Sample Efficient Reinforcement Learning with Gaussian Processes

Robert C. Grande RGRANDE@MIT.EDU
Thomas J. Walsh THOMASJWALSH@GMAIL.COM
Jonathan P. How JHOW@MIT.EDU
Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA 02139 USA

Abstract

This paper derives sample complexity results for using Gaussian Processes (GPs) in both model-based and model-free reinforcement learning (RL). We show that GPs are KWIK learnable, proving for the first time that a model-based RL approach using GPs, GP-Rmax, is sample efficient (PAC-MDP). However, we then show that previous approaches to model-free RL using GPs take an exponential number of steps to find an optimal policy, and are therefore not sample efficient. The third and main contribution is the introduction of a model-free RL algorithm using GPs, DGPQ, which is sample efficient and, in contrast to model-based algorithms, capable of acting in real time, as demonstrated on a five-dimensional aircraft simulator.

1. Introduction

In Reinforcement Learning (RL) (Sutton & Barto, 1998), several new algorithms for efficient exploration in continuous state spaces have been proposed, including GP-Rmax (Jung & Stone, 2010) and C-PACE (Pazis & Parr, 2013). In particular, C-PACE was shown to be PAC-MDP, an important class of RL algorithms that obtain an optimal policy in a polynomial number of exploration steps. However, these approaches require a costly fixed-point computation on *each* experience, making them ill-suited for real-time control of physical systems, such as aircraft. This paper presents a series of sample complexity (PAC-MDP) results for algorithms that use Gaussian Processes (GPs) (Rasmussen & Williams, 2006) in RL, culminating with the introduction of a PAC-MDP model-free algorithm which does not require this fixed-point computation and is better suited for real-time learning and control.

First, using the KWIK learning framework (Li et al., 2011), we provide the first-ever sample complexity analysis of GP learning under the conditions necessary for RL. The result of this analysis actually proves that the previously described model-based GP-Rmax is indeed PAC-MDP. However, it still cannot be used in real time, as mentioned above. Our second contribution is in model-free RL, in which a GP is used to directly model the Q-function. We show that existing model-free algorithms (Engel et al., 2005; Chung et al., 2013) that use a single GP may require an exponential (in the discount factor) number of samples to reach a near-optimal policy.

The third, and primary, contribution of this work is the introduction of the first model-free continuous state space PAC-MDP algorithm using GPs: Delayed-GPQ (DGPQ). DGPQ represents the current value function as a GP, and updates a separately stored value function only when sufficient outlier data has been detected. This operation "overwrites" a portion of the stored value function and resets the GP confidence bounds, avoiding the slowed convergence rate of the naïve model-free approach.

The underlying analogy is that while GP-Rmax and C-PACE are generalizations of model-based Rmax (Brafman & Tennenholtz, 2002), DGPQ is a generalization of model-free Delayed Q-learning (DQL) (Strehl et al., 2006; 2009) to general continuous spaces. That is, while early PAC-MDP RL algorithms were model-based and ran a planner after each experience (Li et al., 2011), DQL is a PAC-MDP algorithm that performs *at most* a single Bellman backup on each step. In DGPQ, the delayed updates of DQL are adapted to continuous state spaces using a GP.

We prove that DGPQ is PAC-MDP, and show that using a sparse online GP implementation (Csató & Opper, 2002) DGPQ can perform in real-time. Our empirical results, including those on an F-16 simulator, show DGPQ is both sample efficient and orders of magnitude faster in per-sample computation than other PAC-MDP continuous-state learners.

Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

2. Background

We now describe background material on Reinforcement Learning (RL) and Gaussian Processes (GPs).

2.1. Reinforcement Learning

We model the RL environment as a Markov Decision Process (Puterman, 1994) $M = \langle S, A, R, T, \gamma \rangle$ with a potentially infinite set of states S, finite actions A, and $0 \leq \gamma < 1$. Each step elicits a reward $R(s, a) \mapsto [0, R_{\max}]$ and a stochastic transition to state $s' \sim T(s, a)$. Every MDP has an optimal value function $Q^*(s, a) = R(s, a) + \gamma \int_{s'} T(s, a, s')V^*(s')$ where $V^*(s) = \max_a Q^*(s, a)$ and corresponding optimal policy $\pi^* : S \mapsto A$. Note the bounded reward function means $V^* \in [0, V_{\max}]$.

In RL, an agent is given S,A, and γ and then acts in M with the goal of enacting π^*. For value-based methods (as opposed to policy search (Kalyanakrishnan & Stone, 2009)), there are roughly two classes of RL algorithms: model-based and model-free. Model-based algorithms, such as KWIK-Rmax (Li et al., 2011), build models of T and R and then use a planner to find Q^*. Many model-based approaches have *sample efficiency* guarantees, that is bounds on the amount of *exploration* they perform. We use the PAC-MDP definition of sample complexity (Strehl et al., 2009). The definition uses definitions of the covering number of a set (adapted from (Pazis & Parr, 2013)).

Definition 1 The **Covering Number** $\mathcal{N}_U(r)$ of a compact domain $U \subset \mathbb{R}^n$ is the cardinality of the minimal set $C = \{c_i, \ldots, c_{N_c}\}$ s.t. $\forall x \in U$, $\exists c_j \in C$ s.t. $d(x, c_j) \leq r$, where $d(\cdot, \cdot)$ is some distance metric.

Definition 2 Algorithm $\mathcal{A}$ (non-stationary policy $\mathcal{A}_t$) with accuracy parameters ϵ and δ in an MDP of size $\mathcal{N}_{SA}(r)$ is said to be Probably Approximately Correct for MDPs (PAC-MDP) if, with probability $1-\delta$, it takes no more than a polynomial (in $\langle \mathcal{N}_{SA}(r), \frac{1}{1-\gamma}, \frac{1}{\epsilon}, \frac{1}{\delta} \rangle$) number of steps where $V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \epsilon$.

By contrast, *model-free* methods such as Q-Learning (Watkins, 1992) build Q^* directly from experience without explicitly representing T and R. Generally model-based methods are more sample efficient but require more computation time for the planner. Model-free methods are generally computationally light and can be applied without a planner, but need (sometimes exponentially) more samples, and are usually not PAC-MDP. There are also methods that are not easily classified in these categories, such as C-PACE (Pazis & Parr, 2013), which does not explicitly model T and R but performs a fixed-point operation for planning.

2.2. Gaussian Processes

This paper uses GPs as function approximators both in model-based RL (where GPs represent T and R) and model-free RL (where GPs represent Q^*), while showing how to maintain PAC-MDP sample efficiency in both cases. A GP is defined as a collection of random variables, any finite subset of which has a joint Gaussian distribution (Rasmussen & Williams, 2006) with prior mean $\mu(x)$ and covariance kernel $k(x, x')$. In this paper, we use the radial basis function (RBF), $k(x, x') = \exp(-\frac{\|x-x'\|^2}{2\theta^2})$.

The elements of the GP kernel matrix $K(X, X)$ are defined as $K_{i,j} = k(x_i, x_j)$, and $k(X, x')$ denotes the kernel vector. The conditional probability distribution of the GP at new data point, x', can then be calculated as a normal variable (Rasmussen & Williams, 2006) with mean $\hat{\mu}(x') = k(X, x')^T [K(X, X) + \omega_n^2 I]^{-1} y$, and covariance $\mathbf{\Sigma}(x') = k(x', x') - k(X, x')^T [K(X, X) + \omega_n^2 I]^{-1} k(X, x')$, where y is the set of observations, X is the set of observation input locations, and ω_n^2 is the measurement noise variance. For computational and memory efficiency, we employ an online sparse GP approximation (Csató & Opper, 2002) in the experiments.

2.3. Related Work

GPs have been used for both model-free and model-based RL. In model-free RL, GP-Sarsa (Engel et al., 2005) has been used to model a value function and extended to off-policy learning (Chowdhary et al., 2014), and for (heuristically) better exploration in iGP-Sarsa (Chung et al., 2013). However, we prove in Section 4.1 that these algorithms may require an exponential (in $\frac{1}{1-\gamma}$) number of samples to reach optimal behavior since they only use a single GP to model the value function.

In model-based RL, the PILCO algorithm trained GPs to represent T, and then derived policies using policy search (Deisenroth & Rasmussen, 2011). However, PILCO does not include a provably efficient (PAC-MDP) exploration strategy. GP-Rmax (Jung & Stone, 2010) does include an exploration strategy, specifically replacing areas of low confidence in the (T and R) GPs with high valued states, but no theoretical results are given in that paper. In Section 3, we show that GP-Rmax actually is PAC-MDP, but the algorithm's planning phase (comparable to the C-PACE planning in our experiments) after each update makes it computationally infeasible for real-time control.

REKWIRE (Li & Littman, 2010) uses KWIK linear regression for a model-free PAC-MDP algorithm. However, REKWIRE needs H separate approximators for finite horizon H, and assumes Q can be modeled as a linear function, in contrast to the general continuous functions considered in our work. The C-PACE algorithm (Pazis & Parr, 2013) has already been shown to be PAC-MDP in the continuous setting, though it does not use a GP representation. C-PACE stores data points that do not have close-enough neighbors to be considered "known". When it adds a new data point, the Q-values of each point are calculated by a value-iteration like operation. The computation for this operation grows with the number of datapoints and so (as shown in our experiments) the algorithm may not be able to act in real time.

3. GPs for Model-Based RL

In model-based RL (MBRL), models of T and R are created from data and a planner derives π^*. Here we review the KWIK-Rmax MBRL architecture, which is PAC-MDP. We then show that GPs are a KWIK-learnable representation of T and R, thereby proving that GP-Rmax (Jung & Stone, 2010) is PAC-MDP given an exact planner.

3.1. KWIK Learning and Exploration

The "Knows What it Knows" (KWIK) framework (Li et al., 2011) is a supervised learning framework for measuring the number of times a learner will admit uncertainty. A KWIK learning agent is given a hypothesis class $H : X \mapsto Y$ for inputs X and outputs Y and parameters ϵ and δ. With probability $1 - \delta$ over a run, when given an adversarially chosen input x_t, the agent must, either (1) predict $\hat{y}_t$ if $||y_t - \hat{y}_t|| \leq \epsilon$ where y_t is the true expected output ($E[h^*(x)] = y_t$) or (2) admit "I don't know" (denoted $\perp$). A representation is *KWIK learnable* if an agent can KWIK learn any $h^* \in H$ with only a polynomial (in $\langle |H|, \frac{1}{\epsilon}, \frac{1}{\delta} \rangle$) number of $\perp$ predictions. In RL, the KWIK-Rmax algorithm (Li et al., 2011), uses KWIK learners to model T and R and replaces the value of any $Q^*(s,a)$ where $T(s,a)$ or $R(s,a)$ is $\perp$ with a value of $V_{\max} = \frac{R_{\max}}{1-\gamma}$. If T and R are KWIK-learnable, then the resulting KWIK-Rmax RL algorithm will be PAC-MDP under Definition 2. We now show that a GP is KWIK learnable.

3.2. KWIK Learning a GP

First, we derive a metric for determining if a GP's mean prediction at input x is ϵ-accurate with high probability, based on the variance estimate at x.

Lemma 1 Consider a GP trained on samples $\vec{y} = [y_1, \ldots, y_t]$ which are drawn from $p(y \mid x)$ at input locations $X = [x_1, \ldots, x_t]$, with $\mathbb{E}[y \mid x] = f(x)$ and $y_i \in [0, V_m]$. If the predictive variance of the GP at $x_i \in X$ is

$$\sigma^2(x_i) \leq \sigma^2_{tol} = \frac{2\omega_n^2 \epsilon_1^2}{V_m^2 \log(\frac{2}{\delta_1})} \qquad (1)$$

then the prediction error at x_i is bounded in probability: $Pr\left\{|\hat{\mu}(x_i) - f(x_i)| \geq \epsilon_1\right\} \leq \delta_1$.

Proof sketch Here we ignore the effect of the GP-prior, which is considered in depth in the Supplementary material. Using McDiarmid's inequality we have that $Pr\{|\mu(x) - f_1(x)| \geq \epsilon_1\} \leq 2\exp\left(-\frac{2\epsilon_1^2}{\sum c_i^2}\right)$, where c_i is the maximum amount that y_i can alter the prediction value $\mu(x)$. Using algebraic manipulation, Bayes law, and noting that the max singular value $\bar{\sigma}(K(X,X)(K(X,X) + \omega^2 I)^{-1}) < 1$, it can be shown that $\sum c_i^2 \leq \sigma^2(x_i) V_m^2 / \omega^2$. Substituting $\sigma^2(x_i)$ to McDiarmid's inequality and rearranging yields (1). See supplemental material.

Lemma 1 gives a sufficient condition to ensure that, with high probability, the mean of the GP is within ϵ_1 of $E[h^*(x)]$. A KWIK agent can now be constructed that predicts $\perp$ if the variance is greater than σ^2_{tol} and otherwise predicts $\hat{y}_t = \mu(x)$, the mean prediction of the GP. Theorem 1 bounds the number of $\perp$ predictions by such an agent, and when $\delta_1 = \frac{\delta}{\mathcal{N}_U(r(\frac{1}{2}\sigma^2_{tol}))}$, establishes the KWIK learnability of GPs. For ease of exposition, we define the equivalent distance as follows:

Definition 3 The **equivalent distance** $r(\epsilon_{tol})$ is the maximal distance s.t. $\forall x, c \in U$, if $d(x,c) \leq r(\epsilon_{tol})$, then the linear independence test $\gamma(x,c) = k(x,x) - k(x,c)^2/k(c,c) \leq \epsilon_{tol}$.

Theorem 1 Consider a GP model trained over a compact domain $U \subset \mathbb{R}^n$ with covering number $\mathcal{N}_U(r(\frac{1}{2}\sigma^2_{tol}))$. Let the observations $\vec{y} = [y_1, \ldots, y_n]$ be drawn from $p(y \mid x)$, with $\mathbb{E}[y \mid x] = f(x)$, and x drawn adversarially. Then, the worst case bound on the number of samples for which

$$Pr\left\{|\hat{\mu}(x) - f(x)| \geq \epsilon_1\right\} \leq \delta_1, \forall x \in U \qquad (2)$$

and the GP is forced to return $\perp$ is at most

$$m = \left(\frac{4V_m^2}{\epsilon_1^2} \log\left(\frac{2}{\delta_1}\right)\right) \mathcal{N}_U\left(r\left(\frac{1}{2}\sigma^2_{tol}\right)\right). \qquad (3)$$

Furthermore, $\mathcal{N}_U\left(r\left(\frac{1}{2}\sigma^2_{tol}\right)\right)$ grows polynomially with $\frac{1}{\epsilon_1}$ and V_m for the RBF kernel.

Proof sketch If $\sigma^2(x) \leq \sigma^2_{tol}$, $\forall x \in U$, then (2) follows from Lemma 1. By partitioning the domain into the Voronoi regions around the covering set C, it can be shown using the covariance equation (see supplemental material), that given n_V samples in a Voronoi region $c_i \in C$, $\sigma^2(x) \leq \frac{n_V \frac{1}{2}\sigma^2_{tol} + \omega^2}{n_V + \omega^2}$ everywhere in the region. Solving for n_V, we find that $n_V \geq \frac{2\omega^2}{\sigma^2_{tol}}$ is sufficient to drive $\sigma^2(x) \leq \sigma^2_{tol}$. Plugging in n_V into (1), we have that $n_V = \left(\frac{V_m^2}{\epsilon_1^2} \log\left(\frac{2}{\delta_1}\right)\right)$ points reduce the variance at c_i below σ^2_{tol}. Therefore, the total number of points we can sample anywhere in U before reducing the variance below

σ^2_{tol} everywhere is equal to the sum of points n_V over all regions, $\mathcal{N}_U$. The total probability of an incorrect prediction is given by the union bound of an incorrect prediction in any region, $\sum_{c_i}\delta_1 = \delta_1\mathcal{N}_U = \delta$. See supplemental material for proof that $\mathcal{N}_U$ grows polynomially with $\frac{1}{\epsilon_1}$ and V_m.

Intuitively, the KWIK learnability of GPs can be explained as follows. By knowing the value at a certain point within some tolerance $\frac{\epsilon}{2}$, the Lipschitz smoothness assumption means there is a nonempty region around this point where values are known within a larger tolerance ϵ. Therefore, given sufficient observations in a neighborhood, a GP is able to generalize its learned values to other nearby points. Lemma 1 relates the error at any of these points to the predictive variance of the GP, so a KWIK agent using a GP can use the variance prediction to choose whether to predict $\perp$ or not. As a function becomes less smooth, the size of these neighborhoods shrinks, increasing the covering number, and the number of points required to learn over the entire input space increases.

Theorem 1, combined with the KWIK-Rmax Theorem (Theorem 3 of (Li et al., 2011)) establishes that the previously proposed GP-Rmax algorithm (Jung & Stone, 2010), which learns GP representations of T and R and replaces uncertainty with $V_{\max}$ values, is indeed PAC-MDP. GP-Rmax was empirically validated in many domains in this previous work but the PAC-MDP property was not formally proven. However, GP-Rmax relies on a planner that can derive Q^* from the learned T and R, which may be infeasible in continuous domains, especially for real-time control.

4. GPs for Model-Free RL

Model-free RL algorithms, such as Q-learning (Watkins, 1992), are often used when planning is infeasible due to time constraints or the size of the state space. These algorithms do not store T and R but instead try to model Q^* directly through incremental updates of the form: $Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha\left(r_t + \gamma V^*_t(s_{t+1})\right)$. Unfortunately, most Q-learning variants have provably exponential sample complexity, including optimistic/greedy Q-learning with a linearly decaying learning rate (Even-Dar & Mansour, 2004), due to the incremental updates to Q combined with the decaying α term.

However, the more recent Delayed Q-learning (DQL) (Strehl et al., 2006) *is* PAC-MDP. This algorithm works by initializing $Q(s,a)$ to $V_{\max}$ and then *overwriting* $Q(s,a) = \frac{\sum_{i=1}^m r_i + \gamma V(s'_i)}{m}$ after m samples of that $\langle s,a\rangle$ pair have been seen. In section 4.1, we show that using a single GP results in exponential sample complexity, like Q-learning; in section 4.2, we show that using a similar overwriting approach to DQL achieves sample efficiency.

4.1. Naïve Model-free Learning using GPs

We now show that a naïve use of a single GP to model Q^* will result in exponentially slow convergence similar to greedy Q-learning with a linearly decaying α. Consider the following model-free algorithm using a GP to store values of $\hat{Q} = Q_t$. A GP for each action (GP_a) is initialized optimistically using the prior. At each step, an action is chosen greedily and GP_a is updated with an input/output sample: $\langle x_t, r_t + \gamma \max_b GP_b(s_{t+1})\rangle$. We analyze the worst-case performance through a toy example and show that this approach requires an exponential number of samples to learn Q^*. This slow convergence is intrinsic to GPs due to the variance reduction rate and the non-stationarity of the $\hat{Q}$ estimate (as opposed to T and R, whose sampled values do not depend on the learner's current estimates). So, any model-free algorithm using a GP that does not reinitialize the variance will have the same worst-case complexity as in this toy example.

Consider an MDP with one state s and one action a that transitions deterministically back to s with reward $r = 0$, and discount factor γ. The Q function is initialized optimistically to $\hat{Q}_0(s) = 1$ using the GP's prior mean. Consider the naïve GP learning algorithm described above, kernel $k(s,s) = 1$, and measurement noise ω^2 to predict the Q-function using the GP regression equations. We can analyze the behavior of the algorithm using induction.

Consider the first iteration: $\hat{Q}_0 = 1$, and the first measurement is γ. In this case, the GP prediction is equivalent to the MAP estimate of a random variable with Gaussian prior and linear measurements subject to Gaussian noise.

$$\hat{Q}_1 = \frac{\omega^2}{\sigma_0^2 + \omega^2}1 + \frac{\sigma_0^2}{\sigma_0^2 + \omega^2}\gamma \tag{4}$$

Recursively, $\hat{Q}_{i+1} = (\frac{\omega^2}{\sigma_i^2+\omega^2} + \frac{\sigma_i^2}{\sigma_i^2+\omega^2}\gamma)\hat{Q}_i$ where $\sigma_i^2 = \frac{\omega^2\sigma_0^2}{\omega^2+i\sigma_0^2}$ is the GP variance at iteration i. Substituting for σ_i^2 and rearranging yields a recursion for the prediction of $\hat{Q}_i$ at each iteration,

$$\hat{Q}_{i+1} = \frac{\omega^2 + \sigma_0^2(i+\gamma)}{\omega^2 + \sigma_0^2(i+1)}\hat{Q}_i \tag{5}$$

From (Even-Dar & Mansour, 2004), we have that a series of the form $\hat{Q}_{i+1} = \frac{i+\gamma}{i+1}\hat{Q}_i$ will converge to ϵ exponentially slowly in terms of $\frac{1}{\epsilon}$ and $\frac{1}{1-\gamma}$. However, for each term in our series, we have

$$\frac{\frac{\omega^2}{\sigma_0^2} + i + \gamma}{\frac{\omega^2}{\sigma_0^2} + i + 1} \geq \frac{i+\gamma}{i+1} \tag{6}$$

The modulus of contraction is always at least as large as the RHS, so the series convergence to ϵ (since the true

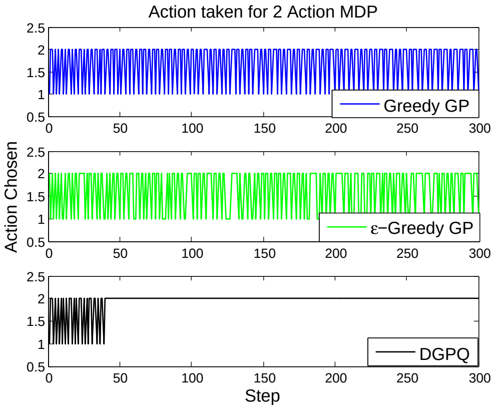

Figure 1. Single-state MDP results: DGPQ converges quickly to the optimal policy while the naïve GP implementations oscillate.

Algorithm 1 Delayed GPQ (DGPQ)

```
 1: Input: GP kernel $k(\cdot,\cdot)$, Lipschitz Constant $L_Q$, Environment $Env$, Actions $A$, initial state $s_0$, discount $\gamma$, threshold $\sigma^2_{tol}$, $\epsilon_1$
 2: for $a \in A$ do
 3:     $\hat{Q}_a = \emptyset$
 4:     $GP_a = GP$.init($\mu = \frac{R_{max}}{1-\gamma}, k(\cdot,\cdot)$)
 5: for each timestep $t$ do
 6:     $a_t = \arg\max_a \hat{Q}_a(s_t)$ by Eq 7
 7:     $\langle r_t, s_{t+1}\rangle = Env$.takeAct($a_t$)
 8:     $q_t = r_t + \gamma \max_a \hat{Q}_a(s_{t+1})$
 9:     $\sigma_1^2 = GP_{a_t}$.variance($s_t$)
10:     if $\sigma_1^2 > \sigma^2_{tol}$ then
11:         $GP_{a_t}$.update($s_t, q_t$)
12:     $\sigma_2^2 = GP_{a_t}$.variance($s_t$)
13:     if $\sigma_1^2 > \sigma^2_{tol} \geq \sigma_2^2$ and
        $\hat{Q}_{a_t}(s_t) - GP_{a_t}$.mean($s_t$) $> 2\epsilon_1$ then
14:         $\hat{Q}_a$.update($s_t, GP_{a_t}$.mean($s_t$) $+ \epsilon_1$)
15:         $\forall a \in A, GP_a = GP$.init($\mu = \hat{Q}_a, k(\cdot,\cdot)$)
```

value is 0) is at least as slow as the example in (Even-Dar & Mansour, 2004). Therefore, the naïve GP implementation's convergence to ϵ is also exponentially slow, and, in fact, has the same learning speed as Q-learning with a linear learning rate. This is because the variance of a GP decays linearly with number of observed data points, and the magnitude of GP updates is proportional to this variance.

Additionally, by adding a second action with reward $r = 1-\gamma$, a greedy naïve agent would oscillate between the two actions for an exponential (in $\frac{1}{1-\gamma}$) number of steps, as shown in Figure 1 (blue line), meaning the algorithm is not PAC-MDP, since the other action is not near optimal. Randomness in the policy with ϵ-greedy exploration (green line) *will not* improve the slowness, which is due to the non-stationarity of the Q-function and the decaying update magnitudes of the GP. Many existing model-free GP algorithms, including GP-Sarsa (Engel et al., 2005), iGP-Sarsa (Chung et al., 2013), and (non-delayed) GPQ (Chowdhary et al., 2014) perform exactly such GP updates without variance reinitialization, and therefore have the same worst-case exponential sample complexity.

4.2. Delayed GPQ for model-free RL

We now propose a new algorithm, inspired by Delayed Q-learning, that guarantees polynomial sample efficiency by consistently reinitializing the variance of the GP when many observations differ significantly from the current $\hat{Q}$ estimate.

Delayed GPQ-Learning (DGPQ: Algorithm 1) maintains *two* representations of the value function. The first is a set of GPs, GP_a for each action, that is updated after each step but *not* used for action selection. The second representation of the value function, which stores values from previously converged GPs is denoted $\hat{Q}(s,a)$. Intuitively, the algorithm uses $\hat{Q}$ as a "safe" version of the value function for choosing actions and retrieving backup values for the GP updates, and only updates $\hat{Q}(s,a)$ when GP_a converges to a significantly different value at s.

The algorithm chooses actions greedily based on $\hat{Q}$ (line 6) and updates the corresponding GP_a based on the observed rewards and $\hat{Q}$ at next the state (line 8). Note, in practice one creates a prior of $\hat{Q}_a(s_t)$ (line 15) by updating the GP with points $z_t = q_t - \hat{Q}_a(s_t)$ (line 11), and adding back $\hat{Q}_a(s_t)$ in the update on line 14. If the GP has just crossed the convergence threshold at point s and learned a value significantly lower ($2\epsilon_1$) than the current value of $\hat{Q}$ (line 13), the representation of $\hat{Q}$ is partially overwritten with this new value plus a bonus term using an operation described below. Crucially, the GPs are then *reset*, with all data erased. This reinitializes the variance of the GPs so that updates will have a large magnitude, avoiding the slowed convergence seen in the previous section. Guidelines for setting the sensitivity of the two tests (σ^2_{tol} and ϵ_1) are given in Section 5.

There are significant parallels between DGPQ and the discrete-state DQL algorithm (Strehl et al., 2009), but the advancements are non-trivial. First, both algorithms maintain two Q functions, one for choosing actions ($\hat{Q}$), and a temporary function that is updated on each step (GP_a), but in DGPQ we have chosen specific representations to handle continuous states and still maintain optimism and sample complexity guarantees. DQL's counting of (up to m) discrete state visits for each state cannot be used in continuous spaces, so DGPQ instead checks the immediate convergence of GP_a at s_t to determine if it should compare $\hat{Q}(s_t,a)$ and $GP_a(s_t)$. Lastly, DQL's discrete learning flags are not applicable in continuous spaces, so DGPQ only compares the two functions as the GP variance crosses a threshold, thereby partitioning the continuous MDP into areas that are known (w.r.t. $\hat{Q}$) and unknown, a property that will be vital for proving the algorithm's convergence.

One might be tempted to use yet another GP to model $\hat{Q}(s,a)$. However, it is difficult to guarantee the optimism of $\hat{Q}$ ($\hat{Q} \geq Q^* - \epsilon$) using a GP, which is a crucial property of most sample-efficient algorithms. In particular, if one attempts to update/overwrite the local prediction values of a GP, unless the kernel has finite support (a point's value only influences a finite region), the overwrite will affect the values of all points in the GP and possibly cause a point that was previously correct to fall below $Q^*(s,a) - \epsilon$.

In order to address this issue, we use an alternative function approximator for $\hat{Q}$ which includes an optimism bonus as used in C-PACE (Pazis & Parr, 2013). Specifically, $\hat{Q}(s,a)$ is stored using a set of values that have been updated from the set of GPs, $(\langle s_i, a_i\rangle, \hat{\mu}_i)$ and a Lipschitz constant L_Q that is used to find an optimistic upper bound for the $\hat{Q}$ function for $\langle s, a\rangle$:

$$\hat{Q}(s,a)=\min\left\{\min_{\langle s_i,a\rangle\in BV}\hat{\mu}_i+L_Q d((s,a),(s_i,a)), V_{\max}\right\} \quad (7)$$

We refer to the set (s_i, a) as the set of *basis vectors* (BV). Intuitively, the basis vectors store values from the previously learned GPs. Around these points, Q^* cannot be greater than $\hat{\mu}_i + L_Q d((s,a),(s_i,a))$ by continuity. To predict optimistically at points not in BV, we search over BV for the point with the lowest prediction including the weighted distance bonus. If no point in BV is sufficiently close, then V_{max} is used instead. This representation is also used in C-PACE (Pazis & Parr, 2013) but here it simply stores the optimistic Q-function; we do not perform a fixed point operation. Note the GP still plays a crucial role in Algorithm 1 because its confidence bounds, which are not captured by $\hat{Q}$, partition the space into known/unknown areas that are critical for controlling updates to $\hat{Q}$.

In order to perform an update (partial overwrite) of $\hat{Q}$, we add an element $\langle (s_i, a_i), \hat{\mu}_i\rangle$ to the basis vector set. Redundant constraints are eliminated by checking if the new constraint results in a lower prediction value at other basis vector locations. Thus, the pseudocode for updating $\hat{Q}$ is as follows: add point $\langle (s_i, a_i), \hat{\mu}_i\rangle$ to the basis vector set; if for any j, $\mu_i + L_Q d((s_i, a),(s_j, a)) \leq \mu_j$, delete $\langle (s_j, a_j), \hat{\mu}_j\rangle$ from the set.

Figure 1 shows the advantage of this technique over the naïve GP training discussed earlier. DGPQ learns the optimal action within 50 steps, while the naïve implementation as well as an ϵ-greedy variant both oscillate between the two actions. This example illustrates that the targeted exploration of DGPQ is of significant benefit compared to untargeted approaches, including the ϵ-greedy approach of GP-SARSA.

5. The Sample Complexity of DGPQ

In order to prove that DGPQ is PAC-MDP we adopt a proof structure similar to that of DQL (Strehl et al., 2009) and refer throughout to corresponding theorems and lemmas. First we extend the DQL definition of a "known state" MDP M_{K_t}, containing state/actions from $\hat{Q}$ that have low Bellman residuals.

Definition 4 During timestep t of DGPQ's execution with $\hat{Q}$ as specified and $\hat{V}(s) = \max_a \hat{Q}(s,a)$, the set of **known states** is given by $K_t = \{\langle s,a\rangle | \hat{Q}(s,a) - (R(s,a) + \gamma \int_{s'} T(s'|s,a)\hat{V}(s')ds') \leq 3\epsilon_1\}$. M_K contains the same MDP parameters as M for $\langle s,a\rangle \in K$ and values of $V_{\max}$ for $\langle s,a\rangle \notin K$.

We now recap (from Theorem 10 of (Strehl et al., 2009)) three sufficient conditions for proving an algorithm with greedy policy values V_t is PAC-MDP. (1) Optimism: $\hat{V}_t(s) \geq V^*(s) - \epsilon$ for all timesteps t. (2) Accuracy with respect to M_{K_t}'s values: $\hat{V}_t(s) - V^{\pi_t}_{M_{K_t}}(s) \leq \epsilon$ for all t. (3) The number of updates to $\hat{Q}$ and the number of times a state outside M_K is reached is bounded by a polynomial function of $\langle \mathcal{N}_S(r), \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma}\rangle$.

The proof structure is to develop lemmas that prove these three properties. After defining relevant structures and concepts, Lemmas 2 and 3 bound the number of possible changes and possible *attempts* to change $\hat{Q}$. We then define the "typical" behavior of the algorithm in Definition 6 and show this behavior occurs with high probability in Lemma 4. These conditions help ensure property 2 above. After that, Lemma 5 shows that the function stored in $\hat{Q}$ is always optimistic, fulfilling property 1. Finally, property 3 is shown by combining Theorem 1 (number of steps before the GP converges) with the number of updates to $\hat{Q}$ from Lemma 2, as formalized in Lemma 7.

We now give the definition of an "update" (as well as "successful update") to $\hat{Q}$, which extends definitions from the original DQL analysis.

Definition 5 An **update** (or **successful update**) of state-action pair (s,a) is a timestep t for which a change to $\hat{Q}$ (an overwrite) occurs such that $\hat{Q}_t(s,a) - \hat{Q}_{t+1}(s,a) > \epsilon_1$. An **attempted update** of state-action pair (s,a) is a timestep t for which $\langle s,a\rangle$ is experienced and $GP_{a,t}.\text{Var}(s) > \sigma^2_{tol} \geq GP_{a,t+1}.\text{Var}(s)$. An attempted update that is not successful (does not change $\hat{Q}$) is an **unsuccessful update**.

We bound the total number of updates to $\hat{Q}$ by a polynomial term κ based on the concepts defined above.

Lemma 2 The total number of successful updates (overwrites) during any execution of DGPQ with $\epsilon_1 = \frac{1}{3}\epsilon(1-\gamma)$

is

$$\kappa = |A|\mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)\left(\frac{3R_{max}}{(1-\gamma)^2\epsilon}+1\right) \tag{8}$$

Proof The proof proceeds by showing that the bound on the number of updates in the single-state case from Section 4.1 is, $\left(\frac{3R_{max}}{(1-\gamma)^2\epsilon}+1\right)$ based on driving the full function from $V_{\max}$ to $V_{\min}$. This quantity is then multiplied by the covering number and the number of actions. See the Supplementary Material.

The next lemma bounds the number of attempted updates. The proof is similar to the DQL analysis and shown in the Supplementary Material.

Lemma 3 The total number of **attempted** updates (overwrites) during any execution of GPQ is $|A|\mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa)$.

We now define the following event, called A2 to link back to the DQL proof. A2 describes the situation where a state/action that currently has an inaccurate value (high Bellman residual) with respect to $\hat{Q}$ is observed m times and this causes a successful update.

Definition 6 Define **Event A2** to be the event that for all timesteps t, if $\langle s,a\rangle \notin K_{k_1}$ and an attempted update of $\langle s,a\rangle$ occurs during timestep t, the update will be successful, where $k_1 < k_2 < ... < k_m = t$ are the timesteps where $GP_{a_k}.\text{Var}(s_k) > \sigma^2_{tol}$ since the last update to $\hat{Q}$ that affected $\langle s,a\rangle$.

We can set an upper bound on m, the number of experiences required to make $GP_{a_t}.\text{Var}(s_t) < \sigma^2_{tol}$, based on m in Theorem 1 from earlier. In practice m will be much smaller but unlike discrete DQL, DGPQ does not need to be given m, since it uses the GP variance estimate to decide if sufficient data has been collected. The following lemma shows that with high probability, a failed update will not occur under the conditions of event A2.

Lemma 4 By setting

$$\sigma^2_{tol} = \frac{2\omega_n^2\epsilon^2(1-\gamma)^4}{9R^2_{max}\log(\frac{6}{\delta}|A|\mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa))} \tag{9}$$

we ensure that the probability of event A2 occurring is $\geq 1-\delta/3$.

Proof The maximum number of attempted updates is $|A|\mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa)$ from Lemma 3. Therefore, by setting $\delta_1 = \frac{\delta}{3|A|\mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa)}$, $\epsilon_1 = \frac{1}{3}\epsilon(1-\gamma)$, and $V_m = \frac{R_{max}}{1-\gamma}$, we have from Lemma 1 that during each overwrite, there is no greater than probability $\delta_1 = \frac{\delta}{3|A|\mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)(1+\kappa)}$ of an incorrect update. Applying the union bound over all of the possible attempted updates, we have the total probability of A2 not occuring is $\frac{\delta}{3}$.

The next lemma shows the optimism of $\hat{Q}$ for all timesteps with high probability $\frac{\delta}{3}$. The proof structure is similar to Lemma 3.10 of (Pazis & Parr, 2013). See supplemental material.

Lemma 5 During execution of DGPQ, $Q^*(s,a) \leq Q_t(s,a) + \frac{2\epsilon_1}{1-\gamma}$ holds for all $\langle s,a\rangle$ with probability $\frac{\delta}{3}$.

The next Lemma connects an unsuccessful update to a state/action's presence in the known MDP M_K.

Lemma 6 If event A2 occurs, then if an unsuccessful update occurs at time t and $GP_a.\text{Var}(s) < \sigma^2_{tol}$ at time $t+1$ then $\langle s,a\rangle \in K_{t+1}$.

Proof The proof is by contradiction and shows that an unsuccessful update in this case implies a previously successful update that would have left $GP_a.\text{Var}(s) \geq \sigma^2_{tol}$. See the Supplemental material.

The final lemma bounds the number of encounters with state/actions not in K_t, which is intuitively the number of points that make updates to the GP from Theorem 1 times the number of changes to $\hat{Q}$ from Lemma 2. The proof is in the Supplemental material.

Lemma 7 Let $\eta = \mathcal{N}_S\left(\frac{\epsilon(1-\gamma)}{3L_Q}\right)$. If event A2 occurs and $\hat{Q}_t(s,a) \geq Q^*(s,a) - \frac{2\epsilon_1}{1-\gamma}$ holds for all t and $\langle s,a\rangle$ then the number of timesteps ζ where $\langle s_t,a_t\rangle \notin K_t$ is at most

$$\zeta = m|A|\eta\left(\frac{3R_{max}}{(1-\gamma)^2\epsilon}+1\right) \tag{10}$$

where

$$m = \left(\frac{36R^2_{max}}{(1-\gamma)^4\epsilon^2}\log\left(\frac{6}{\delta}|A|\eta(1+\kappa)\right)\right)|A|\eta \tag{11}$$

Finally, we state the PAC-MDP result for DGPQ, which is an instantiation of the General PAC-MDP Theorem (Theorem 10) from (Strehl et al., 2009).

Theorem 2 Given real numbers $0 < \epsilon < \frac{1}{1-\gamma}$ and $0 < \delta < 1$ and a continuous MDP M there exist inputs σ^2_{tol} (see (9)) and $\epsilon_1 = \frac{1}{3}\epsilon(1-\gamma)$ such that DGPQ executed on M will be PAC-MDP by Definition 2 with only

$$\frac{R_{\max}\zeta}{\epsilon(1-\gamma)^2}\log\left(\frac{1}{\delta}\right)\log\left(\frac{1}{\epsilon(1-\gamma)}\right) \tag{12}$$

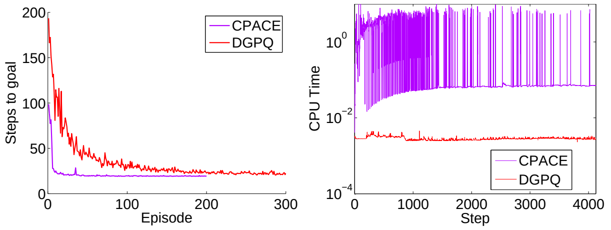

Figure 2. Average (10 runs) steps to the goal and computation time for C-PACE and DGPQ on the square domain.

timesteps where $Q_t(s_t, a_t) < V^*(s_t) - \epsilon$, where ζ is defined in Equation (10).

The proof of the theorem is the same as Theorem 16 by (Strehl et al., 2009), but with the updated lemmas from above. The three crucial properties hold when A2 occurs, which Lemma 4 guarantees with probability $\frac{\delta}{3}$. Property 1 (optimism) holds from Lemma 5. Property 2 (accuracy) holds from the Definition 4 and analysis of the Bellman Equation as in Theorem 16 by (Strehl et al., 2009). Finally, property 3, the bounded number of updates and escape events, is proven by Lemmas 2 and 7.

6. Empirical Results

Our first experiment is in a 2-dimensional square over $[0, 1]^2$ designed to show the computational disparity between C-PACE and DGPQ. The agent starts at $[0, 0]$ with a goal of reaching within a distance of 0.15 of $[1, 1]$. Movements are 0.1 in the four compass directions with additive uniform noise of ± 0.01. We used an $\mathcal{L}_1$ distance metric, $L_Q = 9$, and an RBF kernel with $\theta = 0.05$, $\omega_n^2 = 0.1$ for the GP. Figure 2 shows the number of steps needed per episode (capped at 200) and computational time per step used by C-PACE and DGPQ. C-PACE reaches the optimal policy in fewer episodes but requires orders of magnitude more computation during steps with fixed point computations. Such planning times of over 10s are unacceptable in time sensitive domains, while DGPQ only takes 0.003s per step.

In the second domain, we use DGPQ to stabilize a simulator of the longitudinal dynamics of an unstable F16 with linearized dynamics, which motivates the need for a real-time computable policy. The five dimensional state space contains height, angle of attack, pitch angle, pitch rate, and airspeed. Details of the simulator can be found in (Stevens & Lewis, 2003). We used the reward $r = -|h - h_d|/100\text{ft} - |\dot{h}|/100\text{ft/s} - |\delta_e|$, with aircraft height h, desired height h_d and elevator angle (degrees) δ_e. The control input was discretized as $\delta_e \in \{-1, 0, 1\}$ and the elevator was used to control the aircraft. The thrust input to the engine was fixed as the reference thrust command at the (unstable) equilibrium. The simulation time step size was 0.05s and at each step, the air speed was perturbed with Gaussian noise $\mathcal{N}(0, 1)$ and the angle of attack was perturbed with Gaussian noise $\mathcal{N}(0, 0.01^2)$. A RBF kernel with $\theta = 0.05$, $\omega_n^2 = 0.1$ was used. The initial height was h_d, and if $|h - h_d| \geq 200$, then the vertical velocity and angle of attack were set to zero to act as boundaries.

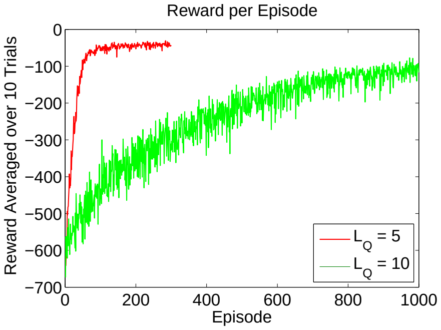

Figure 3. Average (10 runs) reward on the F16 domain.

DGPQ learns to stabilize the aircraft using less than 100 episodes for $L_Q = 5$ and about 1000 episodes for $L_Q = 10$. The disparity in learning speed is due to the curse of dimensionality. As the Lipschitz constant doubles, $\mathcal{N}_c$ increases in each dimension by 2, resulting in a 2^5 increase in $\mathcal{N}_c$. DGPQ requires on average 0.04s to compute its policy at each step, which is within the 20Hz command frequency required by the simulator. While the maximum computation time for DGPQ was 0.11s, the simulations were run in MATLAB so further optimization should be possible. Figure 3 shows the average reward of DGPQ in this domain using $L_Q = \{5, 10\}$. We also ran C-PACE in this domain but its computation time reached over 60s per step in the first episode, well beyond the desired 20 Hz command rate.

7. Conclusions

This paper provides sample efficiency results for using GPs in RL. In section 3, GPs are proven to be usable in the KWIK-Rmax MBRL architecture, establishing the previously proposed algorithm GP-Rmax as PAC-MDP. In section 4.1, we prove that existing model-free algorithms using a single GP have exponential sample complexity, connecting to seemingly unrelated negative results on Q-learning learning speeds. Finally, the development of DGPQ provides the first provably sample efficient model-free (without a planner or fixed-point computation) RL algorithm for general continuous spaces.

Acknowledgments

We thank Girish Chowdhary for helpful discussions, Hassan Kingravi for his GP implementation, and Aerojet-Rocketdyne and ONR #N000141110688 for funding.

References

Brafman, Ronen I. and Tennenholtz, Moshe. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

Chowdhary, Girish, Liu, Miao, Grande, Robert C., Walsh, Thomas J., How, Jonathan P., and Carin, Lawrence. Off-policy reinforcement learning with gaussian processes. *Acta Automatica Sinica*, To appear, 2014.

Chung, Jen Jen, Lawrance, Nicholas R. J., and Sukkarieh, Salah. Gaussian processes for informative exploration in reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2633–2639, 2013.

Csató, L. and Opper, M. Sparse on-line gaussian processes. *Neural Computation*, 14(3):641–668, 2002.

Deisenroth, Marc Peter and Rasmussen, Carl Edward. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 465–472, 2011.

Engel, Y., Mannor, S., and Meir, R. Reinforcement learning with Gaussian processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.

Even-Dar, Eyal and Mansour, Yishay. Learning rates for Q-learning. *Journal of Machine Learning Research*, 5: 1–25, 2004.

Jung, Tobias and Stone, Peter. Gaussian processes for sample efficient reinforcement learning with RMAX-like exploration. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2010.

Kalyanakrishnan, Shivaram and Stone, Peter. An empirical analysis of value function-based and policy search reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 749–756, 2009.

Li, Lihong and Littman, Miachael L. Reducing reinforcement learning to KWIK online regression. *Annals of Mathematics and Artificial Intelligence*, 58(3-4):217–237, 2010.

Li, Lihong, Littman, Michael L., Walsh, Thomas J., and Strehl, Alexander L. Knows what it knows: a framework for self-aware learning. *Machine Learning*, 82(3):399–443, 2011.

Pazis, Jason and Parr, Ronald. PAC optimal exploration in continuous space markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2013.

Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

Rasmussen, C. and Williams, C. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

Stevens, Brian L. and Lewis, Frank L. *Aircraft control and simulation*. Wiley-Interscience, 2003.

Strehl, Alexander L., Li, Lihong, Wiewiora, Eric, Langford, John, and Littman, Michael L. PAC model-free reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 881–888, 2006.

Strehl, Alexander L., Li, Lihong, and Littman, Michael L. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10:2413–2444, 2009.

Sutton, R. and Barto, A. *Reinforcement Learning, an Introduction*. MIT Press, Cambridge, MA, 1998.

Watkins, C. J. Q-learning. *Machine Learning*, 8(3):279–292, 1992.