Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)

Kernelizing LSPE(λ)

Tobias Jung
University of Mainz, Germany
E-mail: tjung@informatik.uni-mainz.de

Daniel Polani
University of Hertfordshire, UK
E-mail: d.polani@herts.ac.uk

***Abstract*— We propose the use of kernel-based methods as underlying function approximator in the least-squares based policy evaluation framework of LSPE(λ) and LSTD(λ). In particular we present the 'kernelization' of model-free LSPE(λ). The 'kernelization' is computationally made possible by using the subset of regressors approximation, which approximates the kernel using a vastly reduced number of basis functions. The core of our proposed solution is an efficient recursive implementation with automatic supervised selection of the relevant basis functions. The LSPE method is well-suited for optimistic policy iteration and can thus be used in the context of online reinforcement learning. We use the high-dimensional Octopus benchmark to demonstrate this.**

I. Introduction

Least squares based policy evaluation is ideally suited for the use with linear models and a very sample-efficient variant of reinforcement learning (RL). In this paper we propose a (non-parametric) kernel-based approach to approximate the value function. in least squares based policy evaluation. The rationale for doing this is that by representing the solution through the data and not by some basis functions chosen before the data becomes available, we can better adapt to the complexity of the unknown function we are trying to estimate. In particular, parameters (i.e. basis functions) are not 'wasted' on parts of the input space that are never visited. The hope is that thereby the exponential growth of parameters due to high-dimensional inputs is bypassed.

To solve the RL problem of optimal control we consider the framework of optimistic policy iteration and the least squares based policy evaluation method LSPE(λ). The LSPE method is formulated with linearly parametrized function approximation in mind and can be easily 'kernelized'. A straightforward application to LSTD(λ) is also possible, but not given here in detail.

We use the subset of regressors method to approximate the kernel using a much reduced subset of basis functions. To select this subset we employ sparse greedy *online* selection, similar to [7], [8], that adds a candidate basis function based on its distance to the span of the previously chosen ones. One improvement is that we consider a *supervised* criterion for the selection of the relevant basis functions that takes into account the reduction of the cost in the original learning task in addition to the reduction of the error incurred from approximating the kernel. Since the per-step complexity during training and prediction depends on the size of the subset, making sure that no unnecessary basis functions are selected ensures more efficient usage of otherwise scarce resources. This way learning in real-time becomes possible.

This paper is structured in three parts: the first part (Section II and Section III) gives a brief introduction on RL and describes the kernelization of LSPE. The second part (Section IV) describes our recursive implementation. The third part (Section V) contains the experiments. One final note: our work is in many respects similar to the GPTD approach from Engel et al. [8], [9], [10]. A longer discussion of the two approaches is deferred to the end of this paper.

II. Least Squares based Reinforcement Learning

Reinforcement learning (RL) is a simulation-based form of approximate dynamic programming, e.g. see [2], [26]. Consider a discrete-time dynamical system with (finite) states $\mathcal{S}$: at every time step t, when the system is in state s_t, a decision maker chooses a control-action a_t (selected from a finite set of admissible actions $\mathcal{A}$) which changes probabilistically the state of the system to s_{t+1}, with distribution $P(s_{t+1}|s_t, a_t)$. Every such transition yields an immediate reward $r_{t+1} = R(s_{t+1}|s_t, a_t)$. The ultimate goal of the decision-maker is to choose a course of actions such that the long-term performance, a measure of the cumulated sum of rewards, is maximized.

A. Model-free Q-value function and optimal control

For a fixed deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ we want to evaluate the state-action value function (Q-function) which is here the expected infinite-horizon discounted sum of rewards

$$Q^\pi(s,a) = E^\pi \left\{ \sum_{t\geq 0} \gamma^t r_{t+1} | s_0 = s, a_0 = a \right\} \quad \forall s, a$$

where $s_{t+1} \sim P(\cdot\,|s_t, \pi(s))$ and $r_{t+1} = R(s_{t+1}|s_t, \pi(s_t))$. Parameter $\gamma \in (0,1)$ denotes the discount factor.

Ultimately we are interested in optimal control, i.e. we seek an optimal policy $\pi^* = \mathrm{argmax}_\pi Q^\pi$. To accomplish that, policy iteration interleaves the two steps policy evaluation and policy improvement: First, compute Q^{π_k} for a fixed policy π_k. Then, once Q^{π_k} is known, derive an improved policy π_{k+1} by choosing in every state the action that achieves the best Q-value, i.e. $\forall s$, $\pi_{k+1}(s) = \mathrm{argmax}_a Q^{\pi_k}(s,a)$. Obtaining the best action is trivial if we employ the Q-notation, otherwise we would need both the transition probabilities and reward function (i.e. a 'model').

B. Policy evaluation

To compute the Q-function, one exploits the fact that Q^π obeys the fixed-point relation $Q^\pi(s,a) =$

$E\{r_{t+1} + \gamma Q^\pi(s', \pi(s'))|s_t = s, a_t = a, s_{t+1} = s'\}$. In principle, it is possible to calculate Q^π exactly by solving the corresponding linear system of equations, provided that the transition probabilities $P(s'|s,a)$ and rewards $R(s'|s,a)$ are known in advance. However, in many practical situations this is not the case. Instead, one employs simulation (i.e. an agent interacts with the environment) to generate a large number of observed transitions; the expectation value is then approximated from these samples. A second need for approximation arises from the number of states, which is often very large or infinite. Then, one can only operate with an approximation of the Q-function, e.g. a linear approximation $\tilde{Q}^\pi(s,a) = \boldsymbol{\varphi}(s,a)^\mathsf{T}\mathbf{w}$, where $\boldsymbol{\varphi}(s,a)$ is a feature vector and $\mathbf{w}$ the adjustable weight vector. An important example employing both approaches is the temporal difference algorithm TD(λ), initially proposed by Sutton [27].

C. Approximate policy evaluation with least squares methods

In what follows we will discuss two related algorithms for approximate policy evaluation, that share most of the advantages of TD(λ) but converge much faster, since they are based on solving a least squares problem in closed form, whereas TD(λ) is based on stochastic gradient descent. Both methods assume that an (infinitely) long[1] trajectory of states and rewards is generated using a simulation of the system. The trajectory starts from an initial state s_0 and consists of tuples $(s_0, a_0), (s_1, a_1), \ldots$ and rewards $r_1, r_2, \ldots$ where action a_i is chosen according to the current policy π and successor states and associated rewards are sampled from the underlying transition probabilities. From now on, to abbreviate these state-action tuples we will understand $\mathbf{x}_t$ as denoting $\mathbf{x}_t := (s_t, a_t)$. Furthermore, we assume that the Q-function is parameterized by

$$\tilde{Q}^\pi(\mathbf{x}) = \langle \boldsymbol{\varphi}(\mathbf{x}), \mathbf{w} \rangle = \boldsymbol{\varphi}(\mathbf{x})^\mathsf{T}\mathbf{w} \tag{1}$$

where $\boldsymbol{\varphi}(\mathbf{x})$ is a (possible infinite dimensional) feature vector (e.g. arising from the Mercer kernel map [23]). In the context of kernel-based learning (1) is often called the primal form.

1) The LSPE(λ) method: The λ-least squares policy evaluation method LSPE(λ) was proposed by [18], [3] and proceeds by making incremental changes to the weights $\mathbf{w}$. Assume that at time t (after having observed t transitions) we have a current weight vector $\mathbf{w}_t$ and observe a new transition from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$ with associated reward r_{t+1}. Then we compute the solution $\hat{\mathbf{w}}_{t+1}$ of the least squares problem

$$\hat{\mathbf{w}}_{t+1} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=0}^{t} \Big\{ \boldsymbol{\varphi}(\mathbf{x}_i)^\mathsf{T}\mathbf{w} - \boldsymbol{\varphi}^\mathsf{T}(\mathbf{x}_i)^\mathsf{T}\mathbf{w}_t - \sum_{k=i}^{t} (\lambda\gamma)^{k-i} d(\mathbf{x}_k, \mathbf{x}_{k+1}; \mathbf{w}_t) \Big\}^2 \tag{2}$$

[1]If we are dealing with an episodic learning task with designated terminal states, we can generate an infinite trajectory in the following way: once an episode ends, we set the discount factor γ to zero and make a zero-reward transition from the terminal state to the start state of the next (following) episode.

where

$$d(\mathbf{x}_k, \mathbf{x}_{k+1}; \mathbf{w}_t) := r_{k+1} + \gamma\boldsymbol{\varphi}(\mathbf{x}_{k+1})^\mathsf{T}\mathbf{w}_t - \boldsymbol{\varphi}(\mathbf{x}_k)^\mathsf{T}\mathbf{w}_t.$$

The new weight vector $\mathbf{w}_{t+1}$ is obtained by setting

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta(\hat{\mathbf{w}}_{t+1} - \mathbf{w}_t) \tag{3}$$

where $\mathbf{w}_0$ is the initial weight vector and $0 < \eta \leq 1$ is a step size.

2) The LSTD(λ) method: The related least squares temporal difference method LSTD(λ) proposed by [6] for $\lambda = 0$ and by [5] for general $\lambda \in [0,1]$ does not proceed by making incremental changes to the weight vector $\mathbf{w}_t$. Instead, at time t, the weight vector $\mathbf{w}_t$ is obtained by solving the fixed-point equation

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=0}^{t} \Big\{ \boldsymbol{\varphi}(\mathbf{x}_i)^\mathsf{T}\mathbf{w} - \boldsymbol{\varphi}(\mathbf{x}_i)^\mathsf{T}\hat{\mathbf{w}} - \sum_{k=i}^{t} (\lambda\gamma)^{k-i} d(\mathbf{x}_k, \mathbf{x}_{k+1}; \hat{\mathbf{w}}) \Big\}^2 \tag{4}$$

for the unique solution $\mathbf{w}_t$.

3) Comparing LSTD(λ) and LSPE(λ): For a linearly parameterized approximation both LSPE(λ) and LSTD(λ) converge to the same limit, which is also the limit to which TD(λ) converges (see [3]). Both methods rely on the solution of a least squares problem (either explicitly as is the case in LSPE or implicitly as is the case in LSTD) and can be efficiently implemented using recursive computations. Computational experiments in [4], [16] indicate that both approaches can perform much better than TD(λ).

Both methods differ as far their role in the approximate policy iteration framework is concerned. LSPE can take advantage of previous estimates of the weight vector and can hence be used in the context of optimistic policy iteration. For LSTD this is not possible; here a more rigid actor-critic approach is called for. In this paper we are interested in online learning with optimistic policy iteration; hence we will only deal with the 'kernelization' of the LSPE variant. Note however, that LSTD allows a very similar 'kernelization', see [14] for details.

III. LSPE WITH KERNELS

A. The primal form

Now we want express (2), (3) using matrices. Define:

$$\boldsymbol{\Phi} := \begin{bmatrix} \boldsymbol{\varphi}(\mathbf{x}_0)^\mathsf{T} \\ \vdots \\ \boldsymbol{\varphi}(\mathbf{x}_t)^\mathsf{T} \end{bmatrix}, \quad \bar{\boldsymbol{\Phi}} := \begin{bmatrix} \boldsymbol{\varphi}(\mathbf{x}_0)^\mathsf{T} - \gamma\boldsymbol{\varphi}(\mathbf{x}_1)^\mathsf{T} \\ \vdots \\ \boldsymbol{\varphi}(\mathbf{x}_t)^\mathsf{T} - \gamma\boldsymbol{\varphi}(\mathbf{x}_{t+1})^\mathsf{T} \end{bmatrix}$$

$$\mathbf{r} := \begin{bmatrix} r_1 \\ \vdots \\ r_{t+1} \end{bmatrix}, \quad \boldsymbol{\Lambda} := \begin{bmatrix} 1 & (\lambda\gamma)^1 & \cdots & (\lambda\gamma)^t \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & 1 & (\lambda\gamma)^1 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

Now we state (2) with an added weight regularizer to improve stability (as in ridge regression [13]); thus $\hat{\mathbf{w}}_{t+1}$ is the solution of

$$\min_{\mathbf{w}} \Big\{ \big\| \boldsymbol{\Phi}_{t+1}\mathbf{w} - \boldsymbol{\Phi}_{t+1}\mathbf{w}_t - \boldsymbol{\Lambda}_{t+1}\big(\mathbf{r}_{t+1} - \bar{\boldsymbol{\Phi}}_{t+1}\mathbf{w}_t\big)\big\|^2 + \sigma^2 \|\mathbf{w} - \mathbf{w}_t\|^2 \Big\}$$

where $\sigma^2 > 0$ is the regularization parameter. Computing the derivative wrt $\mathbf{w}$ and setting it to zero:

$$0 = \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\hat{\mathbf{w}}_{t+1} - \boldsymbol{\Phi}^\mathsf{T}\big(\boldsymbol{\Phi}\mathbf{w}_t + \boldsymbol{\Lambda}_{t+1}(\mathbf{r}_{t+1} - \bar{\boldsymbol{\Phi}}\mathbf{w}_t)\big) + \sigma^2(\hat{\mathbf{w}}_{t+1} - \mathbf{w}_t)$$

thus

$$(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\mathbf{I})\hat{\mathbf{w}}_{t+1} = \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi}\mathbf{w}_t + \boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Lambda}_{t+1}(\mathbf{r}_{t+1} - \bar{\boldsymbol{\Phi}}\mathbf{w}_t) + \sigma^2\mathbf{w}_t$$

and so

$$\hat{\mathbf{w}}_{t+1} = \mathbf{w}_t + (\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\mathbf{I})^{-1}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Lambda}_{t+1}(\mathbf{r}_{t+1} - \bar{\boldsymbol{\Phi}}\mathbf{w}_t) \quad (5)$$

The problem with (5) is that everything that is done depends on the number of features, which may be high or infinite (e.g. for Gaussian kernels). Proceeding similarly as in the 'kernelization' of ridge regression, we can put (5) in dual form.

B. The dual form

Applying the known matrix identity (e.g. see [12])

$$(\mathbf{P}^{-1} + \mathbf{B}^\mathsf{T}\mathbf{R}^{-1}\mathbf{B})^{-1}\mathbf{B}^\mathsf{T}\mathbf{R}^{-1} = \mathbf{P}\mathbf{B}^\mathsf{T}(\mathbf{B}\mathbf{P}\mathbf{B}^\mathsf{T} + \mathbf{R})^{-1}$$

we obtain for the inverse matrix in (5)

$$(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\Phi} + \sigma^2\mathbf{I})^{-1}\boldsymbol{\Phi}^\mathsf{T} = \boldsymbol{\Phi}^\mathsf{T}(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\mathbf{I})^{-1}$$

and thus (5) may be written as

$$\hat{\mathbf{w}}_{t+1} = \mathbf{w}_t + \boldsymbol{\Phi}^\mathsf{T}(\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\mathbf{I})^{-1}\boldsymbol{\Lambda}(\mathbf{r} - \bar{\boldsymbol{\Phi}}\mathbf{w}_t).$$

Obviously solutions $\hat{\mathbf{w}}$ of (5) lie in the column space of $\boldsymbol{\Phi}^\mathsf{T}$. Thus we may express all primal variables $\mathbf{w}$ by dual variables $\boldsymbol{\alpha} = \big(\alpha^{(1)}, \ldots, \alpha^{(m)}\big)^\mathsf{T}$, i.e. $\mathbf{w} = \sum_{i=0}^{t} \alpha^{(i)}\boldsymbol{\varphi}(\mathbf{x}_i)$. This way we can turn a problem depending on the number of features into a problem depending on the number of the data. Define $\mathbf{K} := \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$, the matrix of inner products $[\mathbf{K}]_{ij} = \langle\boldsymbol{\varphi}(\mathbf{x}_i), \boldsymbol{\varphi}(\mathbf{x}_j)\rangle =: k(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{H} := \bar{\boldsymbol{\Phi}}\boldsymbol{\Phi}^\mathsf{T}$ with $[\mathbf{H}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) - \gamma k(\mathbf{x}_{i+1}, \mathbf{x}_j)$. Then we can write (5) using dual variables

$$\hat{\boldsymbol{\alpha}}_{t+1} = \boldsymbol{\alpha}_t + (\mathbf{K} + \sigma^2\mathbf{I})^{-1}\boldsymbol{\Lambda}(\mathbf{r} - \mathbf{H}\boldsymbol{\alpha}_t)$$

and thus the dual LSPE update (3)

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \eta(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\boldsymbol{\Lambda}(\mathbf{r} - \mathbf{H}\boldsymbol{\alpha}_t) \quad (6)$$

Predictions (1) in test points $\mathbf{x}^*$ can also be made just using the dual variables

$$\tilde{Q}^\pi(\mathbf{x}^*) = \langle\boldsymbol{\varphi}(\mathbf{x}^*), \mathbf{w}_{t+1}\rangle = \langle\boldsymbol{\varphi}(\mathbf{x}^*), \sum_{i=0}^{t} \alpha_{t+1}^{(i)}\boldsymbol{\varphi}(\mathbf{x}_i)\rangle = \sum \alpha_{t+1}^{(i)} k(\mathbf{x}^*, \mathbf{x}_i) \quad (7)$$

Kernelizing LSPE is equivalent to using kernel ridge regression in the underlying least squares problem. Also, kernel ridge regression is equivalent to the mean of posterior of Gaussian process regression (GPR), see [21]. If we assume the standard error model for regression that the targets are noisy observations with independent normal noise distribution, then we can exploit this connection to the probabilistic framework to obtain in addition to predictions (7) an expression for the predictive variance:

$$\mathrm{var}(Q^\pi(\mathbf{x}^*)) = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^\mathsf{T}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}^*) \quad (8)$$

where $\mathbf{k}(\mathbf{x}^*) = \big(k(\mathbf{x}^*, \mathbf{x}_0), \ldots, k(\mathbf{x}^*, \mathbf{x}_t)\big)^\mathsf{T}$, see [21] for details. In our application to online reinforcement learning, we will use, as suggested in [9], the predictive variance (8) to guide the choice of actions during exploration.

C. The subset of regressors approximation

Since LSPE is an iterative, online method, solving the full $t \times t$ problem in (6) every time a new transition is observed is clearly computationally infeasible. Instead we need to consider means of approximation. In the subset of regressors (SR) approach [19], [17], [25], [23] one chooses a subset $\{\tilde{\mathbf{x}}_i\}_{i=1}^m$ of the data, with $m \ll t$ and approximates for all $\mathbf{x}$ the feature representation $\boldsymbol{\varphi}(\mathbf{x})$ by

$$\boldsymbol{\varphi}(\mathbf{x}) \approx \sum_{i=1}^{m} a_i\boldsymbol{\varphi}(\tilde{\mathbf{x}}_i). \quad (9)$$

Coefficient vector $\mathbf{a}$ in (9) is found by minimizing the length of the residual

$$\delta(\mathbf{x}) := \min_{\mathbf{a}\in\mathbb{R}^m} \left\| \boldsymbol{\varphi}(\mathbf{x}) - \sum_{i=1}^{m} a_i\boldsymbol{\varphi}(\tilde{\mathbf{x}}_i) \right\|^2. \quad (10)$$

Writing (10) with inner products, computing the derivative w.r.t. $\mathbf{a}$ and equating it with zero gives

$$\mathbf{a} = \mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}) \quad (11)$$

where matrix $[\mathbf{K}_{mm}]_{ij} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ and vector $\mathbf{k}_m(\mathbf{x}) = \big(k(\mathbf{x}, \tilde{\mathbf{x}}_1), \ldots, k(\mathbf{x}, \tilde{\mathbf{x}}_m)\big)^\mathsf{T}$. Applying the approximation (9) to arbitrary $\mathbf{x}, \mathbf{x}'$ we obtain

$$k(\mathbf{x}, \mathbf{x}') \approx \mathbf{k}_m(\mathbf{x})^\mathsf{T}\mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}') \quad (12)$$

which is exact if either $\mathbf{x}$ or $\mathbf{x}'$ belongs to the subset. Likewise we see that the full $(t+1) \times (t+1)$ kernel matrix $\mathbf{K}$ is approximated by $\tilde{\mathbf{K}} = \mathbf{K}_{t+1,m}\mathbf{K}_{mm}^{-1}\mathbf{K}_{t+1,m}^\mathsf{T}$, where $\mathbf{K}_{t+1,m}$ is the $(t+1) \times m$ submatrix $[\mathbf{K}_{t+1,m}]_{ij} = k(\mathbf{x}_i, \tilde{\mathbf{x}}_j)$.

Replacing in (6), (7) every occurrence of $k(\cdot,\cdot)$ by the approximation (12) we obtain (skipping the derivation) in place of (6)

$$\boldsymbol{\alpha}_{t+1,m} = \boldsymbol{\alpha}_{t,m} + \eta(\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{K}_{t+1,m} + \sigma^2\mathbf{K}_{mm})^{-1} \mathbf{K}_{t+1,m}^\mathsf{T}\boldsymbol{\Lambda}_{t+1}(\mathbf{r}_{t+1} - \mathbf{H}_{t,m}\boldsymbol{\alpha}_{t,m}) \quad (13)$$

and in place of (7)

$$\tilde{Q}^\pi(\mathbf{x}^*) = \sum_{i=1}^{m} \alpha_{t+1,m}^{(i)} k(\mathbf{x}, \tilde{\mathbf{x}}_i). \quad (14)$$

Note that here $\boldsymbol{\alpha}_{tm}$ denotes a $m \times 1$ vector. Overall the effect of SR is to reduce the number of dual variables $\boldsymbol{\alpha}$ from $t+1$ to m. Instead of solving a $\mathcal{O}(t^2)$ problem in (6) computional costs per step are reduced to $\mathcal{O}(m^2)$ in (13) and to $\mathcal{O}(m)$ for predicting in (14). Moreover, since LSPE is an incremental, online algorithm, the per-step cost does no longer depend on the number of previously observed transitions t, but on the fixed number m.

Finally, when applying SR, the predictive variance (8) is computed by (see [21])

$$\mathrm{var}(\tilde{Q}^\pi(\mathbf{x}^*)) = \\ \sigma^2 \mathbf{k}_m(\mathbf{x}^*)^\mathsf{T} \left(\mathbf{K}_{t+1,m}^\mathsf{T} \mathbf{K}_{t+1,m} + \sigma^2 \mathbf{K}_{mm}\right)^{-1} \mathbf{k}_m(\mathbf{x}^*). \quad (15)$$

D. Online selection of the subset

In LSPE we assume that the data becomes available sequentially at $t = 1, 2, \ldots$, so that we cannot select the subset $\{\tilde{\mathbf{x}}_i\}_{i=1}^m$ in advance (as for example is done with random selection in [28], selection by incomplete Cholesky in [11], or greedy forward selection using matching pursuit techniques in [24]). Working in the context of GPR, [7] and later [8] have proposed sparse greedy online approximation: start from an empty subset (termed the dictionary or set of basis vectors $\mathcal{BV}$) and build it up incrementally by examining at every time step if the new example $\mathbf{x}_{t+1}$ needs to be included in the current set $\mathcal{BV}$ or if it can be processed without augmenting $\mathcal{BV}$.

The criterion they employ to make that decision is an unsupervised one: at every time step t compute for the new data point $\mathbf{x}_{t+1}$ the error from (10)

$$\delta_{t+1} := k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}_m(\mathbf{x}_{t+1})^\mathsf{T} \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}_{t+1}) \quad (16)$$

incurred from approximating the new data point using the current $\mathcal{BV}$. If δ_{t+1} exceeds a given threshold `TOL1` then it is considered as sufficiently different and added to the dictionary $\mathcal{BV}$.

Note that only the *current* number of elements in $\mathcal{BV}$ at any given time t is considered, the contribution from basis functions that will be added at a later time is ignored (i.e. $\mathbf{a}$ is padded with zeros). Here it might be helpful to visualize the $(t+1) \times m$ data matrix $\mathbf{K}_{t+1,m}$ once $\mathcal{BV}$ is augmented. Adding the new element $\mathbf{x}_{t+1}$ to $\mathcal{BV}$ means adding a new basis function (centered on $\mathbf{x}_{t+1}$) to the model and consequently adding a new associated column $\mathbf{q} = \big(k(\mathbf{x}_0, \mathbf{x}_{t+1}), \ldots, k(\mathbf{x}_t, \mathbf{x}_{t+1})\big)^\mathsf{T}$ to $\mathbf{K}_{t+1,m}$. With sparse online approximation all t past entries in $\mathbf{q}$ are given by $k(\mathbf{x}_i, \mathbf{x}_{t+1}) \approx \mathbf{k}_m(\mathbf{x}_i)^\mathsf{T} \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}_{t+1})$, $i = 0 \ldots, t$, which is exact for the m basis-elements and an approximation for the remaining $t - m$ non-basis elements. Hence, going from m to $m+1$ basis functions, we have that

$$\mathbf{K}_{t+1,m+1} = \begin{bmatrix} \mathbf{K}_{t+1,m} & \mathbf{q} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{t,m} & \mathbf{K}_{t,m}\mathbf{a}_{t+1} \\ \mathbf{k}_m(\mathbf{x}_{t+1})^\mathsf{T} & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix}$$

where $\mathbf{a}_{t+1} := \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}_{t+1})$. The overall effect is, that now we do not need to access the full data set any longer. All costly $\mathcal{O}(tm)$ operations that normally arise from adding a new column (i.e. adding a new basis function, computing the reduction of error during greedy forward selection of basis functions, computing predictive variance with augmentation as in [20]) now become a more affordable $\mathcal{O}(m^2)$.

This is exploited in [15]; here a simple modification of this selection procedure is presented, where in addition to the unsupervised criterion from (16) the contribution to the reduction of the error (i.e. the objective function one is trying to minimize) is taken into account. Here we will also employ this supervised criterion (see Section IV-C.2). Since the per-step complexity during training and then later during prediction critically depends on the size m of the subset $\mathcal{BV}$, making sure that no unnecessary basis functions are selected ensures more efficient usage of otherwise scarce resources and makes online learning in real-time possible.

IV. Implementation

Now we have all the pieces together to formulate an efficient online implementation for kernel-based LSPE(λ) with automatic basis selection. An outline of the implementation is sketched in Fig. 1; the following sections will explain each of the steps in more detail.

A. Recursive updates

Let $\mathbf{x}_{t+1} = (s_{t+1}, \pi(s_{t+1}))$ be the currently observed state-action pair and r_{t+1} the associated reward. Assume that from the past transitions the m examples $\{\tilde{\mathbf{x}}_i\}_{i=1}^m$ were selected into the dictionary $\mathcal{BV}$. In the following we will use a double index (also for vectors) to indicate the dependence on the number of examples t and the number of basis functions m. Every time a new transition $(\mathbf{x}_{t+1}, r_{t+1})$ is observed we will perform one or both of the following update operations:

1) *Normal step*: Process $(\mathbf{x}_{t+1}, r_{t+1})$ using the current fixed set of basis functions $\mathcal{BV}$.
2) *Growing step*: If the new example is sufficiently different from the previous examples in $\mathcal{BV}$ (i.e. the reconstruction error in (16) exceeds a given threshold) and strongly contributes to the solution of the problem (i.e. the decrease of the loss when adding the new basis function is greater than a given threshold) then the current example is added to $\mathcal{BV}$ and the number of basis functions in the model is increased by one.

Consider the dual minimization problem for LSPE when using SR (the result of the skipped computation in Section III-C):

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^m} \; J_{t+1,m}(\boldsymbol{\alpha}) := \\ \|\mathbf{K}_{t+1,m}\boldsymbol{\alpha} - \mathbf{K}_{t+1,m}\boldsymbol{\alpha}_{tm} + \boldsymbol{\Lambda}_{t+1}(\mathbf{r}_{t+1} + \mathbf{H}_{t,m}\boldsymbol{\alpha}_{tm})\|^2 \\ + \sigma^2 (\boldsymbol{\alpha} - \boldsymbol{\alpha}_{tm})^\mathsf{T} \mathbf{K}_{mm} (\boldsymbol{\alpha} - \boldsymbol{\alpha}_{tm}) \quad (17)$$

Algorithm Online policy evaluation with kernelized LSPE

Input: policy π

Output: Q-function: linearly parameterized $\tilde{Q}^\pi(\mathbf{x}) = \mathbf{k}_m(\mathbf{x})^\mathsf{T}\boldsymbol{\alpha}_{t,m}$ (m basis functions)

Generate first state s_0. Choose $a_0 = \pi(s_0)$.

For $t = 0, 1, \ldots$

Execute action a_t (simulate a transition).
Observe next state s_{t+1} and reward r_{t+1}.
Choose action $a_{t+1} = \pi(s_{t+1})$.

1. Check: If $\mathbf{x}_{t+1} := (s_{t+1}, a_{t+1})$ should be added to the subset of basis functions

2. Case 1: Update weight vector $\boldsymbol{\alpha}_{t,m}$ without augmenting the basis
Perform the 'normal step' in Section IV-B

3. Case 2: Update weight vector $\boldsymbol{\alpha}_{t,m}$ and augment the basis
(Prepare the 'growing step' by caching certain terms)
Perform the 'normal step' in Section IV-B
Perform the 'growing step' in Section IV-C

$s_{t+1} \to s_t$, $a_{t+1} \to a_t$

Fig. 1. Online policy evalution with Kernel-LSPE(λ) at $\mathcal{O}(m^2)$ operations per step, where m is the number of current basis functions

To write the expressions more compactly, we introduce the shorthands

$$\begin{aligned}\mathbf{B}_{t+1,m} &:= (\mathbf{K}_{t+1,m}^\mathsf{T}\mathbf{K}_{t+1,m} + \sigma^2\mathbf{K}_{mm})\\ \mathbf{Z}_{t+1,m}^\mathsf{T} &:= \mathbf{K}_{t+1,m}^\mathsf{T}\boldsymbol{\Lambda}_{t+1}\\ \mathbf{b}_{t+1,m} &:= \mathbf{Z}_{t+1,m}^\mathsf{T}\mathbf{r}_{t+1}\\ \mathbf{A}_{t+1,m} &:= \mathbf{Z}_{t+1,m}^\mathsf{T}\mathbf{H}_{t+1,m}\end{aligned}$$

Thus the LSPE update (restated from (13)) when using a fixed basis is

$$\boldsymbol{\alpha}_{t+1,m} = \boldsymbol{\alpha}_{t,m} + \eta\mathbf{B}_{t+1,m}^{-1}(\mathbf{b}_{t+1,m} - \mathbf{A}_{t+1,m}\boldsymbol{\alpha}_{t,m}) \quad (18)$$

and when adding a new basis function

$$\boldsymbol{\alpha}_{t+1,m+1} = \begin{bmatrix}\boldsymbol{\alpha}_{t,m}\\ 0\end{bmatrix} + \eta\mathbf{B}_{t+1,m+1}^{-1}\left(\mathbf{b}_{t+1,m+1} - \mathbf{A}_{t+1,m+1}\begin{bmatrix}\boldsymbol{\alpha}_{t,m}\\ 0\end{bmatrix}\right) \quad (19)$$

Assuming that $\{\boldsymbol{\alpha}_{t,m}, \mathbf{B}_{t,m}^{-1}, \mathbf{A}_{t,m}, \mathbf{b}_{t,m}\}$ are known from the previous step, we need to derive update equations for the two cases:

$$\begin{aligned}\{\boldsymbol{\alpha}_{t,m}, \mathbf{B}_{t,m}^{-1}\} &\to \{\boldsymbol{\alpha}_{t+1,m}, \mathbf{B}_{t+1,m}^{-1}\}\\ \{\boldsymbol{\alpha}_{t+1,m}, \mathbf{B}_{t+1,m}^{-1}\} &\to \{\boldsymbol{\alpha}_{t+1,m+1}, \mathbf{B}_{t+1,m+1}^{-1}\}\end{aligned}$$

Our update operations work along the lines of recursive least squares (RLS), i.e. propagate forward the inverse[2] of the $m \times m$ matrix $\mathbf{B}_{t+1,m}$.

To perform these updates we use the two well-known matrix identities for recursively computing the inverse of a matrix: (for matrices $\mathbf{G}, \mathbf{g}$ with compatible dimensions)

$$\text{if } \mathbf{G}_{t+1} = \mathbf{G}_t + \mathbf{g}\mathbf{g}^\mathsf{T} \text{ then}$$
$$\mathbf{G}_{t+1}^{-1} = \mathbf{G}_t^{-1} - \frac{\mathbf{G}_t^{-1}\mathbf{g}\mathbf{g}^\mathsf{T}\mathbf{G}_t^{-1}}{1 + \mathbf{g}^\mathsf{T}\mathbf{G}_t^{-1}\mathbf{g}} \quad (20)$$

[2]A better alternative (from the standpoint of numerical implementation) would be to not propagate forward the inverse, but instead to work with the cholesky factor. For this paper we chose the first method in the first place because it gives consistent update formulas. For details on the second way, see e.g. [22].

which is used when adding a row to the data matrix. Likewise,

$$\text{if } \mathbf{G}_{t+1} = \begin{bmatrix}\mathbf{G}_t & \mathbf{g}\\ \mathbf{g}^\mathsf{T} & g^*\end{bmatrix} \text{ then}$$
$$\mathbf{G}_{t+1}^{-1} = \begin{bmatrix}\mathbf{G}_t^{-1} & \mathbf{0}\\ \mathbf{0} & 0\end{bmatrix} + \frac{1}{\Delta_b}\begin{bmatrix}-\mathbf{G}_t^{-1}\mathbf{g}\\ 1\end{bmatrix}\begin{bmatrix}-\mathbf{G}_t^{-1}\mathbf{g}\\ 1\end{bmatrix}^\mathsf{T} \quad (21)$$

with $\Delta_b = g^* - \mathbf{g}^\mathsf{T}\mathbf{G}_t^{-1}\mathbf{g}$. This second update is used when adding a column to the data matrix.

B. Normal step: $\{t, m\} \to \{t+1, m\}$

With $\mathbf{k}_{t+1} := \mathbf{k}_m(\mathbf{x}_{t+1})$ and $\mathbf{h}_{t+1} := \mathbf{k}_t - \gamma\mathbf{k}_{t+1}$ one gets

$$\mathbf{K}_{t+1,m} = \begin{bmatrix}\mathbf{K}_{t,m}\\ \mathbf{k}_{t+1}^\mathsf{T}\end{bmatrix}, \mathbf{H}_{t+1,m} = \begin{bmatrix}\mathbf{H}_{t,m}\\ \mathbf{h}_{t+1}^\mathsf{T}\end{bmatrix}, \mathbf{r}_{t+1} = \begin{bmatrix}\mathbf{r}_t\\ r_{t+1}\end{bmatrix}.$$

Thus $\mathbf{B}_{t+1,m} = \mathbf{B}_{t,m} + \mathbf{k}_{t+1}\mathbf{k}_{t+1}^\mathsf{T}$ and we obtain from (20)

$$\mathbf{B}_{t+1,m}^{-1} = \mathbf{B}_{t,m}^{-1} - \frac{\mathbf{B}_{t,m}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^\mathsf{T}\mathbf{B}_{t,m}^{-1}}{\Delta} \quad (22)$$

where $\Delta = 1 + \mathbf{k}_{t+1}^\mathsf{T}\mathbf{B}_{t,m}^{-1}\mathbf{k}_{t+1}$. Skipping some derivations we compute the vector

$$\mathbf{z}_{t+1,m} = (\lambda\gamma)\mathbf{z}_{t,m} + \mathbf{k}_{t+1}$$

used to update $\mathbf{A}_{t,m}, \mathbf{b}_{t,m}$ as follows

$$\mathbf{A}_{t+1,m} = \mathbf{A}_{t,m} + \mathbf{z}_{t+1,m}\mathbf{h}_{t+1}^\mathsf{T} \quad (23)$$
$$\mathbf{b}_{t+1,m} = \mathbf{b}_{t,m} + \mathbf{z}_{t+1,m}r_{t+1} \quad (24)$$

The set of basis functions $\mathcal{BV}$ is not altered during this step. The new weight vector $\boldsymbol{\alpha}_{t+1,m}$ can now be obtained from (18). Operation count is $\mathcal{O}(m^2)$.

C. Growing step:$\{t+1, m\} \to \{t+1, m+1\}$

1) How to add a $\mathcal{BV}$: When adding an additional basis function (centered on $\mathbf{x}_{t+1}$) to the model we augment the set $\mathcal{BV}$ with $\tilde{\mathbf{x}}_{m+1}$ (which is the same as $\mathbf{x}_{t+1}$). Adding a new basis function means appending a new column/row to the relevant matrices. Due to the lack of space we cannot describe

the derivations in more detail; we just give the results: invoking (21) we obtain

$$\mathbf{B}_{t+1,m+1}^{-1} = \begin{bmatrix} \mathbf{B}_{t+1,m}^{-1} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} + \frac{1}{\Delta_b} \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix}^T \quad (25)$$

where

$$\begin{aligned} \mathbf{w}_b &= \mathbf{a}_{t+1} + \frac{\delta}{\Delta} \mathbf{B}_{t,m}^{-1} \mathbf{k}_{t+1} \\ \Delta_b &= \frac{\delta^2}{\Delta} + \sigma^2 \delta \end{aligned}$$

with $\mathbf{a}_{t+1} = \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}_{t+1})$ from (11), $\delta := k_t^* - \mathbf{k}_t^\mathsf{T} \mathbf{a}_{t+1}$, $k_t^* := k(\mathbf{x}_t, \mathbf{x}_{t+1})$. Furthermore, compute $k_{t+1}^* := k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1})$, $h^* := k_t^* - \gamma k_{t+1}^*$ and $z^* := (\lambda\gamma)\mathbf{z}_{t,m}^\mathsf{T} \mathbf{a}_{t+1} + k_t^*$. Then we obtain

$$\mathbf{z}_{t+1,m+1}^\mathsf{T} = \begin{bmatrix} \mathbf{z}_{t+1,m}^\mathsf{T} & z^* \end{bmatrix}^\mathsf{T}$$

and thus with $\mathbf{u} := \mathbf{a}_{t+1}^\mathsf{T} \mathbf{A}_{t,m}$ and $\mathbf{v} := \mathbf{A}_{t,m} \mathbf{a}_{t+1}$

$$\begin{aligned} \mathbf{A}_{t+1,m+1} &= \begin{bmatrix} \mathbf{A}_{t+1,m} & \mathbf{v} + \mathbf{z}_{t+1,m} h^* \\ \mathbf{u} + z^* \mathbf{h}_{t+1} & \mathbf{u}\mathbf{a}_{t+1} + z^* h^* \end{bmatrix} \\ \mathbf{b}_{t+1,m+1} &= \begin{bmatrix} \mathbf{b}_{t+1,m} \\ \mathbf{a}_{t+1}^\mathsf{T} \mathbf{b}_{t,m} + z^* r_{t+1} \end{bmatrix} \end{aligned}$$

Caching and reusing those terms already computed in the preceding step (see Sect. IV-B) we can perform these updates in $\mathcal{O}(m^2)$ operations. Finally, every time we add an example to the $\mathcal{BV}$ set we must also update the inverse kernel matrix $\mathbf{K}_{mm}^{-1}$ needed during the computation of $\mathbf{a}_{t+1}$. This can be done using the formula for partitioned matrix inverses (21).

2) When to add a $\mathcal{BV}$: To decide whether or not the current example $\mathbf{x}_{t+1}$ should be added to the $\mathcal{BV}$ set, we employ a supervised two-part criterion similar to [15]. The first part measures the 'novelty' of the current data point: compute as in [7] the squared norm of the residual from projecting the feature $\boldsymbol{\varphi}(\mathbf{x}_{t+1})$ onto the span of the current $\mathcal{BV}$ set, i.e. we compute δ_{t+1} from (16). If $\delta_{t+1} < \texttt{TOL1}$ for a given threshold `TOL1`, then $\mathbf{x}_{t+1}$ is well represented by the given $\mathcal{BV}$ set and its inclusion would not contribute much to reduce the error from approximating the kernel by the reduced set. On the other hand, if $\delta_{t+1} > \texttt{TOL1}$ then $\mathbf{x}_{t+1}$ is not well represented by the current $\mathcal{BV}$ set and leaving it behind could incur a large error in the approximation of the kernel.

However, using as sole criterion the reduction of the error incurred from approximating the kernel is probably too wasteful of resources, since examples could get selected into the subset that are unrelated to the original task [1]. We want to be more restrictive, particularly because the computational complexity per step scales with the square of basis functions in $\mathcal{BV}$ (so that the size of $\mathcal{BV}$ will soon become the limiting factor).

Aside from novelty, we thus consider as second part of the selection criterion the 'usefulness' of a basis function candidate. Usefulness is taken to be its contribution to the reduction of the regularized costs in the dual minimization problem (17). One can show that for a given $\mathbf{x}_{t+1}$ this reduction is equal to $\Delta_b^{-1}(c - \mathbf{w}_b^\mathsf{T}\mathbf{d})^2$ where $c = \mathbf{a}_{t+1}^\mathsf{T}(\mathbf{b}_{t,m} - \mathbf{A}_{t,m}\boldsymbol{\alpha}_{t,m}) + z^*(r_{t+1} - \mathbf{h}_{t+1}^\mathsf{T}\boldsymbol{\alpha}_{t,m})$ and $\mathbf{d} = \mathbf{b}_{t+1,m} - \mathbf{A}_{t+1,m}\boldsymbol{\alpha}_{t,m}$. Again this reduction can be very cheaply obtained in $\mathcal{O}(m)$ time (thus again independently from the number of total examples).

D. Action selection with augmented predictive variance

Once a dual weight vector $\boldsymbol{\alpha}_{t+1,m}$ is known we can choose a control action a^* for an arbitrary state s^* by taking the action a^* that achieves the maximum value:

$$a^* = \operatorname*{argmax}_a \mathbf{k}_m(s^*, a)^\mathsf{T} \boldsymbol{\alpha}_{t+1,m}.$$

Sometimes however, instead of choosing the best (greedy) action, it is recommended to try out an alternative (non-greedy) action to ensure sufficient exploration. Here we will employ the ε-greedy selection scheme; we choose a random action with a small probability ε, otherwise we pick the greedy action with probability $1 - \varepsilon$. Taking a random action usually means to select one with equal probability. Here we will exploit the predictive variance of (15) to assign probabilities according to the 'novelty' of the state-action pair $\mathbf{x}^* = (s^*, a)$.

Note, that directly computing the predictive variance for SR approximation is not helpful, since the resulting quantity tends to zero[3] if we stray far from the examples saved in the subset $\mathcal{BV}$. This is exactly the opposite of what we really want; the predictive variance should be high for novel test points. As a remedy [20] suggest to use an augmented predictive variance, which adds during every prediction a new basis function centered on the test point. Here again we can benefit from sparse online approximation from Section III-D to carry out what would normally be a $\mathcal{O}(tm)$ operation in only $\mathcal{O}(m^2)$ time. Due to the lack of space we omit the corresponding formulas.

V. Experiments with the octopus arm

The experimental work we carried out for this article uses the publicly available octopus arm benchmark[4]. Learning to control an octopus arm is a high-dimensional control task with both continuous state space (the arm is discretized into 8 compartments which results in 66 dimensions) and continuous actions (the action space has 22 dimensions) and was initially proposed and formulated as a reinforcement learning problem by Engel et al. in [10]. Instead of dealing with continuous actions we are using, as in [10], 7 meaningful predefined activation patterns that each translate into a 22-dimensional action.

A. The reaching task

Figure 2 shows a picture of the octopus problem. The objective for the octopus arm is to reach the goal, which is just a point in the 2D plane. Reaching is successful as soon as

[3] For distance-based kernels, like for example the Gaussian RBF, the kernel goes to zero when the distance to a fixed center goes to infinity. Thus $\mathbf{k}_m(\mathbf{x}^*)$ will also be nearly zero.

[4] from the ICML06 RL benchmarking page: `http://www.cs.mcgill.ca/dprecup/workshops/ICML06/octopus.html`

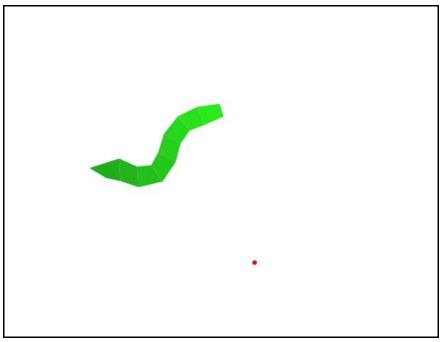

Fig. 2. Illustrating the *octopus* task. The octopus arm must wiggle and stretch considerably to touch the red goal.

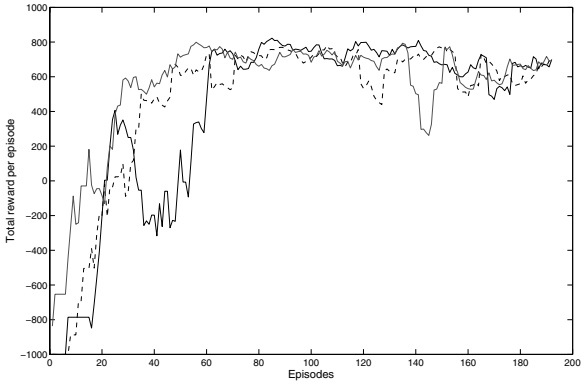

Fig. 3. Online performance with Kernel-LSPE during the first 200 episodes. The results are smoothed and shown for three different runs. A negative total reward means that the agent was not able to reach the goal in 1000 time steps (the maximum time allowed before an episode is truncated). A positive total reward means that the agent was able to reach the goal.

any part of the arm touches this point. For our experiments, we used the 'hardTask' goal configuration, which places the target far away such that the arm needs to perform a rather sophisticated sequence of control actions in order to succeed. The reaching task is an episodic task. It starts from a random position of the arm (these initial states are read in from a file and thus are the same for different runs) and proceeds until the goal is reached. Every step incurs a negative immediate reward of -1, successfully reaching the goal is awarded with +1000 reward. Note that the reward itself does not carry helpful information about the distance to the goal. An episode is truncated if the agent fails to reach the goal after 1000 steps.

B. Setup of LSPE and hyper-parameters

We use online reinforcement learning and pair our kernelized LSPE(λ) with optimistic policy iteration. Thus the underlying policy is continually modified to reflect the changing estimates of Q. To select actions we employ ε-greedy action selection (ε=0.05). Whenever chance indicates that an exploratory action should be chosen, we pick the one with the highest associated predictive variance (see Section IV-D).

Since the kernel is defined for state-action tuples, we employ a product kernel $k([s,a],[s',a']) = k_S(s,s')k_A(a,a')$ as suggested by Engel et al. in [9]. The action kernel $k_A(a,a')$ is taken to be the Kronecker delta, since the actions are discrete and disparate objects. As state kernel $k_S(s,s')$ we chose the Gaussian RBF with length-scale $h = 200$. The other parameters were set to: regularization $\sigma^2 = 0.1$, discount factor for RL $\gamma = 0.999$, $\lambda = 0.6$, and LSPE step size $\eta = 0.5$. The novelty parameter for basis selection was set to $\texttt{TOL1}= 0.1$. For the usefulness part we actually tried out different values to examine the effect supervised basis selection has; we started with $\texttt{TOL2}= 0$ corresponding to the unsupervised case and then examined increasingly higher thresholds.

C. Results

Figure 3 shows the learning curves obtained during the first 200 episodes for independent runs made using the same choice of hyper-parameters. The curves (smoothed by the rolling mean) plot the cumulated reward attained in one episode against the number of episodes the agent is learning. A total reward of -1000 means that the agent failed to reach the goal in 1000 time steps. A positive total reward means that the agent was able to reach the goal in that episode; here a higher number indicates that the agent completed the task more quickly. The plots show that kernel-LSPE is able to eventually solve the problem with a success rate of 100%. What is rather surprising is that this happens within very few episodes (an average of 20-50 episodes was needed before the agent completed the first successful trial).

We would also like to mention the effectiveness of our proposed supervised basis function selection. Moving TOL2 gradually away from zero (which corresponds to the unsupervised case) to larger values selects an increasingly fewer number of basis functions into the dictionary $\mathcal{BV}$; the unsupervised case results in $\sim$ 2000 basis functions, whereas a supervised selection with $\texttt{TOL2}$=0.01 only requires $\sim$ 1200 basis functions. The level of performance is not adversely affected by this reduction.

VI. Discussion and Related Work

We have presented a kernel-based approach for model-free, least squares based policy evaluation in RL using 'kernelized' LSPE (note that the related LSTD algorithm can be 'kernelized' in a very similar manner). One key point is the automatic construction of relevant features used to represent the approximated value function. We presented an efficient supervised basis selection mechanism, which selects a subset of relevant basis functions directly from the data stream.

Kernelized LSPE(λ) is particularly useful in the context of optimistic policy iteration and allows us to apply online RL to solve high-dimensional control tasks. We prove the effectiveness of our approach using the recent Octopus benchmark. The overall results indicate that online learning in RL using kernels is in practice very well possible and recommendable.

One property that makes the kernel-based approach particularly attractive is that it only requires the setting of some fairly general parameters that do not depend on the specific control problem one wants to solve. On the other hand, using traditional (parametric) function approximation in high dimensions, e.g. a fixed basis function network, requires considerable manual effort on part of the programmer to carefully devise problem-specific features and manually choose suitable basis functions.

Engel et al. initially advocated using kernel-based methods in RL and proposed the related GPTD algorithm [8], [9]. Our method using kernel ridge regression develops this idea further. Both methods have in common the online selection of relevant basis functions based on [7]. As opposed to the unsupervised selection in GPTD we use a supervised criterion to further reduce the number of relevant basis functions selected. A more fundamental difference is the policy evaluation method addressed by the respective formulation; GPTD models the Bellman residuals. Thus, in its original formulation GPTD can be only applied to RL problems with deterministic state transitions. In contrast, we provide a unified and concise formulation of LSPE (and LSTD) with kernels, which can deal with stochastic state transitions as well.

REFERENCES

[1] F. R. Bach and M. I. Jordan. Predictive low-rank decomposition for kernel methods. In *Proc. of ICML 22*, 2005.

[2] D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[3] D. P. Bertsekas, V. S. Borkar, and A. Nedić. Improved temporal difference methods with linear function approximation, LIDS Tech. Report 2573, MIT, 2003, also appears in Learning and Approximate Dynamic Programming, by A. Barto, W. Powell, J. Si, (Eds.), IEEE Press, 2004.

[4] D. P. Bertsekas and S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming, LIDS Tech. Report LIDS-P-2349, MIT, 1996.

[5] J. A. Boyan. Least-squares temporal difference learning. In *Proc. of ICML 16*, 1999.

[6] S. J. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

[7] L. Csató and M. Opper. Sparse representation for Gaussian process models. In *Advances in NIPS 13*, pages 444–450, 2001.

[8] Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proc. of ICML 20*, pages 154–161, 2003.

[9] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with gaussian processes. In *Proc. of ICML 22*, 2005.

[10] Y. Engel, P. Szabo, and D. Volkinshtein. Learning to control an octopus arm with gaussian process temporal difference methods. In *Advances in NIPS 17*, 2005.

[11] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representation. *JMLR*, 2:243–264, 2001.

[12] G. Golub and C. Van Loan. *Matrix Computations (3rd Edition)*. John Hopkins University Press, Baltimore, 1996.

[13] A. E. Hoerl and R. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(3):55–67, 1970.

[14] T. Jung and D. Polani. Learning robocup-keepaway with kernels. submitted, 2006.

[15] T. Jung and D. Polani. Sequential learning with ls-svm for large-scale data sets. In *Proc. of ICANN 16*, 2006.

[16] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *JMLR*, 4:1107–1149, 2003.

[17] Z. Luo and G. Wahba. Hybrid adaptive splines. *J. Amer. Statist. Assoc.*, 92:107–116, 1997.

[18] A. Nedić and D. P. Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications*, 13:79–110, 2003.

[19] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of IEEE*, 78:1481–1497, 1990.

[20] C. E. Rasmussen and J. Quinonero-Candela. Healing the relevance vector machine through augmentation. In *Proc. of ICML 22*, 2005.

[21] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[22] A. Sayed. *Fundamentals of Adaptive Filtering*. Wiley Interscience, 2003.

[23] B. Schölkopf and A. Smola. *Learning with Kernels*. Cambridge, MA: MIT Press, 2002.

[24] A. J. Smola and P. L. Bartlett. Sparse greedy Gaussian process regression. In *Advances in NIPS 13*, pages 619–625, 2001.

[25] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proc. of ICML 17*, pages 911–918, 2000.

[26] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[27] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[28] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in NIPS 13*, pages 682–688, 2001.