theta} &= \sum_{j=1}^{n} \int_{\mathcal{S}} \phi_i(\boldsymbol{s}') \varepsilon_j^{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' \\ &= \int_{\mathcal{S}} \phi_i(\boldsymbol{s}') \underbrace{\sum_{j=1}^{n} \varepsilon_j^{\pi_\theta}(\boldsymbol{s}')}_{=1} \mathrm{d}\boldsymbol{s}' \\ &= \int_{\mathcal{S}} \phi_i(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' = 1 \quad \forall i = 1, \ldots, n.\end{aligned}$$

To remove $\mathbf{C}_{\pi_\theta}$ from Equation (37) we need to assure that its inverse exists. In fact, $\mathbf{C}_{\pi_\theta}$ is only singular if two basis functions are the same, which would happen for the deterministic policy case if two samples were coincident. Finally we can express the value weights as

$$\begin{aligned}\mathbf{C}_{\pi_\theta}^{-1} \mathbf{C}_{\pi_\theta} \boldsymbol{q}_{\pi_\theta} &= \mathbf{C}_{\pi_\theta}^{-1} \mathbf{C}_{\pi_\theta} \boldsymbol{r} + \gamma \mathbf{C}_{\pi_\theta}^{-1} \mathbf{C}_{\pi_\theta} \hat{\mathbf{P}}_{\pi_\theta} \boldsymbol{q}_{\pi_\theta} \\ \boldsymbol{q}_{\pi_\theta} &= \boldsymbol{r} + \gamma \hat{\mathbf{P}}_{\pi_\theta} \boldsymbol{q}_{\pi_\theta} \\ \boldsymbol{q}_{\pi_\theta} &= \left(\mathbf{I} - \gamma \hat{\mathbf{P}}_{\pi_\theta} \right)^{-1} \boldsymbol{r} = \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r}. \end{aligned} \tag{38}$$

The matrix $\boldsymbol{\Lambda}_{\pi_\theta}$ is always invertible because its eigenvalues are all non negative. Since $\hat{\mathbf{P}}_{\pi_\theta}$ is a stochastic matrix all its eigenvalues are in absolute value less or equal than 1, and the eigenvalues of $\boldsymbol{\Lambda}_{\pi_\theta}$ are $\geq 1 - \gamma > 0$ for $\gamma \in [0, 1)$ [61].

Therefore, the value function for the whole state space assumes a solution as stated by the following theorem.

Theorem 1. *The closed form solution of the* ***Nonparametric Bellman Equation*** *for every point in the state space, supported by the values from* $\boldsymbol{q}_{\pi_\theta}$*, has a unique fixed-point solution*

$$\hat{V}_{\pi_\theta}(\boldsymbol{s}) \coloneqq \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \boldsymbol{q}_{\pi_\theta} = \boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})^\top \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} \quad \forall \boldsymbol{s} \in \mathcal{S}. \tag{39}$$

Proof. We need to show that

$$\hat{V}_{\pi_\theta}(\boldsymbol{s}) - \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \left(\boldsymbol{r} + \gamma \int_{\mathcal{S}} \boldsymbol{\phi}(\boldsymbol{s}') \hat{V}_{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' \right) = 0 \quad \forall \boldsymbol{s} \in \mathcal{S}. \tag{40}$$

Plugging the solution for the nonparametric bellman equation we get

$$\begin{aligned}&\boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} - \boldsymbol{\varepsilon}_{\pi_\theta}^\top \left(\boldsymbol{r} + \gamma \int_{\mathcal{S}} \boldsymbol{\phi}(\boldsymbol{s}') \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}') \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} \, \mathrm{d}\boldsymbol{s}' \right) \\ &= \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \left(\boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} - \boldsymbol{r} - \gamma \int_{\mathcal{S}} \boldsymbol{\phi}(\boldsymbol{s}') \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}') \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} \, \mathrm{d}\boldsymbol{s}' \right) \\ &= \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \left(\left(\mathbf{I} - \gamma \int_{\mathcal{S}} \boldsymbol{\phi}(\boldsymbol{s}') \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} - \boldsymbol{r} \right) \\ &= \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \left(\boldsymbol{\Lambda}_{\pi_\theta} \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} - \boldsymbol{r} \right) \\ &= 0 \quad \forall \boldsymbol{s} \in \mathcal{S}.\end{aligned}$$

□

4.3. Policy Gradient

With a closed-form solution for the value function in Equation (39) we are now able to derive an analytical gradient of the objective also in closed-form. We start by taking the gradient of the nonparametric value function with respect to the policy parameters

$$\begin{aligned}
\nabla_\theta \hat{V}_{\pi_\theta}(\boldsymbol{s}) &= \nabla_\theta \left(\boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} \right) \\
&= \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} + \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \left(\frac{\partial}{\partial \theta} \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \right) \boldsymbol{r} \\
&= \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} + \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \left(-\boldsymbol{\Lambda}_{\pi_\theta}^{-1} \left(\frac{\partial}{\partial \theta} \boldsymbol{\Lambda}_{\pi_\theta} \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \right) \boldsymbol{r} \\
&= \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} + \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \left(-\boldsymbol{\Lambda}_{\pi_\theta}^{-1} \left(\frac{\partial}{\partial \theta} \left(\mathbf{I} - \gamma \hat{\mathbf{P}}_{\pi_\theta} \right) \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \right) \boldsymbol{r} \\
&= \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} + \gamma \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_{\pi_\theta} \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} \\
&= \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \right) \boldsymbol{q}_{\pi_\theta} + \gamma \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_{\pi_\theta} \right) \boldsymbol{q}_{\pi_\theta} \quad \forall \boldsymbol{s} \in \mathcal{S}.
\end{aligned} \tag{41}$$

Replacing Equation (41) in the objective defined in Equation (26) we obtain

$$\begin{aligned}
\nabla_\theta \hat{J}_{\pi_\theta} &\coloneqq \nabla_\theta \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \hat{V}_{\pi_\theta}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \approx \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \nabla_\theta V_{\pi_\theta}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} = \nabla_\theta J_{\pi_\theta} \\
\nabla_\theta \hat{J}_{\pi_\theta} &= \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \left(\left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \right) \boldsymbol{q}_{\pi_\theta} + \gamma \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_{\pi_\theta} \right) \boldsymbol{q}_{\pi_\theta} \right) \mathrm{d}\boldsymbol{s} \\
&= \left(\int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \right) \boldsymbol{q}_{\pi_\theta} + \gamma \left(\int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \right) \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_{\pi_\theta} \right) \boldsymbol{q}_{\pi_\theta} \\
&= \left(\frac{\partial}{\partial \theta} \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \right) \boldsymbol{q}_{\pi_\theta} + \gamma \boldsymbol{\varepsilon}_{\pi_\theta,0}^\top \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_{\pi_\theta} \right) \boldsymbol{q}_{\pi_\theta} \\
&= \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta,0}^\top \right) \boldsymbol{q}_{\pi_\theta} + \gamma \boldsymbol{\varepsilon}_{\pi_\theta,0}^\top \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_{\pi_\theta} \right) \boldsymbol{q}_{\pi_\theta},
\end{aligned} \tag{42}$$

where we introduced $\boldsymbol{\varepsilon}_{\pi_\theta,0}^\top = \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s}$.
We can further simplify Equation (42) by introducing a new quantity $\boldsymbol{\mu}_{\pi_\theta}$ that we conjecture to be the support points of the (unnormalized) state distribution, since it follows that

$$\begin{aligned}
\boldsymbol{\mu}_{\pi_\theta}^\top &= \boldsymbol{\varepsilon}_{\pi_\theta,0}^\top \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \\
\boldsymbol{\mu}_{\pi_\theta} &= \boldsymbol{\Lambda}_{\pi_\theta}^{-\top} \boldsymbol{\varepsilon}_{\pi_\theta,0} \\
\boldsymbol{\Lambda}_{\pi_\theta}^\top \boldsymbol{\mu}_{\pi_\theta} &= \boldsymbol{\varepsilon}_{\pi_\theta,0} \\
\left(\mathbf{I} - \gamma \hat{\mathbf{P}}_{\pi_\theta}^\top \right) \boldsymbol{\mu}_{\pi_\theta} &= \boldsymbol{\varepsilon}_{\pi_\theta,0} \\
\boldsymbol{\mu}_{\pi_\theta} &= \boldsymbol{\varepsilon}_{\pi_\theta,0} + \gamma \hat{\mathbf{P}}_{\pi_\theta}^\top \boldsymbol{\mu}_{\pi_\theta}.
\end{aligned} \tag{43}$$

Notice the similarities between Equation (43) and Equation (6). For discrete MDPs the first term is the initial state distribution while in a nonparametric modelling we compute an average initial state distribution based on the collected samples. In Chapter 5 we will show empirically that this quantity does provide an estimate of the state distribution over the whole state-space.
Similarly to the extrapolation of the value function for the whole state space (Equation (39)), the state distribution can be computed based on the support points as

$$\mu_{\pi_\theta}(\boldsymbol{s}) \coloneqq \boldsymbol{\varepsilon}_{\pi_\theta}^\top(\boldsymbol{s}) \boldsymbol{\mu}_{\pi_\theta} \quad \forall \boldsymbol{s} \in \mathcal{S}. \tag{44}$$

Replacing the state distribution into Equation (42) we introduce the Nonparametric Off-Policy Policy Gradient Theorem.

Theorem 2. *For a dataset of samples* $D \equiv \{\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'\}_{i=1}^n$ *collected in an off-policy manner with any behavourial policy* β*, transition dynamics modelled with kernel density estimation and reward function with kernel regression, the* ***Nonparametric Off-Policy Policy Gradient*** *is given by*

$$\nabla_\theta \hat{J}_{\pi_\theta} = \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta,0}^\top\right) \boldsymbol{q}_{\pi_\theta} + \gamma \boldsymbol{\mu}_{\pi_\theta}^\top \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_{\pi_\theta}\right) \boldsymbol{q}_{\pi_\theta}. \tag{45}$$

The Nonparametric Off-Policy Policy Gradient has a closed-form solution in the sense that the gradient is not estimated by sampling from the environment (or the model) as in pure on-policy policy gradient. Moreover, once we have *sufficiently many* samples to describe the dynamics and the reward function, no further interaction with the environment is required. Hence, the policy gradient can be estimated in a truly offline and off-policy setting.

Notice that contrary to Off-PAC [15] we started from a well understood and sound objective function and do not exclude any terms when computing the gradient. Therefore, we conjecture (but do not prove it) that the only source of bias in the gradient is if the model is itself biased.

4.4. Why Nonparametric Modelling

An advantage of nonparametric over parametric density estimation models is that they are theoretically guaranteed to be consistent estimators, because as the number of samples grows to infinity and with shrinking bandwidths, the estimates converge to the true density function [45].

The Nadaraya-Watson estimator computes a regression as a local mean and introduces two sources of bias [62]. The first is the design-bias, which is related to how the samples are obtained, since it depends on a term of the form $\beta'(x)/\beta(x)$, where β is the sampling distribution. The second is the boundary bias, resulting from a high bias near the boundaries. It can be shown that with mild continuity conditions on the regression function and on the sampling distribution, with Gaussian kernels and in the limit of infinite number of samples and bandwidths converging to zero, the bias of the Nadaraya-Watson regression tends to zero, and more importantly does not depend on the sampling distribution. This result can be further used to prove that the bias of the solution of the Nonparametric Bellman Equation also shrinks to zero under the same conditions.

Even though principled, kernel density estimation and regression models have two major downsides. They suffer from the curse of dimensionality and their naive inference complexity is $\mathcal{O}(n)$, where n is the number of samples. Therefore, scaling to high-dimensional state and action spaces is not elementary.

4.5. Implementation Details

Although we presented a way to derive an expression for the policy gradient in closed form in Equation (45), a correct implementation still needs to choose which kernel to use, the right bandwidth and estimate quantities involving integrals. For the latter, most quantities can be seen as expectations of random variables and thus straightforwardly solved using Monte Carlo integration techniques [63].

The choice of the kernel functions and their bandwidths should not be overlooked. Different choices are possible, but in order to get smooth and differentiable density models we chose the Multivariate Gaussian [64], which satisfies the conditions imposed in Section 4.2. We consider in practical implementations each dimension of the state and action features to be independent [9, 29], i.e., their bandwidths can be described as diagonal matrices, $\boldsymbol{h}_\psi, \boldsymbol{h}_\phi \in \mathbb{R}^{d_S}$ and $\boldsymbol{h}_\varphi \in \mathbb{R}^{d_A}$, where d_S is the number of state dimensions and d_A the number of action dimensions. With this definition of bandwidth we can write

the kernel function as (for instance for ψ)

$$\psi(\boldsymbol{s}, \boldsymbol{s}_i) = \frac{1}{\sqrt{(2\pi)^k |\mathrm{diag}(\boldsymbol{h}_\psi)|}} \exp\left\{ -\frac{1}{2} (\boldsymbol{s} - \boldsymbol{s}_i)^\top \mathrm{diag}(\boldsymbol{h}_\psi)^{-1} (\boldsymbol{s} - \boldsymbol{s}_i) \right\}$$

where diag refers to the diagonal matrix and $|\mathbf{X}|$ the determinant of the matrix $\mathbf{X}$.

As already referred in Section 2.9, there are multiple ways to estimate the bandwidths. We opt to chose them via cross-validation. The candidate bandwidths are picked from the interval $[h_{\text{silv}} - \alpha; h_{\text{silv}} + \alpha]$, where h_{silv} is the bandwidth estimated with Silverman's rule of thumb, and α an hyperparameter.

We now detail how to compute the quantities involving integral computations. We approximate each entry of $\boldsymbol{\varepsilon}_{\pi_\theta}(\boldsymbol{s})$ and $\boldsymbol{\varepsilon}_{\pi_\theta,0}$ with Monte Carlo sampling as

$$\begin{aligned} \varepsilon_i^{\pi_\theta}(\boldsymbol{s}) &= \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}) \frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^n \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})} \, \mathrm{d}\boldsymbol{a} \\ &\approx \frac{1}{N_\pi^{\text{MC}}} \sum_{z=1}^{N_\pi^{\text{MC}}} \frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a}_z)}{\sum_{j=1}^n \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a}_z)} \quad \text{with } \boldsymbol{a}_z \sim \pi_\theta(\cdot \mid \boldsymbol{s}), \end{aligned} \tag{46}$$

$$\begin{aligned} \varepsilon_i^{\pi_\theta,0} &= \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \varepsilon_i^{\pi_\theta}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \\ &\approx \frac{1}{N_{\mu_0}^{\text{MC}}} \sum_{z=1}^{N_{\mu_0}^{\text{MC}}} \varepsilon_i^{\pi_\theta}(\boldsymbol{s}_z) \quad \text{with } \boldsymbol{s}_z \sim \mu_0(\cdot). \end{aligned} \tag{47}$$

For many environments, although the state distribution might change depending on the task at hand, we can consider the existence of a single initial state $\boldsymbol{s}_0$. For instance, a robot arm may return to an initial position after performing a selected task. In such scenarios, the integral in Equation (47) reduces to $\varepsilon_i^{\pi_\theta,0} = \varepsilon_i^{\pi_\theta}(\boldsymbol{s}_0)$.

To estimate the entries of matrix $\hat{\mathbf{P}}_{\pi_\theta}$ we also use Monte Carlo integration. However, a better approximation is to consider that with a large amount of samples we may set a very small kernel bandwidth and approximate the integral by taking the mean of $\phi_i(\cdot)$, which is just the sampled next state $\boldsymbol{s}'_i$

$$\hat{P}_{ij}^{\pi_\theta} = \int_{\boldsymbol{s}} \phi_i(\boldsymbol{s}') \varepsilon_j^{\pi_\theta}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' \tag{48}$$

$$\approx \frac{1}{N_\phi^{\text{MC}}} \sum_{z=1}^{N_\phi^{\text{MC}}} \varepsilon_j^{\pi_\theta}(\boldsymbol{s}'_z) \quad \text{with } \boldsymbol{s}'_z \sim \phi_i(\cdot) \tag{49}$$

$$\approx \varepsilon_j^{\pi_\theta}(\boldsymbol{s}'_i) \quad \text{with } ||\boldsymbol{h}_\phi|| \text{ small enough.} \tag{50}$$

Due to the nature of kernels, many entries of $\hat{\mathbf{P}}_{\pi_\theta}$ are close to zero, as depicted in Figure 3. Hence, a reasonable (and crucial) approximation is to sparsify $\hat{\mathbf{P}}_{\pi_\theta}$ and only keep the k largest values of each row, at a cost of $\mathcal{O}(k \cdot \log n)$ per row [65], directly reducing the memory of storing $\hat{\mathbf{P}}_{\pi_\theta}$ from $\mathcal{O}(n^2)$ to $O(k\,n)$. Although additional computational cost for sparsifying $\hat{\mathbf{P}}_{\pi_\theta}$ is required, having a sparse matrix outweighs that cost, since many matrix operations are more computational efficient when matrices are sparse. The question that remains is how to choose k. In Figure 4 we depict how the mean KL divergence per row between the original and sparse matrix changes with the number of elements kept. We define the optimal k (k^*), as the minimum k such that the average KL divergence per row is below

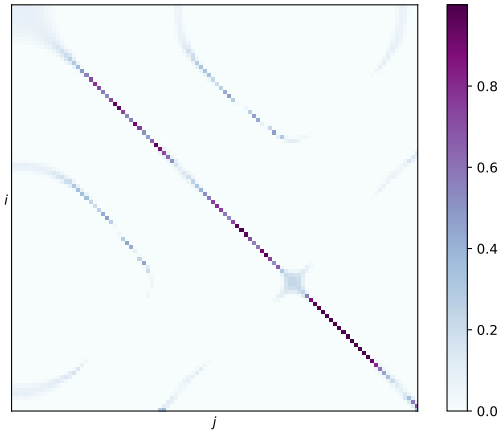


Figure 3.: Example of a $\hat{\mathbf{P}}_{\pi_\theta}$ matrix Each entry in $\hat{\mathbf{P}}_{\pi_\theta} \in \mathbb{R}^{100\times 100}$ was computed by taking the mean of $\phi_i(\boldsymbol{s}')$ as in Equation (50). From the figure it is noticeable that most entries are close to zero, making it suitable for sparsification. Moreover, the visible main diagonal comes from the states $\boldsymbol{s}'_i$ and $\boldsymbol{s}_i$ being close, since the rows are ordered by the order the samples were collected. This figure was generated by collecting 100 samples from the Pendulum-v0 environment [19] by starting in the upright position and applying actions sampled from a mixture of two Gaussians as explained in Section 5.3. The policy being optimized was a neural network of one hidden layer and 50 neurons, with ReLU activation functions and randomly initialized weights.

a desired threshold. Formally k^* is defined as

$$k^* = \operatorname*{arg\,min}_{k=1,\dots,n} \frac{1}{n}\sum_{i=1}^{n} \mathrm{D}_{\mathrm{KL}}\left(\hat{\mathbf{P}}_i^{\pi_\theta} \,\middle\|\, \hat{\mathbf{P}}^{\pi_\theta}_{\mathrm{sparse}_i}(k)\right) \leq \delta_{\mathrm{sparse}}, \tag{51}$$

where $\hat{\mathbf{P}}_i^{\pi_\theta}$ is the i^{th} row of $\hat{\mathbf{P}}_{\pi_\theta}$, $\hat{\mathbf{P}}^{\pi_\theta}_{\mathrm{sparse}_i}(k)$ is constructed with the i^{th} row of $\hat{\mathbf{P}}_{\pi_\theta}$ by keeping the top k values, setting the rest to zero and finally normalizing to 1, and δ_{sparse} is the chosen threshold for the average KL divergence.

Determining k^* is costly and cannot be done in every policy improvement step. Nevertheless, we can compute it once before optimizing the policy and see it as an empirical choice. Unfortunately, we do not have any theoretical guarantees on the effects of sparsification, but we found it to work well empirically.

An important part in the computation of the policy gradient are also the gradients of $\boldsymbol{\varepsilon}_{\pi_\theta,0}$ and $\hat{\mathbf{P}}_{\pi_\theta}$ with respect to the policy parameters. For the former we have

$$\frac{\partial}{\partial\theta}\varepsilon_i^{\pi_\theta,0} = \int_{\mathcal{S}} \mu_0(\boldsymbol{s}) \frac{\partial}{\partial\theta}\varepsilon_i^{\pi_\theta}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}$$

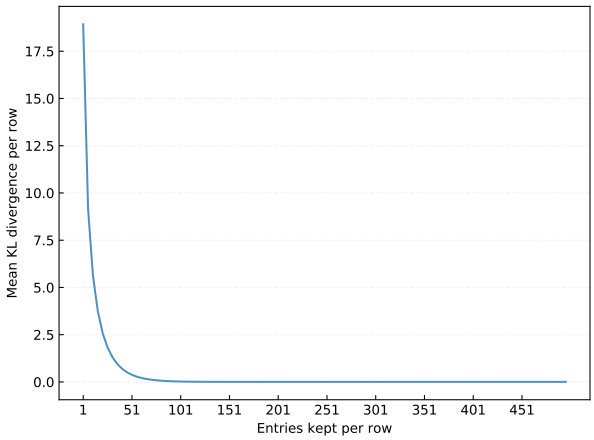


Figure 4.: Mean KL divergence between $\hat{\mathbf{P}}_{\pi_\theta}$ and $\hat{\mathbf{P}}^{\mathbf{sparse}}_{\pi_\theta}$ for different levels of sparsification In this figure we show the mean KL divergence per row if only the top k entries per row of $\hat{\mathbf{P}}_{\pi_\theta}$ are kept. The dataset for this experiment was collected by sampling 500 transitions from the CartPole environment [20] with actions randomly sampled as described in Section 5.3. The curve tells that only a fraction of each row has relevant information. For instance, for this task, we are able to achieve convergence and do not lose substantially more information in $\hat{\mathbf{P}}_{\pi_\theta}$ if we only keep the top 75 entries of each row and set the rest to zero.

where by expanding and applying the log-ratio trick we can compute the derivative also by sampling

$$\begin{aligned}
\frac{\partial}{\partial\theta}\varepsilon_i^{\pi_\theta}(\boldsymbol{s}) &= \frac{\partial}{\partial\theta}\int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}\mid\boldsymbol{s})\frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^n \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})}\,\mathrm{d}\boldsymbol{a} \\
&= \int_{\mathcal{A}} \frac{\partial}{\partial\theta}\pi_\theta(\boldsymbol{a}\mid\boldsymbol{s})\frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^n \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})}\,\mathrm{d}\boldsymbol{a} \\
&= \int_{\mathcal{A}} \pi_\theta(\boldsymbol{a}\mid\boldsymbol{s})\frac{\partial}{\partial\theta}\log\pi_\theta(\boldsymbol{a}\mid\boldsymbol{s})\frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^n \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})}\,\mathrm{d}\boldsymbol{a} \\
&\approx \frac{1}{N_\pi^{\mathrm{MC}}}\sum_{z=1}^{N_\pi^{\mathrm{MC}}}\frac{\partial}{\partial\theta}\log\pi_\theta(\boldsymbol{a}_z\mid\boldsymbol{s})\frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a}_z)}{\sum_{j=1}^n \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a}_z)}\,\mathrm{d}\boldsymbol{a} \quad \text{with } \boldsymbol{a}_z\sim\pi_\theta(\cdot\mid\boldsymbol{s}).
\end{aligned}$$

In addition, if the policy is a (multivariate) Gaussian distribution with a mean parameterized by f_θ and a covariance by g_θ, i.e. $\pi_\theta(\boldsymbol{a}\mid\boldsymbol{s})\sim\mathcal{N}\left(\boldsymbol{a}\,\middle|\,\boldsymbol{\mu}=f_\theta(\boldsymbol{s}),\boldsymbol{\Sigma}=g_\theta(\boldsymbol{s})\right)$, we can propagate the gradient through a stochastic function using the reparameterization trick [44] by writing $\varepsilon_i^{\pi_\theta}(\boldsymbol{s})$ as

$$\begin{aligned}
\varepsilon_i^{\pi_\theta}(\boldsymbol{s}) &= \int_{\mathcal{A}}\mathcal{N}\left(\boldsymbol{a}\,\middle|\,f_\theta(\boldsymbol{s}),g_\theta(\boldsymbol{s})\right)\frac{\psi_i(\boldsymbol{s})\varphi_i(\boldsymbol{a})}{\sum_{j=1}^n \psi_j(\boldsymbol{s})\varphi_j(\boldsymbol{a})}\,\mathrm{d}\boldsymbol{a} \\
&= \int_{\eta}\mathcal{N}\left(\eta\,\middle|\,\mathbf{0},\mathbf{I}\right)\frac{\psi_i(\boldsymbol{s})\varphi_i(f_\theta(\boldsymbol{s})+\eta g_\theta(\boldsymbol{s}))}{\sum_{j=1}^n \psi_j(\boldsymbol{s})\varphi_j(f_\theta(\boldsymbol{s})+\eta g_\theta(\boldsymbol{s}))}\,\mathrm{d}\eta
\end{aligned} \tag{52}$$

and applying the chain rule to compute the derivative with respect to θ.
For $\hat{\mathbf{P}}_{\pi_\theta}$ it is straightforward to compute the gradient as well

$$\frac{\partial}{\partial\theta}\hat{P}_{ij}^{\pi_\theta} = \int_{\boldsymbol{s}}\phi_i(\boldsymbol{s}')\frac{\partial}{\partial\theta}\varepsilon_j^{\pi_\theta}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}'.$$

We move now to compute the support points of the value function and state distribution, which we can view as two systems of linear equations

$$\begin{aligned} \boldsymbol{q}_{\pi_\theta} &= \boldsymbol{\Lambda}_{\pi_\theta}^{-1} \boldsymbol{r} \\ \Leftrightarrow \ \boldsymbol{\Lambda}_{\pi_\theta} \boldsymbol{q}_{\pi_\theta} &= \boldsymbol{r}, \end{aligned} \tag{53}$$

$$\begin{aligned} \boldsymbol{\mu}_{\pi_\theta} &= \boldsymbol{\Lambda}_{\pi_\theta}^{-\top} \boldsymbol{\varepsilon}_{\pi_\theta,0} \\ \Leftrightarrow \ \boldsymbol{\Lambda}_{\pi_\theta}^{\top} \boldsymbol{\mu}_{\pi_\theta} &= \boldsymbol{\varepsilon}_{\pi_\theta,0}. \end{aligned}$$

The naive solution for both is to directly invert $\boldsymbol{\Lambda}_{\pi_\theta}$ (or its transpose), which when naively performed costs $O(n^3)$, where n is the number of collected samples, thus undermining the application of the algorithm for datasets of reasonable size. Instead, we can use other solving approaches such as the Conjugate Gradient method. Assuming $\hat{\mathbf{P}}_{\pi_\theta}$ is sparse so is $\boldsymbol{\Lambda}_{\pi_\theta}$. Hence, we have a sparse system of linear equations, for which the Conjugate Gradient method is still one of the most fitted to solve [66]. This method works straightforwardly if $\boldsymbol{\Lambda}_{\pi_\theta}$ is squared, symmetric and positive-definite. Although the first and third conditions hold, $\boldsymbol{\Lambda}_{\pi_\theta}$ is not necessarily symmetric. Nevertheless, we can always solve the system $\boldsymbol{\Lambda}_{\pi_\theta}^{\top} \boldsymbol{\Lambda}_{\pi_\theta} \boldsymbol{q}_{\pi_\theta} = \boldsymbol{\Lambda}_{\pi_\theta}^{\top} \boldsymbol{r}$, which yields the same result as Equation (53) [66] (the same is true when solving for $\boldsymbol{\mu}_{\pi_\theta}$), at the expense of slower convergence.
With the previously detailed calculations we can compute the policy gradient from Equation (45). To update the policy parameters θ we use the update rule described in Equation (9) with an adaptive learning rate scheme such as ADAM [67].
A high level pseudo-code of a comprehensive algorithm is presented in Algorithm 1. We denote it as the Nonparametric Off-Policy Policy Gradient Algorithm (NOPG), and in particular NOPG Stochastic (NOPG-S) and NOPG Deterministic (NOPG-D) when optimizing a stochastic or deterministic policy, respectively.

4.6. Computational and Memory Complexity

As any nonparametric algorithm, NOPG computational and memory complexities are highly dependent on the number of collected samples n. The most relevant requirements per policy update in Algorithm 1 are:

- In line 2, building the vector $\boldsymbol{\varepsilon}_{\pi_\theta,0}$ takes $\mathcal{O}\left(N_{\mu_0}^{\mathrm{MC}} N_{\pi}^{\mathrm{MC}} n\right)$. For the complexity computation we assume always a constant cost of sampling from a distribution and also drop the fact that we need to normalize the vector, which takes at least two times the size of the vector, since it involves diving each entry the sum of all entries. Storing $\boldsymbol{\varepsilon}_{\pi_\theta,0}$ needs $\mathcal{O}\left(N_{\mu_0}^{\mathrm{MC}} N_{\pi}^{\mathrm{MC}} n\right)$ memory;

- In line 3, we compute $\hat{\mathbf{P}}_{\pi_\theta}$ row by row, by sparsifying each row via selecting the top k elements, leading to a complexity of $\mathcal{O}\left(N_{\phi}^{\mathrm{MC}} N_{\pi}^{\mathrm{MC}} n^2 \log k\right)$, since $\mathcal{O}(n \log k)$ is the cost of processing one row. Storing $\hat{\mathbf{P}}_{\pi_\theta}$ needs $\mathcal{O}\left(N_{\phi}^{\mathrm{MC}} N_{\pi}^{\mathrm{MC}} k n\right)$;

- In line 5 solving the linear system of equations with the Conjugate Gradient method takes $\mathcal{O}\left(\sqrt{\nu}(k+1)n\right)$, where ν is the condition number of $\hat{\mathbf{P}}_{\pi_\theta}$ after sparsification, and $(k+1)n$ the number of nonzero elements [66]. The plus one comes from the computation of $\boldsymbol{\Lambda}_{\pi_\theta}$, since subtracting $\hat{\mathbf{P}}_{\pi_\theta}$ from the identity matrix can lead to a increase of n nonzero elements. Comparing the result with the naive solution involving a matrix inverse, which costs in the order of $\mathcal{O}\left(n^3\right)$, we see the advantage of using the Conjugate Gradient method in sparse matrices. As a side note, computing the condition number ν is computationally intensive, but methods to compute an upper bound exist [68];

Algorithm 1 Nonparametric Off-Policy Policy Gradient (NOPG)

Input:

Dataset $D \equiv \{\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i, t_i\}_{i=1}^{n}$, where t_i indicates a terminal state
Parameterized policy π_θ
Kernels ψ, φ and ϕ for state, action and next state, respectively
Policy updates T
Learning rate α
Discount factor γ
Initial state distribution μ_0

Output:

Optimized policy π_θ

1: **for** $t = 1, \ldots, T$ **do**

2: Build $\boldsymbol{\varepsilon}_{\pi_\theta,0}$ as in Equation (47) (with $\varepsilon_i^{\pi_\theta}$ from Equation (46))

3: Build each entry of $\hat{\mathbf{P}}_{\pi_\theta}$ as in Equation (49) (with $\varepsilon_j^{\pi_\theta}$ from Equation (46)), while sparsifying each row using Equation (51). Set row i of $\hat{\mathbf{P}}_{\pi_\theta}$ to 0 if t_i indicates a terminal state

4: Build matrix $\boldsymbol{\Lambda}_{\pi_\theta}$ as

$$\boldsymbol{\Lambda}_{\pi_\theta} = \mathbf{I} - \gamma \hat{\mathbf{P}}_{\pi_\theta}$$

5: Evaluate $\boldsymbol{\Lambda}_{\pi_\theta}$ at the current policy parameters and solve the linear systems of equations for $\boldsymbol{q}_{\pi_\theta}$ and $\boldsymbol{\mu}_{\pi_\theta}$ using the Conjugate Gradient method

$$\boldsymbol{\Lambda}_{\pi_\theta} \boldsymbol{q}_{\pi_\theta} = \boldsymbol{r} \qquad \boldsymbol{\Lambda}_{\pi_\theta}^{\top} \boldsymbol{\mu}_{\pi_\theta} = \boldsymbol{\varepsilon}_{\pi_\theta,0}$$

6: Compute the policy gradient

$$\nabla_\theta \hat{J}_{\pi_\theta} = \left(\frac{\partial}{\partial \theta} \boldsymbol{\varepsilon}_{\pi_\theta,0}^{\top} \right) \boldsymbol{q}_{\pi_\theta} + \gamma \boldsymbol{\mu}_{\pi_\theta}^{\top} \left(\frac{\partial}{\partial \theta} \hat{\mathbf{P}}_{\pi_\theta} \right) \boldsymbol{q}_{\pi_\theta}$$

7: Update the policy parameters with a gradient ascent technique

$$\theta \leftarrow \theta + \alpha \nabla_\theta \hat{J}_{\pi_\theta}$$

8: **end for**

- In line 6, the cost of the vector-vector multiplication $\left(\frac{\partial}{\partial\theta}\boldsymbol{\varepsilon}_{\pi_\theta,0}^\top\right)\boldsymbol{q}_{\pi_\theta}$ is $\mathcal{O}\left(N_{\mu_0}^{\text{MC}}N_{\pi}^{\text{MC}}n\right)$, and the vector-(sparse) matrix-vector multiplication $\boldsymbol{\mu}_{\pi_\theta}^\top\left(\frac{\partial}{\partial\theta}\hat{\mathbf{P}}_{\pi_\theta}\right)\boldsymbol{q}_{\pi_\theta}$ is $\mathcal{O}\left(N_{\phi}^{\text{MC}}N_{\pi}^{\text{MC}}kn^2\right)$, thus totalling $\mathcal{O}\left(N_{\mu_0}^{\text{MC}}N_{\pi}^{\text{MC}}n + N_{\phi}^{\text{MC}}N_{\pi}^{\text{MC}}kn^2\right)$. Assuming the number of policy parameters M to be much lower than the number of samples, $M << n$, we ignore the gradient computation, since even when using a simple method of differentiation such as finite differences, we would have $\mathcal{O}(M) << \mathcal{O}(n)$.

The reader might be asking why do we consider in every step the terms of the Monte Carlo integration, instead of discarding them, since in the Big-$\mathcal{O}$ notation it is common to drop constant terms. We left these terms to emphasize that the policy parameters are "hidden" inside each entry of $\boldsymbol{\varepsilon}_{\pi_\theta,0}$ and $\hat{\mathbf{P}}_{\pi_\theta}$, and thus we need to keep these terms until we compute the gradient. In fact, modern engines using automatic differentiation for gradient computation such as Tensorflow [69], build a static computational graph to *backpropagate* gradients. Therefore, we cannot simply ignore these constants. The only exception is when computing $\boldsymbol{q}_{\pi_\theta}$ and $\boldsymbol{\mu}_{\pi_\theta}$ with Conjugate Gradient. Here we drop the terms because $\boldsymbol{\Lambda}_{\pi_\theta}$ is evaluated for the current policy parameters, leading to a $\hat{\mathbf{P}}_{\pi_\theta}$ matrix effectively represented with $k \times n$ elements.
Taking into account all the costs, we conclude that the computational complexity of NOPG *per policy update* is

$$\begin{aligned}
&\underbrace{\mathcal{O}\left(N_{\mu_0}^{\text{MC}}N_{\pi_\theta}^{\text{MC}}n\right)}_{\boldsymbol{\varepsilon}_{\pi_\theta,0}} + \underbrace{\mathcal{O}\left(N_{\phi}^{\text{MC}}N_{\pi_\theta}^{\text{MC}}n^2\log k\right)}_{\hat{\mathbf{P}}_{\pi_\theta}} + \underbrace{\mathcal{O}\left(\sqrt{\nu}(k+1)n\right)}_{\text{Conj. Grad. } \boldsymbol{q}_{\pi_\theta},\boldsymbol{\mu}_{\pi_\theta}} \\
&\quad + \underbrace{\mathcal{O}\left(N_{\mu_0}^{\text{MC}}N_{\pi_\theta}^{\text{MC}}n\right)}_{\left(\frac{\partial}{\partial\theta}\boldsymbol{\varepsilon}_{\pi_\theta,0}^\top\right)\boldsymbol{q}_{\pi_\theta}} + \underbrace{\mathcal{O}\left(N_{\phi}^{\text{MC}}N_{\pi_\theta}^{\text{MC}}kn^2\right)}_{\boldsymbol{\mu}_{\pi_\theta}^\top\left(\frac{\partial}{\partial\theta}\hat{\mathbf{P}}_{\pi_\theta}\right)\boldsymbol{q}_{\pi_\theta}} \\
&= \mathcal{O}\left(N_{\mu_0}^{\text{MC}}N_{\pi_\theta}^{\text{MC}}n\right) + \mathcal{O}\left(N_{\phi}^{\text{MC}}N_{\pi_\theta}^{\text{MC}}n^2(k+\log k)\right) + \mathcal{O}\left(\sqrt{\nu}(k+1)n\right),
\end{aligned}$$

and the memory complexity is

$$\underbrace{\mathcal{O}\left(N_{\mu_0}^{\text{MC}}N_{\pi_\theta}^{\text{MC}}n\right)}_{\boldsymbol{\varepsilon}_{\pi_\theta,0}} + \underbrace{\mathcal{O}\left(N_{\phi}^{\text{MC}}N_{\pi_\theta}^{\text{MC}}kn\right)}_{\hat{\mathbf{P}}_{\pi_\theta}} + \underbrace{\mathcal{O}\left(n\right)}_{\boldsymbol{q}_{\pi_\theta}} + \underbrace{\mathcal{O}\left(n\right)}_{\boldsymbol{\mu}_{\pi_\theta}},$$

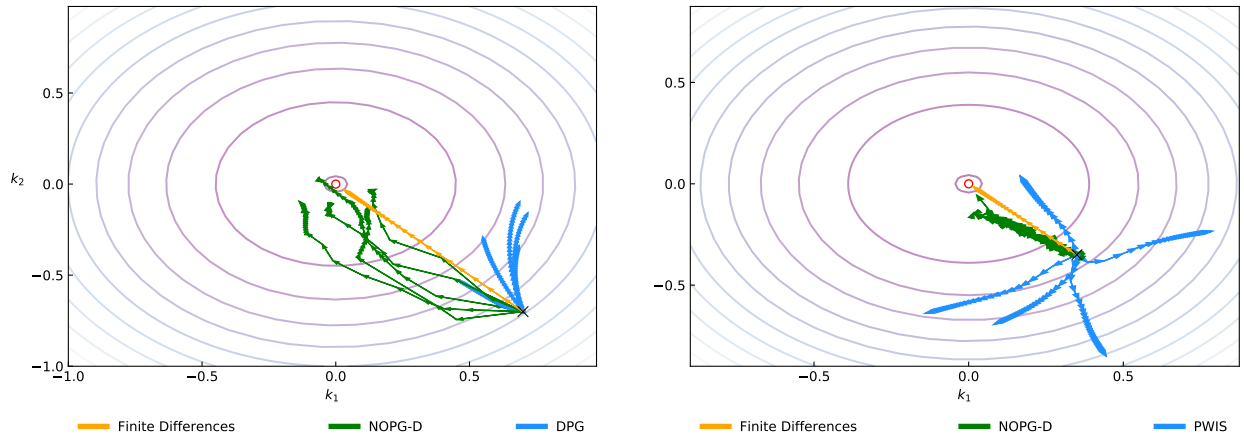
where the quantities in "underbrace" indicate the source of the complexities as described in this chapter. Hence, we state that NOPG has nearly quadratic computational complexity and linear memory complexity with respect to the number of samples n, per policy update.

5. Experiments

In this section we present empirical results between the deterministic and stochastic versions of NOPG and different on- and off-policy algorithms. The algorithms chosen are: Deterministic Policy Gradient (DPG) [16]; Deep Deterministic Policy Gradient (DDPG) [39]; Trust Region Policy Optimization (TRPO) [38] (with implementations from OpenAI baselines [70]); DDPG Offline, which is a modified version of DDPG where the gradient updates use full-batch samples from a fixed replay buffer that is populated with data collected following a behavioural policy; and Gradient of a Partially Observable Markov Decision Process (G(PO)MDP) [54] with Pathwise Importance Sampling correction, which we shorten to PWIS throughout the rest of this work.
We first present in Section 5.1 a qualitative comparison between the gradient estimates of NOPG, DPG and PWIS, followed by quantitative empirical results of NOPG, DDPG, TRPO, DDPG Offline and PWIS in the OpenAI Gym [19] and Quanser Control Systems [20] platforms for different environments. Each experiment is chosen to highlight NOPG's sample complexity, the ability to learn with randomly sampled data and to optimize a policy based on suboptimal trajectories. Moreover, all environments have continuous state-action spaces to mimic closer a real world scenario.
For a fair comparison, in all experiments and algorithms we use a parameterized policy encoded with a neural network. The policy has a single hidden layer with 50 units and Rectified Linear Unit (ReLU) activations. For deterministic policies the output of the neural network is a single action, while for stochastic policies the outputs are the mean and covariance of a Gaussian distribution, as explained in Section 2.8.
A crucial point in the NOPG algorithm is the choice of the kernel and its bandwidth. This is a general problem in kernel density estimation as explained in Section 2.9 and not particular to NOPG. We decided to use Multivariate Gaussian kernels with diagonal bandwidths, as presented in Section 4.5. The bandwidths are estimated using cross validation. For each dimension of the state and action spaces we first compute an initial rough estimate with Silverman's rule of thumb [47], h_{silv}, and after select 30 bandwidths from the linearly spaced interval $[0, 1.5 \cdot h_{\text{silv}}]$, from which we select the best using 5-fold cross validation. Additionally, we can specify an external hyperparameter as a multiplicative scaling factor to adjust the bandwidth with $\boldsymbol{h}_{\text{factor}} = [\ldots, h_i^{\text{factor}}, \ldots]$, where h_i^{factor} is the i-th dimension factor. In the quantitative experiments, we report the policy evaluation results for NOPG using the policy after the last optimization step, while for DDPG Offline and PWIS we take the maximum value obtained, since we noticed that learning was unstable. Detailed information on the experiments and hyperparameters can be found in Appendix B for the experiment in Section 5.1, and in Appendix C for the other experiments.

5.1. Qualitative Gradient Comparison

To assess the quality of NOPG's gradient estimate we set up an experiment with a simple 2-dimensional LQR problem. We compare the estimates for a deterministic policy with NOPG-D and DPG, and for a stochastic policy using NOPG-S and PWIS. For both settings we compute the true gradient of the LQR system using finite differences. The results are depicted in Figure 5.
In both experiments we collected 5 datasets of 100 trajectories of 30 steps each. Each trajectory starts in a given initial state and transitions according to the LQR dynamics using a policy encoded as a diagonal matrix $\mathbf{K} = \text{diag}(k_1, k_2)$. For the deterministic experiment each trajectory is generated by adding Gaussian noise to an initial policy $\mathbf{K}_{\text{init}}$, and thereafter keeping the policy unchanged. For the stochastic policy experiment, in each step of the trajectory the policy is forced to be stochastic by



(a) LQR with deterministic policy

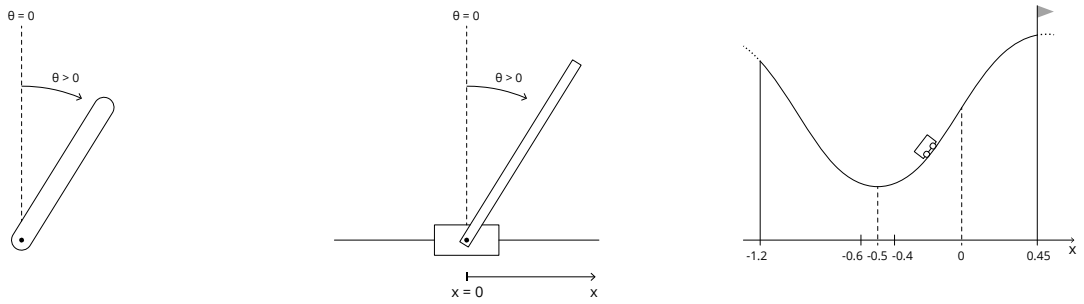
(b) LQR with stochastic policy

Figure 5.: Comparison of gradient estimates in a LQR system In each plot the isolines show the 2-dimensional LQR return for different values of a diagonal policy parameterization. Each arrow trajectory corresponds to the policy gradient estimates of different algorithms. The red circle at $(k_1, k_2) = (0, 0)$ is the best configuration. The initial policy parameters correspond to the black cross (lower right part of the figures), and the red circle the best parameter configuration. Subfigure (a) shows the gradient estimates in the case of a deterministic policy, while Subfigure (b) depicts the stochastic policy case. In both settings NOPG provides a good approximation of the gradient.

adding Gaussian noise to $\mathbf{K}_{\text{init}}$. This stochasticity is important for PWIS, which only admits such class of behavioural policies.
As expected from theory, in Figure 5a we can observe that DPG provides a biased estimate of the gradient. We can also state that because the datasets generated for this experiment used very similar policies, when DPG takes a gradient step that leads the policy to a region without samples it provides a bad estimate. Similarly in Figure 5b one can notice that PWIS provides a high-variance gradient estimate. On the contrary, in both experiments NOPG provides low-bias and low-variance estimates, showing that it can move away from the region of samples collected with the behavioural policy and converge to an approximate good solution.

5.2. Learning with a Uniform Dataset

In this experiment we test the ability of NOPG to learn with a carefully chosen dataset, composed by a uniform grid of states and actions sampled from the Pendulum-v0 environment of OpenAI Gym (referred just as Pendulum). This task consists on balancing a pendulum that is attached to a motor on one end, where a continuous torque $u \in [-2, 2]$ can be applied to make the pendulum rotate. A schematic of this environment is depicted in Figure 6a. The system's state is fully described by $\boldsymbol{s} = (\cos(\theta), \sin(\theta), \dot{\theta})$, where $\theta[\text{rad}] \in [-\pi, \pi]$ is defined as 0 when the pendulum is in the upright position, and $\dot{\theta}$ the angular velocity with $\dot{\theta} \in [-8, 8]$ rad/s. The reward per time-step is given by $R(\theta, \dot{\theta}, u) = -(\text{norm}(\theta)^2 + 10^{-1}\dot{\theta}^2 + 10^{-3}u^2)$, where $\text{norm}(\theta)$ places θ in the interval $[-\pi, \pi]$. The reward is maximal when $(\theta, \dot{\theta}) = (0, 0)$ and $u = 0$, i.e., the goal is to bring the pendulum to the desired state with minimal effort. In the original environment the pendulum can start from a random state. To ensure that we are in the worst possible starting configuration, in our experiments we evaluate the trained policies by setting the starting state to the bottom position with zero angular velocity, $\boldsymbol{s}_0 = (\cos(\pi), \sin(\pi), 0)$, which forces the pendulum to swing up to the top position. In the evaluation we run the optimized policy for 500 steps, where a cumulative reward of -500 is considered a satisfactory return.



(a) Pendulum **(b)** CartPole **(c)** MountainCar

Figure 6.: Environment schematics These figures depict the environments used in the experiments.

In Table 1 we report the performance of the discrete and stochastic versions of NOPG for different uniformly sampled dataset sizes. The configurations can be found in Table C1. From the table it is possible to notice that already with 450 samples both versions of NOPG can solve the task. When comparing with the results from random sampling from Figure 8, we verify that a uniform dataset is a better scenario for NOPG.

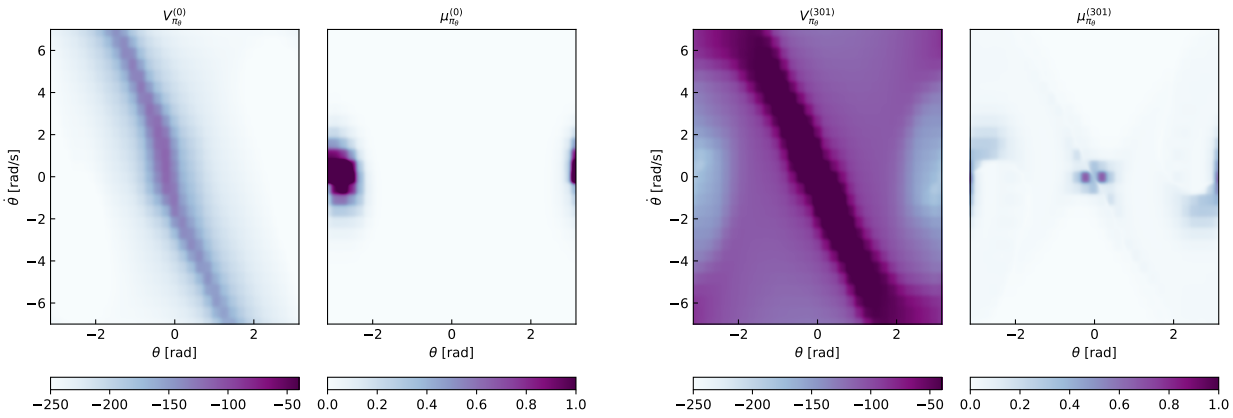
Sample size	NOPG-D	NOPG-S
200	-3794.9 ± 11.9	-3788.5 ± 00.9
450	-576.3 ± 03.1	-594.1 ± 39.8
800	-572.3 ± 02.3	-572.8 ± 03.9
1250	-567.9 ± 03.2	-564.2 ± 01.6
1800	-472.0 ± 28.8	-457.9 ± 26.0
3200	-415.4 ± 00.7	-428.0 ± 26.9

Table 1.: Returns in the Pendulum environment with uniform sampling This table presents the returns obtained by NOPG-D and NOPG-S as a function of the dataset size with samples from an uniform grid. Depicted are the mean and 95% confidence intervals of 10 trials. The datasets' details are found in Table C1.

As argued in Chapter 4, NOPG interpolates the value function and the state distribution of any state using the closed-form expression from Equation (39) and Equation (44), respectively. In Figure 7 we show the value function and state distribution estimates computed with NOPG-D before any policy update and after 301 iterations. It is possible to see that NOPG is able to find a reasonable approximation of the optimal value function and that it can predict that the goal state, located at $(\theta, \dot{\theta}) = (0, 0)$, will be reached, as depicted in the rightmost figure. The ability of NOPG to estimate the state distribution in closed-form is particularly interesting in robotic applications, because knowing the state distribution without needing to interact with the environment provides a safe way to determine if the robot is moving towards a dangerous region of the state space.

5.3. Learning with a Random Behavioural Policy

In contrast to the uniform grid experiment, here we show how NOPG behaves in the presence of a dataset sampled with a random behaviour policy in the Pendulum and CartPole environments.
The CartPole stabilization is a classical control task consisting on a pole attached to an actuated cart that can move horizontally. Here we use the CartPole version from Quanser Control Systems [20] with an OpenAI Gym interface. The goal is to keep the pole in the upright position and prevent it from



(a) Before any policy update

(b) After 301 policy updates

Figure 7.: Phase portrait of the value function and state distribution for the Pendulum environment The plots show the value function and state distribution estimates in the Pendulum environment using NOPG-D, with a uniformly sampled data. The two leftmost plots show the estimates before any policy update and the rightmost after 301 updates. In this experiment, data was collected uniformly from a grid of state-action pairs (30 angles, 30 angular velocities, 2 actions) and NOPG-D trained with the details presented in Table C2.

falling by moving the cart with a motor controlled with a continuous action $u \in [-24, 24]$ Volts. A figure of this environment is found in Figure 6b. The state is fully described as $\boldsymbol{s} = (x, \sin(\theta), \cos(\theta), \dot{x}, \dot{\theta})$, where x is the cart's relative position to the center, $\dot{x}$ the cart's velocity, θ is the angle the pole makes with the vertical axis, defined as 0 in the upright position and positive in the clockwise direction, and $\dot{\theta}$ is the angular velocity. The reward per time-step is defined as $R(\theta) = \cos\theta$, which is maximal when the pole is in the upright position. The starting state is $\boldsymbol{s}_0 = (x = 0, \sin(\theta), \cos(\theta), \dot{x} = 0, \dot{\theta} = 0)$, with θ sampled from a uniform distribution $\theta \sim \mathcal{U}[-10^{-2}\pi, 10^{-2}\pi]$, i.e., with the pole close to the top position. The episode terminates if the absolute value of θ is greater than 0.25 radians or the pre-specified number of steps is reached. The original environment also stops if the cart exceeds some distance to the center. As we are just interested in stabilizing the pole we removed this constraint leading to a smaller state space $\boldsymbol{s} = (\theta, \dot{x}, \dot{\theta})$. We want to refer that we can also keep the cart in the center at $x = 0$, by introducing a penalty in the reward function proportional to the absolute distance from the center, at the cost of more samples. We evaluate the policies with 10^4 steps. Given the reward function, if the return is less than 10^4 it indicates the number of steps taken before the pole falls.

To generate random trajectories for the Pendulum environment we start in the upright position and apply actions sampled from a mixture of two Gaussians, centered in -2 and 2, each with 0.75 standard deviation. If the desired trajectory size is more than 500 transitions, the system is reset to the initial position and the sampling process repeats. This configuration ensures that the desired goal state is seen at least once during sampling. For all algorithms we ensure the same configuration.

In Figure 8 we show the results of the policy evaluation returns of the different algorithms in the Pendulum environment. Notice that the horizontal axis is in logarithmic scale. It is possible to observe that both versions of NOPG need less samples (about one order of magnitude) to achieve the same results as other state-of-the-art algorithms, hence supporting the claim for a more sample-efficient off-policy algorithm. NOPG-D performs worse than the stochastic version because during the optimization procedure the policy might jump to a state-action space location with few samples from where the gradient is close to zero and thus from where it cannot escape. With NOPG-S this issue is solved with the inherent stochasticity of the policy. Moreover, it is interesting to note that for the same number of policy updates as NOPG, the offline version of DDPG is in line with the findings from Fujimoto et. al [17], i.e., it cannot cope with truly off-policy samples.

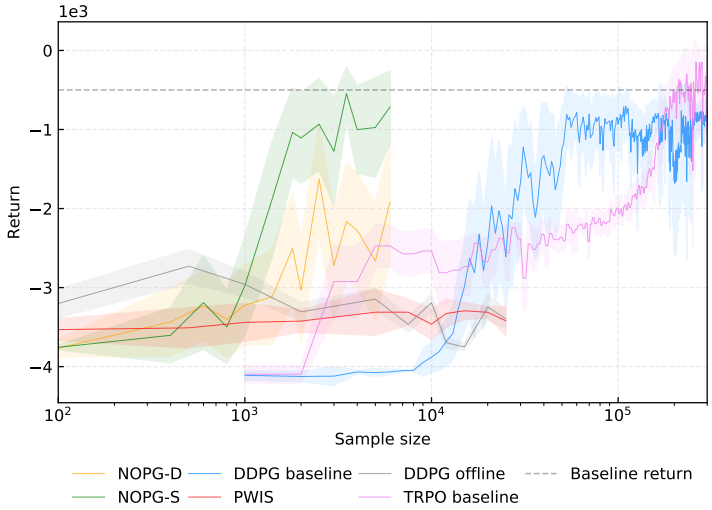


Figure 8.: Returns in the Pendulum environment with randomly sampled data This plot compares the return obtained by NOPG and typical on- and off-policy algorithms in the Pendulum task, as a function of the number of collected samples. NOPG's performance surpasses other algorithms by at least an order of magnitude. The TRPO and DDPG curves start at 10^3 samples because to accelerate training we only evaluate these algorithms after collecting every other 10^3 samples. The solid line represents the mean and the shaded area the 95% confidence interval of 10 runs. The details used in this experiment are presented in Table C3.

Similar to the Pendulum experiment we run a task with the Cartpole environment using random data. Sample collection is done by repeatedly placing the pole in the starting state and applying a random policy that samples from a uniform distribution between -5 and 5. In Figure 9 we plot the results. We can see that eventually TRPO and DDPG learn an optimal policy after collecting around $11 \cdot 10^3$ samples, while the offline version of DDPG and PWIS are not able to learn, similar to the Pendulum task. Again, NOPG shows that it is capable of learning in a total off-policy setting, using an order of magnitude less samples than state-of-the-art algorithms. One could argue that in every iteration NOPG uses the full dataset while other algorithms, such as DDPG, sample a batch from a replay buffer. While this observation is certainly valid, in this particular experiment DDPG runs 50 policy updates after collecting a single new transition by sampling 64 transitions from the current buffer, meaning that for a buffer with 100 samples it almost surely encounters all transitions at least once.
We further applied a policy learned in simulation in the real CartPole system. Although this experiment does not prove the capability of NOPG to learn directly on the real system, it shows that the policy learned can cope with the small intricacies not modelled in simulation, such as the friction forces action on the cart's wheels. These results can be explained because the kernel assumes a noisy environment, and thus the policy is potentially more robust. In Figure 10 we show a snapshot series in the real environment. We want to stress that we tried to learn a policy directly with data collected from the real CartPole system, but unfortunately we were not successful. Many factors could have contributed to this inability, such as the fact that the pole has to be precisely standing still or that the wheels' friction results in very noisy measurements.

5.4. Learning from Suboptimal trajectories

We provide an experiment to show that NOPG finds a policy that improves on suboptimal trajectories given by a human demonstrator. In this setting the data provided is completely offline and off-policy, which mimics one way to produce training data in robotics. It is also worth noting that such setting

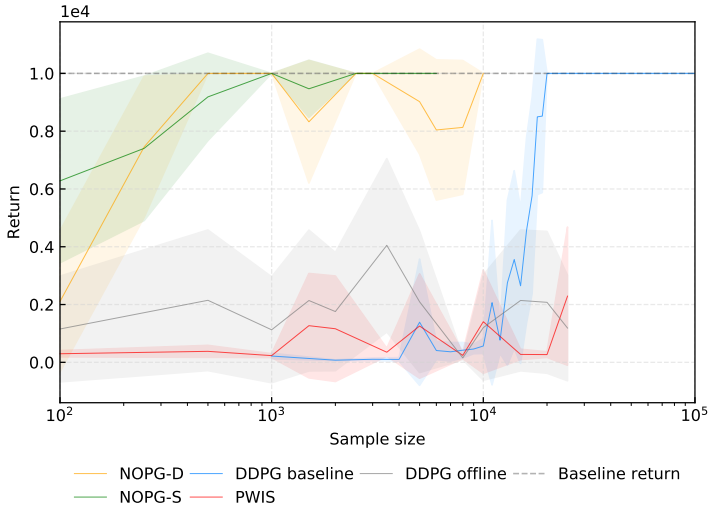


Figure 9.: Returns in the Cartpole stabilization environment with randomly sampled data This plot compares the return obtained by NOPG and typical on- and off-policy algorithms in the CartPole task, as a function of the number of collected samples. NOPG's performance surpasses other algorithms by at least an order of magnitude. The TRPO and DDPG curves start at $5 \cdot 10^2$ samples because to accelerate training we only evaluate these algorithms after collecting every other $5 \cdot 10^2$ samples. The solid line represents the mean and the shaded area the 95% confidence interval of 10 runs. The details used in this experiment are presented in Table C4.

is different from pure imitation learning. Observe that PWIS algorithms cannot learn directly from human demonstrations, because they require the explicit probability distribution of the behaviour policy, contrary to NOPG.

For this task we use the 2-dimensional ContinuousMountainCar-v0 environment of OpenAI Gym (we refer to it as MountainCar), which is an adapted version of the original task proposed in [71], where a car is placed still in a valley between two mountains and on the top of the mountain to the right there is a flag. The schematic of this environment is depicted in Figure 6c. The systems' state is fully described by $\boldsymbol{s} = (x, \dot{x})$, where $x \in [-1.2, 0.6]$ is the relative position of the car, $\dot{x}$ its velocity and the maximum absolute velocity is 0.07. The flag is placed at $x = 0.45$. The car moves with an external force represented by a continuous action $u \in [-1, 1]$. The starting state is $\boldsymbol{s}_0 = (x_0, 0)$, where $x_0 \sim \mathcal{U}[-0.6, -0.4]$, i.e. the car starts in the valley fully stopped. We use a modified reward function as $R = -1$ for each time-step. The task is to bring the car to the flag in the least steps as possible. Therefore, a return tells how many steps it takes for the car to reach the flag. The sole acceleration in the positive x direction does not work. Instead, the car needs to build momentum by first moving away from the goal. To evaluate a policy we set a maximum episode length of 500 steps.

We provide NOPG with an offline dataset generated by a human demonstrator. This collection is composed of 10 deliberately suboptimal trajectories of maximum length equal to 500. In all trajectories the car reaches the goal, with the exception of 2 of them. At each step the human chooses whether to accelerate in the positive ($u = 1$) or negative ($u = -1$) direction or to not accelerate at all ($u = 0$). To mimic the scenario of a continuous behavioural policy we add Gaussian noise $\mathcal{N}(0, 0.25)$ to each action. The mean trajectory as approximately 434 steps (return of -434) with negligible variance, which indicates that they are definitely suboptimal, since it is possible to solve the task with nearly 100 steps.

Figure 11 shows the performance obtained by NOPG with different numbers of trajectories. One can also think of the horizontal axis as the number of samples by roughly assigning 450 samples to each trajectory. Hence, from the figure we conclude that both versions of NOPG can learn an optimal policy

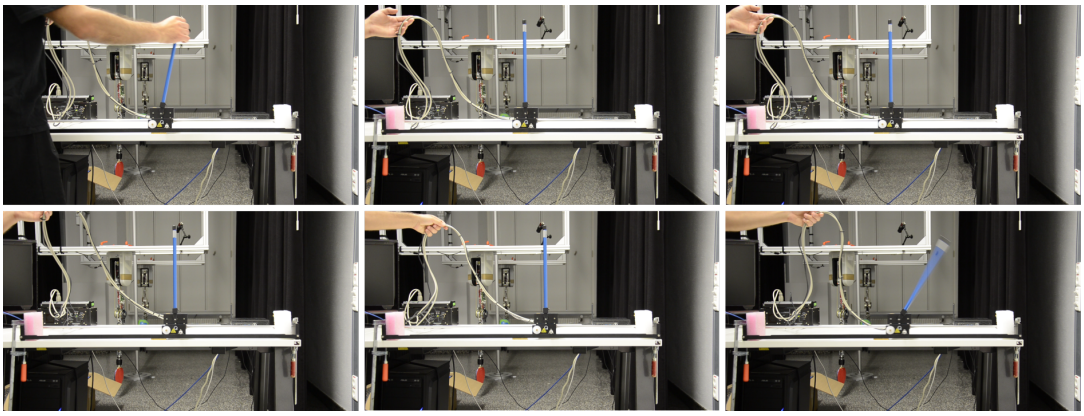


Figure 10.: Evaluation of a policy optimized with NOPG-S in the real CartPole system This stream of figures (left to right, top to bottom ordering) shows the application of a policy learned in simulation with NOPG-S in the real CartPole environment. The policy was trained using 1500 data points from trajectories with actions randomly sampled from a uniform distribution. Although a compliant policy can be learned with less samples, we found its actions alternated between very low and high signals, leading to very rough movements. We noticed that using more samples solved this problem.

already using a minimum of 2 trajectories (roughly 1000 samples). This experiment shows that NOPG is indeed finding a policy that is optimal and not just performing pure imitation learning with the off-policy data.

Like in the Pendulum experiment, in Figure 12 we can also look at the phase portrait of the state space before and after any policy update. With an initial randomly initialized policy it is possible to observe in the right side of Figure 12a that the state distribution is very concentrated in the valley region, from where the car cannot escape. Also, the value function before any policy update is assigning more credit to the positions that are to the right of the starting position (on average -0.5) and have a positive velocity. After some updates, as depicted in Figure 12b, the policy found translates into a value function that is not completely white near $(x, \dot{x}) = (-0.5, 0)$, meaning that the car might be able to escape the valley region. Also, if the car is at the leftmost position and with high velocity in the positive direction, the value function is already closer to what we would expect from an optimal policy. Since the episode terminates when the car crosses the goal position, there is no concept of a stationary state distribution, making it more difficult to analyse the rightmost figure. Nevertheless, it is possible to observe that the policy manages to move the car out of the valley region, visible by the nonzero probability on the area near the goal state as indicated by the state distribution.

In Figure 13 we show two trajectories of evaluation runs of NOPG-D in the MountainCar environment. It is easy to identify that Trajectory 1 needed less steps to reach the goal because it started on the right side of the valley at $x > -0.5$ ($x = -0.5$ is the valley's flat region), which already has some inclination. This starting position allows the car to gain more momentum when accelerating backwards, and it just needs to swing backwards once. In contrast, in trajectory 2 the car started on the left side of the valley ($x < -0.5$), which has an opposite inclination, and hence it needed to swing two times to gain enough momentum.

As we have seen, it is not an easy task to actually quantify the similarity between MDP instances and there exist a lot of

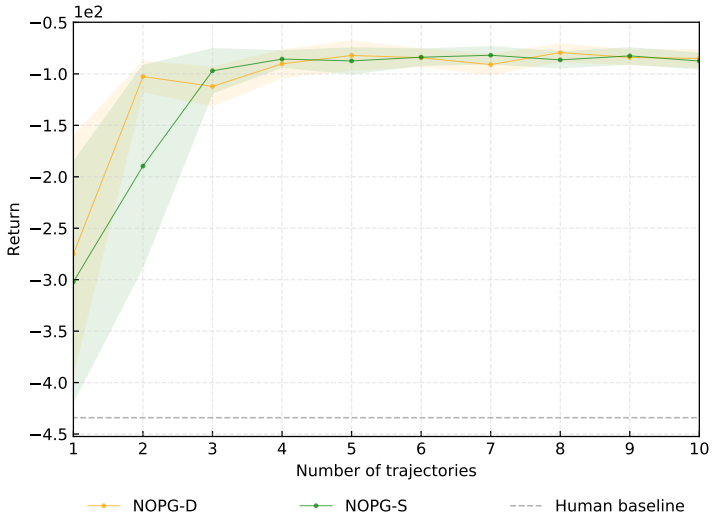
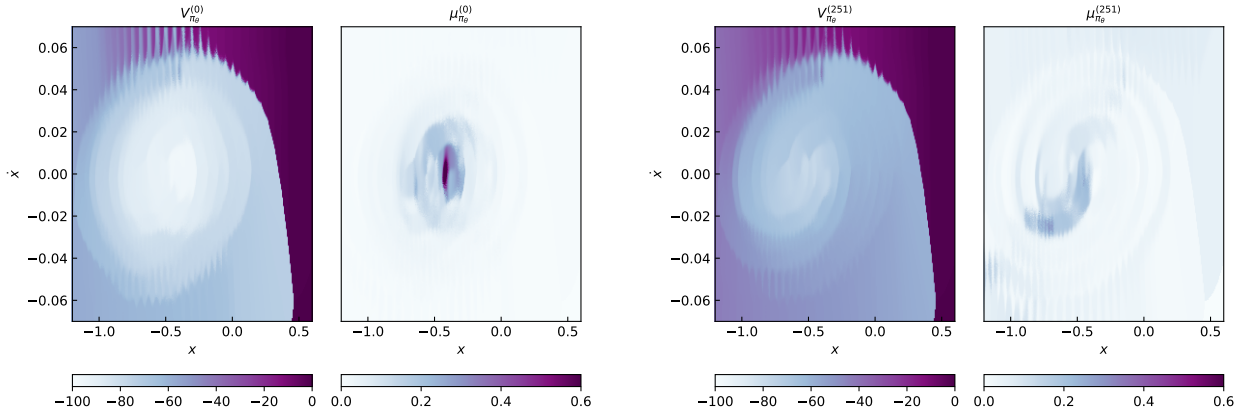


Figure 11.: Returns in the MountainCar environment with varying number of trajectories This plot shows the returns obtained by NOPG-D and NOPG-S as a function of the number of trajectories (roughly each trajectory corresponds to 450 samples). The human baseline corresponds to the average return in the human demonstrations and is depicted by the dashed line. The solid line represents the mean and the shaded area the 95% confidence interval of 10 runs. The details used in this experiment are presented in Table C5.



(a) Before any policy update

(b) After 251 policy updates

Figure 12.: Phase portrait of the value function and state distribution for the MountainCar experiment The plots show the value function and state distribution estimates in the MountainCar environment using NOPG-D. The two leftmost plots show the estimates before any policy update and the rightmost after 251 updates. NOPG-D was trained with 4 trajectories provided by a human demonstrator with the details presented in Table C5.

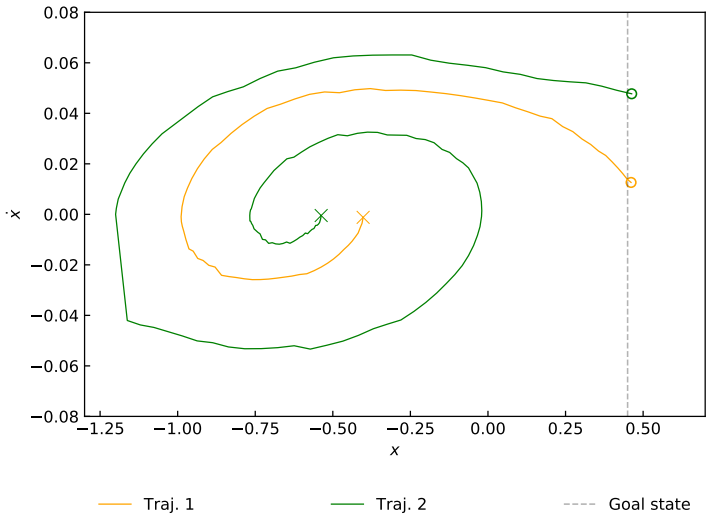


Figure 13.: Trajectories in the state space of the MountainCar environment This figure shows two trajectories from evaluation runs of NOPG-D in the MountainCar environment. Trajectory's 1 return is -79 and trajectory's 2 is -139. The crosses denote the starting states and the circles the final states. The dashed vertical line at $x = -0.45$ depicts the goal state. The policy used was optimized with NOPG-D trained on 4 trajectories provided by a human demonstrator with the details presented in Table C5.

6. Conclusion and Future Research

In this thesis we delved into Policy Gradient estimators for off-policy datasets. In RL tasks, searching directly in the space of parameterized policies is often a better approach than estimating first a value function and extracting a policy thereafter, specially for large and continuous state and action spaces. Additionally, learning with samples collected off-policy seems to be a promising path for robotic applications, due to the promise of higher sample-efficiency and the ability to learn from multiple different policies, for instance through demonstrations.

The Policy Gradient Theorem [35] provides a solid foundation to perform a gradient step to update the policy parameters by letting an agent interact directly with the environment. The crux is that every policy update requires usually more than one trajectory rollout, which hinders its direct application to real world systems. For the reasons explained above, a much more desired approach is to use off-policy datasets. However, the Policy Gradient Theorem fails in this setting because an expectation under the state distribution induced by the policy can no longer be directly computed. In Chapter 3 we presented the current state-of-the-art methods to compute a policy gradient in an off-policy manner, including Importance Sampling, the Off-Policy Policy Gradient Theorem and Model-based RL solutions. IS methods show large variance estimates, only support the class of stochastic policies and tend to only work well when the behavioural and target distributions are close. The Off-Policy Policy Gradient Theorem defines an objective function that in our opinion is not very well understood, and drops a term in the gradient derivation, which is guaranteed to improve the policy only for tabular MDPs. These sources of approximations made by these algorithms and its successors are solved in practice by ensuring the joint distribution of state-action pairs induced by the behavioural and target policies are somewhat close. Therefore, we argue that these algorithms cannot succeed in truly offline and off-policy configurations, as also noticed in other works [17].

The main finding of this work is a novel way to compute a policy gradient for deterministic and stochastic policies in closed-form from an offline dataset with transitions collected using an unspecified behavioural policy. Following the work of Kroemer et. al [18], we started by modelling the transition probability density function as a density estimation problem solved with kernel density estimation, and the reward function approximated with the Nadaraya-Watson kernel regression. With these two constructs we employed a Galerkin Projection to solve the infinite value function constraints, defined over a continuous state space, on a subspace spanned by a set of basis functions based on the samples collected in an off-policy dataset. We obtained a nonparametric value function with a direct dependence on the policy parameters. With this formulation we could compute a policy gradient in closed-form, such that a policy update requires no further interaction with the environment, opposite to the on-policy version of the Policy Gradient Theorem. Another important finding of this work is that with this kind of nonparametric modelling, we are able to compute the state distribution under a policy for the whole state space. Estimating this quantity is of special importance in robotics to prevent the a priori exploration of dangerous areas of the state space. Based on these theoretical ideas we provided a working algorithm which we name as the Nonparametric Off-Policy Policy Gradient (NOPG) algorithm. In Chapter 5 we tested how NOPG compares to other state-of-the-art on- and off-policy learning methods. We empirically showed with a simple 2-dimensional LQR system that the gradient of NOPG converges to true gradient, while IS and DPG show a large variance and biased estimates, respectively. In another experiment we trained a policy parameterized by a small neural network with NOPG using different dataset configurations. We attested that with an uniformly sampled grid of state-action pairs NOPG can learn the Pendulum task with a minimum of 450 samples. Additionally, we set an experiment to show that NOPG can also learn with off-policy data collected with a random behavioural policy as demonstrated with the Pendulum and CartPole examples. The results exhibited that other algorithms

failed to learn or that their sample complexity was at least one order of magnitude larger. Finally, we tested NOPG with a human provided dataset of suboptimal trajectories to solve the MountainCar task and showed that NOPG is able to surpass the human baseline and find an optimized policy.
In conclusion, in this work we showed to be possible to learn controllers with offline and off-policy data provided with random behavioural policies in a sample efficient way, emphasizing the potential to use NOPG in scarce sample regimes such as real robot applications.

Future Research

The main downside of the NOPG algorithm is also its strength, namely the number of available samples. In its current form, every policy update has at least quadratic computational and linear memory complexities in the dataset size. Additionally, nonparametric methods suffer from the curse of dimensionality and they do not work properly in high-dimensional spaces. Further research could begin by expressing a transition model using density mixture models (for instance Gaussian mixtures) and the reward function with mixture of experts models. This formulation could potentially reduce the number of parameters and be able to scale to more data points. More importantly, one could possibly obtain an expression for the value function that is no longer a single scalar but rather a distribution, which would enable guiding the exploration to areas of the state space with higher uncertainty.

Bibliography

[1] C. Szepesvari, *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[2] C. E. Shannon, *Programming a Computer for Playing Chess*, pp. 2–13. New York, NY: Springer New York, 1988.

[3] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," *Machine Learning*, vol. 22, pp. 33–57, Mar 1996.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] OpenAI, "Openai five." `https://blog.openai.com/openai-five/`, 2018.

[8] J. Peters, J. Kober, K. Mülling, O. Kroemer, and G. Neumann, "Towards robot skill learning: From simple skills to table tennis," in *Machine Learning and Knowledge Discovery in Databases, Proceedings of the European Conference on Machine Learning, Part III (ECML 2013)*, vol. LNCS 8190, pp. 627–631, Springer, 2013.

[9] M. P. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, (USA), pp. 465–472, Omnipress, 2011.

[10] N. M. O. Heess, T. Dhruva, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *ArXiv*, vol. abs/1707.02286, 2017.

[11] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, vol. abs/1808.00177, 2018.

[12] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *CoRR*, vol. abs/1812.11103, 2018.

[13] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, pp. 1334–1373, Jan. 2016.

[14] S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[15] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," *CoRR*, vol. abs/1205.4839, 2012.

[16] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. I–387–I–395, JMLR.org, 2014.

[17] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," *CoRR*, vol. abs/1812.02900, 2018.

[18] O. B. Kroemer and J. R. Peters, "A non-parametric approach to dynamic programming," in *Advances in Neural Information Processing Systems 24* (J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, eds.), pp. 1719–1727, Curran Associates, Inc., 2011.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016.

[20] "Quanser - control systems lab solutions." `https://www.quanser.com/solution/control-systems/`. Accessed: 2019-06-08.

[21] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[22] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Athena Scientific, first ed., 2019.

[23] D. Kirk, *Optimal Control Theory: An Introduction*. Dover Books on Electrical Engineering Series, Dover Publications, 2004.

[24] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 1994.

[25] C. Daniel, O. Kroemer, M. Viering, J. Metz, and J. Peters, "Active reward learning with a novel acquisition function," *Auton. Robots*, vol. 39, pp. 389–405, Oct. 2015.

[26] T. Wang, D. Lizotte, M. Bowling, and D. Schuurmans, "Dual representations for dynamic programming," *Journal of Machine Learning Research*, vol. 1, pp. 1–29, 01 2008.

[27] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *CoRR*, vol. abs/1708.05866, 2017.

[28] R. Bellman, "The theory of dynamic programming," *Bull. Amer. Math. Soc.*, vol. 60, pp. 503–515, 11 1954.

[29] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning, Cambridge, MA, USA: MIT Press, Jan. 2006.

[30] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1st ed., 1996.

[31] C. J. C. H. Watkins and P. Dayan, "Q-learning," in *Machine Learning*, pp. 279–292, 1992.

[32] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Found. Trends Robot*, vol. 2, pp. 1–142, Aug. 2013.

[33] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, (New York, NY, USA), pp. 1061–1068, ACM, 2013.

[34] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems 12* (S. A. Solla, T. K. Leen, and K. Müller, eds.), pp. 1008–1014, MIT Press, 2000.

[35] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, (Cambridge, MA, USA), pp. 1057–1063, MIT Press, 1999.

[36] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[37] J. Peters, K. Mülling, and Y. Altün, "Relative entropy policy search," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pp. 1607–1612, AAAI Press, 2010.

[38] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1889–1897, PMLR, 07–09 Jul 2015.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[40] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–1–III–9, JMLR.org, 2013.

[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[42] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Tech. Rep. TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

[43] N. Meuleau, L. Peshkin, and K.-E. Kim, "Exploration in gradient-based reinforcement learning," Tech. Rep. 2001-003, MIT, 2001.

[44] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[45] L. Devroye and L. Gyorfi, *Nonparametric density estimation: the L1 view.* New York; Chichester: John Wiley and Sons, 1985.

[46] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics).* Berlin, Heidelberg: Springer-Verlag, 2006.

[47] B. W. Silverman, *Density estimation for statistics and data analysis.* Monographs on Statistics and Applied Probability, London: Chapman and Hall, 1986.

[48] J. M. Leiva-Murillo and A. Artés-Rodríguez, "Algorithms for maximum-likelihood bandwidth selection in kernel density estimators," *Pattern Recognition Letters*, vol. 33, pp. 1717–1724, 2012.

[49] C. J. Stone, "An asymptotically optimal window selection rule for kernel density estimates," *Ann. Statist.*, vol. 12, pp. 1285–1297, 12 1984.

[50] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, pp. 78–87, Oct. 2012.

[51] E. A. Nadaraya, "On estimating regression," *Theory of Probability and its Applications*, vol. 9, pp. 141–142, 1964.

[52] G. S. Watson, "Smooth regression analysis," *Sankhy: The Indian Journal of Statistics, Series A*, vol. 26, pp. 359–372, 01 1964.

[53] K. E. Atkinson, *The Numerical Solution of Integral Equations of the Second Kind*. Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 1997.

[54] J. Baxter and P. L. Bartlett, "Direct gradient-based reinforcement learning," *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, vol. 3, pp. 271–274 vol.3, 2000.

[55] E. Imani, E. Graves, and M. White, "An off-policy policy gradient theorem using emphatic weightings," in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 96–106, Curran Associates, Inc., 2018.

[56] X. Wang and T. G. Dietterich, "Model-based policy gradient reinforcement learning," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 776–783, AAAI Press, 2003.

[57] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control.," in *IROS*, pp. 5026–5033, IEEE, 2012.

[58] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," *CoRR*, vol. abs/1609.05143, 2016.

[59] J. G. Schneider, "Exploiting model uncertainty estimates for safe dynamic control learning," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, (Cambridge, MA, USA), pp. 1047–1053, MIT Press, 1996.

[60] G. B. Thomas, M. D. Weir, and R. L. Finney, *Calculus and analytic geometry*. Addison-Wesley, 9th ed., 1996.

[61] S. U. Pillai, T. Suel, and S. Cha, "The Perron-Frobenius theorem: Some of its applications," *Signal Processing Magazine, IEEE*, vol. 22, pp. 62–75, March 2005.

[62] L. Wasserman, *All of Nonparametric Statistics (Springer Texts in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[63] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2005.

[64] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[65] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.

[66] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[67] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[68] G. Piazza and T. Politi, "An upper bound for the condition number of a matrix in spectral norm," *Journal of Computational and Applied Mathematics*, vol. 143, no. 1, pp. 141 – 144, 2002.

[69] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[70] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines." `https://github.com/openai/baselines`, 2017.

[71] A. W. Moore, "Efficient memory-based learning for robot control," tech. rep., University of Cambridge, 1990.

A. Proof of the Normalized State Distribution

Consider the discounted expected state visitation following a policy π

$$\mu_\pi(\boldsymbol{s}) = \mu_0(\boldsymbol{s}) + \gamma \int_{\mathcal{S}} \int_{\mathcal{A}} \mu_\pi(\boldsymbol{s}')\pi(\boldsymbol{a} \mid \boldsymbol{s}')P(\boldsymbol{s} \mid \boldsymbol{s}', \boldsymbol{a}) \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{s}' \quad \forall \boldsymbol{s} \in \mathcal{S}.$$

We prove that the normalized discounted state distribution is given by $(1-\gamma)\mu_\pi(\boldsymbol{s})$.
Assume an indicator variable defined as

$$I_t^\pi(\boldsymbol{s}) \coloneqq \begin{cases} 1 & , \text{ if } \boldsymbol{s}_t = \boldsymbol{s} \\ 0 & , \text{ if } \boldsymbol{s}_t \neq \boldsymbol{s} \end{cases},$$

where $\boldsymbol{s}_t$ is the state visited at time t, when following the policy π. Therefore, the discounted state visitation is simply given by $\mu_\pi(\boldsymbol{s}) = \sum_{t=0}^{\infty} \gamma^t I_t^\pi(\boldsymbol{s})$. First we prove that $0 \leq (1-\gamma)\mu_\pi(\boldsymbol{s}) \leq 1 \; \forall \boldsymbol{s} \in \mathcal{S}, \; 0 \leq \gamma < 1$

$$\begin{aligned}
\mu_\pi(\boldsymbol{s}) &= \sum_{t=0}^{\infty} \gamma^t I_t^\pi(\boldsymbol{s}) \\
&\leq \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma} \quad \text{if } \boldsymbol{s}_t = \boldsymbol{s} \; \forall t, \\
\mu_\pi(\boldsymbol{s}) &= \sum_{t=0}^{\infty} \gamma^t I_t^\pi(\boldsymbol{s}) \\
&\geq \sum_{t=0}^{\infty} \gamma^t \cdot 0 = 0 \quad \text{if } \boldsymbol{s}_t \neq \boldsymbol{s} \; \forall t, \\
\implies 0 \leq \mu_\pi(\boldsymbol{s}) &\leq \frac{1}{1-\gamma} \Leftrightarrow \; 0 \leq (1-\gamma)\mu_\pi(\boldsymbol{s}) \leq 1.
\end{aligned}$$

Then we prove that $(1-\gamma)\mu_\pi(\boldsymbol{s})$ is normalized, i.e., $\sum_{\boldsymbol{s}\in\mathcal{S}}(1-\gamma)\mu_\pi(\boldsymbol{s}) = 1$

$$\begin{aligned}
&\sum_{\boldsymbol{s}\in\mathcal{S}} (1-\gamma)\mu_\pi(\boldsymbol{s}) \\
=&\sum_{\boldsymbol{s}\in\mathcal{S}} (1-\gamma) \sum_{t=0}^{\infty} \gamma^t I_t^\pi(\boldsymbol{s}) \\
=&\sum_{\boldsymbol{s}\in\mathcal{S}} \sum_{t=0}^{\infty} (\gamma^t - \gamma^{t+1}) I_t^\pi(\boldsymbol{s}) \\
&\text{Since only one state is visited at every time step } t, \text{ i.e.,} \\
&\text{if } I_t^\pi(\boldsymbol{s}_i) = 1 \text{ then } I_t^\pi(\boldsymbol{s}_j) = 0 \; \forall i \neq j \\
=&\sum_{t=0}^{\infty} (\gamma^t - \gamma^{t+1}) \\
=&\gamma^0 \underbrace{-\gamma^1 + \gamma^1 - \gamma^2 + \gamma^2 - \gamma^3 + \gamma^3 - \ldots}_{=0} - \gamma^\infty \\
=&\gamma^0 - \gamma^\infty = 1 - 0 = 1 \quad \text{for } 0 \leq \gamma < 1.
\end{aligned}$$

B. Gradient Estimates with LQR, NOPG, DPG and PWIS

Here we detail the experiment presented in Section 5.1. We use a discrete infinite-horizon discounted Linear Quadratic Regulator system of the form

$$\begin{aligned}\max_{\boldsymbol{x}_t,\boldsymbol{u}_t} J = \frac{1}{2}\sum_{t=0}^{\infty}\gamma^t\left(\boldsymbol{x}_t^\top\mathbf{Q}\boldsymbol{x}_t + \boldsymbol{u}_t^\top\mathbf{R}\boldsymbol{u}_t\right)\\ \text{s.t. } \boldsymbol{x}_{t+1} = \mathbf{A}\boldsymbol{x}_t + \mathbf{B}\boldsymbol{u}_t \quad \forall t,\end{aligned}$$

where $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$, $\boldsymbol{u}_t \in \mathbb{R}^{d_u}$, $\mathbf{Q} \in \mathbb{R}^{d_x\times d_x}$, $\mathbf{R} \in \mathbb{R}^{d_u\times d_u}$, $\mathbf{A} \in \mathbb{R}^{d_x\times d_x}$, $\mathbf{B} \in \mathbb{R}^{d_x\times d_u}$, $\gamma \in [0,1)$ and $\boldsymbol{x}_0$ given. In this example we use consider a 2-dimensional problem with the following quantities

$$\begin{aligned}\mathbf{A} &= \begin{bmatrix} 1.2 & 0 \\ 0 & 1.1 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.2 \end{bmatrix}\\ \mathbf{Q} &= \begin{bmatrix} -0.5 & 0 \\ 0 & -0.25 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}\\ \boldsymbol{x}_0 &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \gamma = 0.9.\end{aligned}$$

For this LQR problem we impose a linear controller as a diagonal matrix

$$\mathbf{K} = \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix}. \tag{54}$$

Deterministic Experiment (Figure 5a)
For each dataset we run 100 trajectories of 30 steps. Each trajectory is generated by following the dynamics of the described LQR and using at each time step a fixed policy initialized as

$$\mathbf{K} = \begin{bmatrix} k_1+\varepsilon & \varepsilon \\ \varepsilon & k_2+\varepsilon \end{bmatrix},\ \varepsilon \sim \mathcal{N}(0,1),$$

where $k_1 = 0.7$ and $k_2 = -0.7$.
NOPG-D optimized for each dataset a policy encoded as in Equation (54) with: learning rate 0.5 with ADAM optimizer; bandwidths (on average) for the state space $\boldsymbol{h}_\psi = [0.03, 0.05]$ and for the action space $\boldsymbol{h}_\varphi = [0.33, 0.27]$; discount factor $\gamma = 0.9$; and keeping 5 elements per row after sparsification of the $\mathbf{P}$ matrix.
DPG optimized for each dataset a policy encoded as in Equation (54) with: learning rate 0.5 with ADAM optimizer; Q-function encoded as $Q(\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{x}^\top\mathbf{Q}\boldsymbol{x} + \boldsymbol{u}^\top\mathbf{R}\boldsymbol{u}$ (with $\mathbf{Q}$ and $\mathbf{R}$ to be learned); discount factor $\gamma = 0.9$; two target networks are kept to stabilize learning and soft-updated using $\tau = 0.01$ (similar to DDPG).

Stochastic Experiment (Figure 5b)
For each dataset we run 100 trajectories of 30 steps. Each trajectory is generated by following the

dynamics of the described LQR, and using at each time step a stochastic policy as

$$\boldsymbol{u}_t = \mathbf{K}\boldsymbol{x}_t + \boldsymbol{\varepsilon},\ \boldsymbol{\varepsilon} \sim \mathcal{N}\left(\boldsymbol{\mu} = \mathbf{0}, \boldsymbol{\Sigma} = \mathrm{diag}(0.01, 0.01)\right), \tag{55}$$

where $\mathbf{K} = \mathrm{diag}(0.35, -0.35)$.
NOPG-S optimized for each dataset a policy encoded as in Equation (55) with: learning rate 0.25 with ADAM optimizer; bandwidths (on average) for the state space $\boldsymbol{h}_\psi = [0.008, 0.003]$ and for the action space $\boldsymbol{h}_\varphi = [0.02, 0.02]$; discount factor $\gamma = 0.9$; and keeping 10 elements per row after sparsification of the $\mathbf{P}$ matrix.
PWIS optimized for each dataset a policy encoded as in Equation (55) with: learning rate 2.5×10^{-4} with ADAM optimizer; and discount factor $\gamma = 0.9$.

C. Experiments Configurations

In this section we detail in each table the configurations and hyperparameters used in the quantitative experiments of Chapter 5. Every table entry should be self-explanatory, but we want to point one that might not be straightforward. We use a policy encoded as neural network with parameters $\boldsymbol{\theta}$. A deterministic policy can be viewed as a single action $\boldsymbol{a} = f_{\boldsymbol{\theta}}(\boldsymbol{s})$, where $f_{\boldsymbol{\theta}}$ is the output of the learned neural network. A stochastic policy is encoded as a Gaussian distribution with parameters determined by a neural network with two outputs, the mean and covariance. In this case we represent by $f_{\boldsymbol{\theta}}(\boldsymbol{s})$ the slice of the output corresponding to the mean and by $g_{\boldsymbol{\theta}}(\boldsymbol{s})$ the part of the output corresponding to the covariance. In some algorithms we use a latent representation as part of the policy and value function networks, denoted by $\boldsymbol{z} = h_{\boldsymbol{\theta}}(\boldsymbol{s})$, which is then linearly transformed to give the Gaussian policy parameters or to compute the value function.

$\#\theta$	$\#\dot{\theta}$	$\#u$	Sample size
10	10	2	200
15	15	2	450
20	20	2	800
25	25	2	1250
30	30	2	1800
40	40	2	3200

Table C1.: Pendulum uniform grid dataset configurations This table shows the level of discretization for each dimension of the state space ($\#\theta$ and $\#\dot{\theta}$) and the action space ($\#u$). Each line corresponds to a uniformly sampled dataset, where $\theta \in [-\pi, \pi]$, $\dot{\theta} \in [-8, 8]$ and $u \in [-2, 2]$. The entries under the states' dimensions and action dimension correspond to how many linearly spaced states or actions are to be queried from the corresponding intervals. The Cartesian product of states and actions dimensions is taken in order to generate the state-action pairs to query the environment transitions. The rightmost column indicates the total number of corresponding samples.

NOPG	
discount factor γ	0.97
state $\boldsymbol{h}_{\text{factor}}$	1.0 1.0 1.0
action $\boldsymbol{h}_{\text{factor}}$	50.0
policy	neural network parameterized by $\boldsymbol{\theta}$
	1 hidden layer, 50 units, ReLU activations
policy output	$2\tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$ (NOGP-D)
	$\mu = 2\tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$, $\sigma = \text{sigmoid}(g_{\boldsymbol{\theta}}(\boldsymbol{s}))$ (NOGP-S)
learning rate	10^{-2} with ADAM optimizer
$N_{\mu_0}^{\text{MC}}$ (NOPG-S)	15
N_{ϕ}^{MC}	1
$N_{\mu_0}^{\text{MC}}$	(non applicable) fixed initial state
policy updates	$1.5 \cdot 10^3$

Table C2.: NOPG configurations for the Pendulum uniform grid experiment This table contains the configurations of NOPG-D and NOPG-S needed to replicate the results from the Pendulum uniform grid experiment (Table 1).

NOPG

dataset sizes	10^2, $5 \cdot 10^2$, 10^3, $1.5 \cdot 10^3$, $2 \cdot 10^3$, $3 \cdot 10^3$, $5 \cdot 10^3$, $7 \cdot 10^3$, $9 \cdot 10^3$, 10^4
discount factor γ	0.97
state $\boldsymbol{h}_{\text{factor}}$	10.0 10.0 1.0
action $\boldsymbol{h}_{\text{factor}}$	30.0
policy	neural network parameterized by $\boldsymbol{\theta}$ 1 hidden layer, 50 units, ReLU activations
policy output	$2\tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$ (NOGP-D) $\mu = 2\tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$, $\sigma = \text{sigmoid}(g_{\boldsymbol{\theta}}(\boldsymbol{s}))$ (NOGP-S)
learning rate	10^{-2} with ADAM optimizer
$N_{\mu_0}^{\text{MC}}$ (NOPG-S)	10
N_{ϕ}^{MC}	1
$N_{\mu_0}^{\text{MC}}$	(non applicable) fixed initial state
policy updates	$2 \cdot 10^3$

DDPG

discount factor γ	0.97
rollout steps	100
actor	neural network parameterized by $\boldsymbol{\theta}_{\text{actor}}$ 1 hidden layer, 50 units, ReLU activations
actor output	$2\tanh(f_{\boldsymbol{\theta}_{\text{actor}}}(\boldsymbol{s}))$
actor learning rate	10^{-3} with ADAM optimizer
critic	neural network parameterized by $\boldsymbol{\theta}_{\text{critic}}$ 1 hidden layer, 50 units, ReLU activations
critic output	$f_{\boldsymbol{\theta}_{\text{critic}}}(\boldsymbol{s}, \boldsymbol{a})$
critic learning rate	10^{-2} with ADAM optimizer
soft update	$\tau = 10^{-3}$
policy updates	$3 \cdot 10^5$

TRPO

discount factor γ	0.97
rollout steps	1000
policy	neural network parameterized by $\boldsymbol{\theta}$ 1 hidden layer, 50 units, ReLU activations latent representation $\boldsymbol{z}$
policy output	Gaussian distribution with mean and covariance retrieved from $\mathbf{W}_{\text{policy}}\boldsymbol{z} + \boldsymbol{b}_{\text{policy}}$
value function	has the same parameters $\boldsymbol{\theta}$ and latent variable $\boldsymbol{z}$ as the policy network
value function output	$\mathbf{W}_{\text{vf}}\boldsymbol{z} + \boldsymbol{b}_{\text{vf}}$
value function learning rate	$3 \cdot 10^{-4}$ with ADAM optimizer
value function iterations per epoch	3
max KL	10^{-3}
conjugate gradient iterations	10
conjugate gradient damping	10^{-2}
policy updates	$3 \cdot 10^5$

DDPG Offline	
dataset sizes	10^2, $5 \cdot 10^2$, 10^3, $2 \cdot 10^3$, $5 \cdot 10^3$, $7.5 \cdot 10^3$, 10^4, $1.2 \cdot 10^4$, $1.5 \cdot 10^4$, $2 \cdot 10^4$, $2.5 \cdot 10^4$
discount factor γ	0.97
actor	neural network parameterized by $\boldsymbol{\theta}_{\text{actor}}$ 1 hidden layer, 50 units, ReLU activations
actor output	$2\tanh(f_{\boldsymbol{\theta}_{\text{actor}}}(\boldsymbol{s}))$
actor learning rate	10^{-2} with ADAM optimizer
critic	neural network parameterized by $\boldsymbol{\theta}_{\text{critic}}$ 1 hidden layer, 50 units, ReLU activations
critic output	$f_{\boldsymbol{\theta}_{\text{critic}}}(\boldsymbol{s}, \boldsymbol{a})$
critic learning rate	10^{-2} with ADAM optimizer
soft update	$\tau = 10^{-3}$
policy updates	$2 \cdot 10^3$

PWIS	
dataset sizes	10^2, $5 \cdot 10^2$, 10^3, $2 \cdot 10^3$, $5 \cdot 10^3$, $7.5 \cdot 10^3$, 10^4, $1.2 \cdot 10^4$, $1.5 \cdot 10^4$, $2 \cdot 10^4$, $2.5 \cdot 10^4$
discount factor γ	0.97
policy	neural network parameterized by $\boldsymbol{\theta}$ 1 hidden layer, 50 units, ReLU activations
policy output	$\mu = 2\tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$, $\sigma = \text{sigmoid}(g_{\boldsymbol{\theta}}(\boldsymbol{s}))$
learning rate	10^{-2} with ADAM optimizer
policy updates	$2 \cdot 10^3$

Table C3.: Algorithms configurations for the Pendulum random data experiment These tables contain the configurations of the algorithms needed to replicate the results from the Pendulum random data experiment (Figure 8).

NOPG

dataset sizes	10^2, $2.5 \cdot 10^2$, $5 \cdot 10^2$, 10^3, $1.5 \cdot 10^3$, $2.5 \cdot 10^3$, $3 \cdot 10^3$, $5 \cdot 10^3$, $6 \cdot 10^3$, $8 \cdot 10^3$, 10^4
discount factor γ	0.99
state $\boldsymbol{h}_{\text{factor}}$	1.0 1.0 1.0
action $\boldsymbol{h}_{\text{factor}}$	20.0
policy	neural network parameterized by $\boldsymbol{\theta}$ 1 hidden layer, 50 units, ReLU activations
policy output	$5 \tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$ (NOGP-D) $\mu = 5 \tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$, $\sigma = \text{sigmoid}(g_{\boldsymbol{\theta}}(\boldsymbol{s}))$ (NOGP-S)
learning rate	$5 \cdot 10^{-2}$ with ADAM optimizer
$N_{\mu_0}^{\text{MC}}$ (NOPG-S)	15
N_{ϕ}^{MC}	1
$N_{\mu_0}^{\text{MC}}$	15
policy updates	$1.5 \cdot 10^3$

DDPG

discount factor γ	0.99
rollout steps	100
actor	neural network parameterized by $\boldsymbol{\theta}_{\text{actor}}$ 1 hidden layer, 50 units, ReLU activations
actor output	$5 \tanh(f_{\boldsymbol{\theta}_{\text{actor}}}(\boldsymbol{s}))$
actor learning rate	10^{-3} with ADAM optimizer
critic	neural network parameterized by $\boldsymbol{\theta}_{\text{critic}}$ 1 hidden layer, 50 units, ReLU activations
critic output	$f_{\boldsymbol{\theta}_{\text{critic}}}(\boldsymbol{s}, \boldsymbol{a})$
critic learning rate	10^{-2} with ADAM optimizer
soft update	$\tau = 10^{-3}$
policy updates	$2 \cdot 10^5$

TRPO

discount factor γ	0.99
rollout steps	500
policy	neural network parameterized by $\boldsymbol{\theta}$ 1 hidden layer, 50 units, ReLU activations latent representation $\boldsymbol{z}$
policy output	Gaussian distribution with mean and covariance retrieved from $\mathbf{W}_{\text{policy}}\boldsymbol{z} + \boldsymbol{b}_{\text{policy}}$
value function	has the same parameters $\boldsymbol{\theta}$ and latent variable $\boldsymbol{z}$ as the policy network
value function output	$\mathbf{W}_{\text{vf}}\boldsymbol{z} + \boldsymbol{b}_{\text{vf}}$
value function learning rate	$3 \cdot 10^{-4}$ with ADAM optimizer
value function iterations	3
max KL	10^{-3}
conjugate gradient iterations	10
conjugate gradient damping	10^{-2}
policy updates	$2 \cdot 10^5$

DDPG Offline	
dataset sizes	10^2, $5 \cdot 10^2$, 10^3, $2 \cdot 10^3$, $3.5 \cdot 10^3$, $5 \cdot 10^3$, $8 \cdot 10^3$, 10^4, $1.5 \cdot 10^4$, $2 \cdot 10^4$, $2.5 \cdot 10^4$
discount factor γ	0.99
actor	neural network parameterized by $\boldsymbol{\theta}_{\text{actor}}$ 1 hidden layer, 50 units, ReLU activations
actor output	$5 \tanh(f_{\boldsymbol{\theta}_{\text{actor}}}(\boldsymbol{s}))$
actor learning rate	10^{-2} with ADAM optimizer
critic	neural network parameterized by $\boldsymbol{\theta}_{\text{critic}}$ 1 hidden layer, 50 units, ReLU activations
critic output	$f_{\boldsymbol{\theta}_{\text{critic}}}(\boldsymbol{s}, \boldsymbol{a})$
critic learning rate	10^{-2} with ADAM optimizer
soft update	$\tau = 10^{-3}$
policy updates	$2 \cdot 10^3$

PWIS	
dataset sizes	10^2, $5 \cdot 10^2$, 10^3, $2 \cdot 10^3$, $3.5 \cdot 10^3$, $5 \cdot 10^3$, $8 \cdot 10^3$, 10^4, $1.5 \cdot 10^4$, $2 \cdot 10^4$, $2.5 \cdot 10^4$
discount factor γ	0.99
policy	neural network parameterized by $\boldsymbol{\theta}$ 1 hidden layer, 50 units, ReLU activations
policy output	$\mu = 5 \tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$, $\sigma = \text{sigmoid}(g_{\boldsymbol{\theta}}(\boldsymbol{s}))$
learning rate	10^{-3} with ADAM optimizer
policy updates	$2 \cdot 10^3$

Table C4.: Algorithms configurations for the CartPole random data experiment These tables contain the configurations of the algorithms needed to replicate the results from the CartPole random data experiment (Figure 9).

NOPG	
discount factor γ	0.99
state $\boldsymbol{h}_{\text{factor}}$	1.0 1.0
action $\boldsymbol{h}_{\text{factor}}$	50.0
policy	neural network parameterized by $\boldsymbol{\theta}$ 1 hidden layer, 50 units, ReLU activations
policy output	$1 \tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$ (NOGP-D) $\mu = 1 \tanh(f_{\boldsymbol{\theta}}(\boldsymbol{s}))$, $\sigma = \text{sigmoid}(g_{\boldsymbol{\theta}}(\boldsymbol{s}))$ (NOGP-S)
learning rate	10^{-2} with ADAM optimizer
$N_{\mu_0}^{\text{MC}}$ (NOPG-S)	15
N_{ϕ}^{MC}	1
$N_{\mu_0}^{\text{MC}}$	15
policy updates	$1.5 \cdot 10^3$

Table C5.: NOPG configurations for the MountainCar experiment This table contains the configurations of NOPG-D and NOPG-S needed to replicate the results from the MountainCar trajectories experiment (Figure 11).