Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Multi-Fidelity Model-Free Reinforcement Learning with Gaussian Processes

Varun Suryan, Nahush Gondhalekar, Pratap Tokekar
Virginia Tech, USA

Abstract

We study the problem of Reinforcement Learning (RL) using as few real-world samples as possible. A naive application of RL can be inefficient in large and continuous state spaces. We present a Multi-Fidelity Reinforcement Learning (MFRL) model-free algorithm that leverages Gaussian Processes (GPs) to learn the optimal policy in the real world. In the MFRL framework, an agent uses multiple simulators of the real environment to perform actions. With increasing fidelity in a simulator chain, the number of samples used in successively higher simulators can be reduced. By incorporating GPs in the MFRL framework, further reduction in the number of learning samples can be achieved as we move up the simulator chain. We examine the performance of our algorithm through simulations and through real-world experiments for navigation with a ground robot.

Introduction

Reinforcement learning (RL) allows an agent to learn directly from an environment without relying on exemplary supervision or complete models of the environment (Sutton and Barto 1998). Recently, there has been a significant development in RL applied to learning policies for robots (Mnih et al. 2015; LeCun, Bengio, and Hinton 2015; Silver et al. 2016). A major limitation of using RL for learning with robots is the need to obtain a large number of training samples. Obtaining a large number of real-world samples can be expensive and potentially dangerous. In particular, obtaining negative samples may require the robot to collide or fail, which is undesirable. The overall goal of our work is to reduce the number of real-world samples required for learning optimal policies. In this paper, we show how to leverage one or more simulators along with real-world samples to learn a policy for a robot.

We build on the Multi-Fidelity Reinforcement Learning (MFRL) algorithm from (Cutler, Walsh, and How 2015). MFRL leverages multiple simulators with varying fidelity levels to minimize the number of real-world (*i.e.*, highest fidelity simulator) samples. The simulators, denoted by $\Sigma_1, \ldots, \Sigma_d$, have increasing levels of fidelity with respect to the real environment. For example, Σ_1 can be a simple simulator that models only the robot kinematics, Σ_2 can model the dynamics as well as kinematics, and the highest fidelity simulator can be the real-world (Figure 1).

This work has been funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation-sponsored industry/university cooperative research center (I/UCRC) under NSF Award No. IIP-1161036 along with significant contributions from C-UAS industry members.

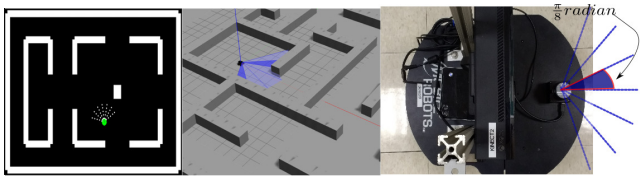


Figure 1: MFRL framework: The first simulator captures only grid-world movements of a point robot while the second simulator has more fidelity modeling the physics as well. Control can switch back and forth between the simulators and real environment which is the third simulator in the chain. We use the Python-based simulator Pygame as Σ_1, Gazebo as Σ_2 and Pioneer P3-DX robot in real-world as Σ_3.

MFRL differs from transfer learning (Taylor, Stone, and Liu 2007), where a transfer of parameters is allowed only in one direction. The MFRL algorithm starts in Σ_1. Once it learns a sufficiently good policy in Σ_1, it switches to a higher fidelity simulator. If it observes that the policy learned in the lower fidelity simulator is no longer optimal in the higher fidelity simulator, it switches back to the lower fidelity simulator. (Cutler, Walsh, and How 2015) showed that the resulting algorithm has polynomial sample complexity and minimizes the number of samples required for the highest fidelity simulator, i.e., the real world, under some technical conditions.

The original MFRL algorithm uses the Knows-What-It-Knows framework (Li et al. 2011) to learn the transition and reward functions at each level. The reward and transition for each state-action pair are learned independently of others. While this is reasonable for general agents, when planning for physically-grounded robots, we can exploit the spatial correlation between neighboring state-action pairs to speed up the learning.

We use Gaussian Process (GP) regression (Rasmussen and Williams 2006) as a function approximator to speed up learning in the MFRL framework. GPs can predict the

learned function value for any query point, and not just for a discretized state-action pair. Furthermore, GPs can exploit the correlation between nearby state-action values by an appropriate choice of a kernel. GPs have been used in conjunction with policy search methods to obtain optimal policies in simulation-aided reinforcement learning (Cutler and How 2016; M. Cutler and J. P. How 2015). We take this further by using GPs in the MFRL setting.

In MFRL, the state-space of Σ_i is a subset of the state space of Σ_j for all $j > i$. Therefore, when the MFRL algorithm switches from Σ_i to Σ_{i+1} it already has a policy (better than naive) for states in $\Sigma_{i+1} \setminus \Sigma_i$. Thus, GPs are particularly suited for MFRL, which we verify through our simulation results.

Our main contribution in this paper is to leverage GP regression for model-free MFRL by directly estimating the optimal Q-values (GPQ-MFRL) and subsequently calculating the optimal policy. We verify the performance of the GP-based MFRL algorithm through simulations as well as experiments with a ground robot. Our empirical evaluation shows that the GP-based MFRL algorithm learns the optimal policy faster than the original MFRL algorithm using even fewer real-world samples. The original MFRL work was for a model-based approach. We extend it to introduce a model-free version. Model-free algorithms have been used in conjunction with transfer learning in the past (Taylor, Stone, and Liu 2007).

The rest of the paper is organized as follows. In the next section, we present the background on RL and GPs followed by a survey of related work. Next, we present the GP-MFRL algorithm (GPQ-MFRL) followed by experimental results along with comparisons with other MFRL and non-MFRL techniques. We conclude with a discussion of the future work.

Background

Reinforcement Learning

RL problems can be formulated as a Markov Decision Process (MDP): $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathcal{P}(s_t, a_t, s_{t+1}) \mapsto [0, 1]$, reward function $\mathcal{R}(s_t, a_t) \mapsto \mathbb{R}$ and discount factor $\gamma \in [0, 1)$ (Sutton and Barto 1998; Puterman 2014). A policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ maps states to actions. Together with the initial state s_0, a policy forms a trajectory $\zeta = \{[s_0, a_0, r_0], [s_1, a_1, r_1], \ldots\}$ where $a_t = \pi(s_t)$. r_t and s_{t+1} are sampled from the reward and transition functions, respectively.

We consider a scenario where the goal is to maximize the infinite horizon discounted reward starting from a state s_0. The value function for a state s_0 is defined as $\mathcal{V}^\pi(s_0) = \mathbb{E}[\sum_{t=0}^{t=\infty} \gamma^t r_t(s_t, a_t) | a_t = \pi(s_t)]$. The state-action value function or Q-value of each state-action pair under policy π is defined as $Q^\pi(s, a) = \mathbb{E}[\sum_{t=0}^{t=\infty} \gamma^t r_{t+1}(s_{t+1}, a_{t+1}) | s_0 = s, a_0 = a]$ which is the expected sum of discounted rewards obtained starting from state s, taking action a and following π thereafter. The optimal Q-value function Q^* for a state-action pair (s, a) satisfies $Q^*(s, a) = \max_\pi Q^\pi(s, a) = \mathcal{V}^*(s)$ and can be written recursively as,

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r(s_t, a_t) + \gamma \mathcal{V}^*(s_{t+1})]. \quad (1)$$

Our objective is to find the optimal policy $\pi^*(s) = \text{argmax}_a Q^*(s, a)$ when $\mathcal{R}$ and $\mathcal{P}$ are not known to the agent. In model-based approaches, the agent learns $\mathcal{R}$ and $\mathcal{P}$ first and then finds an optimal policy by calculating optimal Q-values from Equation 1. The most commonly used model-based approach is VI (Jung and Stone 2010; Brafman and Tennenholtz 2002). We can also directly estimate the optimal Q-values (model-free approaches) (Strehl et al. 2006; Grande, Walsh, and How 2014) or directly calculate the optimal policy (policy-gradient approaches) (Sutton et al. 2000). The most commonly used model-free algorithm is Q-learning (Watkins and Dayan 1992). For our GPQ-MFRL implementation, we use Q-learning to perform the policy update using GP regression.

In this work, we focus on the model-free version of GP based MFRL since they are computationally and memory efficient (Brafman and Tennenholtz 2002).

Gaussian Processes

GPs are Bayesian non-parametric function approximators. GPs can be defined as a collection of infinitely many random variables, any finite subset $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ [1] of which is jointly Gaussian with mean vector $\mathbf{m} \in \mathbb{R}^k$ and covariance matrix $\mathbf{K} \in \mathbb{R}^{k \times k}$ (Rasmussen and Williams 2006).

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ denote the set of the training inputs. Let $\mathbf{y} = \{y_1, \ldots, y_k\}$ denote the corresponding training outputs. GPs can be used to predict the output value at a new test point, $\mathbf{x}$, conditioned on the training data. Predicted output value at $\mathbf{x}$ is normally distributed with mean $\hat{\mu}(\mathbf{x})$ and variance $\hat{\sigma}^2(\mathbf{x})$ given by,

$$\hat{\mu}(\mathbf{x}) = \mu(\mathbf{x}) + \mathbf{k}(\mathbf{x}, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \omega^2 \mathbf{I}]^{-1} \mathbf{y}, \quad (2)$$

$$\hat{\sigma}^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x}, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \omega^2 \mathbf{I}]^{-1} \mathbf{k}(\mathbf{X}, \mathbf{x}) + \omega^2, \quad (3)$$

where $\mathbf{K}(\mathbf{X}, \mathbf{X})$ is the kernel. The entry $\mathbf{K}_{\mathbf{x}_l, \mathbf{x}_m}$ gives the covariance between two inputs $\mathbf{x}_l$ and $\mathbf{x}_m$. $\mu(\mathbf{x})$ in Equation 2 is the prior mean of output value at $\mathbf{x}$.

We use a zero-mean prior and a squared-exponential kernel where $\mathbf{K}_{\mathbf{x}_l, \mathbf{x}_m}$ is given by,

$$\mathbf{K}_{\mathbf{x}_l, \mathbf{x}_m} = \sigma^2 \exp\left(-\frac{1}{2} \sum_{d=1}^{d=D} \left(\frac{\mathbf{x}_{dl} - \mathbf{x}_{dm}}{l_d}\right)^2\right) + \omega^2, \quad (4)$$

σ^2, l_d and ω^2 are hyperparameters that can be either set by the user or learned online through the training data.

In the GPQ-MFRL algorithm, we use GPs to learn Q-values. GPs are proved to be consistent function approximators in RL with convergence guarantees (Melo, Meyn, and Ribeiro 2008; Chowdhary et al. 2014). A set of state-action pairs is the input to GP and Q–values are the output/observation values to be predicted.

[1] Upper and lower bold face letters represent matrices and vectors respectively. Scalar values are represented by a lower-case letter.

Related Work

In model-based approaches, GPs are commonly used to learn transition models for agents moving in the real world (Dames, Tokekar, and Kumar 2015) and have been used in RL to learn the transition function (Rasmussen and Kuss 2003) and the reward function (Deisenroth 2010). GPs have also been used in model-free approaches for approximating the Q-values in continuous state-action spaces (Engel, Mannor, and Meir 2003). This was extended to the GP-SARSA algorithm which includes online action selection and policy improvement steps (Engel, Yaakov and Mannor, Shie and Meir, Ron 2005). The authors used the posterior variance to compute confidence intervals around the value estimate and note that the variance could be used for exploration.

Using multiple approximations of real-world environments has previously been considered in the literature (Abbeel, Quigley, and Ng 2006; Taylor, Stone, and Liu 2007; Torrey and Shavlik 2009). (Yao and Doretto 2010) extended the transfer learning framework for transferring knowledge from multiple sources. (Yosinski et al. 2014) show the transfer of features in deep neural networks and demonstrate that initializing a network with transferred features from almost any number of layers can generalize to fine-tuning to the target dataset. Unlike these methods, the MFRL algorithm allows for bi-directional switching, where the agent is allowed to go back to lower fidelity simulator to gather additional samples.

The MFRL algorithm was introduced by (Cutler, Walsh, and How 2015) where they showed how to leverage the model-based RMax algorithm to reduce the number of samples. Our empirical results demonstrate that the number of samples for MFRL can be brought further down by leveraging GPs.

Algorithm Description

In this section, we first describe our algorithm. We compare the proposed algorithm with baseline strategies through simulations. A flowchart of the proposed algorithm is shown in Figure 2.

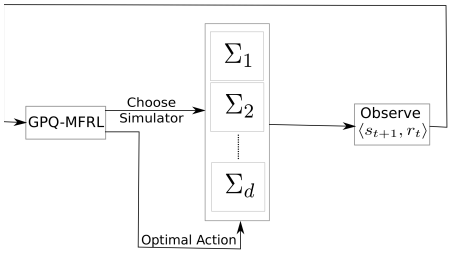


Figure 2: Overview of our model-free algorithm (GPQ-MFRL). Simulators are represented by $\Sigma_1, \Sigma_2, \ldots, \Sigma_d$. GPQ-MFRL directly estimates Q-values using GPs.

GPQ-MFRL Algorithm

The agent learns the optimal Q-values using GPs directly, instead of learning the model first. The underlying assumption is that nearby state-action pairs will produce similar Q-values. This assumption can also be applied to problems where the states and actions are discrete but the transition function implies some sense of continuity. We choose the squared-exponential kernel because it models the spatial correlation we expect to see in a robot. However, any appropriate kernel can be used. We use a separate GP per simulator to estimate the Q-values using only data collected in that simulator.

Algorithm 1 GPQ-MFRL Algorithm

1: **procedure**
2: **Input:** confidence parameters σ_{th} and σ_{th}^{sum}; simulator chain $\langle \mathbf{\Sigma},$ fidelity parameters $\boldsymbol{\beta},$ state mappings $\boldsymbol{\rho} \rangle$; $\mathcal{L}$.
3: **Initialize:** $\hat{Q}_i$ = initialize GP for $i \in \{1, \ldots, d\}$; state s_0 in simulator Σ_1; $i \leftarrow 1$; $change =$ FALSE.
4: **Initialize:** $t \leftarrow 0$; $\mathcal{D}_i \leftarrow \{\}$ for $i \in \{1, \ldots, d\}$.
5: **while** terminal condition is not met
6: $a_t \leftarrow$ CHOOSEACTION(s_t, i)
7: **if** $\sigma_i(s_t, a_t) \leq \sigma_{th}$: $change =$ TRUE
8: **if** $\sigma\left(\rho_i(s_t), a_t\right) > \sigma_{th}$ and $change$ and $i > 1$
9: $s_t \leftarrow \rho_i(s_t)$, $i \leftarrow i - 1$, continue
10: $\langle r_t, s_{t+1} \rangle \leftarrow$ execute action a_t in Σ_i
11: append $\langle s_t, a_t, s_{t+1}, r_t \rangle$ to $\mathcal{D}_i$
12: $\mathcal{Y}_i \leftarrow \{\}$
13: **for** $\langle s_t, a_t, s_{t+1}, r_t \rangle \in \mathcal{D}_i$ //batch training//
14: $y_t \leftarrow r_t + \gamma \max_a \hat{Q}_i\left(s_{t+1}, a\right)$
15: append $\langle s_t, a_t, y_t \rangle$ to $\mathcal{Y}_i$
16: $\hat{Q}_i \leftarrow$ update GP$_i$ using $\mathcal{Y}_i$
17: **if** $\sum_{j=t}^{j=t-\mathcal{L}} \sigma_i(s_j, a_j) \leq \sigma_{th}^{sum}$ and $t > \mathcal{L}$ and $i < d$
18: $s_t \leftarrow \rho_{i+1}^{-1}\left(s_t\right)$, $i \leftarrow i + 1$
19: $change =$ FALSE
20: **end procedure**
21:
22: **procedure** CHOOSEACTION(s, i)
23: **for** $a \in \mathcal{A}(s)$
24: $Q(s, a) = \hat{Q}_{i-1}\left(\rho_i(s), a\right) + \beta_i$
25: **for** $k \in \{i, \ldots, d\}$
26: $s_k = \rho_k^{-1} \ldots \rho_{i+2}^{-1} \rho_{i+1}^{-1}(s)$
27: **if** $\sigma_k(s_k, a) \leq \sigma^{th}$: $Q(s, a) = \hat{Q}_k(s_k, a)$
28: return $\arg\max_a Q(s, a)$
29: **end procedure**

Algorithm 1 gives the details of the proposed framework. GPQ-MFRL continues to collect samples in the same simulator until the agent is confident about its optimal actions. If the running sum of the variances is below a threshold (Line 17), this suggests that the robot has found a good policy with high confidence in the current simulator and it must advance to the next one (Line 18).

GPQ-MFRL keeps track of the variance of the $\mathcal{L}$ most recently visited state-action pairs in the current simulator. If the running sum of the variances is below a threshold (Line 15), this suggests that the robot is confident about its actions in the current simulator and can advance to the next one. In the original work (Cutler, Walsh, and How 2015),

the agent switches to the higher fidelity simulator after a certain number of known state-action pairs were encountered. In our implementation (Line 7), the model of current environment changes if the posterior variance for a state-action pair drops below a threshold value (*i.e.*, agent has a sufficiently accurate estimate of the transitions from that state). The algorithm checks if the agent has a sufficiently accurate estimate of optimal Q-values in the previous simulator (Line 8). Lines 10–15 describe the main body of the algorithm where the agent records the observed transitions in $\mathcal{D}_i$. We update target values (Line 14) for every transition as more data gets collected in $\mathcal{D}_i$ (Line 13). The GP model is updated after every step (Line 16).

The agent utilizes the experiences collected in higher simulators (Lines 25–27) to choose the optimal action in the current simulator (Line 6). Specifically, it checks for the maximum fidelity simulator in which the posterior variance for (s, a) is less than a threshold σ_{th}. If one exists, it utilizes the Q-values from the highest known simulator to choose the next action in the current simulator. If no such higher simulator exists, the Q-values from the previous simulator (Line 24) are considered to choose the next action in the current simulator with an additive fidelity parameter β.

GPQ-MFRL performs a batch retraining every time the robot collects the new sample in a simulator (Lines 13–15). During the batch retraining, the algorithm updates the target values in previously collected training data using the knowledge gained by collecting new samples. Then these updated target values are used to predict the Q-values using GPs (Line 16). As the amount of data grows, updating the GP can become computationally expensive. However, we can prune the dataset using sparse GP techniques (Engel, Yaakov and Mannor, Shie and Meir, Ron 2005). It is non-trivial to choose values for confidence bounds but for the current experiments we chose the σ_{th}^{sum} to be ten percent of the maximum Q-value possible and σ_{th} to be one fifth of σ_{th}^{sum}.

Results

We use three environments to simulate GPQ-MFRL. Σ_1 is Python-based simulator Pygame (Shinners 2011), Σ_2 is a Gazebo environment, and Σ_3 is the real world.

Evaluating the GPQ-MFRL Algorithm

We use three environments (Figure 1) to demonstrate the GPQ-MFRL algorithm. The task of the robot is to navigate through a given environment without crashing into the obstacles, assuming the robot has no prior information about the environments. There is no goal state.

The robot has a laser sensor that gives the distance readings from the obstacles along seven equally spaced directions. The angle between two consecutive measurement directions is $\frac{\pi}{8}$ radians and range of measurements is 5 meters. The actual robot has a Hokuyo laser sensor that operates in the same configuration. Distance measurements along the seven directions serve as the state in the environment. Hence, we have a seven-dimensional continuous state space: $\mathcal{S} \in (0, 5]^7$ (Figure 1), where individual state dimension corresponds to the distance measurement along a particular direction. Note that this is a different state representation than the standard one used in navigation where the X and Y coordinates of the robot are used as the state. Since we use the sensor input directly as the state, the learned policy maps the sensor inputs directly to the actions, thereby learning a polygon that avoids collisions. This can easily generalize to any environment.

The linear speed of the robot is held constant at 0.2 m/sec. The robot can choose its angular velocity from nineteen possible options: $\{-\frac{\pi}{9}, -\frac{\pi}{8}, \ldots, \frac{\pi}{9}\}$. The reward in each state is set to be the sum of laser readings from seven directions except when the robot hits the obstacle when it gets a reward of -50.

We train the GP regression, $Q(\mathbf{s}, a) : \mathbb{R}^8 \rightarrow \mathbb{R}$. Hyperparameters of the squared-exponential kernel were calculated off-line by minimizing the negative log marginal likelihood of 2000 training points which were collected by letting the robot run in the real world directly. The parameter values for experiments in this section are given in Table 1.

Table 1: Parameters used in GPQ-MFRL

Description	Type	Value
Hyperparameters	σ	102.74
	l	[2.1, 5.1, 14, 6.2, 15, 2, 2, 1]
	ω^2	20
Confidence parameters	σ_{th}^{sum}	60
	σ_{th}	15
Algorithm	$\mathcal{L}$	5

Average Cumulative Reward in the Real-World In Figure 3, we compare GPQ-MFRL algorithm with three other baseline strategies by plotting the average cumulative reward collected by the robot as a function of samples collected in the real world. Three baseline strategies are,

1. Directly collecting samples in the real world without the simulators (Direct),
2. Collect hundred samples in one simulator and transfer the policy to the Pioneer robot with no further learning in the real world (Frozen Policy) and
3. Collect hundred samples in one simulator and transfer the policy to the robot while continuing to learn in the real world (Transferred Policy).

Policy Improvement over Time Figure 4 shows the absolute percentage change in the sum of the value functions with respect to last estimated sum of value functions and average predictive variance for states $\{1, 3, 5\}^7$ in all the three simulators. Observe that initially most of the samples are collected in the simulator, whereas over time the samples are collected mostly in the real world. The simulators help the robot to make its value estimates converge quickly as observed by a sharp dip in the first white region. Note that GP updates for i^{th} simulator $(\hat{Q}_i)$ are made only when the robot is running in i^{th} simulator.

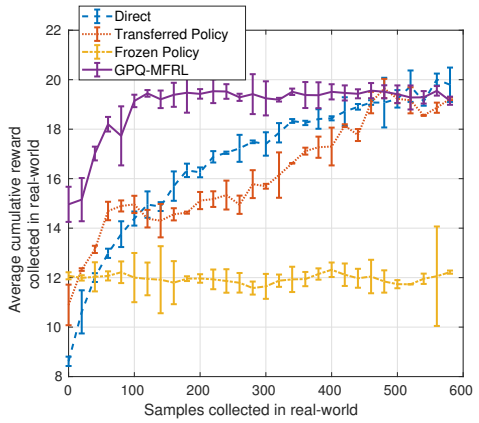
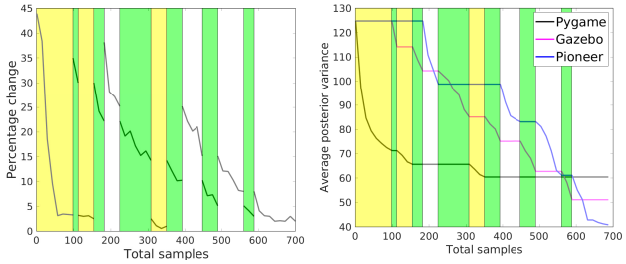


Figure 3: Average cumulative reward collected by Pioneer in real-world as a function of the samples collected in real-world. The plot shows the average and standard deviation of 2 trials.



(a) Sum of absolute change in value functions

(b) Average variances in value function estimations

Figure 4: Yellow, green and white regions correspond to the samples collected in the Pygame, Gazebo and real-world environments respectively. Plots are for state set $\{1, 3, 5\}^7$.

Discussion and Future Work

The GP-based MFRL algorithm provides a general RL technique that is particularly suited for robotics. We demonstrated empirically that the GP-based MFRL algorithm finds the optimal policies using fewer samples than the baseline algorithms. We plan to analyze the algorithms in order to provide theoretical bounds for the sample complexity. (Strehl et al. 2006) show that the sample complexity of RMax algorithm after ignoring logarithmic factors is $\tilde{O}\left(\frac{S^2A}{\epsilon^3(1-\gamma)^6}\right)$. Here S is the size of state space, A is the number of actions available to the agent, γ is the discount factor and ϵ denotes the desired accuracy until the algorithm converges.

One disadvantage of using GPs is that as the number of observations increase, the time taken to perform GP updates also increases in with cubic complexity. However, we can use adaptive sample selection techniques (Osborne 2010) as well as numerical optimization techniques (Engel, Yaakov and Mannor, Shie and Meir, Ron 2005) to speed up this process.

References

Abbeel, P.; Quigley, M.; and Ng, A. Y. 2006. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, 1–8. ACM.

Brafman, R. I., and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3(Oct):213–231.

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

Cutler, M., and How, J. P. 2016. Autonomous drifting using simulation-aided reinforcement learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 5442–5448.

Cutler, M.; Walsh, T. J.; and How, J. P. 2015. Real-world reinforcement learning via multifidelity simulators. *IEEE Transactions on Robotics* 31(3):655–671.

Dames, P.; Tokekar, P.; and Kumar, V. 2015. Detecting, localizing, and tracking an unknown number of moving targets using a team of mobile robots. In *International Symposium on Robotics Research (ISRR)*.

Deisenroth, M. P. 2010. *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing.

Engel, Y.; Mannor, S.; and Meir, R. 2003. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 154–161.

Engel, Yaakov and Mannor, Shie and Meir, Ron. 2005. Reinforcement learning with Gaussian Processes. In *Proceedings of the 22nd international conference on Machine learning*, 201–208. ACM.

Grande, R.; Walsh, T.; and How, J. 2014. Sample efficient reinforcement learning with gaussian processes. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1332–1340.

Jung, T., and Stone, P. 2010. Gaussian processes for sample efficient reinforcement learning with rmax-like exploration. In *Proceedings of the European Conference on Machine Learning*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

Li, L.; Littman, M. L.; Walsh, T. J.; and Strehl, A. L. 2011. Knows what it knows: a framework for self-aware learning. *Machine learning* 82(3):399–443.

M. Cutler and J. P. How. 2015. Efficient reinforcement learning for robots using informative simulated priors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2605–2612.

Melo, F. S.; Meyn, S. P.; and Ribeiro, M. I. 2008. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*, 664–671. ACM.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Osborne, M. 2010. *Bayesian Gaussian processes for sequential prediction, optimisation and quadrature*. Ph.D. Dissertation, University of Oxford.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Rasmussen, C. E., and Kuss, M. 2003. Gaussian processes in reinforcement learning. NIPS.

Rasmussen, C. E., and Williams, C. K. I. 2006. *Gaussian Processes for Machine Learning*. the MIT Press.

Shinners, P. 2011. Pygame. `http://pygame.org/`.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Strehl, A. L.; Li, L.; Wiewiora, E.; Langford, J.; and Littman, M. L. 2006. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, 881–888. ACM.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

Taylor, M. E.; Stone, P.; and Liu, Y. 2007. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research* 8(Sep):2125–2167.

Torrey, L., and Shavlik, J. 2009. Transfer learning: Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

Yao, Y., and Doretto, G. 2010. Boosting for transfer learning with multiple sources. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, 1855–1862. IEEE.

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H. 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, 3320–3328.