Sparse Approximations to Value Functions in Reinforcement Learning

Hunor S. Jakab and Lehel Csató

Abstract. We present a novel sparsification and value function approximation method for on-line reinforcement learning in continuous state and action spaces. Our approach is based on the kernel least squares temporal difference learning algorithm. We derive a recursive version and enhance the algorithm with a new sparsification mechanism based on the topology obtained from proximity graphs. The sparsification mechanism – necessary to speed up the computations – favors datapoints minimizing the divergence of the target-function gradient, thereby also considering the *shape* of the target function. The performance of our sparsification and approximation method is tested on a standard benchmark RL problem and comparisons with existing approaches are provided.

1 Introduction

Reinforcement learning addresses the problem of learning optimal decision making strategies (policies) for diverse control problems. RL puts special emphasis on the autonomous nature of the learning process. The learning agent develops an efficient policy to reach a predefined goal, without external inference, the only information source being its state and the reward signal that it receives from the environment. Whilst the definition of autonomy is difficult, the design of a *generic* autonomous learning "agent" is based on the idea that the agent explores its environment and aims to collect "rewards" – i.e. to exploit too – during exploration [22]. Given a reinforcement learning algorithm, one needs to build the reward function for it to be applied to a given application domain.

Approximate reinforcement learning (RL) methods are algorithms dealing with real-world problems that are characterized by continuous, high dimensional

Hunor S. Jakab · Lehel Csató
Babeş-Bolyai University, Faculty of Mathematics and Computer Science
e-mail: {jakabh,csatol}@cs.ubbcluj.ro

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_14

state-action spaces and noisy measurements. The purpose of this chapter is to investigate the problem of accurately estimating value functions for a given policy in continuous RL problems, also called *policy evaluation*[22].

Policy evaluation is the process of estimating the utility of a state or state-action pair when the learning agent starts from that state (state-action pair) and follows a fixed strategy (policy) for an infinite number of steps. The function that maps states to utility values is called a value function and it needs to be represented by a function approximation framework. Many reinforcement learning methods rely on the accurate approximation of value functions, which can be used to derive greedy policies that choose the action which leads to the state with highest utility value. Thus the accuracy of value functions around decision boundaries (regions where the utility values change rapidly) is especially important. Traditionally the representation of value functions on continuous domains has been treated using linear maps with non-linear features (*ex. using different order polinomial, fourier or wavelet basis*)[11], neural networks [16], non-parametric approximators [7] etc. To improve the accuracy of the approximated value functions a large variety of algorithms have been proposed.

In this article we focus on linear least-squares algorithms for temporal difference learning [2]. These algorithms similarly to most RL algorithms, require a lot of hand-tuning to work in different application domains. One of the major research areas within RL is to eliminate the need for hand-tuning and to develop methods which can be applied generically to a large variety of problems. A popular way to address this design issue is by applying a nonlinear extension of the algorithm using "kernelization". Kernelization in turn raises the problem of complexity, over-fitting, and increased computational cost. To avoid these problems, different sparsification mechanisms are employed which are meant to eliminate non-essential training-data based on a given selection criteria. The quality of the sparsification procedure influences the accuracy and the generalization capability of the function approximator.

The ability of an RL algorithm to operate on-line, to acquire training-data sequentially by interacting with its environment, is crucial. In this setting, the training data acquired during environment interaction presents some spatio-temporal characteristics which in our opinion can be exploited to develop a superior approximation framework. To do this we introduce a proximity-graph based sparsification mechanism which is based on the on-line construction of a proximity-graph that captures the temporal correlation of the training data. The resulting graph is a coarse representation of the manifold upon which the training data lies, and it allows the sparsification to fine-tune itself by increasing the density of *support points*[1] in the areas where the function changes rapidly. At the same time our algorithm reduces the density of support-points in areas where the target function hardly changes, keeping computational costs down. We apply the sparsification mechanism to the kernel least squares temporal difference learning (KLSTD) [29] algorithm. We also introduce a recursive version of KLSTD that is much better suited to the on-line learning scenario frequently seen in reinforcement learning.

[1] Support points are similar to the support points in Parzen-windowing or in vector-quantization summarizing in a few – weighted – samples a large amount of input location.

The rest of this paper is structured as follows: In Section 2 we introduce some of the basic notions related to reinforcement learning and the notation which we will be using throughout the rest of the paper. Section 3 surveys some of the recent value function approximation methods from the literature. In section 4 we introduce the kernel least squares algorithm for value approximation, while in section 4.1 we present our first major contribution, a recursive version of KLSTD.The sparsification problem is introduced in Section 5, and in Section 5.1 the second major contribution of the paper, the laplacian sparsification mechanism is presented. Section 6 deals with the problem of iteratively constructing proximity graphs to be used in our laplacian sparsification mechanism. The complete value approximation algorithm based on our recursive version of KLSTD together with the laplacian sparsification method is presented in detail in Section 7. Section 8 presents the experimental results and Section 9 concludes the paper.

2 Notation and Background

We introduce the notation we will be using and sketch the background material required to understand the proposed methods. We mention first that in RL data results from interacting with the environment and consequently the successive states of the agent, the actions taken in those states and the corresponding rewards define the training data. Moreover – bringing the method closer to applicability – we assume that data is acquired on-line. The size of the training data therefore is not fixed and it expands continuously as the interaction process takes place.

To describe the environment where the learning process takes place we use the formalism of Markov decision processes (MDP) [15], commonly used for modeling reinforcement learning problems:

An MDP is a quadruple $M(S,A,P,R)$, where S is the (possibly infinite) set of states, A is the set of actions, $P(s'|s,a) : S \times S \times A \to [0,1]$ describes the usually unknown transition probabilities, and $R(s,a) : S \times A \to R$ is the instantaneous reward function defining the feedback to the learner.

The decision making mechanism is modeled with the help of an action selection *policy*: $\pi : S \times A \to [0,1]$. Here π is the conditional probability distribution – $\pi(a|s)$ – of taking action a while being in the state s. In reinforcement learning the goal is to find the *optimal policy* π^*, that maximizes the expected discounted cumulative reward received by the learner:

$$\pi^* = \arg\max_{\pi} E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$$

An important element of many RL algorithm is a representation of the *utility* of the state or state-action pairs as value functions $V_\pi : S \to \mathbb{R}$ or action-value functions $Q_\pi : S \times A \to \mathbb{R}$:

$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s\right]; \quad Q_\pi(s,a) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s, a_0 = a\right]$$

Value functions express the expected long term discounted reward received when starting from a given state or state-action pair and following a given policy π. The majority of reinforcement learning algorithms can be included in two main categories: value based and model-based methods. Model-based algorithms [7], [6] approximate both the system dynamics and the state-value function and apply dynamic programming for the derivation of the optimal policy. In both of these categories the accurate and consistent estimation of state or state-action values plays an essential role.

Since this paper focuses on value approximation, from now on we consider the action selection policy to be fixed. This restriction gives rise to a so-called Markov Reward process [23] which we will refer to as MRP. Throughout the rest of this paper for ease of exposition we will omit the policy from the notation. For better readability we will use state value functions $V(\cdot)$ for the derivation of our algorithms, since their extension to state-action value functions is straightforward.

3 Related Work in Value Approximation

The approximation of value functions can be accomplished by different function approximation architectures. There have been attempts to use neural networks [16], hierarchical function approximation [26], non-parametric methods [19],[20],[9].

Estimation and approximation of value functions is easiest with a linear model: the value is a linear function of the state. These models are too simple to be applied for RL problems. A non-linear extension is to consider models that are linear in their parameters but *non-linearly* map the inputs to the output space, called *linear-in weight models* [21], or *generalized linear models* [13]. Linear regression models with non-linear features have been preferred for approximate policy evaluation, due to their good convergence properties and relative ease of analysis. The main drawback of linear-in-weight regression models is the difficulty in obtaining *good features* for each problem. One way to reduce the hand-tuning required is to decouple the approximation of value functions or state-transition dynamics from the construction of problem-specific feature functions which transform the input data into a more interpretable form. The proto-value function framework presented in [12] achieves this by using diffusion wavelets and the eigenvectors of a graph laplacian for the automatic construction of feature functions. Another popular approach is to use kernel methods for this purpose [18] by formulating the algorithms in such a way that feature functions only appear in dot product form. Gaussian processes are a good example of this, and have been successfully used in a number of references for approximating both value functions and system transition dynamics [6],[7]. Another class of value approximation algorithms is based on the least squares minimization principle and use linear architectures with updates known from temporal difference learning [2],[1]. Least squares based methods are believed to have better sample efficiency, however the expressive power of linear architectures is largely determined by the quality of the feature functions that they use. The learning

algorithms from this class can also be kernelised as seen in [29]. An overview of kernel-based value approximation frameworks can be found in [24].

4 Kernel Least Squares Value Approximation

LSTD is based on a generalized linear approximation to the value function using a set of predefined feature vectors:

$$\tilde{V}(s) = \phi(s)^T \mathbf{w}^H \quad \text{, where} \quad \phi(s) = [\phi_1(s), \ldots, \phi_d(s)]^T. \tag{1}$$

Here $\mathbf{w}^H = [w_1, \ldots, w_d]^T$ is the vector containing the feature weights, and d is the dimension of the feature space. Notice that $\mathbf{w}^H \in \mathbb{R}^d$, *i.e.* it has to match the dimension of the feature space where the input s is projected and superscript H signifies this fact. In temporal difference we incrementally update the value function based on the *temporal difference error* [22]:

$$\delta_{t+1} = R_{t+1} + \gamma \tilde{V}_t(s_{t+1}) - \tilde{V}_t(s_t) = R_{t+1} - (\phi(s_t) - \gamma\phi(s_{t+1}))^T \mathbf{w}_t^H \tag{2}$$

and the parameter update – using the learning parameter α_t – is the following:

$$\mathbf{w}_{t+1}^H \leftarrow \mathbf{w}_t^H + \alpha_t \delta_{t+1} \phi(s_t).$$

where t is time and, since each datum is new, it is also the sample number. In the limit of infinite time t, the algorithm leads to a converged vector $\mathbf{w}^H$ that satisfies $E[\phi(s_t)\delta_{t+1}] = 0$ [23]. We are looking at its equilibrium value, by replacing the theoretical average with the sample average, to obtain the following equation:

$$\frac{1}{n} \sum_{t=0}^{n} \phi(s_t)\delta_{t+1} = 0 \tag{3}$$

Substituting δ_{t+1} from eq. (2) – and dropping n, the sample size – we obtain:

$$\left(\sum_{t=0}^{n} \phi(s_t) (\phi(s_t) - \gamma\phi(s_{t+1}))^T \right) \mathbf{w} = \sum_{t=0}^{n} \phi(s_t) R_t \tag{4}$$

If we use the following notation:

$$\tilde{A}^H = \sum_{t=0}^{n} \phi(s_t) (\phi(s_t) - \gamma\phi(s_{t+1}))^T, \quad \tilde{b}^H = \sum_{t=0}^{n} \phi(s_t) R_t,$$

Then the equation becomes: $\tilde{A}^H \mathbf{w}^H = \tilde{b}^H$, a linear problem. For this system we have the following properties:

- The matrix $\tilde{A}^H$ is a sum of individual outer products, therefore it can be calculated incrementally;
- Each component in the sum is a square matrix with the size of the feature space;
- When $\tilde{A}^H$ is invertible, there is a direct solution for $\mathbf{w}^H$, see *e.g.* [2].

The problem with the solution presented above is that manually constructing suitable feature representations for each problem-domain is time-consuming and error prone. To alleviate this drawback, a non-linear extension via "kernelisation" of the above algorithm has been introduced in [28], briefly sketched in what follows. To eliminate the direct projection into the feature space, we assume that the projection is into a *reproducing kernel Hilbert space* [25, 18]. The solutions to the linear systems can then be written as a linear combination of the feature images of the inputs, *i.e.* $\mathbf{w}^H \stackrel{\text{def}}{=} \sum_{t=1}^{n} w_i \phi(s_i)$. This is equivalent with a re-parametrization of the problem from eq. (4), where (1) we are looking for $\mathbf{w} = [w_1, \ldots, w_n]$ as coefficients solving the problem, and (2) we express all equations in terms of a *dot product* $\phi(s_i)^T \phi(s_j) \rightarrow k(s_i, s_j)$ (called kernel transformation of the problem). The re-parameterized problem optimizes $\mathbf{w}$ and the system is $\mathbf{w}^* = \tilde{A}^{-1}\tilde{\mathbf{b}}$ with parameters:

$$\tilde{A} = \sum_{t=0}^{n} \mathbf{k}(s_t) \left[\mathbf{k}^T(s_t) - \gamma \mathbf{k}^T(s_{t+1})\right], \quad \tilde{\mathbf{b}} = \sum_{t=0}^{n} \mathbf{k}(s_t) R_t, \tag{5}$$

where $\tilde{A}$, $\tilde{\mathbf{b}}$ are the re-parametrized entities of eq. (4) and $k(\cdot,\cdot)$ is an arbitrary kernel function [18]. $\mathbf{k}(s) = [k(s,s_1), \ldots, k(s,s_n)]^T$ is the vector of kernels at the new data s and the training data set $\{s_i \,|\, i = \overline{1,n}\}$. An important consequence is that the expression for the value function – originally in feature space, eq. (1) – is expressed using kernels too:

$$\tilde{V}(s) = \sum_{i=0}^{n} w_i k(s, s_i), \quad \forall\, s \in S. \tag{6}$$

In what follows we reduce the computational time required to obtain the vector $\mathbf{w}$, by first proposing a recursive calculation of the matrix $\tilde{A}^{-1}$ and vector b, then present a sparsification mechanism that reduces the number of parameters in the system.

4.1 Recursive Computation

We assume that we are reading the elements of the data-set one by one. When n inputs have been processed, the size $\tilde{A}$ is n, consequently the computation of $\mathbf{w}$ is $O(n^3)$. On-line learning makes this problem worse since it requires the inversion of $\tilde{A}$ to be performed at each step which is highly impractical. The computational burden of the matrix inversion can be mitigated by keeping $C_t \stackrel{\text{def}}{=} \tilde{A}_t^{-1}$ and updating it incrementally. Suppose we have processed t data-points – therefore $C_t \in \mathbb{R}^{t \times t}$ and $\tilde{\mathbf{b}} \in \mathbb{R}^{t \times 1}$ – and let us write implicitly update to $\tilde{A}_{t+1}$:

$$\begin{aligned} \tilde{A}_{t+1} &= \frac{t}{t+1}\left(\begin{bmatrix} \tilde{A}_t & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \begin{bmatrix} k(s_t,s_1) \\ \vdots \\ k(s_t,s_{t+1}) \end{bmatrix} \cdot \begin{bmatrix} k(s_t,s_1) - \gamma k(s_{t+1},s_1) \\ \vdots \\ k(s_t,s_{t+1}) - \gamma k(s_{t+1},s_{t+1}) \end{bmatrix}^T\right) \\ &\stackrel{\text{def}}{=} \frac{t}{t+1}\left(\begin{bmatrix} \tilde{A}_t & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{u} \\ u^* \end{bmatrix}\begin{bmatrix} \mathbf{v} \\ v^* \end{bmatrix}^T\right) \end{aligned} \tag{7}$$

where – in the second line – we defined, for a more concise notation, the vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{t\times 1}$ and the scalars u^* and v^*. We also see that the inclusion of the new input increases the size of the matrix $\tilde{A}_{t+1}$. With the above notation we can express the recursion for the new value of the *inverse*, that is:

$$C_{t+1} = \tilde{A}_{t+1}^{-1} = \frac{t+1}{t}\begin{bmatrix}\tilde{A}_t + \mathbf{u}\mathbf{v}^T & v^*\mathbf{u} \\ u^*\mathbf{v}^T & u^*v^*\end{bmatrix}^{-1} = \begin{bmatrix} C_t & -C_t\mathbf{u}/u^* \\ -\mathbf{v}^T C_t/v^* & \frac{1+\mathbf{v}^T C_t \mathbf{u}}{u^*v^*}\end{bmatrix} \tag{8}$$

where we used the inversion rules for block matrices and the Sherman-Woodbury formula, similarly to [5]. Notice the particular simplicity of the update: as its name suggests, it is indeed a rank-one update, involving only the last line and column of a – potentially large – matrix. The iterative update of $\mathbf{b}_t$ is:

$$\tilde{\mathbf{b}}_{t+1} = \frac{t}{t+1}\left(\begin{bmatrix}\tilde{\mathbf{b}}_t \\ 0\end{bmatrix} + \begin{bmatrix}\mathbf{u} \\ u^*\end{bmatrix} R_t\right) \tag{9}$$

Combining the above defined representation of the inverse of $\tilde{A}$ with Eq. (5), the optimal coefficients for the approximation of the target function after processing the $(t+1)$-th input data-point can be obtained as:

$$\begin{aligned}\mathbf{w}_{t+1} &= C_{t+1}\tilde{\mathbf{b}}_{t+1} = \begin{bmatrix} C_t & -C_t\mathbf{u}/u^* \\ -\mathbf{v}^T C_t/v^* & \frac{1+\mathbf{v}^T C_t \mathbf{u}}{u^*v^*}\end{bmatrix}\left(\begin{bmatrix}\tilde{\mathbf{b}}_t \\ 0\end{bmatrix} + \begin{bmatrix}\mathbf{u} \\ u^*\end{bmatrix} R_t\right) \\ &= \begin{bmatrix}\mathbf{w}_t \\ \left(R_t - \mathbf{v}^T\mathbf{w}_t\right)/v^*\end{bmatrix} = \begin{bmatrix}\mathbf{w}_t \\ \left(R_t - (\tilde{V}(s_t) - \gamma\tilde{V}(s_{t+1}))\right)/v^*\end{bmatrix}\end{aligned} \tag{10}$$

In the rest of this paper we will refer to the presented iterative algorithm as kernel recursive least squares temporal difference learning algorithm or (KRLSTD). For a detailed derivation of a policy iteration algorithm based on the least squares value function approximation method, see [10].

In the next section we present a new type of sparsification mechanism to keep the matrix $\tilde{A}$ at a small fixed size. Since the sparsification depends on the *values of the inputs*, we need a *"good"* subset, the criteria presented on the next section.

5 Sparsification of the Representation

Sparsification reduces the computational cost of kernel-based algorithms by finding a small set of data-points called *dictionary*, which will be used as basis set for subsequent computation. The subject of selecting a representative set of data-points for reducing computational cost in case of kernel methods has been studied in-depth in the scientific literature. To achieve sparsity a large number of different approaches have been employed. Error-insensitive cost functions in support vector machine regression, reduction of the rank of the kernel matrix by low-rank approximations, greedy feature construction algorithms, using a Bayesian prior to force sparsity in case of fully probabilistic models *etc.* For non-parametric kernel

methods this process can also be called feature selection, since the feature representation of a data-point is determined by the subset of data-points in the dictionary.

The commonly used sparsification method in RL [8] uses approximate linear independence (ALD) as a criterion for discarding data-points. In this context a new point is approximated by the linear combination of dictionary points in feature space, and the approximation error is as follows:

$$\varepsilon = \min_{\alpha} \| \sum_{i=1}^{d} \alpha_i \phi(s_i) - \phi(s^*) \|^2 \tag{11}$$

Here $\phi(s)$ are the feature images of the corresponding data-points, s^* is the new data point and α_i are the variational coefficients. Minimizing the error from (11) and applying the kernel trick leads to the following expression fo the approximation coefficients:

$$\alpha = K^{-1}\mathbf{k}(\mathbf{s}^*) \tag{12}$$

where $K_{i,j} = k(s_i, s_j)$ is the kernel matrix and $\mathbf{k}(\mathbf{s}^*) = \left[k(s_1, s^*) \dots k(s_n, s^*)\right]$ is the vector of kernel values computed on the new data-point s^*.

If the approximation error is below a predefined threshold ν the data-point is discarded and the kernel-matrix is updated using the linear combination coefficients α_i[2]. Otherwise the new data-point is incorporated into the dictionary. In [4] a sparsification mechanism for Gaussian processes is presented which considers the influence of individual data-points in the approximation accuracy. The method is based on calculating the Kullback-Leibler divergence between two posterior Gaussian process distributions: one with a new data-point included and one without. The data-point that causes the smallest change in the posterior mean is discarded. Unfortunately this can only be applied when we have a full probabilistic model available as in the case of Gaussian Processes. Moreover the kernel hyper-parameters have a major influence on the accuracy of this method.

Proponents of sparsification methods based on approximate linear dependency argue that one of its major advantages is that it is not affected by noise in the training data, since it does not take into account the training labels, only the linear dependency of inputs in feature space. In the problem-domain of reinforcement learning, however this becomes a drawback since for the accurate approximation of value functions, the information contained in the training labels needs to be taken into account.

In what follows we introduce a sparsification mechanism that puts a larger emphasis on the characteristics of the target-function in the selection process of dictionary data-points. Our method can be applied on-line, is not dependent on the initial values of the kernel hyper-parameters and it has only one parameter that requires tuning.

5.1 Laplacian Sparsification Criteria

The main idea of our method is to approximate the manifold upon which the training-data lies, and use the characteristics of this manifold to adjust the density of

[2] For a detailed description of the update procedure see [3]

training data-points contained in the dictionary in specific regions of the input space. To obtain a coarse representation of the manifold we propose the on-line construction of a similarity graph based on the observed state-transitions of the system. The graph construction mechanism uses the information contained in the sequential nature of the training data to create edges between vertexes(states or state-action pairs) that are successively traversed[3]. The spectral information contained in the Laplacian of a properly constructed similarity graph can be exploited when deciding on the inclusion or removal of a new data-point to the dictionary.

To provide the necessary background some notions from graph theory are needed.

Let $\mathscr{G}(E,V)$ be an undirected graph with E and V denoting the set of edges and vertices respectively. We define the unnormalized graph Laplacian associated with $G(\mathscr{E},\mathscr{V})$ as: $L^{\mathscr{G}} = D^{\mathscr{G}} - A^{\mathscr{G}}$, where $A^{\mathscr{G}}$ is the weighted adjacency matrix of $\mathscr{G}$ and $D^{\mathscr{G}}_{i,i} = \sum_{j \neq i} A^{\mathscr{G}}_{i,j}$ is a diagonal matrix containing the node degrees on the diagonal.

An interesting property of the graph Laplacian is that it is symmetric, positive semi-definite, and it gives rise to the discrete Laplacian operator Δ:

$$f : V \to \mathbb{R} \quad \Delta f(s_i) = \sum_{j=1}^{n} A^{\mathscr{G}}_{ij} [f(s_i) - f(s_j)] \tag{13}$$

For a given set of data-points consisting of the vertexes of $\mathscr{G} : s_i \in V \quad i = \overline{1,|V|}$ the Laplacian operator applied to the function f is expressed as:

$$\Delta \mathbf{f} = L^{\mathscr{G}} \mathbf{f} \quad \text{where} \quad \mathbf{f} = [f(s_1) \dots f(s_n)]^T \tag{14}$$

For now we assume that the graph approximates the manifold upon which the training data-points are situated, and the shortest paths calculated on $\mathscr{G}$ between two points approximate the geodesic distances between them[4].

Let us denote the approximated function f and the neighbor set of a vertex s_i as $\text{neig}(s_i) = \{s_j \in V | A^{\mathscr{G}}_{i,j} \neq 0\}$. The vertexes of the graph $\mathscr{G}(E,V)$ are the location of samples from the target function. To obtain a sparse dictionary we introduce a score function $\mu(\cdot)$ for a vertex $s_i \in V$ as the weighted sum of the squared differences between target function values of s_i and its neighbouring vertexes:

$$\mu(s_i \in V) = \sum_{s_j \in \text{neig}(s_i)} A^{\mathscr{G}}_{ij} [f(s_i) - f(s_j)]^2 \tag{15}$$

Summing up the individual vertex scores gives an overall measure about the quality of the dictionary set:

$$\sum_{s_i \in V} \mu(s_i) = \sum_{i,j=1}^{n} A^{\mathscr{G}}_{ij} [f(s_i) - f(s_j)]^2 = \mathbf{f}^T L^{\mathscr{G}} \mathbf{f} \tag{16}$$

[3] The on-line construction of the similarity graph is explained in detail in section 6

[4] Geodesic distance is the distance between states along the manifold determined by the transition dynamics of the MRP.

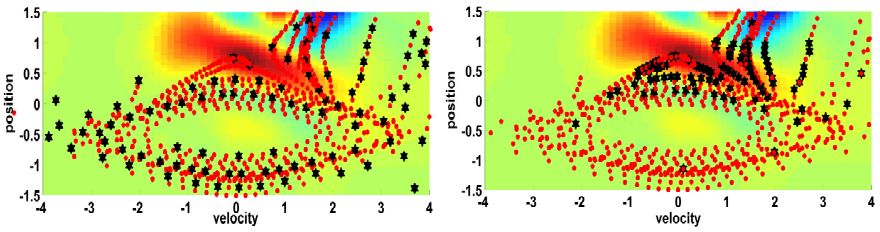


Fig. 1 Dictionary points (*black stars*) obtained after processing 5 roll-outs ($\approx$ 450 data-points) – *red dots* – for the mountain-car control problem using different sparsification techniques. Figure (a): ALD sparsification. Figure (b) our proximity graph-based sparsification. The maximum number of dictionary points was set to 100, KLSTD approximation used.

The above presented score is equivalent to applying the discrete Laplacian operator to the target function, and can be interpreted as the overall divergence of the gradient of f. In Section 7 we describe the complete sparsification algorithm that uses the score from Eq. 16 as a criterion to construct an optimal dictionary.

5.2 Illustrative Example

To illustrate how the above proposed sparsification mechanism changes the density of the dictionary data-points in different regions of the approximated function, we performed a simple experiment on approximating the value function for a fixed policy π on the mountain-car control problem (A detailed description is given in section 8.2). Figure 1 illustrates the differences between approximate linear dependence-based sparsification and our Laplacian-based method. The degree of sparseness in case of ALD is controlled by the threshold parameter ν from section 5. To be able to compare the two algorithms, we adjusted the upper limit on the dictionary size in case of our method to match the number of dictionary points produced by the threshold ν in case of ALD.

The red dots on figure 1 show all the training data-points, the black stars show the elements of the dictionary after sparsification has been performed. The horizontal and vertical axis correspond to the two elements of the state vector and the state values are color coded in the background. Performing sparsification based on the maximization of the score from (16) leads to a higher sample-density in regions where the function changes rapidly (fig. 1b). At the same time the algorithm keeps a sufficient number of data-points in slowly changing regions to obtain good approximation accuracy.

6 On-Line Proximity Graph Construction

The information contained in the sequential nature of the training data in RL can be used to construct a manifold that reflects the state-transitions of the underlying MDP and the true geodesic distance between training data-points. We present

two modalities for storing this information iteratively in so-called proximity graphs encountered in dimensionality reduction and surface reconstruction methods from point-cloud sets.

Let $\mathscr{G}(E,V)$ be a proximity graph that is the representation of a manifold upon which the training data-points are located.

We denote the new data-point which needs to be added to the existing graph structure with s_i and an edge between two vertexes s_i and s_j as e_{s_i,s_j}. After the addition of the new data-point to the graph vertex set $V = V \cup \{s_i\}$ new edges need to be constructed which connect it to existing vertexes in $\mathscr{G}$. Existing work [17], [27] on graph edge construction has focused on treating training data as either i.i.d or batch data. Since the approximation algorithm proposed by us operates on-line we present iterative versions of two well-known edge construction methods, the k-nearest-neighbor(KNN) and the extended sphere of influence (ε-SIG) methods.

6.1 Extended Sphere of Influence Graphs

The extended sphere of influence graph (eSIG) produces a good description of non-smooth surfaces and can accommodate variations in the point sample density [17]. In this approach, edges are constructed between vertexes with intersecting spheres of influence:

$$\begin{aligned} &E = E \cup \{e_{s_i,s_j} = \|s_i - s_j\|^2 \quad | \quad (R(s_i) + R(s_j)) > \|s_i - s_j\|\} \\ &R(s_i) = \|s_i - s_k\|^2 \end{aligned} \tag{17}$$

In the above formula $R(s_i)$ denotes the radius of the sphere of influence of s_i and s_k is the k-th nearest neighbor of s_i. The sphere of influence of a graph vertex is the sphere centered at the point, with radius given by its distance to its k-th nearest neighbor. Disconnected components can appear with this construction, however adjusting k the sphere of influence radius can be increased which helps in alleviating this problem.

6.2 KNN Edge Construction

The K-nearest neighbor method eliminates the problem of unconnected sub-graphs but introduces new problems such as : connections between points that are too far in state-space, asymmetric property of the adjacency matrix and increased computational cost. This strategy effectively limits the number of outgoing connections that a node can have by selecting the k nearest neighbors of x_t and creating an edge to them with length equal to their spatial distance.

$$e_{s_i,s_j} = \begin{cases} \|s_i - s_j\|^2 & \text{if} \quad s_j \in knn(s_i) \\ 0 & \text{otherwise} \end{cases} \qquad i = 1 \ldots n \tag{18}$$

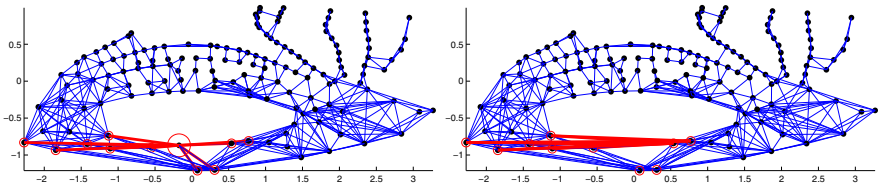


Fig. 2 Dictionary points (*black dots*) and graph structure (*blue lines*) before and after node removal. Figure (a) shows the node to be removed (*red circle*), its neighbours (*red circles with dots*) and the removable edges (*red lines*). On (b) we see the resulting graph with the newly created edges shown in red.

Here we used $knn(s_i)$ to denote the set containing the k vertexes nearest to s_i. The value of k also determines the number of edges present in the graph.

To achieve symmetry in the obtained proximity graph, we use undirected edges. As a consequence, when creating edges between data-points s_i and s_j we update the adjacency matrix as follows: $A^{\mathscr{G}}_{i,j} = A^{\mathscr{G}}_{j,i} = \|s_i - s_j\|^2$.

6.3 Updating the Graph Structure

The deletion of a vertex from the graph $\mathscr{G}(E,V)$ also removes the edges connecting it to its neighbors. In order to keep the information contained in the original edges about the topology we use the following pruning algorithm:

Let us denote the vertex which is to be removed by s^*. After removal of s^* we get a new graph structure $\mathscr{G}'(V',E')$ with the following components:

$$V' = V \setminus \{s^*\} \quad E' = E \setminus \{e(s^*,s) | s \in \text{neig}(s^*)\} \cup E_{new} \tag{19}$$

where $e(s^*,s)$ denotes an edge between the vertexes s^* and s of the graph G(E,V). The set E_{new} contains the new edges between former neighbors of s^* with weights equal to the sum of the edge weights that connected them to s^*:

$$E_{new} = \{e(s_i,s_j) = \|s_i,s^*\| + \|s_j,s^*\| | s_i,s_j \in \text{neig}(s^*), e(s_i,s_j) \notin E\} \tag{20}$$

Figure 2 illustrates the removal procedure on a sample graph structure obtained after interacting for 400 steps with the mountain-car environment.

The addition of the new edges serves two purposes: First the connected property of the graph is maintained, secondly it can be shown that using the previously defined update the shortest path distance matrix of the graph will remain the same except for removing the row and column corresponding to the index of s^*. In consequence the geodesic distances between the remaining graph nodes are preserved.

7 The KSLTD Algorithm with On-Line Laplacian Sparsification

The algorithm described in this section is used to incrementally construct a similarity graph, decide upon the addition of a new data-point to the dictionary and calculate the linear coefficients for the approximation of the value function.

To obtain a representative set of data-points we use the score function defined in Eq. (15) and the iterative update steps of the KLSTD data structures from Eq. (8) and (9). In the KLSTD setting, the target function values $f(s)$ from the laplacian sparsification criterion in Eq. (16) are replaced by the approximated values given by the KLSTD coefficients w^* and the actual composition of the dictionary D, according to Eq. (10). According to this the score function for an individual data-point takes the following form:

$$\mu(s_i) = \sum_{s_j \in neig(s_i)} A^{\mathscr{G}}_{i,j} (\sum_{t=1}^{|D|} w^*_t (k(s_i,s_t) - k(s_j,s_t))^2 \tag{21}$$

Let us denote the upper limit to the size of the dictionary by *maxBV*. Whenever the upper limit has been reached a dictionary point is selected to be discarded, based on the score from Eq. 21. Discarding a data-point has two consequences: (1) the KRLSTD coefficients C and $\tilde{\mathbf{b}}$ and (2) the structure of the graph $\mathscr{G}(E,V)$ need to be updated accordingly. Due to the linear relationship between the approximation coefficients and the KRLSTD parameters, the parameters w^*, the rows and columns of C and the corresponding entries of $\tilde{\mathbf{b}}$ can be arbitrarily permuted. For example if we want to discard data-point s_i from D where $|D| = n$ The following permutation can be performed:

$$\begin{aligned} w^* &= [w^*_1 \dots w^*_i \dots w^*_n]^T \rightarrow [w^*_1 \dots w^*_n \dots w^*_i]^T \\ \tilde{\mathbf{b}} &= \left[\tilde{b}_1 \dots \tilde{b}_i \dots \tilde{b}_n\right]^T \rightarrow \left[\tilde{b}_1 \dots \tilde{b}_n \dots \tilde{b}_i\right]^T \end{aligned}$$

The i-th row and column of C can be exchanged with the last row and column:

$$C = \begin{bmatrix} C_{1,1} \dots C_{1,i} \dots C_{1,n} \\ \vdots \\ C_{i,1} \dots C_{i,i} \dots C_{i,n} \\ \vdots \\ C_{n,1} \dots C_{n,i} \dots C_{n,n} \end{bmatrix} \rightarrow \begin{bmatrix} C_{1,1} \dots C_{1,n} \dots C_{1,i} \\ \vdots \\ C_{n,1} \dots C_{n,n} \dots C_{n,i} \\ \vdots \\ C_{i,1} \dots C_{i,n} \dots C_{i,i} \end{bmatrix} \tag{22}$$

The final step of the update after the above transformations is to simply delete the last row and column of C and the last entry from the approximation coefficient vector w^* and $\tilde{\mathbf{b}}$

Algorithm 1. KRLSTD with Laplacian sparsification

KRLSTD data structures: $\tilde{A}_1 = k(s_1,s_1), C_1 = \frac{1}{k(s_1,s_1)}, \quad \tilde{b}_1 = k(s_1,s_1)\cdot r_1$
Dictionary parameters: $D_1 = \{s_1\}$, maxBV=const, n=1
Graph parameters: $\mathscr{G}(E,V) \quad E = \emptyset \quad V = s_1$
repeat
  get a new training data-point $\{s_t\ , R_t\}$
  $D_t = D_{t-1} \cup \{s_t\}$
  expand proximity graph $\mathscr{G}(E,V)$ Eq. (17),(18)
  update $C_t \to C_{t+1}$, $\tilde{\mathbf{b}}_t \to \tilde{\mathbf{b}}_{t+1}$ Eq. (8),(9)
  calculate new approximation coefficients w_t^* Eq. (10)
  if $|D_t| >= maxBV$ **then**
    calculate scores: $\mu(s_i) \quad i = \overline{1,n} \quad n = |D_t|$ Eq. (21)
    select least significant point $s^* = \arg\min_s(\mu(s)|s \in D_t)$
    eliminate s^* from the dictionary: $D_t = D_t \setminus \{s_r\}$
    update C_{t+1} , $\tilde{\mathbf{b}}_{t+1}$ and w_{t+1}^* Eq. (22),(22)
    update $\mathscr{G}(E,V)$ Eq. (19),(20)
  else
    dictionary remains unchanged
  end if
until approximation coefficients w^* converged
Output: $D \quad w^*$

8 Performance Evaluation

To measure the performance of our policy evaluation framework in a continuous Markov reward process we make use of two quality measures: the Bellman Error (BE) and the mean absolute error with respect to the true value function.

The Bellman error expresses the ability of the approximator to predict the value of the next state based on the current state's approximated value. It also indicates how well the function approximator has converged

$$BE(\tilde{V}(s)) = R(s) + \gamma \int_{s' \in S} P(s'|s,a) \int_{a \in A} \pi(a|s)\tilde{V}(s')dads' - \tilde{V}(s) \tag{23}$$

According to a study in [24] the Bellman error gives an upper bound on the approximation of $\tilde{v}$:

$$\|V - \tilde{V}\|_\infty \leq \frac{\|BE(\tilde{V})\|_\infty}{1-\gamma} \tag{24}$$

The mean absolute error simply compares the accuracy of the approximated value of a set of states to the values given by the true value function. It is defined as the sum of the absolute differences between approximated and true state values:

$$MAE(\tilde{V}(.)) = \int_{s \in S} |\tilde{v}(s) - V(s)|ds \tag{25}$$

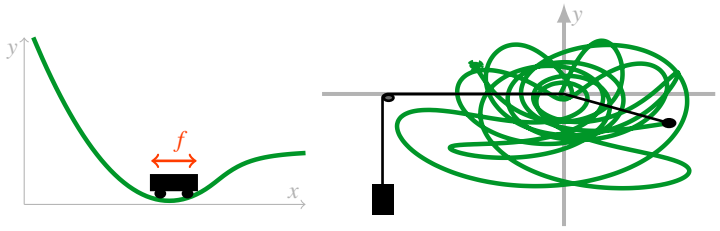


Fig. 3 Illustrations of the (a) car on the hill and (b) swinging Atwood's machine dynamics systems

In our experiments we approximate the above integral by the sample average evaluated over a predefined set of data-points sampled from the state space of the corresponding MRP.

The true value function V used as a baseline in our experiments is calculated in all experimental settings using exhaustive Monte Carlo approximation and can be considered very close to the actual value function induced by the policy π.

To test the performance of our methods on higher dimensional data we choose the problem of learning the state value functions of highly non-linear dynamical systems. In this setting we model the value function of the dynamical system with the help of a KLSTD approximator, where inputs are states and the outputs are the associated values:

$$\tilde{V}_\pi(s_t)\sum_{i=0}^{n} w_i^* k(s_i,s_t) \sim E_\pi\left[\sum_{i=0}^{\infty}\gamma^i r_i | r_i = R(s_i), s_0 = s_t\right] \tag{26}$$

For testing we used two well-known toy problems from reinforcement learning: the Car on hill (a.k.a mountain car) (Fig. 3a) control and the swinging Atwood's machine (Fig. 3b).

8.1 Experimental Setup

To assess the performance of the algorithms we averaged the results of a large number of experiments performed on data generated from the dynamical systems. In each experiment we generated training and test-points, sampled from trajectories[5] where the start states were chosen randomly from within the state-boundaries of the system. For each sampled state-action vector we also collected the corresponding immediate reward, defined by the reward function.

To generate the system inputs we used a fixed Gaussian policy with a linear controller of the following form: $\pi(a,s) \sim N(ctrl(s),\Sigma)$ where $ctrl(s) = \sum_{i=1}^{d} \alpha_i s_i$ is the controller, α_i being the linear coefficients and d the dimensionality of the state vector, $\Sigma = diag(\sigma)$ is the d dimensional diagonal covariance matrix.

[5] A trajectory is a set of successive states starting from an initial state and applying a fixed policy, until a terminal state has been reached.

We also perturbed the transition probabilities of the systems with a Gaussian distributed noise of variance: $\sigma = 0.01$ to simulate real-world conditions. Another important detail of our experimental setup is the initialization of the kernel hyperparameters in case of ALD and the initialization of the nearest neighbor count value k in the graph construction algorithms. The kernel function that we used in our experiments was the Gaussian kernel with added Gaussian noise. The kernel hyperparameters (the characteristic length scale, the noise variance and the signal variance) were optimized using a scaled conjugate gradient optimization to fit the data as well as possible. For setting the nearest neighbor count value k we used the dimensionality of the input data-points as a reference.

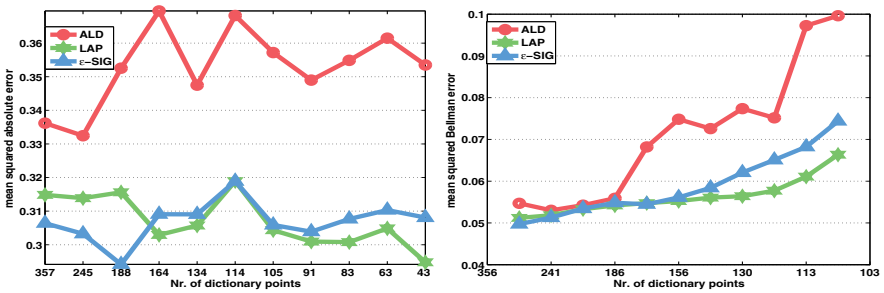
In what follows we describe each of the studied dynamical systems and present the test results.

8.2 Car on Hill Control Problem

The first test system is the well-known mountain-car control problem [22], shown schematically in Fig. 3.a. A car is placed in the a valley and the goal is to apply a sequence forces along the longitudinal axis of the car such that it will climb out the valley. This problem imposes a two dimensional continuous state space $s = \begin{bmatrix} x & v \end{bmatrix}$ composed of the position of the car x and its speed $v = \dot{x}$. The applied action is one dimensional and continuous. To demonstrate the benefits of our Laplacian-based sparsification mechanism for the approximation accuracy we performed 15 experiments each having 150 episodes consisting of 350 steps each – ~ 45000 training-points – for this problem. To see the effects of the sparsification on the approximation accuracy, we performed the same experiments for a number of different maximum dictionary sizes. Figure 4.a shows the mean absolute errors with respect to the true value function.

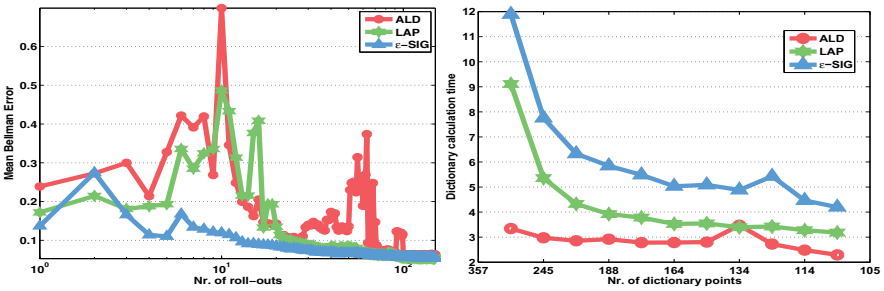
According to our experiments, the Laplacian-based sparsification mechanism leads to a lower approximation error than standard ALD for all maximum dictionary sizes. The approximation accuracy even increases slightly when the dictionary size is drastically reduced as opposed to ALD where having fewer dictionary points raises the approximation error. This may be attributed to the better placement of data-points into regions of the state-space where the target function changes more rapidly.

Figure 4.b shows the evolution of the Bellman error from the same perspective. It can be seen that the Laplacian-based sparsification mechanism with *knn* or $\varepsilon - SIG$ proximity graphs performs better at low dictionary sizes than ALD, the difference becoming more obvious as the dictionary size is further decreased. Figure 5.a illustrates the evolution of the mean Bellman error as a function of the number of training points used for the estimation of the approximation parameters $\mathbf{w}$ from section 4. We calculated the dictionary beforehand based on a fixed training data set using each of the three sparsification methods. When small number of training data is used, the three dictionaries perform similarly. However when the training-data size increases the dictionary obtained by ALD leads to unstable value estimates, whereas using the



(a) (b)

Fig. 4 Evolution of the mean absolute error for different maximum dictionary sizes averaged over 15 experiments, in case of standard *ALD* sparsification, and our Laplacian based sparsification with *knn* and $\varepsilon - SIG$ proximity graphs. The horizontal axis represents the maximum number of dictionary points, the vertical axis on figure (a) shows the mean squared absolute error while on figure (b) the mean squared Bellman error. Measurement of the errors was done on 3600 equally distributed points from the state space.



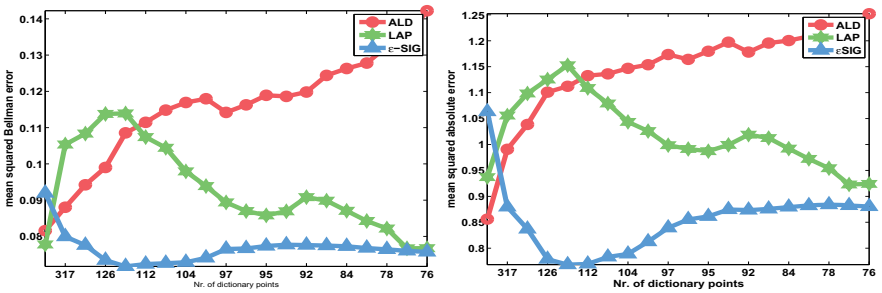
(a) (b)

Fig. 5 (a) Mean Bellman error as a function of training data set size. (b)Time required for the computation of the dictionary from approximately 45000 data-points using ALD and Laplacian-based sparsification with *knn* and $\varepsilon - SIG$ proximity graphs.

dictionary obtained by the Laplacian-based sparsification we obtain more accurate and stable values.

8.3 Swinging Atwood's Machine

The swinging Atwood's machine is composed of two weights connected by a rope and two pulleys, the larger weight can be moved only in the vertical plane, while the smaller weight can rotate freely around the second pulley. A schematic representation of the system with a sample trajectory described by the smaller weight can be seen on fig. 3(b). The goal of the control problem is to guide the vertical force applied to the larger weight in such a manner that the small weight moves roughly along a circle with a given radius. The state representation is four



(a) (b)

Fig. 6 Evolution of the mean squared bellman error (a) and the mean absolute error (b) as a function of maximum dictionary size, for the swinging Atwood's machine control problem. The horizontal axis shows the maximum number of basis vectors allowed while the vertical axis shows the error values. The uneven distribution of the numbers on the horizontal axis is due to the fact that ALD does not put on upper limit on the dictionary size, it is determined by the threshold parameter. Therefore the Laplacian and eSIG algorithms are adjusted to contain the same number of dictionary points as resulted from ALD sparsification

dimensional, $s \stackrel{\text{def}}{=} [l, \dot{l}, \theta, \omega]$, where l is the length of the rope from the second pulley to the smaller weight; $\dot{l}$ is the rate of change in the rope's length; θ is the angle of the smaller weight; ω is the angular velocity. Our implementation of this system is based on a simplified version of the dynamic equations from [14], where a detailed description of the system and its unstable dynamics is also given.

Figure 8.3 illustrates the performance of the three sparsification algorithms (ALD, Laplacian with *knn* (LAP) and Laplacian with extended sphere of influence graphs (eSIG)) as a function of the maximum size of the dictionary. The two sub-figures correspond to the mean squared absolute error and the mean squared Bellman error. Similarly to the case of the car-on-hill control problem, when the maximum size of the dictionary is reduced the two laplacian based sparsification mechanisms produce more accurate estimates both when measured according to the Bellman and the absolute error. In case of the swinging Atwood's machine the extended sphere of influence graph construction produces the best results.

9 Conclusion

We have seen a number of proposed methods for reducing the computational costs of kernel-based value function approximation. As a major contribution of this work we presented a new method that exploits the spatio-temporal relationship between training data-points and the shape of the target function to be approximated. Our method can adjust the density of the dictionary set according to the characteristics of the target function, leading to better approximation accuracy in value function approximation and as a consequence faster convergence rates in value-based reinforcement learning algorithms.

The computational complexity of the presented approximation framework is influenced by the nearest neighbor search in case of graph expansion and the search for vertexes with minimal score in case of the sparsification mechanism. The calculation of the individual scores of each graph vertex $v_i \in V$ has a computational complexity of $O(0)$ since the scores can be stored and updated on-line for each vertex. Compared to the cost of approximate linear independence test our methods are slower as it can be seen from 5(b), but not by orders of magnitude, the difference becomes significant only by large dictionary sizes. The better approximation accuracy and faster convergence rates compensate for the higher computational requirements.

The use of proximity graphs for sparsification enables the extension of our methods with different distance-substitution kernels operating on graphs. An superficial exploration of this subject can be found in [9]. Our method also opens up ways to different exploration strategies like probabilistic road-maps or rapidly exploring random trees and experience replay mechanisms aimed at reducing the number of actual trials needed for learning. The use of proximity graphs for sparsification enables the extension of our methods with different distance-substitution kernels operating on graphs. A superficial exploration of this subject can be found in [9]. Our method also opens up ways to different exploration strategies like probabilistic road-maps or rapidly exploring random trees and experience replay mechanisms aimed at reducing the number of actual trials needed for learning.

The authors acknowledge the support of the Romanian Ministry of Education and Research via grant PN-II-RU-TE-2011-3-0278.

References

1. Boyan, J.A.: Technical update: Least-squares temporal difference learning. Machine Learning 49(2-3), 233–246 (2002)
2. Bradtke, S.J., Barto, A.G., Kaelbling, P.: Linear least-squares algorithms for temporal difference learning. Machine Learning, 22–33 (1996)
3. Csató, L.: Gaussian Processes – Iterative Sparse Approximation. PhD thesis, Neural Computing Research Group (March 2002)
4. Csató, L., Opper, M.: Sparse representation for Gaussian process models. In: Leen, T.K., Dietterich, T.G., Tresp, V. (eds.) NIPS, vol. 13, pp. 444–450. The MIT Press (2001)
5. Csató, L., Opper, M.: Sparse on-line Gaussian Processes. Neural Computation 14(3), 641–669 (2002)
6. Deisenroth, M.P., Rasmussen, C.E.: PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In: Getoor, L., Scheffer, T. (eds.) Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA (June 2011)
7. Deisenroth, M.P., Rasmussen, C.E., Peters, J.: Gaussian process dynamic programming. Neurocomputing 72(7-9), 1508–1524 (2009)
8. Engel, Y., Mannor, S., Meir, R.: The kernel recursive least squares algorithm. IEEE Transactions on Signal Processing 52, 2275–2285 (2003)
9. Jakab, H., Csató, L.: Manifold-based non-parametric learning of action-value functions. In: Verleysen, M. (ed.) European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium, pp. 579–585. UCL, KULeuven (2012)
10. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. J. Mach. Learn. Res. 4, 1107–1149 (2003)

11. Maei, H., Szepesvari, C., Bhatnagar, S., Precup, D., Silver, D., Sutton, R.: Convergent temporal-difference learning with arbitrary smooth function approximation. In: Advances in Neural Information Processing Systems NIPS 22, pp. 1204–1212 (2009)
12. Mahadevan, S., Maggioni, M.: Value function approximation with diffusion wavelets and laplacian eigenfunctions. In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems 18, pp. 843–850. MIT Press, Cambridge (2006)
13. McCullagh, P., Nelder, J.A.: Generalized Linear Models. Chapman & Hall, London (1989)
14. Olivier, P., Perez, J.-P., Simó, C., Simon, S., Weil, J.-A.: Swinging atwood's machine: Experimental and numerical results, and a theoretical study. Physica D 239, 1067–1081 (2010)
15. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York (1994)
16. Riedmiller, M.: Neural fitted q iteration: first experiences with a data efficient neural reinforcement learning method. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 317–328. Springer, Heidelberg (2005)
17. Ruggeri, M.R., Saupe, D.: Isometry-invariant matching of point set surfaces. In: Eurographics Workshop on 3D Object Retrieval (2008)
18. Schölkopf, B., Smola, A.J.: Learning with Kernels. The MIT Press, Cambridge (2002)
19. Seeger, M.W., Kakade, S.M., Foster, D.P.: Information consistency of nonparametric gaussian process methods
20. Sugiyama, M., Hachiya, H., Kashima, H., Morimura, T.: Least absolute policy iteration for robust value function approximation. In: Proceedings of the 2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Piscataway, NJ, USA, pp. 699–704. IEEE Press (2009)
21. Sugiyama, M., Kawanabe, M.: Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation (2012)
22. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
23. Szepesvári, C.: Algorithms for Reinforcement Learning. Morgan & Claypool Publishers (2011)
24. Taylor, G., Parr, R.: Kernelized value function approximation for reinforcement learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, New York, NY, USA, pp. 1017–1024. ACM (2009)
25. Vapnik, V.N.: Statistical learning theory. John Wiley (1997)
26. Vollbrecht, H.: Hierarchic function approximation in kd-q-learning. In: Proc. Fourth Int. Knowledge-Based Intelligent Engineering Systems and Allied Technologies Conf., vol. 2, pp. 466–469 (2000)
27. von Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing 17(4) (2007)
28. Xu, X., Hu, D., Lu, X.: Kernel-based least squares policy iteration for reinforcement learning. IEEE Transactions on Neural Networks, 973–992 (2007)
29. Xu, X., Xie, T., Hu, D., Lu, X.: Kernel least-squares temporal difference learning. International Journal of Information Technology, 55–63 (2005)