In *Proceedings of the 25th International Conference on Machine Learning* (ICML 08),
Helsinki, Finland,, July 2008.

Online Kernel Selection for Bayesian Reinforcement Learning

Joseph Reisinger JOERAII@CS.UTEXAS.EDU
Peter Stone PSTONE@CS.UTEXAS.EDU
Risto Miikkulainen RISTO@CS.UTEXAS.EDU
Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712

Abstract

Kernel-based Bayesian methods for Reinforcement Learning (RL) such as Gaussian Process Temporal Difference (GPTD) are particularly promising because they rigorously treat uncertainty in the value function and make it easy to specify prior knowledge. However, the choice of prior distribution significantly affects the empirical performance of the learning agent, and little work has been done extending existing methods for prior model selection to the online setting. This paper develops Replacing-Kernel RL, an online model selection method for GPTD using sequential Monte-Carlo methods. Replacing-Kernel RL is compared to standard GPTD and tile-coding on several RL domains, and is shown to yield significantly better asymptotic performance for many different kernel families. Furthermore, the resulting kernels capture an intuitively useful notion of prior state covariance that may nevertheless be difficult to capture manually.

1. Introduction

Bayesian methods are a natural fit for Reinforcement Learning (RL) because they represent prior knowledge compactly and allow for rigorous treatment of value function uncertainty. Modeling such uncertainty is important because it offers a principled solution for balancing exploration and exploitation in the environment. One particularly elegant Bayesian RL formulation is Gaussian Process Temporal Difference (GPTD) (Engel et al., 2005). GPTD is an efficient adaptation of Gaussian processes (GPs) to the problem of online value-function estimation. In GPs, prior knowledge in the form of value covariance across states is represented compactly by a Mercer kernel (Rasmussen & Williams, 2006), offering a conceptually simple method for biasing learning.

An important open question for Bayesian RL is how to perform model selection efficiently and online. In GPTD, model selection determines the particular form of the prior covariance function and the settings of any hyperparameters. This paper contributes towards answering this question in two ways: (1) It demonstrates empirically the importance of model selection in Bayesian RL; and (2) it outlines *Replacing-Kernel Reinforcement Learning* (RKRL), a simple and effective sequential Monte-Carlo procedure for selecting the model online. RKRL not only improves learning in several domains, but does so in a way that cannot be matched by any choice of standard kernels.

Although conceptually similar to methods combining evolutionary algorithms and RL (Whiteson & Stone, 2006), RKRL is novel for two reasons: (1) The sequential Monte-Carlo technique employed is simpler and admits a clear *empirical Bayesian* interpretation (Bishop, 2006), (2) Since GPs are nonparametric, it is possible to replace kernels online during learning without discarding any previously acquired knowledge, simply by maintaining the dictionary of saved training examples between kernel swaps. This online replacing procedure significantly improves performance over previous methods, because learning does not need to start from scratch for new kernels.

This paper is divided into seven main sections: Section 2 introduces GPTD, Section 3 describes RKRL, Section 4 details the experimental setup using Mountain Car, Ship Steering and Capture Go as example domains, and the last three sections give results, future work and conclusions.

2. Gaussian Process Reinforcement Learning

In RL domains with large or infinite state spaces, function approximation becomes necessary as it is impractical or impossible to store a table of all state values (Sutton & Barto, 1998). Gaussian Processes (GPs) have emerged as a principled method for solving regression problems in Machine Learning (Rasmussen & Williams, 2006), and have recently been extended to performing function approxima-

Appearing in *Proceedings of the 25^{th} International Conference on Machine Learning*, Helsinki, Finland, 2008.

tion in RL as well (Engel et al., 2005). In this section, we briefly review GPs and their application to temporal difference learning.

GPs are a class of statistical generative models for Bayesian nonparametric inference. Instead of relying on a fixed functional form as in parametric model, GPs are defined directly in some (infinite-dimensional) function space (Rasmussen & Williams, 2006). Specifically, an indexed collection of random variables $V : \mathcal{X} \rightarrow \Re$ over a common probability space is a GP if the distribution of any finite subset of V is Gaussian. Gaussian processes are completely specified by prior mean and covariance functions. In this paper, the mean function is assumed to be identically zero and the prior covariance function is specified as a Mercer kernel $k(\cdot, \cdot)$. Mechanistically, a GP is composed of the set of training data and a prior covariance function (kernel) that defines how to interpolate between those points.

Following the formulation of (Engel, 2005), consider a statistical generative model of the form

$$R(\mathbf{x}) = \mathbf{H}V(\mathbf{x}) + N(\mathbf{x}), \tag{1}$$

where V is the unknown function to be estimated, N is a noise model, $\mathbf{H}$ is a linear transformation, and R is the observed regression function. Given a set of data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=0}^{t}$, the model reduces to a system of linear equations $R_t = \mathbf{H}_t V_t + N_t$, where $R_t = (R(\mathbf{x}_0), \ldots, R(\mathbf{x}_t))^\top$, $V_t = (V(\mathbf{x}_0), \ldots, V(\mathbf{x}_t))^\top$, and $N_t = (N(\mathbf{x}_0), \ldots, N(\mathbf{x}_t))^\top$.

Assuming that $V \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_t)$ is a zero-mean GP with $[\mathbf{K}_t]_{i,j} \stackrel{\text{def}}{=} k(\mathbf{x}_i, \mathbf{x}_j)$ for $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}$ and $N \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_t)$, then the Gauss-Markov theorem gives the posterior distribution of V conditional on the observed R:

$$\begin{aligned} \hat{V}_t(\mathbf{x}) &= \mathbf{k}_t(\mathbf{x})^\top \boldsymbol{\alpha}_t, & (2)\\ P_t(\mathbf{x}) &= k(\mathbf{x}, \mathbf{x}) - \mathbf{k}_t(\mathbf{x})^\top \mathbf{C}_t \mathbf{k}_t(\mathbf{x}), & (3) \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{\alpha}_t &= \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \mathbf{\Sigma})^{-1} \mathbf{r}_{t-1}, \\ \mathbf{C}_t &= \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \mathbf{\Sigma})^{-1} \mathbf{H}_t, \end{aligned}$$

and $\mathbf{k}_t(\mathbf{x}) \stackrel{\text{def}}{=} (k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_t))^\top$. This closed-form posterior can be used to calculate the predicted value of V at some new test point $\mathbf{x}^*$. In RL, the sequence of observed reward values are assumed to be related by some (possibly stochastic) environment dynamics that are captured through the matrix $\mathbf{H}$.

In order to adapt GPs to RL, the standard Markov Decision Process (MDP) framework needs to first be formalized as follows. Let $\mathcal{X}$ and $\mathcal{U}$ be the state and action spaces, respectively. Define $R : \mathcal{X} \rightarrow \Re$ to be the reward function and let $p : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \rightarrow [0, 1]$ be the state transition probabilities.

A policy $\mu : \mathcal{X} \times \mathcal{U} \rightarrow [0, 1]$ is a mapping from states to action selection probabilities. The *discounted return* for a state $\mathbf{x}$ under policy μ is defined as

$$D(\mathbf{x}) = \sum_{i=0}^{\infty} \gamma^i R(\mathbf{x}_i) | \mathbf{x}_0 = \mathbf{x},$$

where $\mathbf{x}_{i+1} \sim p^\mu(\cdot | \mathbf{x}_i)$, the policy-dependent state transition probability distribution, and $\gamma \in [0, 1]$ is the discount factor. The goal of RL is to compute a *value function* that estimates the discounted reward for each state under a policy μ, $V(\mathbf{x}) = \mathbf{E}_\mu[D(\mathbf{x})]$.

GPs can be used to model the latent value function given a sequence of observed rewards and an appropriate noise model. Reward is related to value by

$$R(\mathbf{x}) = V(\mathbf{x}) - \gamma V(\mathbf{x}') + N(\mathbf{x}, \mathbf{x}'),$$

where $\mathbf{x}' \sim p^\mu(\cdot | \mathbf{x})$. Extending this model temporally to a series of states $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t$ yields the system of equations $R_{t-1} = \mathbf{H}_t V_t + N_t$, where R and V are defined as before, and

$$\begin{aligned} N_t &= (N(\mathbf{x}_0, \mathbf{x}_1), \ldots, N(\mathbf{x}_{t-1}, \mathbf{x}_t))^\top, \\ \mathbf{H}_t &= \begin{bmatrix} 1 & -\gamma & 0 & \ldots & 0 \\ 0 & 1 & -\gamma & \ldots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \ldots & 1 & -\gamma \end{bmatrix} \end{aligned}$$

and $N_t \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_t)$. If the environment dynamics are assumed to be deterministic, the covariance of the state-dependent noise can be modeled as $\mathbf{\Sigma}_t = \sigma^2 \mathbf{I}$. For stochastic environments, the noise model $\mathbf{\Sigma}_t = \sigma^2 \mathbf{H}_{t+1} \mathbf{H}_{t+1}^\top$ is more suitable. See (Engel, 2005) for a complete derivation for these models.

Given $\mathbf{H}_t$, $\mathbf{\Sigma}_t$, and a sequence of states and reward values, the posterior moments $\hat{V}_t$ and P_t can be computed to yield value function estimates. Thus GPs fit naturally into the RL framework: Learning is straightforward and does not require setting unintuitive parameters such as α or λ; prior knowledge of the problem can be built in through the covariance function; the full distribution of the posterior is available, making it possible to select actions in more compelling ways, e.g. via interval estimation. Furthermore, the value estimator $\hat{V}_t$ and covariance P_t can be computed incrementally online as each new state action pair is sampled, without having to invert a $t \times t$ matrix at each step. For details of this procedure, see (Engel et al., 2005).

One issue with using GPs for RL is that the size of $\mathbf{K}_t$, $\mathbf{k}(\cdot)$, $\mathbf{H}_t$ and $\mathbf{r}_t$ each grow linearly with the number of states visited, yielding a computational complexity of $O(|\mathcal{D}|^2)$ for each step. Since it is not practical to remember every single experience in online settings, the GP dictionary size must be limited in some way. To this end, Engel et al. derive a *kernel sparsification* procedure based on an approximate linear dependence (ALD) test. As the number of observed training examples tends to infinity, the number of examples that need to be saved tends to zero (Engel et al., 2005). A matrix $\mathbf{A}_t$ contains approximation coefficients for the ALD test and a parameter ν controls how "novel" a particular training example must be before it is remembered by the GP, making it possible to tune how compact and computationally efficient the value function representation is.

Finally, note that GPTD can be extended to the case where no environment model is available, simply by defining the covariance function over state-action pairs $k(\mathbf{x}, \mathbf{u}, \mathbf{x}', \mathbf{u}')$. This procedure will be termed GP-SARSA in this paper.

GPTD has been shown to be successful, but in practice performance relies on a good choice of kernel. The next section will focus on a particular online method for performing such kernel selection.

3. Online Model Selection

A common requirement in RL is that learning take place *online*, e.g. the learner must maximize total reward accrued. However, traditional model selection techniques applied to GPs, such as cross-validation, or Bayesian Model Averaging, are not designed to address this constraint. The main contribution of this paper is to introduce *Replacing-Kernel Reinforcement Learning* (RKRL), an online procedure for model selection in RL. In section 3.1 an online sequential Monte-Carlo method developed and used to implement RKRL, as described in section 3.2.

3.1. Sequential Monte-Carlo Methods

Given a set of kernels $\{k_{\boldsymbol{\theta}}(\cdot, \cdot) | \boldsymbol{\theta} \in \mathcal{M}\}$ parameterized by a random variable $\boldsymbol{\theta}$ and a prior $p(\boldsymbol{\theta})$ over these parameterizations, a fully Bayesian approach to learning involves integrating over all possible settings of $\boldsymbol{\theta}$, yielding the posterior distribution

$$p(R|\mathcal{D}) = \iint p(R|V, \mathcal{D}, \boldsymbol{\theta}) p(V|\mathcal{D}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) dV d\boldsymbol{\theta}.$$

The integration over V given $\boldsymbol{\theta}$ is carried out implicitly when using GPs, however, the remaining integral over $\boldsymbol{\theta}$ is generally intractable for all but the most simple cases. Instead of integrating over all possible model settings $\boldsymbol{\theta}$, we can use the data distribution $\mathcal{D}$ to infer reasonable settings for $\boldsymbol{\theta}$ via $p(\boldsymbol{\theta}|\mathcal{D})$. Such *evidence approximation* can is more computationally efficient and is an example of an empirical Bayes approach, where likelihood information is used to guide prior selection (Bishop, 2006).

Monte-Carlo methods can be used to sample from $p(\boldsymbol{\theta}|\mathcal{D})$, however, such methods assume that this distribution is stationary (Bishop, 2006). In the RL case, stationarity implies that when evaluating $\boldsymbol{\theta}$, previous data acquired while evaluating $\boldsymbol{\theta}'$ cannot be used. To avoid this inefficiency, we instead employ a *sequential* Monte-Carlo (SMC) method adapted from (Gordon et al., 1993) that relaxes the stationarity assumption.

Algorithm 1 Sequential Monte Carlo

Parameters: n, μ, τ, Λ

Draw $\{\boldsymbol{\theta}_i^{(0)}\}_{i=1}^n \sim p(\boldsymbol{\theta})$
for $t = 0, 1, \ldots$ **do**
  Calculate $\{w_i^{(t)}\}_{i=1}^n$ from equation 4.
  Draw $\{\tilde{\boldsymbol{\theta}}_i^{(t+1)}\}_{i=1}^n$ by resampling $\{(\boldsymbol{\theta}_i^{(t)}, w_i^{(t)})\}_{i=1}^n$.
  $\boldsymbol{\theta}_i^{(t+1)} \leftarrow \tilde{\boldsymbol{\theta}}_i^{(t+1)} + (c_0\phi_0, \ldots, c_k\phi_k)^\top$ where $c_k \sim$ Bernoulli(μ) and $\phi_k \sim \mathcal{N}(0, 1)$.
end for

SMC approximates the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ at time t empirically via a set of n samples and weights $\{(\boldsymbol{\theta}_i^{(t)}, w_i^{(t)})\}_{i=1}^n$ where

$$w_i \stackrel{\text{def}}{=} \frac{p(\mathcal{D}|\boldsymbol{\theta}_i^{(t)}) p(\boldsymbol{\theta}_i^{(t)})}{\sum_m p(\mathcal{D}|\boldsymbol{\theta}_m^{(t)}) p(\boldsymbol{\theta}_m^{(t)})}, \tag{4}$$

where $p(\mathcal{D}|\boldsymbol{\theta}_i^{(t)})$ is the likelihood of $\boldsymbol{\theta}_i^{(t)}$ and $p(\boldsymbol{\theta}_i^{(t)})$ is the prior. Inference proceeds sequentially with samples for time $t+1$ drawn from the empirical distribution

$$p(\boldsymbol{\theta}^{(t+1)}|\mathcal{D}) = \sum_l w_l^{(t)} p(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}_l^{(t)}), \tag{5}$$

where $p(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)})$ is the *transition kernel*, defining how the hyperparameter space should be explored. In this paper, the prior over models $p(\boldsymbol{\theta})$ is the uniform distribution over $[0, 1]$ for each of the kernel hyperparameters (listed in table 1) and the transition kernel $p(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)})$ is defined mechanistically as $\boldsymbol{\theta}^{(t+1)} \leftarrow \tilde{\boldsymbol{\theta}}^{(t+1)} + (c_0\phi_0, \ldots, c_k\phi_k)^\top$ where $c_k \sim$ Bernoulli(μ) and $\phi_k \sim \mathcal{N}(0, 1)$. Pseudocode for this procedure is given in algorithm 1.

3.2. Replacing-Kernel Reinforcement Learning

In Replacing-Kernel Reinforcement Learning (RKRL), SMC is used to select good kernel hyperparameter settings. Rather than calculating the true model likelihood $p(\mathcal{D}|\boldsymbol{\theta}_i)$ in equation 4, RKRL instead weights models based on their relative *predictive likelihood*, $\tilde{p}(\mathcal{D}|\boldsymbol{\theta}_i)$, where

$$\log \tilde{p}(\mathcal{D}|\boldsymbol{\theta}_i) \stackrel{\text{def}}{=} \tau^{-1} \sum_t r_t,$$

and $(r_0, r_1, \ldots)$ is the sequence of rewards obtained by evaluating the hyperparameter setting $\boldsymbol{\theta}_i$ for Λ episodes.

Table 1. Basic kernel functions and the corresponding extended parameterizations.

KERNEL	BASIC	EXTENDED
NORM	$k(\mathbf{x},\mathbf{x}') = 1 - \frac{\|\|\mathbf{x}-\mathbf{x}'\|\|^2}{\alpha}$	$k(\mathbf{x},\mathbf{x}') = 1 - \sum_i w_i(x_i - x_i')^2$
GAUSSIAN	$k(\mathbf{x},\mathbf{x}') = \exp\left[\frac{-\|\|\mathbf{x}-\mathbf{x}'\|\|^2}{\sigma^2}\right]$	$k(\mathbf{x},\mathbf{x}') = \exp\left[-\sum_i w_i(x_i - x_i')^2\right]$
POLYNOMIAL	$k(\mathbf{x},\mathbf{x}') = (\langle\mathbf{x},\mathbf{x}'\rangle + 1)^d$	$k(\mathbf{x},\mathbf{x}') = (\sum_i w_i x_i x_i' + 1)^d$
TANH NORM	$k(\mathbf{x},\mathbf{x}') = \tanh(v\|\|\mathbf{x}-\mathbf{x}'\|\|^2 - c)$	$k(\mathbf{x},\mathbf{x}') = \tanh(\sum_i w_i(x_i - x_i')^2 - 1)$
TANH DOT	$k(\mathbf{x},\mathbf{x}') = \tanh(v\langle\mathbf{x},\mathbf{x}'\rangle - c)$	$k(\mathbf{x},\mathbf{x}') = \tanh(\sum_i w_i x_i x_i' - 1)$

The parameter τ is introduced to control how strongly model search should focus on hyperparameter settings that yield high reward. Maximizing predictive ability directly is preferable as it is more closely related to the goal of learning than maximizing the fit to the observed data. In tabular methods these two approaches indeed coincide in the limit of large data, however when using function approximation, they may differ.

When using GPTD, the current value function estimate is formed from the combination of the kernel parameterization $\boldsymbol{\theta}$ determining the prior covariance function and the dictionary $\tilde{\mathcal{D}} \subseteq \mathcal{D}$ gathered incrementally from observing state transitions. In this paper we consider two variants of RKRL: Standard RKRL and *Experience-Preserving* RKRL (EP-RKRL) that differ based on their treatment of the saved experience $\tilde{\mathcal{D}}$. In Standard RKRL, $\tilde{\mathcal{D}}$ is discarded at the start of each new kernel evaluation (making $p(\boldsymbol{\theta}|\mathcal{D})$ stationary). In contrast, in EP-RKRL each kernel parameterization sample $\boldsymbol{\theta}^{(t)}$ inherits $\tilde{\mathcal{D}}$[1] from the sample $\boldsymbol{\theta}^{(t-1)}$ that generated it in equation 5.

RKRL naturally spends more time evaluating hyperparameter settings that correspond to areas with high predictive likelihood, i.e. maximizes online reward. Each sampling step increases information about the predictive likelihood in the sample (exploitation), while sampling from the transition kernel reduces such information (exploration).

4. Experimental Setup

Standard RKRL and EP-RKRL are compared against GP-SARSA on three domains. This section gives the parameter settings, kernel classes and domains used.

4.1. Parameters

In all experiments, the TD discount factor was fixed at $\gamma = 1.0$ and ϵ-greedy action selection was employed with $\epsilon = 0.01$. The GP-SARSA parameters for prior noise variance (σ) and dictionary sparsity (ν) were $\sigma = 1.0$ and $\nu = 0.001$. For each RKRL evaluation, GP-SARSA is run for Λ episodes using the specified kernel parameterization. The RKRL parameters were set to $n = 25$, $\mu = 0.01$ and $\tau = 0.5$. Performance of RKRL is insensitive to changes doubling Λ or μ. Higher settings of n improve the initial performance, but reduce the total number of epochs possible given a fixed number of episodes. The setting of τ significantly impacts performance, although the main results of this paper are insensitive for $0.25 \leq \tau \leq 1.0$. All kernels are extended to functions of both the state and action, with actions treated as extra state variables.

4.2. Kernels

Although RKRL automates the choice of kernel hyperparameters, there is still a need to choose a set of kernels that represents the search space for RKRL. General kernel classes are derived from basic classes commonly found in the literature (table 1) by replacing the standard inner products and norms with weighted variants (cf. *automatic relevance determination*), yielding kernel classes with significantly more hyperparameters. Setting these hyperparameters is the model selection task; as more hyperparameters are added, the model becomes more general, but the corresponding difficulty of inferring the model parameters increases as well.

In order to give a fair baseline GP-SARSA comparison, the best hyperparameter setting for each basic kernel class was derived manually for each domain using grid search. Note that although they are common, the hyperbolic tangent kernels are not positive semi-definite; however they still yield good performance in practice (Smola & Schölkopf, 2004).

4.3. Test Domains

GP-SARSA, RKRL and EP-RKRL are compared across three domains: Mountain Car, Ship Steering and Capture Go. Each domain highlights a different aspect of complexity found in RL problems: Mountain Car and Ship Steering have continuous state spaces and thus require function approximation, Ship Steering also has a large (discrete) action space, and Capture Go is stochastic with a high-dimensional state space.

[1]For efficiency the sufficient statistics $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\mathbf{C}}$ for sparsified GP-SARSA are also inherited, though they can be recalculated. The matrix of approximation coefficients $\mathbf{A}_t$ is not recalculated, although doing so should lead to be better performance in general.

4.3.1. MOUNTAIN CAR

In *Mountain Car*, the learning agent must drive an underpowered car up a steep hill (Sutton & Barto, 1998). The available actions are $a \in \{-1, 0, 1\}$, i.e., brake, neutral and accelerate. The state $\mathbf{x}_t = (x_t, \dot{x}_t) \in \Re^2$ is comprised of the position and velocity. The environment is deterministic with state evolution governed by

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_{t+1} \\ \dot{x}_{t+1} &= \dot{x}_t + 0.001a_t + -0.0025\cos(3x_t) \end{aligned}$$

where $-1.2 \leq x \leq 0.5$ and $|\dot{x}| \leq 0.07$. Reward is -1 for each time step the car has not passed the goal at $x = 0.5$. In all RKRL experiments with Mountain Car, $\Lambda = 100$ (100 episodes per epoch) and each episode is limited to 1000 steps to reduce computation time.

4.3.2. SHIP STEERING

In *Ship Steering*, the learning agent must properly orient a sailboat to a specific heading and travel as fast as possible (White, 2007). Actions are two-dimensional rudder position (degrees) and thrust (Newtons), $\mathbf{a}_t = (r_t, T_t) \in [-90, 90] \times [-1, 2]$. Possible rudder settings are discretized at 3-degree increments and thrust increments are 0.5 Newtons, yielding 427 possible actions. The state is a 3-tuple consisting of the heading, angular velocity and velocity $\mathbf{x}_t = (\theta_t, \dot{\theta}_t, \dot{x}_t) \in \Re^3$. State evolution is described by

$$\begin{aligned} \dot{x}_{t+1} &= \dot{x}_t + \frac{1}{250}(30T_t - 2\dot{x}_t - 0.03\dot{x}_t(5\theta_t + r_t^2)) \\ \dot{\theta}_{t+1} &= \dot{\theta}_t + \frac{\dot{x}_t r_t + \dot{x}_t}{1000} \\ \theta_{t+1} &= \theta_t + 0.5(\dot{\theta}_{t+1} + \dot{\theta}_t) \end{aligned}$$

Reward at time step t is equal to $\dot{x}_t$ if $|\theta_t| < 5$ and zero otherwise. In all RKRL experiments with Ship Steering, $\Lambda = 1$. By comparing results in Ship Steering to Mountain Car, it is possible to elucidate how the learner's performance depends on the size of the action space.

4.3.3. CAPTURE GO

The third domain used in this paper is *Capture Go*, a simplified version of Go played where the first player to make a capture wins.[2] The learner plays against a fixed random opponent on a 5×5 board, and receives reward of -1 for a loss and +1 for a win. The board state $\mathbf{x}_t \in \{-1, 0, 1\}^{25}$ is encoded as a vector where -1 entries correspond to opponent pieces, 0 entries correspond to blank territory and 1 entries correspond to the agent's pieces. The agent is given knowledge of *afterstates*, that is, knowledge of how its moves affect the state. In all RKRL experiments using Capture Go, $\Lambda = 1000$. This domain was chosen because it has a high dimensional state vector and stochastic dynamics.

[2]`http://www.usgo.org/teach/capturegame.html`

Table 2. Asymptotic performance on Mountain car. Bold numbers represent statistical significance.

KERNEL	GP-SARSA	RKRL	EP-RKRL
POLYNOMIAL	-67.6 ±0.3	-149 ±5.5	**-63.7 ±0.3**
GAUSSIAN	-230 ±16	-521 ±84	**-66.9 ±0.9**
TANH NORM	-638 ±72	-569 ±41	**-130 ±8.7**
TANH DOT	-482 ±37	-532 ±113	**-97.0 ±2.0**

Table 3. Asymptotic performance ($\times 10^2$) on Ship Steering.

KERNEL	GP-SARSA	RKRL	EP-RKRL
POLYNOMIAL	34.0 ±1.0	3.3 ±0.7	**171 ±241**
GAUSSIAN	2.5 ±0.3	5.0 ±0.6	12.9 ±9.1
TANH NORM	2.1 ±0.8	4.5 ±0.7	**662 ±183**
TANH DOT	2.9 ±0.6	3.0 ±0.8	**19.5 ±15.3**

Table 4. Asymptotic performance (% wins) on Capture Go.

KERNEL	GP-SARSA	RKRL	EP-RKRL
NORM	90.9 ±0.2	76.1 ±4.4	**94.3 ±0.5**
POLYNOMIAL	89.7 ±0.4	69.5 ±1.1	**92.6 ±1.3**
GAUSSIAN	90.3 ±0.5	78.3 ±0.7	**93.3 ±0.1**
TANH NORM	55.7 ±0.2	78.7 ±3.7	**94.5 ±0.6**
TANH DOT	62.4 ±2.8	70.5 ±1.5	**89.1 ±1.1**

5. Results

GP-SARSA, RKRL and EP-RKRL were applied to three RL domains. Section 5.1 summarizes asymptotic performance in the three domains, Section 5.2 compares asymptotic dictionary sizes, Section 5.3 evaluates the learned kernel performance as a stand-alone static kernel and Section 5.4 analyzes the learned kernel hyperparameter settings in Capture Go.

5.1. Asymptotic Reward

Asymptotic performance is evaluated across three domains: Mountain Car, Ship Steering and Capture Go. In each domain EP-RKRL significantly outperforms both GP-SARSA and RKRL over most kernel classes.

5.1.1. MOUNTAIN CAR

In Mountain Car, learning trials are run for 125,000 episodes and asymptotic performance is measured as the average reward over the last 100 episodes. EP-RKRL significantly outperforms both GP-SARSA and RKRL across all kernel classes asymptotically (table 2). GP-SARSA performance using the POLYNOMIAL kernel reaches a peak at -51.6 after 33 episodes, which is significantly better than

EP-RKRL ($p < 10^{-5}$). However, performance degrades significantly with more episodes. This is a phenomenon common to neural-network based function approximators (Sutton & Barto, 1998).

The Mountain Car problem has been studied extensively in RL literature. The best asymptotic results from the Reinforcement Learning Library stand at -53.92 (White, 2007). NEAT+Q, a similar method for combining TD and evolutionary algorithms, achieves -52.0 (Whiteson & Stone, 2006). However, in the former case, different values for ϵ and γ are used and in the latter case, the learner is run for significantly more episodes, making direct comparison difficult. Running Mountain Car using a standard tile-coding function approximator (Sutton & Barto, 1998) with 8 tilings and the same RL parameter settings yields asymptotic performance of -108.9, significantly better than GP-SARSA across all kernels except POLYNOMIAL, but significantly worse than EP-RKRL under all kernels except TANH NORM.

Note that EP-RKRL significantly outperforms RKRL because it discards less experience over the course of learning. Since kernels can only describe smoothness properties of the value functions, the data points themselves become more important for learning; hence discarding them at each model selection step significantly reduces performance. This contrasts with Whiteson's NEAT+Q work precisely because neural networks are more expressive.

5.1.2. SHIP STEERING

In Ship Steering, each learner trains for 2500 episodes (1000 steps each) and the asymptotic performance is measured as the average reward obtained in the last 10 episodes. EP-RKRL significantly outperforms GP-SARSA and RKRL in all kernel classes except GAUSSIAN (table 3). In the remaining three cases, however, EP-RKRL outperforms both methods by several orders of magnitude. Tile coding with 8 tilings yields asymptotic performance of 0.17, significantly higher performance than GP-SARSA in all cases[3] except for the POLYNOMIAL kernel class ($p < 10^{-9}$), but significantly worse than EP-RKRL in all cases.

5.1.3. CAPTURE GO

In Capture Go, each learner trains for $3.75 \cdot 10^6$ episodes, and asymptotic performance is measured as the average number of wins over the last 1000 episodes. EP-RKRL outperforms GP-SARSA and RKRL across all kernel classes (table 4). GP-SARSA's average reward peaks early and declines under the TANH DOT kernel, achieving a maximum of 78.7% wins after 10,000 episodes, still significantly lower than EP-RKRL ($p < 10^{-6}$).

[3]Performance values in table 3 are scaled by a factor of 100.

Table 5. Asymptotic dictionary size for Mountain Car. Bold numbers indicate statistical significance.

KERNEL	GP-SARSA	RKRL	EP-RKRL
POLYNOMIAL	21.6 ±0.4	**10.3 ±0.5**	13.6 ±0.2
GAUSSIAN	29.6 ±0.5	**7.2 ±0.6**	12.5 ±0.2
TANH NORM	**2.5 ±0.1**	8.5 ±0.8	12.6 ±0.2
TANH DOT	7.7 ±0.7	**5.1 ±0.4**	11.7 ±0.4

Table 6. Asymptotic dictionary size for Ship steering.

KERNEL	GP-SARSA	RKRL	EP-RKRL
POLYNOMIAL	15.3 ±0.8	**4.0 ±0.1**	6.2 ±0.3
GAUSSIAN	**12.3 ±0.5**	15.5 ±0.3	13.7 ±0.1
TANH NORM	**3.8 ±0.6**	5.1 ±0.2	12.3 ±1.0
TANH DOT	7.7 ±0.8	**3.2 ±0.1**	5.3 ±0.2

Table 7. Asymptotic dictionary size for Capture Go.

KERNEL	GP-SARSA	RKRL	EP-RKRL
NORM	28.5 ±0.2	5.9 $\pm\epsilon$	**3.0** $\pm\epsilon$
POLYNOMIAL	147.1 ±3.3	**25.2 ±1.4**	40.4 ±1.4
GAUSSIAN	**66.1 ±0.5**	71.7 ±3.4	91.9 ±6.1
TANH NORM	62.0 ±1.1	19.6 ±0.4	**17.5 ±0.7**
TANH DOT	329.6 ±14.4	**25.6 ±1.5**	28.7 ±1.2

5.2. Dictionary Size

RKRL and GP-SARSA can be compared in terms of computational complexity by measuring the final dictionary sizes $|\tilde{\mathcal{D}}|$ of each learning agent. At each decision point, the computational complexity of GPTD is $O(|\tilde{\mathcal{D}}|^2)$, arising from matrix-vector multiplications and partitioned matrix inversion (Engel, 2005). Furthermore, in practice the $O(|\tilde{\mathcal{D}}|)$ cost of computing $\mathbf{k}(\mathbf{x}) \stackrel{\text{def}}{=} (k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_t))^\top$ for $\mathbf{x}_i \in \tilde{\mathcal{D}}$ can carry a high constant overhead for complex kernels. Thus, keeping $|\tilde{\mathcal{D}}|$ small is critical for online performance.

The dictionary sizes for each kernel and learning algorithm pair is given in table 5. In eight of the thirteen cases, EP-RKRL kernels generate significantly smaller dictionaries for $\nu = 0.001$ than GP-SARSA kernels, and likewise in ten of the thirteen cases RKRL generates significantly smaller dictionaries. Thus in the majority of cases employing model selection yields faster learning both in terms of episodes and in terms of computation. However, these dictionary sizes never differ by more than a single order of magnitude, with the largest difference being between RKRL and GP-SARSA the TANH DOT kernel for Capture Go.

5.3. Generalization

How well a particular kernel hyperparameter setting found by EP-RKRL performs depends on what training examples it encounters during learning. Determining to what degree

Table 8. Generalization performance of EP-RKRL kernel parameterizations for Capture Go. In all cases asymptotic performance declines after the original dictionary is discarded.

KERNEL	BASELINE	RELEARNED
NORM	94.3 ±0.5	68.6 ±1.7
POLYNOMIAL	92.6 ±1.3	72.5 ±0.8
GAUSSIAN	93.3 ±0.1	81.1 ±1.0
TANH NORM	94.5 ±0.6	90.7 ±1.9
TANH DOT	89.1 ±1.1	74.6 ±0.7

this performance depends on the exact set of saved training examples yields a notion of how general the kernel parameterization is. In order to evaluate this generalization in EP-RKRL, final kernel parameterizations for Capture Go were saved and all stored training examples were discarded. Learning was then restarted with the kernel parameterization fixed. Table 8 summarizes the results. Across all kernels, the asymptotic performance decreases significantly. Because most training examples are acquired in the first several hundred episodes, this result indicates that the kernel hyperparameter settings are perhaps overfitting to the particular dictionary of saved experience. In other words RKRL is exploiting the acquired data to pick a highly biased covariance function that has low generalization error given that particular set of stored experience.

Although kernels learned through RKRL overfit the dictionary, this is not a serious problem as it improves generalization performance. Overfitting in this nonparametric case simply means that the learned parameterizations are not transferable between agents with different experience. Thus the saved dictionary should be considered part of the model parameters being optimized.

5.4. Analysis of Learned Kernels

Because the hyperparameter space for kernels in Capture Go is high dimensional, RKRL can exploit many kinds of symmetries and patterns in the value function. It is enlightening to analyze whether the learned kernel parameter settings correspond to intuitively meaning covariance functions. Figure 1 plots the 25 hyperparameter values for the TANH NORM kernel averaged over the entire sample for both RKRL and EP-RKRL. The average pattern of weights at the final epoch differs significantly from the expected average over the prior $p(\boldsymbol{\theta})$. Furthermore, the pattern learned in EP-RKRL has a regular structure: Each weight corresponding to a particular board location takes the opposite sign of its neighbors, yielding a regular "checkerboard" pattern of positive and negative weights. This pattern enforces a simple strategy whereby the learner plays to capture isolated stones. Such a strategy is effective against opponents that do not pay attention to stones in danger of being captured.

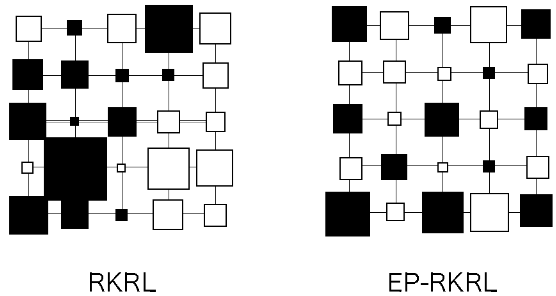

Figure 1. "Hinton diagram" of the average learned hyperparameters by board position for the TAHN NORM kernel in Capture Go. Filled boxes correspond to positive weights and white boxes to negative; box area is proportional to the weight. The parameterization generated by EP-RKRL shows a significant amount of structure, biasing play towards moves that surround single stones.

To further elucidate this result, a second trial was run using the same kernel class, but with only two hyperparameters, corresponding to the positive and negative parameter settings observed above. This *translation invariant* TANH NORM kernel $k(\mathbf{x}, \mathbf{x}') = \tanh(\sum_{i=0}^{25} w_{i \bmod 2}(x_i - x'_i)^2 - 1)$ can express the same alternating positive and negative weights in a more compact form, thus trading off generality compared to the original parameterization. Under the same experimental setup, the translation invariant kernel only reaches an asymptotic performance of 88.6% wins, compared to 94.5% wins with the more general parameterization ($p < 10^{-7}$). Furthermore, the resulting dictionary size of the translation invariant kernel is 32.4, significantly larger than more general parameterization (17.4; $p < 0.001$). The general TANH NORM kernel parameterization also outperforms a variant with built-in rotational symmetry. The *rotationally symmetric* TANH NORM kernel obtains asymptotic performance of 89.3% wins ($p < 10^{-4}$). Taken together these results highlight some of the difficulties of manually building in prior knowledge.

6. Related and Future Work

This paper has presented an efficient and conceptually simple online method for selecting kernels in Bayesian RL. There is a growing body of model selection literature in machine learning and statistics, both on theory and applications, e.g. (Hastie et al., 2001; Seeger, 2001). In RL, model selection has been performed previously using regularization (Jung & Polani, 2006; Loth et al., 2007) and evolutionary methods (Whiteson & Stone, 2006). RKRL is most similar to the latter approach, differing in its use of GPs, ability to save training data across lineages, and simpler hyperparameter optimization procedure.

There are several interesting areas of future work. First, RKRL can be naturally applied to more general kernel classes, e.g. the Matérn kernels, that admit many basic kernels as special cases (Genton, 2002). In particular, the tradeoff between kernel class complexity and performance should be explored.

Second, the learned kernel parameterizations acquired under one dictionary do not perform well under different dictionaries, indicating that the dictionaries themselves can be thought of as hyperparameters to be optimized. Developing such a theory of nonparametric sparsification in RL may lead to significantly better value function approximations.

Third, it is possible to derive a *full kernel-replacing* procedure where all covariance function evaluations share the same accumulated learning, updated after every fitness evaluation of every individual. Such an approach would further reduce the sample complexity of RKRL, and also makes possible the use of *Bayesian Model Averaging* techniques within a single learning agent, i.e. averaging over an ensemble of GP value functions weighted by their predictive likelihoods (Hastie et al., 2001).

Finally, there is a deep connection between action kernels and the concept of Relocatable Action Models (Leffler et al., 2007). It may be possible to cast the latter in terms of state-independent prior covariance functions, yielding a powerful framework for model selection.

7. Conclusion

This paper developed RKRL, a simple online procedure for improving the performance of Gaussian process temporal difference learning by automatically selecting the prior covariance function. In several empirical trials, RKRL yielded significantly higher asymptotic reward than the best hand-picked parameterizations for common covariance functions, even in cases where a large number of hyperparameters must be adapted. Furthermore, the learned covariance functions exhibit highly structured knowledge of the task that would have been difficult to build in *a priori* without significant knowledge of the domain. Overall these initial results are promising, and suggest that leveraging work in statistical model selection will significantly improve online learning.

Acknowledgements

Thanks to the anonymous reviewers for providing very poignant feedback on the original draft, also thanks to Yaakov Engel for help with some implementation details and Tobias Jung and Bryan Silverthorn for helpful discussions. This work was supported in part by an NSF Graduate Research Fellowship to the first author and NSF CAREER award IIS-0237699.

References

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Engel, Y. (2005). *Algorithms and representations for reinforcement learning*. Doctoral dissertation, Hebrew University.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with gaussian processes. *Proc. of ICML-05* (pp. 201–208). New York, NY, USA: ACM Press.

Genton, M. G. (2002). Classes of kernels for machine learning: a statistics perspective. *Journal of Machine Learning Research*, *2*, 299–312.

Gordon, N. J., Salmond, D. J., & Smith, A. F. M. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, *140*, 107–113.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning*. Springer.

Jung, T., & Polani, D. (2006). Least squares svm for least squares td learning. *ECAI* (pp. 499–503). IOS Press.

Leffler, B. R., Littman, M. L., & Edmunds, T. (2007). Efficient reinforcement learning with relocatable action models. *Proc. of AAAI-07* (pp. 572–577). Menlo Park, CA, USA: The AAAI Press.

Loth, M., Davy, M., & Preux, P. (2007). Sparse temporal difference learning using lasso. *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. Hawaii, USA.

Rasmussen, C. E., & Williams, C. K. (2006). *Gaussian processes for machine learning*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press.

Seeger, M. (2001). Covariance kernels from bayesian generative models. *NIPS* (pp. 905–912). MIT Press.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, *14*, 199–222.

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning*. Cambridge, MA, USA: MIT Press.

White, A. (2007). The University of Alberta Reinforcement Learning Library. `http://rlai.cs.ualberta.ca/RLR/`. Edmonton, Alberta: University of Alberta.

Whiteson, S., & Stone, P. (2006). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, *7*, 877–917.