Least Squares SVM for Least Squares TD Learning

Jung Tobias[1] and **Polani Daniel**[2]

Abstract. We formulate the problem of least squares temporal difference learning (LSTD) in the framework of least squares SVM (LS-SVM). To cope with the large amount (and possible sequential nature) of training data arising in reinforcement learning we employ a subspace based variant of LS-SVM that sequentially processes the data and is hence especially suited for online learning. This approach is adapted from the context of Gaussian process regression and turns the unwieldy original optimization problem (with computational complexity being cubic in the number of processed data) into a reduced problem (with computional complexity being linear in the number of processed data). We introduce a QR decomposition based approach to solve the resulting generalized normal equations incrementally that is numerically more stable than existing recursive least squares based update algorithms. We also allow a forgetting factor in the updates to track non-stationary target functions (i.e. for the use with optimistic policy iteration). Experimental comparison with standard CMAC function approximation indicate that LS-SVMs are well-suited for online RL.

1 Introduction

Least-squares TD (LSTD) One very important subproblem in reinforcement learning (RL) is the policy evaluation problem, e.g. see [13]. The goal is to compute the value function under a fixed policy π (we assume a deterministic π), which is taken to be the infinite horizon, discounted sum of rewards $V^\pi(\mathbf{x}) = E^\pi\left\{\sum_{i=0}^{\infty}\gamma^i r_i \,|\mathbf{x}_0=\mathbf{x}\right\}$ for the Markov chain resulting from starting in state $\mathbf{x}$ and choosing actions according to policy π afterwards. Here r_i denotes the reward obtained while traversing from state $\mathbf{x}_{i-1}$ to $\mathbf{x}_i$, and $\gamma \in [0,1)$ denotes a discount factor. The expectation E^π is meant wrt. the distribution of complete trajectories. However, the Bellman equation allows us to write V^π just using the distribution of the successor states $\mathbf{x}'$, i.e. $V^\pi(\mathbf{x}) = E^\pi\left\{r_t+\gamma V^\pi(\mathbf{x}')\,|\mathbf{x}_{t-1}=\mathbf{x}\right\}$. To compute V^π from actual trajectories we can employ temporal difference learning, e.g. TD(0), which employs *incremental* updates of the form $[r+\gamma V(\mathbf{x}')]-V(\mathbf{x})$ for the observed transition from $\mathbf{x}$ to $\mathbf{x}'$ as an unbiased estimate for the true (unknown) expectation. It works well in conjunction with parametrized function approximation and is usually trained with (stochastic) gradient descent.

A variant that computes a solution in closed form is LSTD [2, 8], minimizing the Bellman residuals for *all* observed transitions in a least squares sense. This approach is more efficient since it makes simultaneous use of all transitions seen so far. One disadvantage[3] of LSTD is that it involves solving (recursively) the corresponding normal equations which is more costly than mere gradient descent and has limited its use to parametrized function approximation.

In this paper we aim to exploit the data efficiency of LSTD and the superior generalization capabilities of kernel-based methods as underlying function approximator. Kernel-based methods (e.g. SVM) are a flexible and more powerful alternative to parametric methods, since the solution does not depend on a small number of features chosen beforehand but is expanded in the training data itself. In the past, kernel-based methods have mostly been used for batch RL [4]. More recently, based on the online sparsification for Gaussian process regression (GPR) from [3], a GPR approach for online TD learning was proposed in [5].

Our approach is in some respects similar to theirs (since generally the predictor obtained from LS-SVM equals the posterior mean of the GPR and the regularization parameter just corresponds to the noise estimate). Our contribution lies 1.) in an alternative, elegant derivation using LS-SVM, 2.) presenting a QR decomposition based update scheme that is numerically more stable, 3.) including a forgetting factor to track non-stationary target functions (i.e. for the use with optimistic policy iteration) and 4.) providing further experimental evidence that kernel-based learning is well-suited for RL.

LS-SVM for LSTD Assume we have observed the t deterministic transitions $\{(\mathbf{x}_{i-1} \rightarrow \mathbf{x}_i, r_i)\}_{i=1}^t$, $\mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^d$, $r_i \in \mathcal{Y} \subset \mathbb{R}$. The goal is to find a function $V : \mathcal{X} \rightarrow \mathcal{Y}$ which minimizes the residuals for the given data and generalizes well to unseen data. We proceed in the usual way: choose the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ of functions $V : \mathcal{X} \rightarrow \mathcal{Y}$ endowed with kernel k, where $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{Y}$ is a symmetric, positive function (e.g. think of Gaussian radial basis functions). To learn the unknown function we solve a Tikhonov functional of the special form

$$\min_{V\in\mathcal{H}} \frac{1}{t}\sum_{i=1}^{t}\Big[\big(V(\mathbf{x}_{i-1})-\gamma V(\mathbf{x}_i)\big)-r_i\Big]^2+\frac{\Lambda}{t}\|V\|_{\mathcal{H}}^2 . \quad (1)$$

The first term measures the error in the approximation (of temporal differences) and the second term penalizes the complexity (i.e. the roughness) of the candidate V. The regularization parameter Λ/t controls the trade-off between them. The Representer theorem tells us that every solution to (1) is the sum of kernels centered on the data: i.e. $V(\cdot)=\sum_{i=1}^t \alpha_i k(\mathbf{x}_i,\cdot)$ where $\boldsymbol{\alpha}=\big(\alpha_1,\ldots,\alpha_t\big)^T$ are the coefficients that we need to determine. Define a $t\times(t+1)$ band matrix $\mathbf{D}$ by setting $d_{ii}=1$ and $d_{i,i+1}=-\gamma$. Replacing V in (1) by its expansion and using that in RKHS we have the property $\langle k(\mathbf{x}_i,\cdot),k(\mathbf{x}_j,\cdot)\rangle_{\mathcal{H}}=k(\mathbf{x}_i,\mathbf{x}_j)$, we then arrive at

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^t}\ \frac{1}{t}\|\mathbf{D}\mathbf{K}_{tt}\boldsymbol{\alpha}-\mathbf{r}\|^2+\frac{\Lambda}{t}\boldsymbol{\alpha}^T\mathbf{K}_{tt}\boldsymbol{\alpha} \quad (2)$$

[1] University of Mainz, Germany, email:tjung@informatik.uni-mainz.de
[2] University of Hertfordshire, UK, email: d.polani@herts.ac.uk
[3] Also, if the transitions are stochastic, we need a second independent sample for each transition [1]. To allow a clearer exposition of our algorithm we will only consider deterministic transitions. For stochastic MDPs this problem can be side-stepped by least squares fi xed point approximation [2, 8].

with $\mathbf{K}_{tt}$ being the $t \times t$ kernel matrix $[\mathbf{K}_{tt}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{r} = \big(r_1, \ldots, r_t\big)^T$ being the sequence of observed rewards. Solving for $\boldsymbol{\alpha}$ we obtain

$$\big(\mathbf{H}_{tt}^T \mathbf{H}_{tt} + \Lambda \mathbf{K}_{tt}\big)\boldsymbol{\alpha} = \mathbf{H}_{tt}^T \mathbf{r}\,. \tag{3}$$

where the $t \times t$ matrix $\mathbf{H}_{tt}$ is defined as $\mathbf{H}_{tt} =_{\text{def}} \mathbf{D}\mathbf{K}_{tt}$.

The central difficulty when using a non-parametric approach like (1) is that the computational and memory complexity scales badly in the data: the kernel matrix is dense and requires $\mathcal{O}(t^2)$ storage, solving the generalized normal equations (3) requires $\mathcal{O}(t^3)$ operations and even every single prediction using the learned model requires $\mathcal{O}(t)$ operations. Clearly, this would be completely infeasable when used in conjunction with RL. In the next section we review *sparse approximation* as means to overcome this problem.

2 Background: Projection on a subset of kernels

The technique of sparse approximation attempts to approximate the kernel matrix using only a small subset of the data and avoids explicit use and storage of the full $t \times t$ kernel matrix. In the subsequent computations it allows us to consider a greatly reduced number of variables. It is a very common technique in the context of Gaussian process regression [3, 5, 11, 14].

Sparse approximation Sparse approximation is based on the observation that the kernel matrix often has rapidly decaying eigenvalues. Vanishing eigenvalues indicate that the data spans – approximately – a rather low dimensional manifold in the feature space. Thus, instead of using all the data points we can restrict ourselves to use only those composing a basis of this manifold and project the remaining ones onto their span. Let this basis be $\{k(\tilde{\mathbf{x}}_i, \cdot)\}_{i=1}^m$ for a subset of m chosen data points[4] ($m \leq t$). Now, for arbitrary $\mathbf{x}_i$ kernel $k(\mathbf{x}_i, \cdot)$ is approximated using only the m chosen basis elements

$$k(\mathbf{x}_i, \cdot) \approx \sum_{j=1}^{m} a_{ij} k(\tilde{\mathbf{x}}_j, \cdot) \quad i = 1 \ldots t. \tag{4}$$

In particular $\mathbf{a}_i$ becomes the $j-$th unit vector if $\mathbf{x}_i$ equals basis element $\tilde{\mathbf{x}}_j$ for some index j, so that the m data points constituting the basis are represented exactly in the feature space. The remaining $t - m$ ones are represented only approximately. To obtain the coefficients a_{ij} we minimize in RKHS $\mathcal{H}$ the distance $d_i := \left\| k(\mathbf{x}_i, \cdot) - \sum_{j=1}^m a_{ij} k(\tilde{\mathbf{x}}_j, \cdot) \right\|_{\mathcal{H}}^2$. Writing this in terms of the inner product and setting its derivative to zero we arrive at

$$\mathbf{a}_i = \mathbf{K}_{mm}^{-1} \mathbf{k}_i \tag{5}$$

where $\mathbf{K}_{mm}$ is the $m \times m$ kernel matrix corresponding to the basis elements with $[\mathbf{K}_{mm}]_{ij} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ and $\mathbf{k}_i = \big(k(\tilde{\mathbf{x}}_1, \mathbf{x}_i), \ldots, k(\tilde{\mathbf{x}}_m, \mathbf{x}_i)\big)^T$. Once we have determined $\mathbf{a}_i$ for input $\mathbf{x}_i$, we obtain the corresponding approximation error as

$$d_i = k_{ii} - \mathbf{k}_i^T \mathbf{a}_i \tag{6}$$

where $k_{ii} = k(\mathbf{x}_i, \mathbf{x}_i)$. This quantity tells us how well the basis is able to approximate the i-th data point.

[4] To avoid using cumbersome index sets we just mark data points selected into the basis by a tilde

Online selection of the subset Next we need to deal with the selection of the subset $\{\tilde{\mathbf{x}}_i\}_{i=1}^m$ which is going to make up our basis. Possibilities include a greedy and iterative approach, adding the element to the basis that best represents all remaining non-basis elements [12], the incomplete Cholesky factorization [7] or just random selection [14]. However, all methods but the latter are unsuitable when the data arrives incrementally. For this particular purpose, [3] and later [6] have proposed a greedy online scheme that builds up the basis incrementally and expands it whenever it becomes necessary. We will apply their method in our approach without modification: assume the data arrives sequentially at $t = 1, 2, \ldots$ and that we can examine every data point once but cannot revise our decision later. We maintain a list of currently chosen basis-elements $\mathcal{D}_m$ (the dictionary), the inverse $\mathbf{K}_{mm}^{-1}$ and assume that after seeing $t-1$ samples dictionary $\mathcal{D}_m$ contains m elements. When a new example $\mathbf{x}_t$ arrives we compute how well it can be approximated using $\mathcal{D}_m$, i.e. we compute (6). If this distance is below some chosen threshold ν, the current basis represents $\mathbf{x}_t$ well enough and we do not need to consider it further. If the distance exceeds the chosen tolerance, we add $k(\mathbf{x}_t, \cdot)$ to the basis: expand $\mathcal{D}_{m+1} = \mathcal{D}_m \cup \{\mathbf{x}_t\}$ and update $\mathbf{K}_{mm}^{-1}$. Since the change to $\mathbf{K}_{mm}$ just involves appending $\mathbf{k}_t$ the new inverse can be obtained recursively either using the partitioned matrix inversion formula or the Cholesky factorization. Computational cost and storage is $\mathcal{O}(m^2)$.

It can be shown [6] that for threshold $\nu > 0$ the number m of selected elements is bound by *Const* $\cdot \nu^{-d}$, where d is the dimensionality of the input domain $\mathcal{X} \subset \mathbb{R}^d$. Hence, for a continuous stream of data $\{\mathbf{x}_t\}_{t=1}^{\infty}$ the dictionary remains finite. For any reasonable choice of ν (e.g. $\nu = 10^{-2}$) the number of selected elements m is usually only a fraction of the number t of observed data points and further stops increasing during the later stages of learning.

Solving a reduced problem Distinguish between batch and online selection of the subset. First consider the batch case, where we compute the $\mathbf{a}_i$ *after* we have determined the m basis elements. Define a $t \times m$ matrix $\mathbf{A}$ with rows $\mathbf{a}_i^T$ from (5). Then we can write

$$\mathbf{A}^T = \mathbf{K}_{mm}^{-1} \mathbf{K}_{tm}^T \tag{7}$$

with $\mathbf{K}_{tm}$ being the m columns of the full kernel matrix corresponding to the m indices of the basis elements. We obtain that $\hat{\mathbf{K}}_{tt} =_{\text{def}} \mathbf{A}\mathbf{K}_{mm}\mathbf{A}^T = \mathbf{K}_{tm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{tm}^T$ is a low-rank approximation of the full kernel matrix $\mathbf{K}_{tt}$. To benefit from this approximation [12] suggest the following: solve the regularized risk functional (1) using all data points, but allow only the coefficients of the m points constituting the basis to have non-zero values. This leads to a modified problem (c.f. the original problem (2)):

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^m} \frac{1}{t} \left\| \mathbf{D}\mathbf{K}_{tm}\boldsymbol{\alpha} - \mathbf{r} \right\|^2 + \frac{\Lambda}{t} \boldsymbol{\alpha}^T \mathbf{K}_{mm} \boldsymbol{\alpha} \tag{8}$$

and to the system of linear equations

$$\big(\mathbf{H}_{tm}^T \mathbf{H}_{tm} + \Lambda \mathbf{K}_{mm}\big)\boldsymbol{\alpha} = \mathbf{H}_{tm}^T \mathbf{r} \tag{9}$$

where the $t \times m$ matrix $\mathbf{H}_{tm}$ is defined as $\mathbf{H}_{tm} = \mathbf{D}\mathbf{K}_{tm}$. Thus we end up solving the $m \times m$ system (9) rather than the $t \times t$ system (3), which reduces computational and storage costs to manageable $\mathcal{O}(m^3)$ and $\mathcal{O}(m^2)$ respectively.

If we assume online selection of the basis elements, to compute the $\mathbf{a}_i$ we can use only the basis elements found up to the current time step (whereas in the batch case we would use the complete basis). Indices in $\mathbf{a}_i$ corresponding to elements added in future are set to

zero. Instead of (7) we obtain that $\tilde{\mathbf{K}}_{tm} =_{\text{def}} \mathbf{A}\mathbf{K}_{mm}$ is only an approximition of the true submatrix $\mathbf{K}_{tm}$. To indicate this difference, we will use the notation $\tilde{\mathbf{H}}_{tm}$ instead of $\mathbf{H}_{tm}$ in (9).

3 Time recursive solution using the QR method

The usual way to solve the normal equations (9) incrementally for each observed transition ($\mathbf{x}_{t-1} \rightarrow \mathbf{x}_t, r_t$) is to employ recursive least squares methods. Our approach is different though, we present a QR based solution adapted from the adaptive filtering literature, e.g. [10], which comes at the same cost of $\mathcal{O}(m^2)$ operations per step but has increased numerical stability.

Concerning notation: the first subscript t will denote the current number of transitions and the second subscript m will denote the current number of basis elements. Substituting in (9) the $m \times m$ cross-product matrix by $\mathbf{\Phi}_{tm} = (\tilde{\mathbf{H}}_{tm}^T \tilde{\mathbf{H}}_{tm} + \Lambda \mathbf{K}_{mm})$, and the rhs by $m \times 1$ vector $\mathbf{s}_{tm} = \tilde{\mathbf{H}}_{tm}^T \mathbf{r}$ eq. (9) becomes

$$\mathbf{\Phi}_{tm} \boldsymbol{\alpha}_t = \mathbf{s}_{tm} \tag{10}$$

Next we introduce the Cholesky factorization $\mathbf{\Phi}_{tm} = \mathbf{\Phi}_{tm}^{1/2} \mathbf{\Phi}_{tm}^{T/2}$ and an auxiliary $m \times 1$ vector $\mathbf{q}_{tm}$. To solve (10) we first solve $\mathbf{\Phi}_{tm}^{1/2} \mathbf{q}_{tm} = \mathbf{s}_{tm}$ and then $\mathbf{\Phi}_{tm}^{T/2} \boldsymbol{\alpha}_t = \mathbf{q}_{tm}$.

Assume we have observed $t-1$ transitions and selected m basis functions. Each update t now proceeds as follows: we check if $\mathbf{x}_t$ is represented by the current basis well enough or if we will need to augment the basis. The involved quantities $\{\mathbf{\Phi}_{tm}^{1/2}, \mathbf{q}_{tm}, \mathbf{K}_{mm}^{1/2}\}$ are then updated accordingly. At the end we just solve $\mathbf{\Phi}_{tm}^{T/2} \boldsymbol{\alpha}_t = \mathbf{q}_{tm}$ to obtain the desired $\boldsymbol{\alpha}_t$. Below we sketch the derivation of the central steps (see Fig. 1 for the complete algorithm).

Step 1: Processing new data without augmenting the basis: $\mathbf{x}_t$ is approximately represented by the current dictionary, therefore we only add the row vector $\mathbf{h}_t^T = (\mathbf{k}_{t-1} - \gamma \mathbf{k}_t)^T$ to the designmatrix. Thus,

$$\begin{aligned} \mathbf{\Phi}_{tm} &= \begin{bmatrix} \tilde{\mathbf{H}}_{t-1,m} \\ \mathbf{h}_t^T \end{bmatrix}^T \begin{bmatrix} \tilde{\mathbf{H}}_{t-1,m} \\ \mathbf{h}_t^T \end{bmatrix} + \Lambda \mathbf{K}_{mm} \\ &= \mathbf{\Phi}_{t-1,m} + \mathbf{h}_t \mathbf{h}_t^T \end{aligned}$$

and hence

$$\mathbf{\Phi}_{tm} \mathbf{q}_{tm} = \mathbf{\Phi}_{t-1,m} \mathbf{q}_{t-1,m} + r_t \mathbf{h}_t.$$

To carry out this update we employ the *array* notation from [10]: we start by arranging the quantities $\{\mathbf{\Phi}_{t-1,m}^{1/2}, \mathbf{q}_{t-1,m}\}$ from the previous step and the newly obtained information $\{\mathbf{h}_t, r_t\}$ in a clever way. If we then generate a orthogonal transformation Θ_t to lower triangulize the resulting $(m+1) \times (m+1)$ array

$$\begin{bmatrix} \mathbf{\Phi}_{t-1,m}^{1/2} & \mathbf{h}_t \\ \mathbf{q}_{t-1,m} & r_t \end{bmatrix} \Theta_t = \begin{bmatrix} \mathbf{\Phi}_{tm}^{1/2} & \mathbf{0} \\ \mathbf{q}_{tm} & * \end{bmatrix} \tag{11}$$

we can directly read off the desired quantities $\{\mathbf{\Phi}_{tm}^{1/2}, \mathbf{q}_{tm}\}$ from the right hand side[5] of (11) due to the norm and inner product preserving properties of orthogonal transformations. To determine Θ_t we can, for example, apply m Givens rotations to successively annihilate the rightmost column $\mathbf{h}_t$ in (11).

[5] The $*$ denotes a scalar quantity that is not relevant for us.

Step 2: Processing new data and augmenting the basis: $\mathbf{x}_t$ is *not* approximately represented by the current dictionary. Now we actually have to perform two substeps. The first substep is to account for the change when adding a new example (which adds the new row $\mathbf{h}_t^T = (\mathbf{k}_{t-1} - \gamma \mathbf{k}_t)^T$ to the designmatrix and is step 1 from above) and the second substep is to account for the change when adding a new basis function (which adds the new column $\tilde{\mathbf{H}}_{t-1,m} \mathbf{a}_t$ to the designmatrix). Setting $h_{tt} = k(\mathbf{x}_{t-1}, \mathbf{x}_t) - \gamma k(\mathbf{x}_t, \mathbf{x}_t)$ we obtain the recursion

$$\begin{aligned} \mathbf{\Phi}_{t,m+1} &= \begin{bmatrix} \tilde{\mathbf{H}}_{t-1,m} & \tilde{\mathbf{H}}_{t-1,m} \mathbf{a}_t \\ \mathbf{h}_t & h_{tt} \end{bmatrix}^T \begin{bmatrix} \tilde{\mathbf{H}}_{t-1,m} & \tilde{\mathbf{H}}_{t-1,m} \mathbf{a}_t \\ \mathbf{h}_t & h_{tt} \end{bmatrix} \\ &\quad + \Lambda \begin{bmatrix} \mathbf{K}_{mm} & \mathbf{k}_t \\ \mathbf{k}_t^T & k_{tt} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{\Phi}_{tm} & \mathbf{\Phi}_{t-1,m} \mathbf{a}_t + h_{tt} \mathbf{h}_t \\ \mathbf{a}_t^T \mathbf{\Phi}_{t-1,m}^T + h_{tt} \mathbf{h}_t^T & \zeta \end{bmatrix} \end{aligned}$$

where ζ is short for $\zeta = \mathbf{a}_t^T \mathbf{\Phi}_{t-1,m} \mathbf{a}_t + h_{tt}^2 + \Lambda(k_{tt} - \mathbf{a}_t^T \mathbf{k}_t)$. After computing ζ (where we need $\mathbf{\Phi}_{t-1,m}$) we perform Step 1 (above) to obtain $\mathbf{\Phi}_{tm}$. Then we update the Cholesky factor $\mathbf{\Phi}_{tm}^{1/2}$ by solving $\mathbf{\Phi}_{tm}^{1/2} \mathbf{u} = \mathbf{\Phi}_{t-1,m} \mathbf{a}_t + h_{tt} \mathbf{h}_t$ for $\mathbf{u}$ and setting scalar $\beta = \sqrt{\zeta - \mathbf{u}^T \mathbf{u}}$. Finally we obtain the desired quantities $\{\mathbf{\Phi}_{t,m+1}^{1/2}, \mathbf{q}_{t,m+1}\}$ via

$$\begin{aligned} \mathbf{\Phi}_{t,m+1}^{1/2} &= \begin{bmatrix} \mathbf{\Phi}_{tm}^{1/2} & \mathbf{0} \\ \mathbf{u}^T & \beta \end{bmatrix} \\ \mathbf{q}_{t,m+1} &= \begin{bmatrix} \mathbf{q}_{tm} \\ (\mathbf{a}_t^T \mathbf{\Phi}_{t-1,m}^{1/2} \mathbf{q}_{t-1,m} + h_{tt} r_t - \mathbf{u}^T \mathbf{q}_{tm})/\beta \end{bmatrix} \end{aligned}$$

Exponential forgetting Our contribution to track a non-stationary target function (e.g. when using this approach for optimistic policy iteration and slowly changing policies π) is to include a forgetting factor $0 \ll \lambda < 1$ (e.g. $\lambda = 0.999$) that puts exponentially less emphasis on past experiences. Instead of (9) we solve a modified problem with $\mathbf{\Sigma}_t =_{\text{def}} \text{diag}(\lambda^{t-1}, \ldots, \lambda, 1)$

$$\big(\tilde{\mathbf{H}}_{tm}^T \mathbf{\Sigma}_t \tilde{\mathbf{H}}_{tm} + \Lambda \lambda^t \mathbf{K}_{mm}\big) \boldsymbol{\alpha} = \tilde{\mathbf{H}}_{tm}^T \mathbf{\Sigma}_t \mathbf{r}$$

The derivation is similar to above, and the resulting updates are summarized in the complete algorithm in Fig. 1.

4 Experiments

Here we use simulation experiments both for policy evaluation (offline learning) and optimal control (online learning) to examine if our approach indeed benefits from better generalization when compared with two standard parametric methods: (1) grid-based tilecoding (CMAC) and (2) radial basis function networks.

Policy evaluation The first experiment is a puddle-world consisting of 101x101 cells (scaled to [0,1]). Possible actions are moving into one of the four adjacent cells. The goal state is the cell in the upper right corner. Each step not leading into the goal yields the reward $r = -0.1$. Three overlapping "puddles" (Gaussians) are placed at random and incur an additional penalty proportional to their activation. The location and spread of the puddles is depicted in Fig. 3. We computed the optimal value function and policy for this domain and generated transitions for 500 different episodes, each starting from a randomly chosen state and following the optimal policy to the goal (resulting in a training set of $\sim$50,000 observed transitions). We then

Algorithm Online policy evaluation with sparse LS-SVM and exponential forgetting

Store: Dictionary $\mathcal{D}_m = \{\tilde{\mathbf{x}}_i\}_{i=1}^m$, $\mathbf{\Phi}_{tm}^{1/2}, \mathbf{q}_{tm}, \mathbf{K}_{mm}^{1/2}$, current number of basis functions m
Parameter: Accuracy ν, regularization Λ, discount factor γ, forgetting factor λ
Output: Weights $\boldsymbol{\alpha}_t$ for the approximated value function $V(\cdot) = \sum_{i=1}^m \boldsymbol{\alpha}_t k(\tilde{\mathbf{x}}_i, \cdot)$
For $t = 1, 2, \ldots$ observe transition $(\mathbf{x}_{t-1} \to \mathbf{x}_t, r_t)$ under (fixed) policy π

1. Sparse approximation for $\mathbf{x}_t$
Compute $\mathbf{k}_t, \mathbf{h}_t, k_{tt}, h_{tt}$
Obtain $\mathbf{p}$ from $\mathbf{K}_{mm}^{1/2}\mathbf{p} = \mathbf{k}_t$ and $\mathbf{a}_t$ from $\mathbf{K}_{mm}^{T/2}\mathbf{a}_t = \mathbf{p}$
If $k_{tt} - \mathbf{a}_t^T\mathbf{k}_t > \nu$
Add $\mathbf{x}_t$ to the dictionary, $\mathbf{K}_{mm}^{1/2} \leftarrow [\mathbf{K}_{mm}^{1/2}, \mathbf{0}; \mathbf{p}^T, \sqrt{k_{tt} - \mathbf{p}^T\mathbf{p}}]$, **Goto 3.**
Else goto 2.

2. Add data point but do not augment basis ($\{\mathbf{\Phi}_{t-1,m}^{1/2}, \mathbf{q}_{t-1,m}\} \mapsto \{\mathbf{\Phi}_{tm}^{1/2}, \mathbf{q}_{tm}\}$):
Generate Θ_t to lower triangularize the array

$$\begin{bmatrix} \lambda^{1/2}\mathbf{\Phi}_{t-1,m}^{1/2} & \mathbf{h}_t \\ \lambda^{1/2}\mathbf{q}_{t-1,m} & r_t \end{bmatrix} \Theta_t = \begin{bmatrix} \mathbf{\Phi}_{tm}^{1/2} & \mathbf{0} \\ \mathbf{q}_{tm} & * \end{bmatrix}$$

and read $\mathbf{\Phi}_{tm}^{1/2}, \mathbf{q}_{tm}$ from the rhs. Solve $\mathbf{\Phi}_{tm}^{T/2}\boldsymbol{\alpha}_t = \mathbf{q}_{tm}$ if necessary.

3. Add data point and augment basis ($\{\mathbf{\Phi}_{t-1,m}^{1/2}, \mathbf{q}_{t-1,m}\} \mapsto \{\mathbf{\Phi}_{t,m+1}^{1/2}, \mathbf{q}_{t,m+1}\}$):
Compute $\mathbf{c} = \mathbf{\Phi}_{t-1,m}^{T/2}\mathbf{a}_t$ and $d = \mathbf{c}^T\mathbf{q}_{t-1,m}$. Then get $\mathbf{\Phi}_{tm}^{1/2}, \mathbf{q}_{tm}$ from 2.
Obtain $\mathbf{u}$ from $\mathbf{\Phi}_{tm}^{1/2}\mathbf{u} = \lambda\mathbf{\Phi}_{t-1,m}^{1/2}\mathbf{c} + h_{tt}\mathbf{h}_t$
Compute $\beta = \sqrt{\lambda\mathbf{c}^T\mathbf{c} + h_{tt}^2 + \Lambda\lambda^t(k_{tt} - \mathbf{a}_t^T\mathbf{k}_t) - \mathbf{u}^T\mathbf{u}}$
Update

$$\mathbf{\Phi}_{t,m+1}^{1/2} = \begin{bmatrix} \mathbf{\Phi}_{tm}^{1/2} & \mathbf{0} \\ \mathbf{u}^T & \beta \end{bmatrix}, \quad \mathbf{q}_{t,m+1} = \begin{bmatrix} \mathbf{q}_{tm} \\ (\lambda d + h_{tt}r_t - \mathbf{u}^T\mathbf{q}_{tm})/\beta \end{bmatrix}$$

and solve $\mathbf{\Phi}_{t,m+1}^{T/2}\boldsymbol{\alpha}_t = \mathbf{q}_{t,m+1}$ if necessary.

Figure 1. Online policy evalution with sparse LS-SVM at $\mathcal{O}(m^2)$ operations per step, m being the number of basis functions

wished to examine how fast learning occurs when coupling LSTD with different types of function approximation. The transitions were fed successively into the algorithm and the error between approximation and true value function was measured. Function approximators were: (1) our sparsified LS-SVM (Gaussian kernel, $\sigma = 0.02$ with $\nu = 0.1$ and $\nu = 0.01$, $\Lambda = 10^{-3}$), (2) a CMAC (resolution 7x7x7 and 10x10x10), and (3) a RBF-net with fixed centers (spread uniformly on a 12 x 12 grid, $\sigma = 0.02$). The results are shown in Fig. 3 and indicate that our approach is in fact superior as far as generalization is concerned. Also compare the required resources: CMAC uses 343 and 1000 weights, the RBF-net uses 144 weights and our LS-SVM uses 122 and 202 weights. [6]

Optimal control In the second experiment we examine online learning and pit our LSTD-LSSVM against iterative sarsa(λ) with CMAC, which is the standard setup in RL and works well for many problems. To use LSTD for optimal control we employ optimistic policy iteration, and hence include a forgetting factor $\lambda = 0.999$ in the LS-SVM to cope with the non-stationarity of the value function due to the changing policy. The remaining parameters for LS-SVM were unchanged. Since we only consider learning value functions, we supply a generative model for policy improvement. The sarsa(λ) ($\lambda = 0.5$) setup was appropriately modified. As testbed we used the puddle-world from above and the puck-on-hill task from [9]. Both tasks are episodic with designated terminal states: each episode starts from a random state and proceeds until either a terminal state is reached or the number of steps exceeds a fixed limit of 500 steps. Various performance criteria were considered, see Fig. 2 for the results. In both cases LS-SVM outperforms CMAC in terms of generalization, i.e. number of observed transitions to achieve good performance. Again sparse LS-SVM requires far less weights to achieve this level of accuracy, to obtain satisfactory results with the CMAC in the puddle-world we even needed to double the resolution (20x20x10). Fig. 3 illustrate the general benefit when using LSTD-LSSVM policy evaluation instead of iterative TD: just after running 20 trials and visiting only a small fraction of the state space the approximated value function has the same overall shape as the true value function.

[6] Unfortunately we cannot directly compare CPU-time, since our algorithm is implemented in C whereas the other two run in Matlab. Some numbers: our algorithm processes 10,000 training examples with 122 weights in $\sim$2 secs. Dominating this cost with $\mathcal{O}(m^2)$ is the number of weights m.

5 Summary and future work

We formulated the problem of estimating the value function from actual trajectories in a regularization framework using least-squares SVM. The key contribution is a QR-based solution algorithm that is fully incremental, i.e. solves the resulting normal equations independent of the number of observed transitions and can hence be applied to *online* RL. In the future we wish to extend the scope of our work to the full RL problem. Among other things, we will address the issue of exploration, stochasticity in transitions, model-free learning through Q-values, and more comprehensive experiments with high-dimensional control tasks. We also aim to explore other criteria for the subset selection mechanism in sparse approximation (e.g. taking

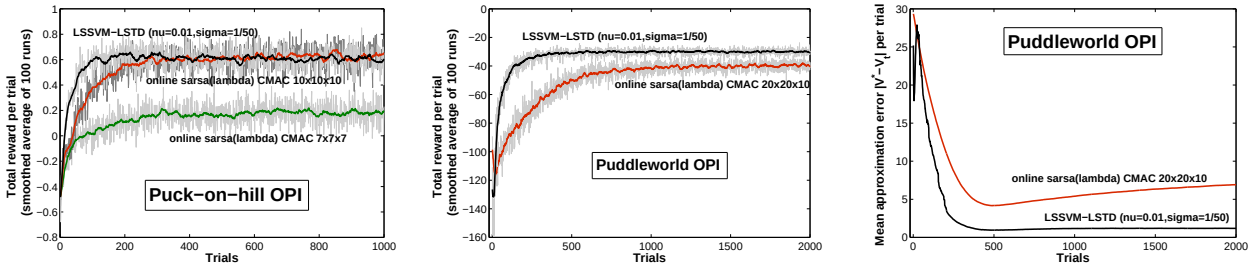

Figure 2. Online learning for optimal control with LS-SVM via optimistic policy iteration (OPI) for two toy problems

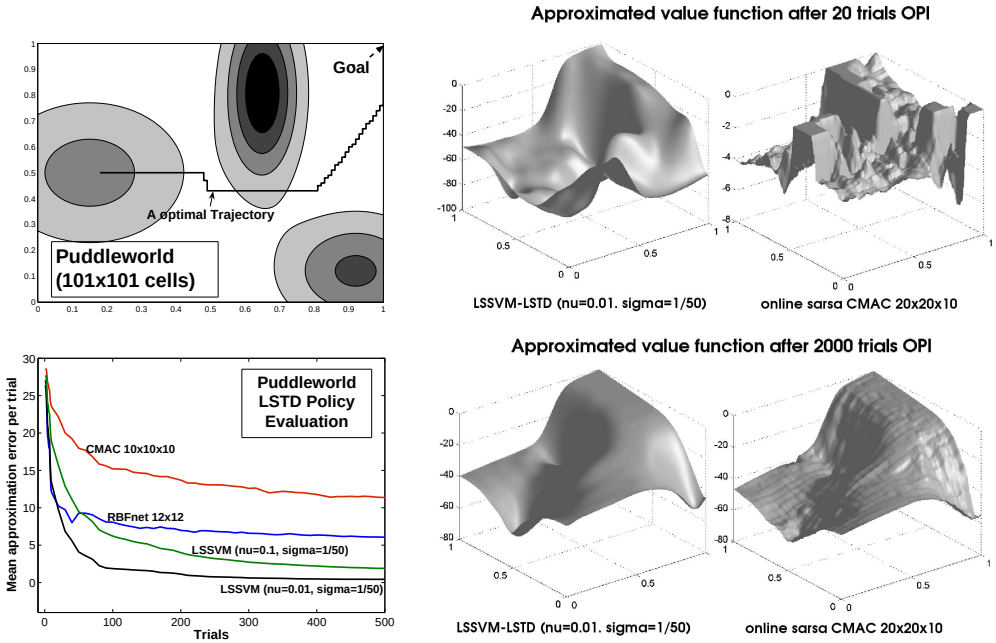

Figure 3. LS-SVM in policy evaluation (left) and OPI (right) for the puddle-world domain

into account the decrease of the error for the final predictor).

We believe that kernel-based methods are very well-suited for RL, since they are data-oriented and thus more flexible than traditional parametric methods: they place parameters where they are needed and when new data arrives and do not waste resources on parts of the state-space that are never visited. Though there is yet not much experimental evidence we believe that this approach could help to tackle high-dimensional control tasks that are yet unsolvable.

REFERENCES

[1] L. C. Baird, 'Residual algorithms: Reinforcement learning with function approximation', in *Proc. of ICML 12*, pp. 30–37, (1995).

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

[3] L. Csató and M. Opper, 'Sparse representation for Gaussian process models', in *Advances in NIPS 13*, (2001).

[4] T. Dietterich and X. Wang, 'Batch value function approximation via support vectors', in *Advances in NIPS 14*, (2002).

[5] Y. Engel, S. Mannor, and R. Meir, 'Bayes meets Bellman: The Gaussian process approach to temporal difference learning', in *Proc. of ICML 20*, (2003).

[6] Y. Engel, S. Mannor, and R. Meir, 'The kernel recursive least squares algorithm', *IEEE Trans. on Sig. Proc.*, **52**(8), 2275–2285, (2004).

[7] S. Fine and K. Scheinberg, 'Effi cient SVM training using low-rank kernel representation', *JMLR*, **2**, 243–264, (2001).

[8] M. G. Lagoudakis and R. Parr, 'Least-squares policy iteration', *JMLR*, **4**, 1107–1149, (2003).

[9] A. W. Moore and C. G. Atkeson, 'The parti-game algorithm for variable resolution reinforcement learning in multi-dimensional state-spaces', *Machine Learning*, **21**(3), 199–233, (1995).

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

[11] A. J. Smola and P. L. Bartlett, 'Sparse greedy Gaussian process regression', in *Advances in NIPS 13*, (2001).

[12] A. J. Smola and B. Schölkopf, 'Sparse greedy matrix approximation for machine learning', in *Proc. of ICML 17*, (2000).

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

[14] C. Williams and M. Seeger, 'Using the Nyström method to speed up kernel machines', in *Advances in NIPS 13*, (2001).