Bayesian RL for Goal-Only Rewards

Philippe Morere
The University of Sydney & Data61
Sydney, Australia
philippe.morere@sydney.edu.au

Fabio Ramos
The University of Sydney
Sydney, Australia
fabio.ramos@sydney.edu.au

Abstract: We address the challenging problem of reinforcement learning under *goal-only rewards* [1], where rewards are only non-zero when the goal is achieved. This reward definition alleviates the need for cumbersome reward engineering, making the reward formulation trivial. Classic exploration heuristics such as Boltzmann or ϵ-greedy exploration are highly inefficient in domains with goal-only rewards. We solve this problem by leveraging value function posterior variance information to direct exploration where uncertainty is higher. The proposed algorithm (EMU-Q) achieves data-efficient exploration, and balances exploration and exploitation explicitly at a policy level granting users more control over the learning process. We introduce general features approximating kernels, allowing to greatly reduce the algorithm complexity from $O(N^3)$ in the number of transitions to $O(M^2)$ in the number of features. We demonstrate EMU-Q is competitive with other exploration techniques on a variety of continuous control tasks and on a robotic manipulator.

Keywords: Reinforcement learning, Exploration, Continuous control

1 Introduction

In robotics, learning to perform tasks from no prior knowledge and little interaction is a hard challenge. Reinforcement Learning (RL) provides a framework for learning policies based on robot interaction with the real world. However, its application to robotics is not trivial: much overhead work is needed to design reward functions leading to efficient policy learning; many data are often required before the desired policy is learned.

Efficient RL exploration is paramount to robots succeeding in finding high rewards. Exploration success is directly dependent upon reward formulation, which is often engineered to guide robots towards unexplored states. This greatly hinders the applicability of RL to robotics as reward expert knowledge is needed. It appears more natural to reward robots only when reaching a goal, termed *goal-only rewards*, which becomes trivial to define [1]. Goal-only rewards, defined as unit reward for reaching a goal and zero elsewhere, cause classic exploration techniques based on random-walk such as ϵ-greedy and control input noise [2], or optimistic initialization to become highly inefficient. For example, Boltzmann exploration [3] requires a training time exponential in the number of states [4]. Such data requirement is unacceptable in robotics. Most solutions to this problem rely on redesigning rewards to avoid dealing with the problem of exploration. Reward shaping helps learning [5], and translating rewards to negative values triggers *optimism in the face of uncertainty* [6, 7, 8]. This approach suffers from two shortcomings: proper reward design is difficult and requires expert knowledge; improper reward design often degenerates to unexpected learned behaviour.

Intrinsic motivation [9] proposes a different approach to exploration by defining an additive guiding reward. Building on this idea, we investigate the elegant solution of leveraging model uncertainty to direct exploration where uncertainty is higher. Understanding where uncertainty comes from is crucial to achieve efficient exploration. Uncertainty in state-action value originates from three sources: (i) stochastic transitions and/or rewards; (ii) non-stationary and/or stochastic policies; (iii) unexplored state-action pairs. Our method takes advantage of this decomposition to enhance exploration.

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

1.1 Related Work

Intrinsic motivation has recently received much attention. Extensive work has focused on simple or discrete spaces, defining intrinsic rewards based on information theory [10], state visit count [11, 12, 13, 14] or approximate value function variance [15]. Exploration values are also mentioned as an alternative to additive rewards [13, 14]. Model prediction error [16, 17] proved to be successful in model-based RL, while prohibitively expensive in complex domains and often introducing unrealistic assumptions. Little work attempted to extend intrinsic exploration to continuous action spaces, as was proposed in [18] for policy gradient RL. However, these methods typically suffer from high data requirements. To the best of our knowledge, there is no data-efficient solution to the exploration problem in fully continuous domains with goal-only rewards.

Modelling state-action values using a probabilistic model enables reasoning about the whole distribution instead of just its expectation. Bayesian Q-learning [19] was first proposed to provide value function posterior information in the fully discrete case, then extended to more complicated domains by using Gaussian processes to model the state-action function [20]. Distribution over returns were proposed to design risk-sensitive algorithms [21], and approximated to enhance RL stability in [22]. However, no strategy can take advantage of posterior information for exploration.

Recent RL successes often focus on leveraging more data to learn policies, which widens the applicability gap between RL and robotics. Bayesian Optimization [23] (BO) provides a data-efficient approach for finding the optimum of an unknown objective. Exploration is achieved by building a probabilistic model and exploiting its posterior variance. BO was applied to direct policy search [24, 25], although operating at a policy level is less data-efficient and recently acquired step information is not used to improve exploration. Also, using BO to optimize policy parameters greatly restricts parameter dimensionality, hence typically requires few expressive and hand-crafted features.

1.2 Contributions

We propose to leverage BO to achieve data-efficient exploration in the goal-only reward setting. Building a probabilistic model of the value function, BO is performed over the action space, generally of low dimension. Operating at a transition level enhances data-efficiency, while not paying the price of increased complexity typically associated with model-based RL. Our contributions are threefold: (i) We formulate a framework combining ideas from BO and model-free RL, in which the balance between exploration and exploitation is made explicit and brought to a policy level; (ii) We propose a data-efficient Bayesian model-free RL method for domains with goal-only rewards, guiding exploration toward regions of higher value function uncertainty; (iii) We propose to use Random Fourier features to approximate shift-invariant kernels, reducing underlying probabilistic RL model complexity from $O(N^3)$ in the number of transitions to $O(M^2)$ in the number of features.

2 Preliminaries

2.1 Goal-only Reward MDPs

A MDP is defined by the tuple $< S, A, T, R, \gamma >$. $\mathcal{S}$ and $\mathcal{A}$ are spaces of states s and actions a respectively. The transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the probability to transition to state s' when executing action a in state s, i.e. $T(s, a, s') = p(s'|s, a)$. The reward distribution R of support $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ defines the reward r associated with transition (s, a, s'). In the simplest case, goal-only rewards are deterministic and unit rewards are given for absorbing goal states, potential negative unit rewards are given for penalized absorbing states, and zero-reward is given elsewhere. $\gamma \in [0, 1)$ is a discount factor. Solving a MDP is equivalent to finding the optimal policy π^* starting from s_0:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_i], \tag{1}$$

with $a_i \sim \pi(s_i)$, $s_{i+1} \sim T(s_i, a_i, \cdot)$, and $r_i \sim R(s_i, a_i, s_{i+1})$. Following a policy for a fixed number of steps or to an absorbing state constitutes an episode. Model-free RL learns an action-value function Q, which encodes the expected long-term discounted value of a state-action pair

$$Q(s, a) = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_i]. \tag{2}$$

Equation 2 can be rewritten recursively, also known as the Bellman equation

$$Q(s,a) = R(s,a,s') + \gamma \mathbb{E}_{s',a'|s,a}[Q(s',a')], \tag{3}$$

$s' \sim p(s'|s,a)$, $a' \sim \pi(s')$, where we dropped the expectation over deterministic rewards.

2.2 Leveraging Uncertainty for Exploration

An naive way to leverage posterior variance for exploration is to augment rewards with an exploration bonus, as presented in the intrinsic RL literature [9], yielding reward r_{total} for transition (s,a,r,s'):

$$r_{total} = r + \xi \mathbb{V}[Q(s,a,s')], \tag{4}$$

where ξ is a parameter balancing exploration and exploitation, and $\mathbb{V}$ is the variance operator. The second term encourages agents to select state-action pairs for which their model on Q is uncertain. While this formulation could help exploration in simple scenarios, it suffers from multiple limitations: (i) The posterior variance of a state-action pair reflects model uncertainty at a specific time of the learning process. It is initially high and decreases after transitions are experienced, making it a non-stationary target for Q and resulting in higher data requirements. (ii) The exploration bonus given for reaching new areas of the state-action space persists in the estimate of Q. As a consequence, agents tend to over explore and may be stuck oscillating between neighbouring states. (iii) There is no dynamic control over the exploration-exploitation balance, as changing parameter ξ only affects future total rewards. Furthermore, it would be desirable to control generating trajectories for pure exploration or exploitation, as these two quantities may conflict. The next section presents a method for enhancing exploration using posterior variance, which does not suffer from these limitations.

3 Methodology

3.1 Overview

We propose making the trade-off between exploration and exploitation explicit, by bringing this balance to a policy level. Policies need to make use of information from two separate models Q and U, representing exploitation and exploration respectively. While exploitation value function Q is learned from external rewards, exploration value function U is modelled using exploration rewards. Exploration rewards, presented in Subsection 3.2, translate reductions in uncertainty of a probabilistic model on Q. Learning probabilistic models for Q and U is described in Section 3.3.

Aiming to define policies which combine exploration and exploitation, we draw inspiration from Bayesian Optimization [24], which seeks to find the maximum of a function using very few samples. It relies on an acquisition function to determine the most promising locations to sample next. The Upper-Confidence Bounds (UCB) acquisition function [26] is popular for its explicit balance between exploitation and exploration controlled by $\kappa \in [0, \infty)$. Adapting UCB to our setting leads to:

$$\pi(s) = \arg\max_a Q(s,a) + \kappa U(s,a). \tag{5}$$

Contrary to most intrinsic RL approaches, this formulation keeps the exploration-exploitation trade-off at a policy level, as in traditional RL. This allows for adapting the exploration-exploitation balance while learning without sacrificing data-efficiency. Also, policy level balance can be used to control agent learning, e.g. stop exploration after a budget is reached, or encourage more exploration if the agent converged to a sub-optimal solution; see supplementary. Lastly, generating trajectories from exploration or exploration only grants experimenters insight over the learning process.

3.2 Modelling Discounted Exploration Returns

We define the discounted exploration return G^{exp} similarly to the classic discounted return. G^{exp} represents the discounted sum of future exploration rewards, following the current policy thereafter:

$$G^{exp} = \sum_{i=0}^{\infty} \eta^i r_i^{exp}, \text{and } r_i^{exp} \sim R^{exp}(\cdot|s_i, a_i, s_{i+1}), \tag{6}$$

where r^{exp} is an exploration reward generated by R^{exp} and $\eta \in [0,1)$ a discount factor. Exploration behaviour is achieved by maximizing the expected discounted exploration return U:

$$U(s,a) = \mathbb{E}[\sum_{i=0}^{\infty} \eta^i r_i^{exp}], \tag{7}$$

where the expectation is over the whole MDP. Note that, as we will define shortly, r^{exp} depends on Q, and so U is a function of Q. For clarity, we omit this dependency in notations. Similarly to Q, the exploration value of a state-action is encoded in $U(s, a)$, and we derive a recursive definition for U:

$$U(s, a) = \mathbb{E}_{s',a',r|s,a}[r^{exp} + \eta U(s', a')], \tag{8}$$

which is similar to that of Q in Equation 3. Learning both U and Q can be seen as combining two agents to solve separate MPDs for goal reaching and exploration. This formulation is general in that any reinforcement learning algorithm can be used to learn U.

We are now left with the task of defining r^{exp} to guide exploration towards parts of the state-action space where the variance of Q is high. We define r^{exp} to be proportional to the variance of Q evaluated at the resulting state s' of a transition, to favour transitions that result in discovery:

$$r^{exp} = \int \mathbb{V}[Q(s', a)]da - \mathbb{V}_{max}, \tag{9}$$

where the variance is integrated over all possible actions, and $\mathbb{V}_{max} = \arg\max_a \mathbb{V}[Q(s', a)]$ ensures r^{exp} is always negative. By combining an optimistic model for U to negative rewards, *optimism in the face of uncertainty* guarantees efficient exploration. The resulting model creates a gradient of U values, encoding trajectories to reach unexplored areas of the state-action space.

With continuous actions, Equation 9 might not have a closed form solution and the integral can be estimated with approximate integration techniques. In domains with discrete actions however, the integral is replaced by a sum over all possible actions.

Note that goal-only rewards are deterministic by definition, and because our framework handles exploration in a deterministic way, we focus on deterministic policies. Although state-action values are non-stationary (because π is non-stationary), their remaining stochasticity originates from transitions only. We assume transition variance is uniform over the state-action space, which we claim is a relatively mild assumption for the control problems we are interested in. Thus, fluctuations in state-action value posterior variance only result from state-action visits and can be used to guide exploration.

3.3 Bayesian Linear Regression for Q-Learning

We now seek to obtain a model-free RL algorithm able to explore with few environment interactions, and providing a full predictive distribution on state-action values. Kernel-based method GPTD [20] and Least-Squares TD [27] are among the most data-efficient model-free techniques. While the former suffers from prohibitive computation requirements, the latter offers an appealing trade-off between data-efficiency and complexity. We now derive a Bayesian RL algorithm that combines the strengths of both kernel methods and LSTD.

The distribution of long-term discounted returns G can be defined recursively as

$$G(s, a) = R(s, a, s') + \gamma G(s', a'), \tag{10}$$

which is an equality in the distributions of the two sides of the equation. Let us decompose the discounted return G into its mean Q and a random zero-mean residual q so that $Q(s, a) = \mathbb{E}[G(s, a)]$:

$$\underbrace{R(s, a, s') + \gamma Q(s', a')}_{\text{target } t} = Q(s, a) + \underbrace{q(s, a) - \gamma q(s', a')}_{\text{noise } \epsilon}.$$

Assuming rewards from R are disturbed by zero-mean Gaussian noise implies the difference of residuals ϵ is Gaussian with zero-mean and precision β. By modelling Q as a linear function of a feature map $\boldsymbol{\phi}_{s,a}$ so that $Q(s, a) = \boldsymbol{w}^T \boldsymbol{\phi}_{s,a}$, estimation of state-action values becomes a linear regression problem of target t and noise ϵ. The likelihood function takes the form

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|r_i + \gamma \boldsymbol{w}^T \boldsymbol{\phi}_{s'_i,a'_i}, \beta^{-1}), \tag{11}$$

where independent transitions are denoted $x_i = (s_i, a_i, r_i, s'_i, a'_i)$. We now treat the weights as random variables with Gaussian prior $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I})$. The weight posterior distribution is

$$p(\boldsymbol{w}|\boldsymbol{t}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_Q, \boldsymbol{S}) \tag{12}$$

$$\boldsymbol{m}_Q = \beta \boldsymbol{S} \boldsymbol{\Phi}^T_{\boldsymbol{s},\boldsymbol{a}} (\boldsymbol{r} + \gamma \boldsymbol{Q}') \tag{13}$$

$$\boldsymbol{S} = (\alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T_{\boldsymbol{s},\boldsymbol{a}} \boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}})^{-1}, \tag{14}$$

where $\boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}} = \{\boldsymbol{\phi}_{s_i,a_i}\}_{i=1}^N$, $\boldsymbol{Q'} = \{Q(s'_i, a'_i)\}_{i=1}^N$, and $\boldsymbol{r} = \{r_i\}_{i=1}^N$. The predictive distribution $p(t|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta)$ is also Gaussian, yielding

$$Q(s,a) = \mathbb{E}[p(t|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta)] = \boldsymbol{\phi}_{s,a}^T \boldsymbol{m}_Q, \tag{15}$$

$$\sigma^2(s,a) = \mathbb{V}[p(t|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta)] = \beta^{-1} + \boldsymbol{\phi}_{s,a}^T \boldsymbol{S} \boldsymbol{\phi}_{s,a}. \tag{16}$$

The predictive variance $\sigma^2(s,a)$ encodes the uncertainty in $Q(s,a)$, and is used to compute r^{exp}. The derivation for U is similar, replacing r with r^{exp}. Note that because $\boldsymbol{S}$ does not depend on rewards, it is shared by both models. Hence, with $\boldsymbol{U'} = \{U(s'_i, a'_i)\}_{i=1}^N$,

$$U(s,a) = \boldsymbol{\phi}_{s,a}^T \boldsymbol{m}_U, \text{ with } \boldsymbol{m}_U = \beta \boldsymbol{S} \boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}}^T (\boldsymbol{r}^{exp} + \eta \boldsymbol{U'}). \tag{17}$$

This model gracefully adapts to iterative updates, by substituting the current prior with the previous posterior. Furthermore, using the Sherman-Morrison equality to update matrix $\boldsymbol{S}$ saves the cost of inverting a matrix at every step and hence reduces the complexity cost from $O(M^3)$ to $O(M^2)$ in the number of features. See Algorithm 1 and supplementary code[1] for an optimized implementation.

Algorithm 1 EMU-Q: Exploring by Minimizing Uncertainty of Q values

1: **Input:** initial state s, parameters α, β, κ. **Output:** Policy π paramatrized by $\boldsymbol{m}_Q$ and $\boldsymbol{m}_U$.
2: $\boldsymbol{S} = \alpha^{-1}\boldsymbol{I}$, $\boldsymbol{t}_Q = \boldsymbol{t}_U = \boldsymbol{m}_Q = \boldsymbol{m}_U = \boldsymbol{0}$
3: **for** episode $l = 1, 2, ..$ **do**
4: **for** step $h = 1, 2, ..$ **do**
5: $\pi(s) = \arg\max_a \boldsymbol{\phi}_{s,a} \boldsymbol{m}_Q + \kappa \boldsymbol{\phi}_{s,a} \boldsymbol{m}_U$
6: Execute $a = \pi(s)$, observe s' and r, and store $\boldsymbol{\phi}_{s,a}, r, s'$ in D.
7: $\boldsymbol{S} = \boldsymbol{S} - \beta \frac{(\boldsymbol{S}\boldsymbol{\phi}_{s,a})(\boldsymbol{\phi}_{s,a}^T \boldsymbol{S})}{1 + \beta \boldsymbol{\phi}_{s,a}^T \boldsymbol{S} \boldsymbol{\phi}_{s,a}}$
8: $\boldsymbol{t}_Q = \boldsymbol{t}_Q + \beta \boldsymbol{\phi}_{s,a}^T (r + \gamma \boldsymbol{\phi}_{s,a} \boldsymbol{m}_Q)$
9: $\boldsymbol{t}_U = \boldsymbol{t}_U + \beta \boldsymbol{\phi}_{s,a}^T (r^{exp} + \eta \boldsymbol{\phi}_{s,a} \boldsymbol{m}_U)$, with r^{exp} from Equation 9 with s'.
10: $\boldsymbol{m}_Q = \boldsymbol{S}\boldsymbol{t}_Q$, and $\boldsymbol{m}_U = \boldsymbol{S}\boldsymbol{t}_U$
11: **end for**
12: From D, draw $\boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}}, \boldsymbol{r}, \boldsymbol{s'}$, and compute $\boldsymbol{\Phi}_{\boldsymbol{s'},\pi(\boldsymbol{s'})}$.
13: Update $\boldsymbol{m}_Q = \boldsymbol{S} \beta \boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}}^T (\boldsymbol{r} + \gamma \boldsymbol{\Phi}_{\boldsymbol{s'},\pi(\boldsymbol{s'})} \boldsymbol{m}_Q)$ until change in $\boldsymbol{m}_Q < \epsilon$.
14: Compute $\boldsymbol{r}^{exp}$ with Equation 9 and $\boldsymbol{s'}$.
15: Update $\boldsymbol{m}_U = \boldsymbol{S} \beta \boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}}^T (\boldsymbol{r}^{exp} + \eta \boldsymbol{\Phi}_{\boldsymbol{s'},\pi(\boldsymbol{s'})} \boldsymbol{m}_U)$ until change in $\boldsymbol{m}_U < \epsilon$.
16: **end for**

End of episode updates for $\boldsymbol{m}_Q$ and $\boldsymbol{m}_U$ (line 13 onward) are analogous to policy iteration, and although not mandatory, greatly improve convergence speed. Note that because r_{exp} is a non-stationary target, recomputing it after each episode with the updated posterior on Q provides the model on U with more accurate targets, thereby improving learning speed.

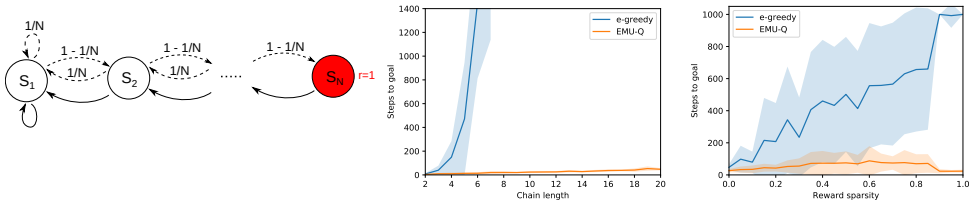
3.4 Kernel Approximation Features for RL

We presented a simple method to learn Q an U as linear functions of state-action features. While powerful when using a good feature map, linear models typically require experimenters to define meaningful features for specific problems. In this section, we introduce random Fourier features (RFF) [28], a kernel approximation technique which allows linear models to enjoy the expressivity of kernel methods. It should be noted that these features are different from Fourier basis [29], which do not approximate kernel functions. Further comparison is given in supplementary material. Although RFF were recently used to learn policy parametrizations [30], to the best of our knowledge, this is the first time RFF are applied to the value function approximation problem in RL.

For any shift invariant kernel, which can be written as $k(\tau)$ with $\tau = \boldsymbol{x} - \boldsymbol{x'}$, a representation based on the Fourier transform can be computed with Bochner's theorem [31]. We have

$$k(\tau) = \int_{\mathbb{R}^D} e^{-i\tau\boldsymbol{\omega}} p(\boldsymbol{\omega}) d\boldsymbol{\omega} \approx \frac{1}{M} \sum_{j=1}^{M} e^{-i\tau\boldsymbol{\omega}_j} = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x'}) \rangle, \tag{18}$$

[1] Code available at https://github.com/PhilippeMorere/EMU-Q



(a) (b) (c)

Figure 1: (a) Chain domain described in [4]. (b) Steps to goal (mean and standard deviation) in goal-only chain domain, for increasing chain lengths (30 runs). (c) Steps to goal in semi-sparse 10-state chain, as a function of reward sparsity, with maximum of 1000 steps (100 runs).

where p is the spectral density of k, $\boldsymbol{\phi}(\boldsymbol{x})$ is an approximate feature map, and M the number of spectral samples from p. In practice, the feature map approximating $k(\boldsymbol{x}, \boldsymbol{x}')$ is

$$\boldsymbol{\phi}(\boldsymbol{x}) = \frac{1}{\sqrt{M}}[\cos(\boldsymbol{x}^T\boldsymbol{\omega}_1), ..., \cos(\boldsymbol{x}^T\boldsymbol{\omega}_M), \sin(\boldsymbol{x}^T\boldsymbol{\omega}_1), ..., \sin(\boldsymbol{x}^T\boldsymbol{\omega}_M)], \tag{19}$$

where the imaginary part was set to zero, as required for real kernels. In the case of the RBF kernel defined as $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{1}{2\sigma^2}||\boldsymbol{x} - \boldsymbol{x}'||_2^2)$, the kernel spectral density is Gaussian $p = \mathcal{N}(0, 2\sigma^{-2}I)$. Feature maps can be computed by drawing $M/2 \times d$ samples from p one time only, and computing Equation 19 on new inputs $\boldsymbol{x}$ using these samples. Resulting features are not domain specific and require no feature engineering. Users only need to choose a kernel that represents adequate distance measures in the state-action space, and can benefit from numerous kernels already provided by the literature. As the number of features increases, kernel approximation error decreases [32], and Bayesian linear regression with RFF approximates a Gaussian process. Additionally, sampling frequencies according to a quasi-random sampling scheme (used in our experiments) reduces kernel approximation error compared to classic Monte-Carlo sampling with the same number of features [33].

Bringing our contributions together, EMU-Q combines the ease-of-use and expressivity of kernel methods brought by RFF with the convergence properties and speed of linear models.

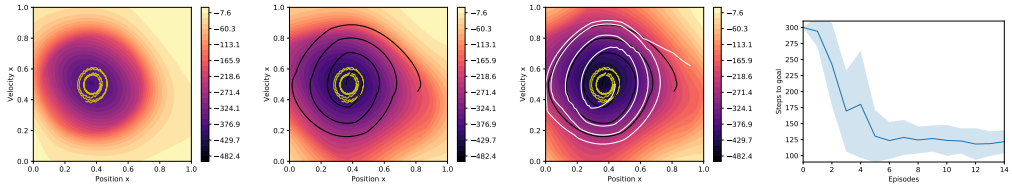
4 Experiments

We qualitatively and quantitatively evaluate the performance of our method EMU-Q on a toy chain MDP example, 7 widely-used continuous control domains, and a robotic manipulator problem. Experiments aim at measuring exploration capabilities in domains with goal-only rewards. Domains feature one absorbing goal state with positive unit reward, and penalizing absorbing states with reward -1. All other rewards are zero, resulting in very sparse reward functions, and rendering guidance from reward gradient information inapplicable.

4.1 Synthetic Chain Domain

Goal-only Rewards: We investigate EMU-Q's exploration capabilities on a classic domain known to be hard to explore. It is composed of a chain of N states and two actions, displayed in Figure 1a. Action right (dashed) has probability $1 - 1/N$ to move right and probability $1/N$ to move left. Action left (solid) is deterministic. Rewards are goal-only with goal state S_N. Classic exploration such as ϵ-greedy was shown to have exponential regret with the number of states in this domain [4]. Achieving better performance on this domain is therefore essential to any advanced exploration technique. We compare EMU-Q to ϵ-greedy exploration for increasing chain lengths, in terms of number of steps before goal-state S_N is found. Results in Figure 1b illustrate the exponential regret of ϵ-greedy while EMU-Q achieves much lower exploration time, scaling linearly with chain length.

Semi-Sparse Rewards: We now investigate the impact of reward structure by decreasing the chain domain's reward sparsity. In this experiment *only*, agents are given additional -1 rewards with probability $1 - p$ for every non-goal state, effectively guiding them towards the goal state (goal-only



(a) (b) (c) (d)

Figure 2: Goal-only MountainCar. (a,b,c) Background: exploration value function U (for action *left*) after 1, 2, and 3 episodes. State trajectories for 3 episodes are plain lines (yellow, black and white respectively). (d) Steps to goal ($x > 0.9$), with policy refined after goal state was found (30 runs).

rewards are recovered with $p = 0$). The average number of steps before the goal is reached as a function of p is compared for ϵ-greedy and EMU-Q in Figure 1c. Results show that ϵ-greedy performs very poorly for high p, but improves as guiding reward density increases. Conversely, our algorithm seems unaffected by reward density and performs equally well for all values of p. When $p = 0$, agents receive -1 reward in every non-goal state, and ϵ-greedy performs similarly to EMU-Q.

4.2 Classic Control

We further evaluate EMU-Q on *goal-only* and fully continuous versions of standard control problems [34], in which classic exploration methods are unable to reach goal states. Details on reward functions used and experimental setup are given in supplementary material.

Exploration Behaviour on goal-only MountainCar: We first provide intuition behind what exploration values learn and illustrate its typical behaviour on a continuous goal-only version of MountainCar. In this problem, the agent is granted a unit reward for reaching the top of the right hill, and zero elsewhere. Figure 2 displays the state-action exploration value function U at different stages of learning, overlaid by the state-space trajectories followed during learning. The first episode (yellow line) exemplifies action babbling, and the car does not exit the valley (around $x = 0.4$). On the next episode (black line), the agent finds sequences of action that allow exiting the valley and exploring further areas of the state-action space. Lastly, in episode three (white line), the agent finds the goal ($x > 0.9$). This is done by adopting a strategy that quickly leads to unexplored areas, as shown by the increased distance between white lines. The exploration value function U reflects high uncertainty about unexplored areas (yellow), which shrink as more data is gathered, and low and decreasing uncertainty for often visited areas such as starting states (purple). Function U also features a gradient which can be followed from any state to find new areas of the state-action space to explore. Figure 2d shows EMU-Q's exploration capabilities enables to find the goal state within one or two episodes.

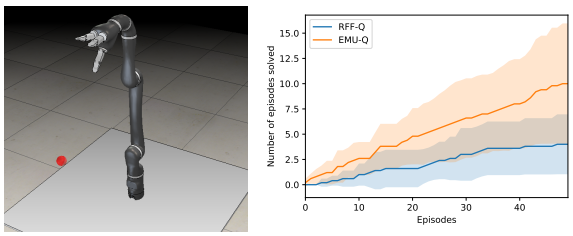
Continuous control benchmark: We now compare our algorithm on the complete benchmark of 7 continuous control goal-only tasks. Random Fourier Features approximating square exponential kernels are used for both state and action spaces. Exploration parameter κ is set to $1/\mathbb{V}_{max}$.

Most methods in the sparse rewards literature address domains with discrete states and/or action spaces, making it difficult to find baselines to compare EMU-Q to. Furthermore, classic exploration techniques such as ϵ-greedy fail on these domains. We compare our algorithm to three baselines. VIME [18] defines exploration as maximizing information gain about the agent's belief of the environment dynamics. DORA [14], which we run on discretized action spaces, extends visit counts to continuous state spaces. Q-Learning with ϵ-greedy exploration and RFF is denoted RFF-Q and run on domains with classic rewards. We are interested in comparing exploration performance, favouring fast discovery of goal states. To reflect exploration performance, we measure the number of episodes required before the first positive reward is obtained. This metric reflects how long pure exploration is required for before goal-reaching information can be exploited to refine policies.

Results displayed in Table 1 indicate that EMU-Q is more consistent than VIME or DORA in finding goal states on all domains, illustrating better exploration capabilities. The average number of episodes to reach goal states is computed *only* on successful runs. EMU-Q displays better goal finding on

Domain	**EMU-Q**		**VIME**		**DORA** (discrete)		**RFF-Q** (classic reward)	
	Success	Episodes to goal	Success	Episodes to goal	Success	Episodes to goal	Success	Episodes to goal
SinglePendulum	**100**%	1.80 (1.07)	95%	2.05 (2.04)	35%	3.00 (4.11)	100%	1.0 (0.00)
MountainCar	**100**%	2.95 (0.38)	65%	5.08 (2.43)	0%	–	100%	8.6 (8.05)
DoublePendulum	**100**%	1.10 (0.30)	90%	3.61 (2.75)	0%	–	100%	4.20 (2.25)
CartpoleSwingup	**90**%	12.40 (16.79)	65%	3.23 (2.66)	35%	48.71 (30.44)	100%	9.70 (12.39)
LunarLander	**100**%	28.75 (29.57)	75%	4.47 (2.47)	30%	35.17 (31.38)	95%	19.15 (24.06)
Reacher	**100**%	19.70 (20.69)	95%	3.68 (2.03)	35%	1.00 (0.00)	95%	26.55 (25.58)
Hopper	**60**%	52.85 (39.32)	40%	5.62 (3.35)	20%	30.50 (11.80)	80%	41.15 (35.72)

Table 1: Results for all 7 domains, as success rate of goal finding within 100 episodes and mean (and standard deviation) of number of episodes before goal is found. Success rate rate is more important than number of episodes to goal. Results averaged over 20 runs. DORA was run with discretized action, and RFF-Q on domains with classic rewards.



(a) (b)

Figure 3: (a) Manipulator task: learning to reach a randomly located target (red ball). (b) EMU-Q's directed exploration yields higher performance compared to RFF-Q with ϵ-greedy exploration.

lower dimension domains, while VIME tends to find goals faster on domains with higher dimensions but fails in more occasions. Observing similar results with RFF-Q confirms EMU-Q can deal with goal-only rewards without sacrificing performance.

4.3 Jaco manipulator

Lastly, we apply EMU-Q to a robotics problem in which we want to learn to control a Jaco manipulator solely from joint configuration. Although computing inverse kinematics on robotic arms is studied extensively, we focus here on learning a mapping from joint configuration to joint torques on a damaged manipulator. We model damage by immobilizing four of the joints, making previous inverse kinematics invalid. The task requires an RL algorithm to learn a policy to reach different areas of the space, as shown in Figure 3a, in which rewards are only given when the goal is reached. Results in Figure 3b show EMU-Q learns faster and manages to complete the task more consistently than RFF-Q, confirming directed exploration is beneficial to this robotics scenario.

5 Conclusion

We presented a novel method for data-efficient exploration in RL domains with goal-only rewards, guiding exploration toward regions of high state-action value function uncertainty. We defined the expected exploration return, which represents the exploration potential of state-action pairs, and brought the exploration-exploitation trade-off to a policy level. We introduced features approximating kernels, bringing expressivity and reduced cost to our algorithm. Evaluated on a toy problem, a benchmark of continuous goal-only reward control tasks and a robotic manipulator task, our method provides competitive exploration performance. It excels in smaller domains, where it consistently finds goals within a couple of episodes.

Modelling exploration and exploitation as separate quantities results in increased flexibility and new ways to dynamically control exploration and learning. Combined with easy reward specification and data efficiency, our method makes RL extremely accessible to robotics.

References

[1] C. Reinke, E. Uchibe, and K. Doya. Average reward optimization with multiple discounting reinforcement learners. In *International Conference on Neural Information Processing*, pages 789–800. Springer, 2017.

[2] 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 (ICML-15)*, pages 1889–1897, 2015.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4, 1996.

[4] I. Osband, B. Van Roy, and Z. Wen. Generalization and exploration via randomized value functions. *arXiv preprint arXiv:1402.0635*, 2014.

[5] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[6] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49, 2002.

[7] R. I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3, 2002.

[8] T. Jaksch, R. Ortner, and P. Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11, 2010.

[9] N. Chentanez, A. G. Barto, and S. P. Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.

[10] P.-Y. Oudeyer and F. Kaplan. How can we define intrinsic motivation? In *Proceedings of the 8th International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, 2008.

[11] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*, pages 206–214, 2012.

[12] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[13] I. Szita and A. Lőrincz. The many faces of optimism: a unifying approach. In *Proceedings of the 25th international conference on Machine learning*, pages 1048–1055, 2008.

[14] L. Fox, L. Choshen, and Y. Loewenstein. DORA the explorer: Directed outreaching reinforcement action-selection. In *International Conference on Learning Representations*, 2018.

[15] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, 2016.

[16] B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[17] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[18] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[19] R. Dearden, N. Friedman, and S. Russell. Bayesian q-learning. In *AAAI/IAAI*, 1998.

[20] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005.

[21] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka. Nonparametric return distribution approximation for reinforcement learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[22] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, 2017.

[23] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13, 1998.

[24] E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[25] A. Wilson, A. Fern, and P. Tadepalli. Using trajectory data to improve bayesian optimization for reinforcement learning. *The Journal of Machine Learning Research*, 15, 2014.

[26] D. D. Cox and S. John. A statistical method for global optimization. In *Systems, Man and Cybernetics, 1992., IEEE International Conference on*. IEEE, 1992.

[27] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[28] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, 2008.

[29] G. Konidaris, S. Osentoski, and P. S. Thomas. Value function approximation in reinforcement learning using the fourier basis. In *AAAI*, 2011.

[30] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade. Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, pages 6553–6564, 2017.

[31] I. Gihman and A. Skorohod. The theory of stochastic processes, vol. i, 1974.

[32] D. J. Sutherland and J. Schneider. On the error of random fourier features. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, 2015.

[33] J. Yang, V. Sindhwani, H. Avron, and M. Mahoney. Quasi-monte carlo feature maps for shift-invariant kernels. In *Proceedings of The 31st International Conference on Machine Learning (ICML-14)*, 2014.

[34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[35] C. M. Bishop. Pattern recognition and machine learning. 2006.

A Derivation of Bayesian linear regression for Q-learning

A.1 Weight posterior distribution

The likelihood function is defined as follows

$$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_i|r_i + \gamma \boldsymbol{w}^T \boldsymbol{\phi}_{s_i', a_i'}, \beta^{-1}), \tag{20}$$

where independent transitions are denoted $x_i = (s_i, a_i, r_i, s_i', a_i')$. we treat the linear regression weights $\boldsymbol{w}$ as random variables and introduce a Gaussian prior

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}) \tag{21}$$

The weight posterior can be computed analytically with Bayes rule, resulting in a normal distribution

$$p(\boldsymbol{w}|\boldsymbol{t}) = \frac{p(\boldsymbol{t}|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{t})} = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}_Q, \boldsymbol{S}) \tag{22}$$

Expressions for the mean $\boldsymbol{m}_Q$ and variance $\boldsymbol{S}$ follow from general results of products of normal distributions [35]:

$$\boldsymbol{m}_Q = \beta \boldsymbol{S} \boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}}^T (\boldsymbol{r} + \gamma \boldsymbol{Q}') \tag{23}$$

$$\boldsymbol{S} = (\alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}}^T \boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}})^{-1}, \tag{24}$$

where $\boldsymbol{\Phi}_{\boldsymbol{s},\boldsymbol{a}} = \{\boldsymbol{\phi}_{s_i,a_i}\}_{i=1}^N$, $\boldsymbol{Q}' = \{Q(s_i', a_i')\}_{i=1}^N$, and $\boldsymbol{r} = \{r_i\}_{i=1}^N$. The predictive distribution $p(t|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta)$ can be obtain by weight marginalization and is also normal

$$p(t|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta) = \int p(t|\boldsymbol{w}, \beta) p(\boldsymbol{w}|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta) d\boldsymbol{w} = \mathcal{N}(t|Q, \sigma^2) \tag{25}$$

Expressions for Q and σ^2 follow from general results [35], yielding

$$Q(s, a) = \mathbb{E}[p(t|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta)] = \boldsymbol{\phi}_{s,a}^T \boldsymbol{m}_Q, \tag{26}$$

$$\sigma^2(s, a) = \mathbb{V}[p(t|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta)] = \beta^{-1} + \boldsymbol{\phi}_{s,a}^T \boldsymbol{S} \boldsymbol{\phi}_{s,a}. \tag{27}$$

A.2 Sherman-Morisson update

The Sherman-Morrison inverse matrix equality is used to compute a rank-1 update of matrix $\boldsymbol{S}$ with the feature map $\boldsymbol{\phi}_{\boldsymbol{s},\boldsymbol{a}}$ of a data point:

$$\boldsymbol{S}_{t+1} = \boldsymbol{S}_t - \beta \frac{(\boldsymbol{S}_t \boldsymbol{\phi}_{s,a})(\boldsymbol{\phi}_{s,a}^T \boldsymbol{S}_t)}{1 + \beta \boldsymbol{\phi}_{s,a}^T \boldsymbol{S}_t \boldsymbol{\phi}_{s,a}} \tag{28}$$

This update only requires matrix-to-vector multiplications and saves the cost of inverting a matrix after every step. Hence the complexity cost is reduced from $O(M^3)$ to $O(M^2)$ in the number of features M. This update is included in the optimized implementation of EMU-Q.

B Comparison between random Fourier features and Fourier basis

Fourier basis features are described in [29] as a linear function approximation based on Fourier series decomposition. Formally, the order-n feature map for state $\boldsymbol{s}$ is defined as follows:

$$\boldsymbol{\phi}(\boldsymbol{s}) = cos(\pi \boldsymbol{s}^T \boldsymbol{C}), \tag{29}$$

where C is the Cartesian product of all $c_j \in \{0, ..., n\}$ for $j = 1, .., d_{\mathcal{S}}$. Note that Fourier basis features do not scale well. Indeed, the number of features generated is exponential with state space dimension.

While Fourier basis features approximate value functions with periodic basis functions, random Fourier features are designed to approximate a kernel function with similar basis functions. As such,

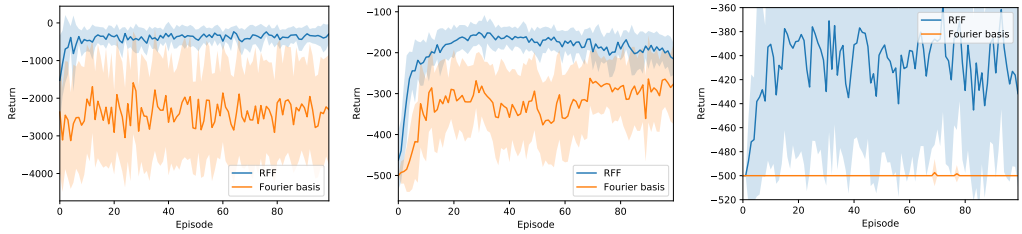


Figure 4: Return mean and standard deviation for Q-learning with random Fourier features (RFF) or Fourier basis features on SinglePendulum (a), MountainCar (b), and DoublePendulum (c) domain with classic rewards. Results are computed using classic Q-learning with ϵ-greedy policy, and averaged over 20 runs for each method.

they allow recovering properties of kernel methods in the limit of the number of features. Additionally, random Fourier features scale better with higher dimensions

We compare both types of features on several classic RL domains using Q-learning. The number of random Fourier features was set to 300, and the order of Fourier basis was set to 5 for SinglePendulum and MountainCar and to 3 for DoublePendulum. Higher state and action space dimensions on subsequent domains make using Fourier basis features prohibitively expensive. Results displayed in Figure 4 show RFF outperforms Fourier basis both in terms of learning speed and asymptotic performance. In DoublePendulum, the low order of Fourier basis features (still leading to more than 2000 features) seems insufficient to learn a policy completing the task.

C Comparison of our method on RL benchmark

C.1 Domain description

All domains make use of OpenAI Gym [34], and are modified to feature goal only rewards and continuous state and action spaces whose dimensions are detailed in Table 2.

Reward functions are modified to the following:

- In SinglePendulum, the agent receives a unit reward when $\theta < 0.05\ rad$ where θ is the pole angle.
- In MountainCar, the agent is given a unit reward for reaching the top of the right hill $x > 0.9$.
- In DoublePendulum, the agent is given a unit reward when the distance between the tip of the pendulum is within a distance $d < 1$ from the tallest point it can reach.
- In CartpoleSwingUp, the agent receives a unit reward when $cos(\theta) > 0.8$, with θ the pole angle. In LunarLander, the agent receives a unit reward for reaching the landing pad within distance $d < 0.05$ of its centre point on both axis, and is given negative unit reward for crashing or exiting the flying area.
- In Reacher, the agent is given unit reward for reaching the target within a distance $d < 0.015$.
- In Hopper, the agent is given a unit reward for jumping to height $h > 1.3$, and a negative unit reward for falling $|\theta| > 0.2$ where θ is the leg angle.

C.2 Experimental setup

Parameters γ and η are set to 0.99 for all domains. Episodes are capped at 500 steps for all domains. State spaces are normalized for each domain, and Random Fourier Features approximating square exponential kernels are used for both state and action spaces. The state and action kernel lengthscales are denoted as $l_{\mathcal{S}}$ and $l_{\mathcal{A}}$ respectively. Exploration and exploitation trade-off parameter κ is set to $1/\mathbb{V}_{max}$ for all experiments. Algorithm parameters were manually fixed to reasonable values given in Table 2.

Domain	$d_{\mathcal{S}}$	$d_{\mathcal{A}}$	$l_{\mathcal{S}}$	$l_{\mathcal{A}}$	M	α	β
SinglePendulum	3	1	0.3	0.3	300	0.001	1.0
MountainCar	2	1	0.3	10	300	0.1	1.0
DoublePendulum	6	1	0.3	0.3	500	0.01	1.0
CartpoleSwingUp	4	1	0.8	1.0	500	0.01	1.0
LunarLander	8	2	0.5	0.3	500	0.01	1.0
Reacher	11	2	0.3	0.3	500	0.001	1.0
Hopper	11	3	0.3	0.3	500	0.01	1.0

Table 2: Experimental parameters for all 7 domains

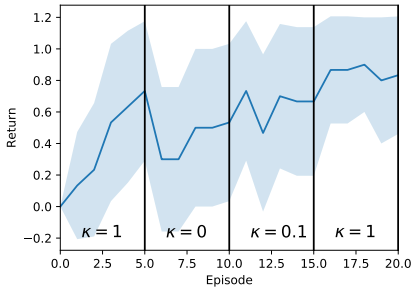


Figure 5: Influence of using schedules for κ while learning on goal-only MountainCar: higher values lead to increases in return while lower ones tend to stabilize performance (averaged over 30 runs).

D Schedule for κ while learning

Bringing the exploration-exploitation trade-off at a policy level allows for dynamically controlling such balance. In this section, we show a basic example of learning behavior when parameter κ is being changed during the learning process. Figure 5 shows agent performance in goal-only MountainCar when κ is first set to 1, then 0 and increased again. Performance increases proportionally to κ when it is greater than 0, while returns tends to stay constant when it is set to 0.