Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning

Yaakov Engel YAKI@ALICE.NC.HUJI.AC.IL
Interdisciplinary Center for Neural Computation, Hebrew University, Jerusalem 91904, Israel

Shie Mannor SHIE@MIT.EDU
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139

Ron Meir RMEIR@EE.TECHNION.AC.IL
Department of Electrical Engineering, Technion Institute of Technology, Haifa 32000, Israel

Abstract

We present a novel Bayesian approach to the problem of value function estimation in continuous state spaces. We define a probabilistic generative model for the value function by imposing a Gaussian prior over value functions and assuming a Gaussian noise model. Due to the Gaussian nature of the random processes involved, the posterior distribution of the value function is also Gaussian and is therefore described entirely by its mean and covariance. We derive exact expressions for the posterior process moments, and utilizing an efficient sequential sparsification method, we describe an on-line algorithm for learning them. We demonstrate the operation of the algorithm on a 2-dimensional continuous spatial navigation domain.

1. Introduction

Gaussian Processes (GPs) have been used extensively in recent years in supervised learning tasks such as classification and regression (e.g. Gibbs & MacKay, 1997; Williams, 1999). Based on a probabilistic generative model, GP methods are theoretically attractive since they allow a Bayesian treatment of these problems, yielding a full posterior distribution rather than a point estimate as in non-Bayesian methods. Moreover, the Gaussian structure often yields closed form expressions for the posterior moments, thus completely avoiding the difficulties associated with optimization algorithms and their convergence behavior. A Gaussian process is a (finite, countably, or uncountably infinite) set of jointly Gaussian random variables. A special case, which is our main focus here is the case where the random variables are indexed by a vector in $\mathbb{R}^d$. In this case each instantiation of the process F is simply a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, and $F(\mathbf{x})$ is a random variable whose distribution is jointly Gaussian with the other components of the process[1]. The prior distribution of the process is fully specified by its mean, which we take to be zero, and its covariance $\mathbf{E}\left(F(\mathbf{x})F(\mathbf{x}')\right) \overset{\text{def}}{=} k(\mathbf{x},\mathbf{x}')$, where $\mathbf{E}(\cdot)$ denotes expectation. In order for $k(\cdot,\cdot)$ to be a legitimate covariance function it is required to be symmetric and positive-definite. Interestingly, these are exactly the requirements made of Mercer kernels, used extensively in kernel machines (Schölkopf & Smola, 2002).

Reinforcement Learning (RL) is a field of machine learning concerned with problems that can be formulated as Markov Decision Processes (MDPs) (Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998). An MDP is a tuple $\{\mathcal{S},\mathcal{A},R,p\}$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively; $R : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ is the immediate reward which may be a random process[2]; $p : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \rightarrow [0,1]$ is the conditional transition distribution. A *stationary policy* $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ is a mapping from states to action selection probabilities. The objective in RL problems is usually to learn an optimal or good suboptimal policy, based on simulated or real experience, but without exact knowledge of the MDP's transition model $\{p, R\}$. Optimality is defined with respect to a *value function*. The value $V(\mathbf{x})$ of a state $\mathbf{x}$ is defined (for some fixed policy π) as the total expected, possibly discounted payoff col-

[1] Here and in the sequel we use capital letters to denote random variables and processes (e.g. $F(\mathbf{x})$ and F, respectively) and lower case letters for their instantiations (e.g. f, $f(\mathbf{x})$). Bold lower case letters denote vectors (e.g. $\mathbf{x}$) with indexed lower case components (e.g. x_i); $\mathbf{0}$ denotes a vector or a matrix of zeros whose dimensions should be clear from the context.

[2] The general case is $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$; to simplify the exposition we ignore the dependence on the action.

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

lected along trajectories starting at that state[3]. The value function of a policy π is defined as

$$V(\mathbf{x}) = \mathbf{E}\left(\sum_{t=0}^{\infty} \gamma^t \bar{R}(\mathbf{x}_t, \mathbf{x}_{t+1}) | \mathbf{x}_0 = \mathbf{x}\right), \qquad (1.1)$$

where the expectation is taken over the policy dependent state transition distribution $p^\pi(\mathbf{x}'|\mathbf{x}) = \sum_{u\in\mathcal{A}} p(\mathbf{x}'|u,\mathbf{x})\pi(u|\mathbf{x})$. Here $\gamma \in [0,1]$ is a discount factor[4], and $\bar{R}(\mathbf{x}_t, \mathbf{x}_{t+1})$ is the expected reward.

The value function can be shown to be the solution to the Bellman equation, which should be satisfied for all $\mathbf{x} \in \mathcal{X}$ [5],

$$V(\mathbf{x}) = \int_{\mathbf{x}'\in\mathcal{X}} p^\pi(\mathbf{x}'|\mathbf{x})\Big(\bar{R}(\mathbf{x},\mathbf{x}') + \gamma V(\mathbf{x}')\Big)d\mathbf{x}'. \quad (1.2)$$

If the state space is finite the integral in Eq. (1.2) is replaced by a sum over $\mathbf{x}' \in \mathcal{X}$. Generally speaking, RL is concerned with finding a policy π^* delivering the highest possible total payoff from each state. Many algorithms for solving this problem are based on the policy iteration method, in which the value function must be estimated for a sequence of fixed policies, making value estimation a crucial algorithmic component. Undoubtedly, the best known method for value estimation is a family of algorithms known as TD(λ) (Sutton, 1988) which utilize temporal differences to make on-line updates to their estimate $\hat{v}$ of the value function. The simplest member of the TD(λ) family is TD(0). The TD(0) update at the transition from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$ is simply

$$\begin{aligned} \hat{v}(\mathbf{x}_t) &:= \hat{v}(\mathbf{x}_t) + \eta_t\delta_t\,, \quad \text{where} \\ \delta_t &= r_t + \gamma\hat{v}(\mathbf{x}_{t+1}) - \hat{v}(\mathbf{x}_t) \end{aligned} \qquad (1.3)$$

is the 1-step temporal difference at time t, r_t is the reward sampled in the transition and η_t is a time dependent learning rate.

Up to now, kernel methods have been largely ignored in the field of RL. This is due to two reasons. First and foremost, kernel algorithms generally scale superlinearly in the number of samples, both in terms of space and time. Second, most of these algorithms require random and/or repeated access to training samples. For these reasons, kernel algorithms, GPs included, were considered ill-fitting to the RL domain which normally requires on-line operation. As far as we are aware the only exception is Dietterich and Wang (2001) in which the value function is estimated off-line using a support vector machine.

The rest of the paper is organized as follows. In the following section we present our generative model for the value function and derive the exact posterior value distribution. In Section 3 we describe our on-line sparsification method; Section 4 unifies the results of the two preceding sections in the description of an on-line sparse algorithm for GP temporal difference (GPTD) learning. In Section 5 we present experiments with GPTD on continuous 2-dimensional mazes, and we conclude in Section 6.

2. Gaussian Processes for TD Learning

In analogy to GP regression we impose a Gaussian prior over value functions, i.e., $V \sim \mathcal{N}(0, k(\cdot,\cdot))$, which means that V is a GP for which, a priori, $\mathbf{E}(V(\mathbf{x})) = 0$ and $\mathbf{E}(V(\mathbf{x})V(\mathbf{x}')) = k(\mathbf{x},\mathbf{x}')$ for all $\mathbf{x},\mathbf{x}' \in \mathcal{X}$. The form of the function k should reflect our prior knowledge concerning the similarity of states in the domain at hand. Recalling the definition of the temporal differences (1.3), we propose the following generative model for the sequence of rewards corresponding to the trajectory $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$:

$$R(\mathbf{x}_i, \mathbf{x}_{i+1}) = V(\mathbf{x}_i) - \gamma V(\mathbf{x}_{i+1}) + N(\mathbf{x}_i)\,, \qquad (2.4)$$

where N is a white Gaussian noise process that does not depend on V but may be input dependent, i.e., $N \sim \mathcal{N}(0,\Sigma)$ with $\Sigma(\mathbf{x},\mathbf{x}') = \sigma_0(\mathbf{x})^2\delta(\mathbf{x}-\mathbf{x}')$, where δ denotes the Dirac delta function. To simplify matters we will assume that $\sigma_0(\mathbf{x})$ is constant over $\mathcal{X}$ and denote it simply by σ_0. Eq. (2.4) may be viewed as a latent variable model in which the value process plays the role of the latent or hidden variable while the reward process plays the role of the observable output variable. For a finite trajectory of length t we define the (finite dimensional) random processes

$$\begin{aligned} R_t &= (R(\mathbf{x}_1), \ldots, R(\mathbf{x}_t))^\top\,, \\ V_t &= (V(\mathbf{x}_1), \ldots, V(\mathbf{x}_t))^\top\,, \\ N_t &= (N(\mathbf{x}_1), \ldots, N(\mathbf{x}_t))^\top\,, \end{aligned} \qquad (2.5)$$

and the vector and matrices (respectively)

$$\begin{aligned} \mathbf{k}_t(\mathbf{x}) &= (k(\mathbf{x}_1,\mathbf{x}), \ldots, k(\mathbf{x}_t,\mathbf{x}))^\top\,, \\ \mathbf{K}_t &= [\mathbf{k}_t(\mathbf{x}_1), \ldots, \mathbf{k}_t(\mathbf{x}_t)]\,, \\ \mathbf{\Sigma}_t &= \mathrm{diag}(\sigma_0^2, \ldots, \sigma_0^2)\,, \end{aligned} \qquad (2.6)$$

where diag($\cdot$) denotes a diagonal matrix whose diagonal is the argument vector. Using these definitions we

[3] The value should strictly be denoted as V^π; however, since most of this paper deals with a fixed policy (with the exception of Section 5), we retain the simpler notation V.

[4] When $\gamma = 1$ the policy must be proper, see Bertsekas and Tsitsiklis (1996).

[5] This is a degenerate special case of the Bellman equation which results from our problem statement, namely evaluating the value function of a *fixed* policy.

may write

$$\begin{pmatrix} N_t \\ V_t \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} \Sigma_t & 0 \\ 0 & \mathbf{K}_t \end{bmatrix} \right\}. \qquad (2.7)$$

Defining the $(t-1) \times t$ matrix

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -\gamma \end{bmatrix}, \qquad (2.8)$$

we may write (2.4) concisely as

$$R_{t-1} = \mathbf{H}_t V_t + N_{t-1} \, . \qquad (2.9)$$

Using standard results on jointly Gaussian random variables (Scharf, 1991) we obtain

$$\begin{pmatrix} R_{t-1} \\ V(\mathbf{x}) \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} \mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \Sigma_{t-1} & \mathbf{H}_t \mathbf{k}_t(\mathbf{x}) \\ \mathbf{k}_t(\mathbf{x})^\top \mathbf{H}_t^\top & k(\mathbf{x},\mathbf{x}) \end{bmatrix} \right\},$$

and the posterior distribution of the value at some point $\mathbf{x}$, conditioned on the observed sequence of rewards $\mathbf{r}_{t-1} = (r_1, \dots, r_{t-1})^\top$ is given by

$$(V(\mathbf{x}) | R_{t-1} = \mathbf{r}_{t-1}) \sim \mathcal{N}\left\{ \hat{v}_t(\mathbf{x}), p_t(\mathbf{x}) \right\}, \qquad (2.10)$$

where

$$\begin{aligned} \hat{v}_t(\mathbf{x}) &= \mathbf{k}_t(\mathbf{x})^\top \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1} \, , \\ p_t(\mathbf{x}) &= k_{xx} - \mathbf{k}_t(\mathbf{x})^\top \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{H}_t \mathbf{k}_t(\mathbf{x}) \, , \\ \text{with} \quad \mathbf{Q}_t &= (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \Sigma_{t-1})^{-1} \, , \\ \text{and} \quad k_{xx} &= k(\mathbf{x},\mathbf{x}) \, . \end{aligned} \qquad (2.11)$$

The expressions above can be written in a somewhat more familiar way, by separating the input dependent term $\mathbf{k}_t(\mathbf{x})$ from the learned terms:

$$\hat{v}_t(\mathbf{x}) = \mathbf{k}_t(\mathbf{x})^\top \alpha_t \, , \; p_t(\mathbf{x}) = k_{xx} - \mathbf{k}_t(\mathbf{x})^\top \mathbf{C}_t \mathbf{k}_t(\mathbf{x}) \, , \qquad (2.12)$$

where $\alpha_t = \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{r}_{t-1}$ and $\mathbf{C}_t = \mathbf{H}_t^\top \mathbf{Q}_t \mathbf{H}_t$.

While this concludes the derivation of the value GP posterior, there are several points worth noting. First, note that in addition to the value estimate $\hat{v}_t(\mathbf{x})$ we are provided with an uncertainty measure $p_t(\mathbf{x})$ for that estimate. This opens the way to a wide range of possibilities, some of which we touch upon in Section 6. Second, the assumptions we make on the noise process N seem to be questionable. Let us consider the validity of these assumptions. The value process satisfies the Bellman equation (1.2) while we assume the model (2.4). Equating the expressions for $V(\mathbf{x}_i)$ from these two equations we get

$$N(\mathbf{x}_i) = \gamma \int \left(p^\pi(\mathbf{x}'|\mathbf{x}_i) - \delta(\mathbf{x}' - \mathbf{x}_{i+1})\right) V(\mathbf{x}') d\mathbf{x}'$$

$$+ \int \left(p^\pi(\mathbf{x}'|\mathbf{x}_i) \bar{R}(\mathbf{x}_i, \mathbf{x}') - \delta(\mathbf{x}' - \mathbf{x}_{i+1}) R(\mathbf{x}_i, \mathbf{x}')\right) d\mathbf{x}' \, .$$

When state transitions are deterministic $p^\pi(\mathbf{x}'|\mathbf{x}) = \delta(\mathbf{x}' - \mathbf{x}_{i+1})$, the first integral vanishes and we are left with

$$N(\mathbf{x}_i) = \bar{R}(\mathbf{x}_i, \mathbf{x}_{i+1}) - R(\mathbf{x}_i, \mathbf{x}_{i+1}) \, ,$$

which is exact if $R(\mathbf{x}, \mathbf{x}') \sim \mathcal{N}\left\{\bar{R}(\mathbf{x}, \mathbf{x}'), \sigma_0^2(\mathbf{x})\right\}$ $\forall \mathbf{x}, \mathbf{x}'$. As the state transitions move away from determinicity this equality turns into an approximation. In Section 5 we experimentally demonstrate the effect of non-deterministic transitions on the value estimate.

Finally, we note that computing the exact GP estimators is computationally unfeasible in all but the smallest and simplest domains, since we need to compute and store α_t ($t \times 1$) and $\mathbf{C}_t$ ($t \times t$), incurring a cost of $O(t^3)$ time and $O(t^2)$ space (The $O(t^3)$ dependence arises from inverting the matrix $\mathbf{Q}_t$). In the following section we describe an efficient on-line sparsification method that reduces this computational burden to a level that allows us to compute a good approximation of the GP posterior on-line.

3. On-Line Sparsification

In order to render the algorithm described above practical we need to reduce the computational burden associated with the computation of the functions $\hat{v}_t$ and p_t in (2.11). In Engel et al. (2002) an on-line sparsification method for kernel algorithms was proposed in the context of Support Vector Regression. This method is based on the following observation: Due to Mercer's Theorem the kernel function $k(\cdot,\cdot)$ may be viewed as an inner product in a high (possibly infinite) dimensional Hilbert space $\mathcal{H}$ (see Schölkopf & Smola, 2002 for details). This means that there exists a generally non-linear mapping $\phi : \mathcal{X} \to \mathcal{H}$ for which $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}} = k(\mathbf{x}, \mathbf{x}')$. Although the dimension of $\mathcal{H}$ may be exceedingly high, the *effective* dimensionality of the manifold spanned by the set of vectors $\{\phi(\mathbf{x}_i)\}_{i=1}^t$ is at most t, and may be significantly lower. Consequently, any expression describable as a linear combination of these vectors may be expressed, to arbitrary accuracy, by a smaller set of linearly independent feature vectors which *approximately* span this manifold[6].

[6] Based on similar ideas, several other sparsification algorithms have been previously proposed, e.g. Burges, 1996;

Taking advantage of this insight we maintain a dictionary of samples that suffice to approximate any training sample (in feature space) up to some predefined accuracy threshold. Our method starts with an empty dictionary $\mathcal{D}_0 = \{\}$. It observes the sequence of states $\mathbf{x}_1, \mathbf{x}_2, \ldots$ one state at a time, and admits $\mathbf{x}_t$ into the dictionary only if its feature space image $\phi(\mathbf{x}_t)$ cannot be approximated sufficiently well by combining the images of states already in $\mathcal{D}_{t-1} = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_{m_{t-1}}\}$. Given the set of m_{t-1} linearly independent feature vectors corresponding to the dictionary states at time $t-1$, $\{\tilde{\phi}(\tilde{\mathbf{x}}_j)\}_{j=1}^{m_{t-1}}$, we seek the least squares approximation of $\phi(\mathbf{x}_t)$ in terms of this set. It can be easily verified that this approximation is given by the solution to the following quadratic problem:

$$\min_{\boldsymbol{a}} \left\{ \boldsymbol{a}^\top \tilde{\mathbf{K}}_{t-1} \boldsymbol{a} - 2\boldsymbol{a}^\top \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) + k_{tt} \right\}, \qquad (3.13)$$

where $\tilde{\mathbf{K}}_{t-1}$ is the kernel matrix of the dictionary states a time $t-1$ (i.e., $[\tilde{\mathbf{K}}_{t-1}]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ with $i, j = 1, \ldots, m_{t-1}$), $\tilde{\mathbf{k}}_{t-1}(\mathbf{x}) = (k(\tilde{\mathbf{x}}_1, \mathbf{x}), \ldots, k(\tilde{\mathbf{x}}_{m_{t-1}}, \mathbf{x}))^\top$ and $k_{tt} = k(\mathbf{x}_t, \mathbf{x}_t)$. The solution to (3.13) is $\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$ and substituting it back to (3.13) we obtain the minimal squared error incurred by the approximation,

$$\varepsilon_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \boldsymbol{a}_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) . \qquad (3.14)$$

If $\varepsilon_t > \nu$, where ν is an accuracy threshold parameter, we add $\mathbf{x}_t$ to the dictionary and we set $\boldsymbol{a}_t = (0, \ldots, 1)^\top$ since $\tilde{\mathbf{x}}_t$ is exactly represented by itself. If $\varepsilon_t \leq \nu$ the dictionary remains unchanged. Either way we are assured that all the feature vectors corresponding to states seen until time t can be approximated by the dictionary at time t with a maximum squared error ν,

$$\phi(\mathbf{x}_i) = \sum_{j=1}^{m_i} a_{i,j} \phi(\tilde{\mathbf{x}}_j) + \phi_i^{res}, \quad \text{where } \|\phi_i^{res}\|^2 \leq \nu . \qquad (3.15)$$

In order to be able to compute $\boldsymbol{a}_t$ at each time step, we need to update $\tilde{\mathbf{K}}_t^{-1}$ whenever the dictionary is appended with a new state. This is done via the partitioned matrix inversion formula (Scharf, 1991):

$$\begin{aligned} \tilde{\mathbf{K}}_t &= \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^\top & k_{tt} \end{bmatrix}, \\ \tilde{\mathbf{K}}_t^{-1} &= \frac{1}{\varepsilon_t} \begin{bmatrix} \varepsilon_t \tilde{\mathbf{K}}_{t-1}^{-1} + \hat{\boldsymbol{a}}_t \hat{\boldsymbol{a}}_t^\top & -\hat{\boldsymbol{a}}_t \\ -\hat{\boldsymbol{a}}_t^\top & 1 \end{bmatrix}, \end{aligned} \qquad (3.16)$$

where $\hat{\boldsymbol{a}}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$. Note that $\hat{\boldsymbol{a}}_t$ equals the vector $\boldsymbol{a}_t$ computed in solving (3.13), so there is no need to recompute it.

Defining the matrices[7] $[\mathbf{A}_t]_{i,j} = a_{i,j}$, $\mathbf{\Phi}_t = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_t)]$, and $\mathbf{\Phi}_t^{res} = [\phi_1^{res}, \ldots, \phi_t^{res}]$ we may write Eq. (3.15) for all time-steps up to t, concisely as

$$\mathbf{\Phi}_t = \tilde{\mathbf{\Phi}}_t \mathbf{A}_t^\top + \mathbf{\Phi}_t^{res} . \qquad (3.17)$$

By pre-multiplying (3.17) with its transpose we obtain a decomposition of the full $t \times t$ kernel matrix $\mathbf{K}_t = \mathbf{\Phi}_t^\top \mathbf{\Phi}_t$ into two matrices:

$$\mathbf{K}_t = \mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top + \mathbf{K}_t^{res} , \qquad (3.18)$$

where $\tilde{\mathbf{K}}_t = \tilde{\mathbf{\Phi}}_t^\top \tilde{\mathbf{\Phi}}_t$. The matrix $\mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top$ is a rank m_t approximation of $\mathbf{K}_t$. It can be shown that the norm of the residual matrix $\mathbf{K}_t^{res}$ is bounded above by a factor linear in ν. Consequently we make the following approximations:

$$\mathbf{K}_t \approx \mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top , \quad \mathbf{k}_t(\mathbf{x}) \approx \mathbf{A}_t \tilde{\mathbf{k}}_t(\mathbf{x}) . \qquad (3.19)$$

The symbol $\approx$ implies that the difference between the two sides of the equation is $O(\nu)$. We note that the computational cost per time step of this sparsification algorithm is $O(m_t^2)$ which, assuming m_t does not depend asymptotically on t, is independent of time[8], allowing this algorithm to operate on-line. Our on-line version of the GPTD algorithm draws its computational leverage from the low rank approximation provided by this sparsification algorithm. In the next section we show how.

4. On-Line GPTD

We are now ready to combine the ideas developed in the last two sections. Substituting the approximations (3.19) into the exact GP solution (2.12) we obtain

$$\begin{aligned} \hat{v}_t(\mathbf{x}) &= \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\boldsymbol{\alpha}}_t , \\ p_t(\mathbf{x}) &= k_{xx} - \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\mathbf{C}}_t \tilde{\mathbf{k}}_t(\mathbf{x}) , \end{aligned} \qquad (4.20)$$

where we define

$$\begin{aligned} \tilde{\mathbf{H}}_t &= \mathbf{H}_t \mathbf{A}_t , \\ \tilde{\mathbf{Q}}_t &= (\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\top + \mathbf{\Sigma}_t)^{-1} , \\ \tilde{\boldsymbol{\alpha}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \mathbf{r}_{t-1} , \\ \tilde{\mathbf{C}}_t &= \tilde{\mathbf{H}}_t^\top \tilde{\mathbf{Q}}_t \tilde{\mathbf{H}}_t . \end{aligned} \qquad (4.21)$$

Note that the parameters we are required to store and update in order to evaluate the posterior mean and

Smola & Bartlett, 2001; Williams & Seeger, 2001; Csató & Opper, 2001, to mention a few. However, all but the latter are inapplicable to the on-line setting.

[7] Due to the sequential nature of the algorithm, for $j > m_i$, $[\mathbf{A}_t]_{i,j} = 0$.

[8] We can bound m_t under mild assumptions on the kernel and the space $\mathcal{X}$, however space does not allow us to pursue this and other interesting issues concerning our sparsification method any further here.

covariance are now $\tilde{\alpha}_t$ and $\tilde{\mathbf{C}}_t$ whose dimensions are $m_t \times 1$ and $m_t \times m_t$, respectively. We will now derive recursive update formulas for $\tilde{\alpha}_t$ and $\tilde{\mathbf{C}}_t$. At each time step t we may be faced with either one of the following two cases. Either $\mathcal{D}_t = \mathcal{D}_{t-1}$ or $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$.

Case 1. $\mathcal{D}_t = \mathcal{D}_{t-1}$: Hence $\tilde{\mathbf{K}}_t = \tilde{\mathbf{K}}_{t-1}$,

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1} \\ a_t^\top \end{bmatrix}, \quad \tilde{\mathbf{H}}_t = \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \\ a_{t-1}^\top - \gamma a_t^\top \end{bmatrix}.$$

Defining $\tilde{\mathbf{h}}_t = a_{t-1} - \gamma a_t$,

$$\tilde{\mathbf{Q}}_t^{-1} = \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} \\ \tilde{\mathbf{h}}_t^\top \end{bmatrix} \tilde{\mathbf{K}}_{t-1} \left[\tilde{\mathbf{H}}_{t-1}, \tilde{\mathbf{h}}_t\right] + \begin{bmatrix} \Sigma_{t-1} & 0 \\ 0^\top & \sigma_0^2 \end{bmatrix}$$
$$= \begin{bmatrix} \tilde{\mathbf{Q}}_{t-1}^{-1} & \tilde{\mathbf{H}}_{t-1}\tilde{\mathbf{K}}_{t-1}\tilde{\mathbf{h}}_t \\ (\tilde{\mathbf{H}}_{t-1}\tilde{\mathbf{K}}_{t-1}\tilde{\mathbf{h}}_t)^\top & \tilde{\mathbf{h}}_t^\top\tilde{\mathbf{K}}_{t-1}\tilde{\mathbf{h}}_t + \sigma_0^2 \end{bmatrix}.$$

Defining $\Delta\tilde{\mathbf{k}}_{t-1} \overset{\text{def}}{=} \tilde{\mathbf{K}}_{t-1}\tilde{\mathbf{h}}_t = \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_{t-1}) - \gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)$, we get

$$\tilde{\mathbf{Q}}_t^{-1} = \begin{bmatrix} \tilde{\mathbf{Q}}_{t-1}^{-1} & \tilde{\mathbf{H}}_{t-1}\Delta\tilde{\mathbf{k}}_{t-1} \\ (\tilde{\mathbf{H}}_{t-1}\Delta\tilde{\mathbf{k}}_{t-1})^\top & \tilde{\mathbf{h}}_t^\top\Delta\tilde{\mathbf{k}}_{t-1} + \sigma_0^2 \end{bmatrix}.$$

We obtain $\tilde{\mathbf{Q}}_t$ using the partitioned matrix inversion formula.

$$\tilde{\mathbf{Q}}_t = \frac{1}{s_t} \begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + g_t g_t^\top & -g_t \\ -g_t^\top & 1 \end{bmatrix},$$

where $g_t = \tilde{\mathbf{Q}}_{t-1}\tilde{\mathbf{H}}_{t-1}\Delta\tilde{\mathbf{k}}_{t-1}$, $s_t = \sigma_0^2 - \mathbf{c}_t^\top\Delta\tilde{\mathbf{k}}_{t-1}$ and $\mathbf{c}_t = \tilde{\mathbf{H}}_{t-1}g_t - \tilde{\mathbf{h}}_t$. Let us now compute $\tilde{\alpha}_t$.

$$\tilde{\alpha}_t = \tilde{\mathbf{H}}_t^\top\tilde{\mathbf{Q}}_t\mathbf{r}_{t-1}$$
$$= \frac{1}{s_t}\left[\tilde{\mathbf{H}}_{t-1}^\top, \tilde{\mathbf{h}}_t\right] \begin{bmatrix} s_t\tilde{\mathbf{Q}}_{t-1} + g_t g_t^\top & -g_t \\ -g_t^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_{t-2} \\ r_{t-1} \end{bmatrix}$$
$$= \tilde{\alpha}_{t-1} + \frac{\mathbf{c}_t}{s_t}\left(\Delta\tilde{\mathbf{k}}_{t-1}^\top\tilde{\alpha}_{t-1} - r_{t-1}\right). \tag{4.22}$$

Similarly, we get

$$\tilde{\mathbf{C}}_t = \tilde{\mathbf{C}}_{t-1} + \frac{1}{s_t}\mathbf{c}_t\mathbf{c}_t^\top. \tag{4.23}$$

Case 2. $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$: Here, $\tilde{\mathbf{K}}_t$ is given by (3.16). Furthermore, $a_t = (0, \ldots, 1)^\top$ since $\phi(\mathbf{x}_t)$ is exactly representable by itself; therefore

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1} & 0 \\ 0^\top & 1 \end{bmatrix}, \quad \tilde{\mathbf{H}}_t = \begin{bmatrix} \tilde{\mathbf{H}}_{t-1} & 0 \\ a_{t-1}^\top & -\gamma \end{bmatrix}.$$

Going through the algebra we get

$$\tilde{\mathbf{Q}}_t^{-1} = \begin{bmatrix} \tilde{\mathbf{Q}}_{t-1}^{-1} & \tilde{\mathbf{H}}_{t-1}\Delta\tilde{\mathbf{k}}_{t-1} \\ (\tilde{\mathbf{H}}_{t-1}\Delta\tilde{\mathbf{k}}_{t-1})^\top & \Delta k_{tt} + \sigma_0^2 \end{bmatrix},$$

where $\Delta k_{tt} \overset{\text{def}}{=} a_{t-1}^\top\left(\Delta\tilde{\mathbf{k}}_{t-1} - \gamma\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)\right) + \gamma^2 k_{tt}$. Denoting $\hat{s}_t = \sigma_0^2 + \Delta k_{tt} - \Delta\tilde{\mathbf{k}}_{t-1}^\top\tilde{\mathbf{C}}_{t-1}\Delta\tilde{\mathbf{k}}_{t-1}$ and using again the partitioned matrix inversion formula we get

$$\tilde{\mathbf{Q}}_t = \frac{1}{\hat{s}_t} \begin{bmatrix} \hat{s}_t\tilde{\mathbf{Q}}_{t-1} + g_t g_t^\top & -g_t \\ -g_t^\top & 1 \end{bmatrix},$$

with g_t as before. Defining $\hat{\mathbf{c}}_t = \tilde{\mathbf{H}}_{t-1}g_t - a_{t-1}$, we can now compute $\tilde{\alpha}_t$ and $\tilde{\mathbf{C}}_t$:

$$\tilde{\alpha}_t = \tilde{\mathbf{H}}_t^\top\tilde{\mathbf{Q}}_t\mathbf{r}_{t-1}$$
$$= \begin{bmatrix} \tilde{\alpha}_{t-1} + \frac{\hat{\mathbf{c}}_t}{\hat{s}_t}\left(\Delta\tilde{\mathbf{k}}_{t-1}^\top\tilde{\alpha}_{t-1} - r_{t-1}\right) \\ \frac{\gamma}{\hat{s}_t}\left(\Delta\tilde{\mathbf{k}}_{t-1}^\top\tilde{\alpha}_{t-1} - r_{t-1}\right) \end{bmatrix}, \tag{4.24}$$

$$\tilde{\mathbf{C}}_t = \begin{bmatrix} \tilde{\mathbf{C}}_{t-1} + \frac{1}{\hat{s}_t}\hat{\mathbf{c}}_t\hat{\mathbf{c}}_t^\top & \frac{\gamma}{\hat{s}_t}\hat{\mathbf{c}}_t \\ \frac{\gamma}{\hat{s}_t}\hat{\mathbf{c}}_t^\top & \frac{\gamma^2}{\hat{s}_t} \end{bmatrix}. \tag{4.25}$$

At any point in time (say t) we can compute the prediction for the value and its variance at some arbitrary state $\mathbf{x}$ using the analog of (2.12):

$$\hat{v}_t(\mathbf{x}) = \tilde{\mathbf{k}}_t(\mathbf{x})^\top\tilde{\alpha}_t, \quad p_t(\mathbf{x}) = k_{xx} - \tilde{\mathbf{k}}_t(\mathbf{x})^\top\tilde{\mathbf{C}}_t\tilde{\mathbf{k}}_t(\mathbf{x}). \tag{4.26}$$

Note that the $\Delta\tilde{\mathbf{k}}_{t-1}^\top\tilde{\alpha}_{t-1} - r_{t-1}$ term appearing in the update rules for $\tilde{\alpha}_t$ (4.22) and (4.24) is (up to a sign) the temporal difference δ_{t-1} defined in (1.3). Another point worth noting is that evaluating $\hat{v}_t(\mathbf{x})$ and $p_t(\mathbf{x})$ costs only $O(m_t)$ and $O(m_t^2)$ time, respectively. Table 1 outlines the algorithm in pseudo-code.

So far we have described the on-line GPTD algorithm for infinite horizon discounted problems. It is straightforward to modify the algorithm to handle episodic tasks. In this kind of tasks the learning agent's experience is composed of a series of learning episodes or trials. At the beginning of each trial the agent is normally placed at a random starting position and then follows some trajectory until it reaches one of the absorbing terminal states (i.e., having a zero outgoing transition probability), and whose value is fixed to zero. Algorithmically, all that is required is to temporarily set γ to zero when a goal state is reached, and make a zero-reward transition to the starting point of the next trial. The extension to semi-Markov decision processes, where transitions may require different amounts of time, is also possible.

5. Experiments

In this section we present several experiments meant to demonstrate the strengths and weaknesses of the

Table 1. The On-Line GPTD Algorithm

```
**Parameters:** $\nu$, $\sigma_0$, $\gamma$
**Initialize:** $\mathcal{D}_1 = \{\mathbf{x}_1\}$, $\tilde{\boldsymbol{\alpha}}_1 = 0$, $\tilde{\mathbf{C}}_1 = 0$, $\tilde{\mathbf{K}}_1^{-1} = 1/k_{11}$
**for** $t = 2, 3, \ldots$
  **Observe transition:** $\mathbf{x}_t$, $r_{t-1}$
  **Sparsification test:**
    **if** $\varepsilon_t > \nu$ (3.14)    % add $\mathbf{x}_t$ to dictionary
      $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\tilde{\mathbf{x}}_t\}$
      Compute $\tilde{\mathbf{K}}_t^{-1}$ (3.16)
      Compute $\tilde{\boldsymbol{\alpha}}_t$, $\tilde{\mathbf{C}}_t$ (4.24,4.25)
    **else**    % dictionary unchanged
      Compute $\tilde{\boldsymbol{\alpha}}_t$, $\tilde{\mathbf{C}}_t$ (4.22,4.23)

**Output:** $\mathcal{D}_t$, $\tilde{\boldsymbol{\alpha}}_t$, $\tilde{\mathbf{C}}_t$
```

GPTD algorithm. We start with a simple maze in order to demonstrate GPTD's ability to provide an uncertainty measure for the value estimate, and its data efficiency, i.e. its ability to extract reasonable value maps from very little data. We then move beyond value estimation to the more challenging task of learning the optimal policy (or a good approximation thereof). We use a more difficult maze and experiment with both deterministic and stochastic state transitions.

Our experimental test-bed is a continuous 2-dimensional square world with unit-length sides. An agent roaming this world may be located at any point, but can perform only a finite number of actions. The actions are 0.1-long steps in one of the 8 major compass winds, with an added Gaussian noise with a standard deviation of 0.05. Time is also discrete $t = 1, 2, 3, \ldots$. In this world there may be one or more rectangular goal regions and possibly also obstacles - piecewise linear curves, which the agent cannot cross. As long as the agent is not in a goal region it receives a reward of -1 per time-step. Upon reaching a goal state the agent is given a zero reward and is then placed at some random state to begin a new trial.

We begin with a simple experiment. The maze, shown in Fig. 1, has a single goal region stretching the entire length of the south wall of the maze (from y=0 to y=0.1). We chose the non-homogeneous polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + 1)^5$, which corresponds to features that are monomials of up to degree 5 in the coordinates (Schölkopf & Smola, 2002), and subtracted 0.5 from each coordinate to avoid any asymmetric bias. The exploration policy is a stochastic one in which a southward move is taken with probability 0.8, otherwise a random move is performed. In Fig. 1 we show the results after a *single trial* in which 12 states were visited including a final goal state. This example illustrates the efficiency of the algorithm when the kernel function is chosen judiciously. As can be seen at the bottom of Fig. 1 a single policy iteration sweep (i.e., choosing the greedy action with respect to the value function estimate) extracts a near-optimal policy for a large section of the maze surrounding the states visited. Looking at the variance map it can be seen that in the proximity of the visited states the uncertainty in the value prediction is significantly lower than in other regions.

As we mentioned in the introduction, a value function estimation algorithm is usually a component in a larger RL system whose aim is to learn the *optimal* policy, namely, the one maximizing the total payoff per trial, or in our case, the one minimizing the time it takes the agent to reach a goal region. One such RL algorithm that has worked surprisingly well in certain domains (e.g. Tesauro, 1995), although it possess no theoretical guarantees for convergence, is Optimistic Policy Iteration (OPI) (Bertsekas & Tsitsiklis, 1996). In OPI the agent does not follow a fixed stationary policy; instead, at each time step it utilizes a model of its environment and its current value estimate to guess the expected payoff for each of the actions available to it. It then greedily chooses the highest ranking action. We ran an agent utilizing OPI on the maze shown in Fig. 2 for 100 trials, once with deterministic transitions, and a second time with the noisy transitions described above. The kernel we used here was Gaussian $k(\mathbf{x}, \mathbf{x}') = \exp\left(\|\mathbf{x} - \mathbf{x}'\|^2/(2\sigma_k^2)\right)$, where $\sigma_k = 0.1$. The feature space induced by this choice is infinite-dimensional (Schölkopf & Smola, 2002). The value maps learned and their corresponding greedy policies are shown in Fig. 2. Note that while the policies learned are both similar and quite close to optimal, the value estimates are different. More specifically, the value estimates in the stochastic case seem to be dampened. The variance maps are omitted, since over the entire maze the variance estimate is close to zero.

6. Discussion

We presented a novel temporal difference learning algorithm based on Gaussian processes and an efficient sparsification scheme. Using these two essential ingredients we were able to suggest a practical and efficient method for value function approximation in large and even infinite state spaces.

Several approaches to RL in continuous space were considered to date, mostly using parametric function approximation architecture that are usually employed

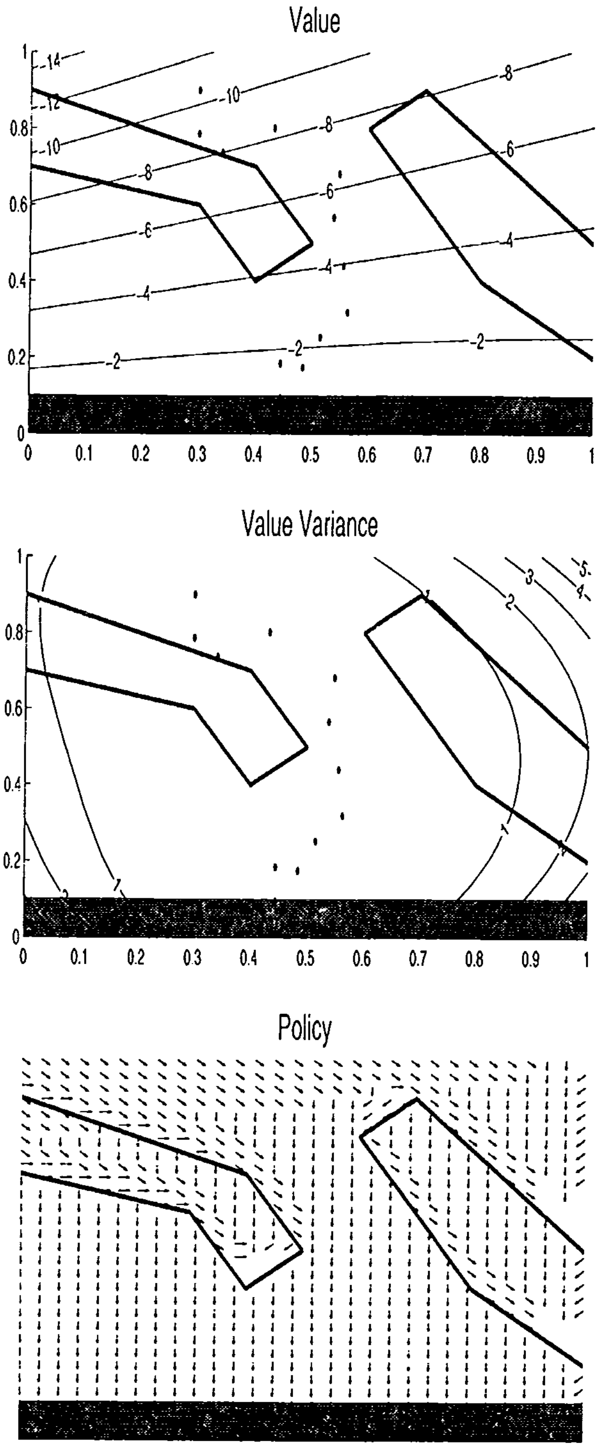

Figure 1. The results of a single 12-step trial on the simple maze shown in the figures, sampled on a 30 by 30 grid. From top to bottom: Top - the points visited during the trial and contour lines of the value function estimate. Center - The variance of the value estimates. Bottom - A greedy policy with respect to the value estimate.

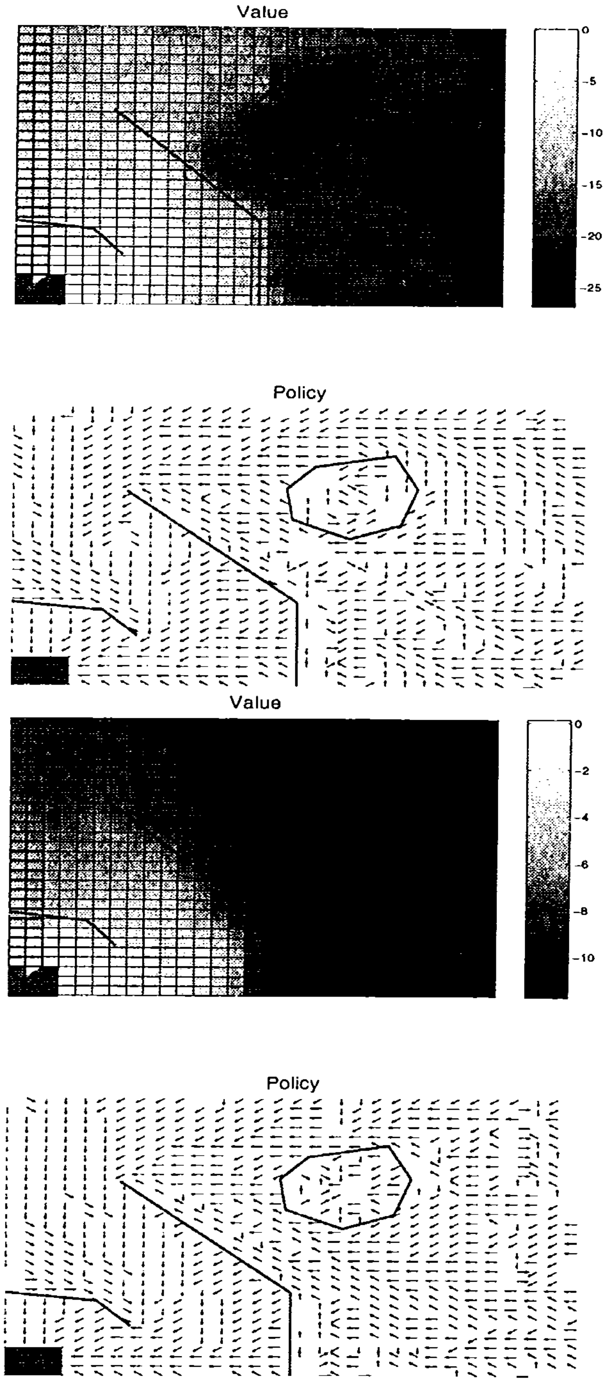

Figure 2. The results after 100 trials on the more difficult maze shown in the figures. GPTD with OPI is used to find a near-optimal policy. The upper two figures show the results for deterministic state transition while the lower two for stochastic ones. For each pair the final value function is shown at the top and its corresponding greedy policy at the bottom. The results shown are samples over a 30 by 30 grid.

in conjunction with gradient based learning rules (e.g. Neural networks, CMACs, see Bertsekas and Tsitsiklis (1996); Sutton and Barto (1998) for a review). In such architectures one must choose the complexity (i.e., the number of tunable parameters) of the model a priori. Other approximation methods include fuzzy sets, hard and soft state aggregation, and non-parametric methods such as variable resolution, instance-based learning, viscosity solutions (Munos, 2000) and local averaging (Ormoneit & Sen, 2002) (We refer the reader to Sutton and Barto (1998) for a review of all but the last two of these methods).

Our approach is fundamentally different from the above. We postulate a probabilistic generative model for the value and use Bayesian analysis to extract a posterior model. Moreover, our approach is non-parametric - the complexity of the value representation (the size of the dictionary) is adapted on-line to match the task at hand. Our method goes beyond Reinforcement Learning and can be also applied to solve control problem when the dynamics are *known* but difficult to capture. Specifically, our method can replace discretization based methods (see, e.g. Rust, 1997 and references therein),

A significant advantage of the Gaussian process approach is that an error estimate is provided at no extra cost. Apart from the benefit of having confidence intervals for the value estimate, there are several other ways in which this may be put into use. For instance, the RL agent may use this information to actively direct its exploration (see Dearden et al., 1998); another possibility would be to use the confidence measure to provide stopping conditions for trials in episodic tasks, as suggested by Bertsekas and Tsitsiklis (1996).

There are several natural directions for future research. First, an extension of GPTD to GPTD(λ) is called for. We conjecture that such an extension will help alleviate the problems experienced with GPTD in stochastic environments. Second, as mentioned above, we plan to test new RL methods that make use of the value uncertainty measure provided by GPTD. Third, we would like to establish convergence guarantees in the spirit of Tsitsiklis and Van-Roy (1997). Finally, we aim at reliably solving the complete RL problem, i.e., finding the optimal policy. This may be achieved using an Actor-Critic architecture (Sutton & Barto, 1998).

Acknowledgments The authors are grateful to N. Shimkin and I. Menache for helpful discussions. S.M. was partially supported by the MIT-Merrill Lynch Partnership.

References

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming.* Athena Scientific.

Burges, C. J. C. (1996). Simplified support vector decision rules. *Proceedings of the Thirteenth International Conference on Machine Learning* (pp. 71–77).

Csató, L., & Opper, M. (2001). Sparse representation for Gaussian process models. *Advances in Neural Information Processing Systems 13* (pp. 444–450).

Dearden, R., Friedman, N., & Russell, S. J. (1998). Bayesian Q-Learning. *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (pp. 761–768).

Dietterich, T. G., & Wang, X. (2001). Batch value function approximation via support vectors. *Advances in Neural Information Processing Systems 14* (pp. 1491–1498).

Engel, Y., Mannor, S., & Meir, R. (2002). Sparse online greedy support vector regression. *13th European Conference on Machine Learning* (pp. 84–96).

Gibbs, M., & MacKay, D. (1997). Efficient implementation of Gaussian processes. Draft.

Munos, R. (2000). A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, *40*, 265–299.

Ormoneit, D., & Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning*, *49*, 161–178.

Rust, J. (1997). Using randomization to break the curse of dimensionality. *Econometrica*, *65*, 487–516.

Scharf, L. L. (1991). *Statistical signal processing.* Addison-Wesley.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels.* MIT Press.

Smola, A. J., & Bartlett, P. L. (2001). Sparse greedy Gaussian process regression. *Advances in Neural Information Processing Systems 13* (pp. 619–625).

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R., & Barto, A. (1998). *Reinforcement learning.* MIT Press.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Comm. ACM*, *38*, 58–68.

Tsitsiklis, J. N., & Van-Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Trans. on Automatic Control*, *42*, 674–690.

Williams, C. (1999). Prediction with Gaussian processes: from linear regression to linear prediction and beyond. *Learning in Graphical Models.* Cambridge, Massachusetts.

Williams, C., & Seeger, M. (2001). Using the Nyström method to speed up kernel machines. *Advances in Neural Information Processing Systems 13* (pp. 682–688).