Noname manuscript No.
(will be inserted by the editor)

A review of LSPI and improvement KLSPI

Zhiyuan Hu

Received: date / Accepted: date

Abstract TD method is widely used in reinforcement learning problem. In this article we review an important algorithm of TD method – LSPI algorithm. Meanwhile, we discuss some issues of LSPI like improvement variants, regularization, feature selection and numerical stability. The choice of feature is a fundamental problem of LSPI. To solve this problem we discuss the kernelized LSPI algorithm. Using ALD analysis, KLSPI uses an automatically constructed subset of sample dataset to compute kernel, which could keep the generalization ability without huge loss of precision.

Keywords LSPI · LSTD · KLSPI · Kernel Machine

1 Introduction

Reinforcement Learning (RL) attracts many research interests not only in machine learning but also in many other disciplines like finance, robotic or optimal control [7]. RL Approaches give an exciting prospect since they are generic, highly automated and could be model-free [18]. In this paper we focus on the method of value function estimation, which could be split mainly into three classes: (i) dynamic programming (DP) including policy iteration and value iteration, (ii) temporal difference (TD) methods such as TD (λ), Q-learning and (iii) rollout-based Monte Carlo method [8].
This paper mainly focus the least square policy iteration algorithm (LSPI) and its variants, which combine (i) and (ii). DP is the basic algorithm for RL problem but not practical in complex real world system since DP requires the

Zhiyuan Hu
TU Darmstadt
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: zhiyuan.hu@stud.tu-darmstdat.de

state transition model and reward model, which is normally unknown.
TD methods compute value functions, or approximations of these. The unique thing is that it tries to minimize the error of temporal consecutive predictions instead of an overall prediction error [11], which is very similar to dopamine mechanism in animal brain [5]. LSTD methods are an enhancement of original TD methods, using mean squared error instead of gradient descent to minimize the error of gradient. This approach is very time-efficient in comparison to TD methods and it does not require learning rate.
LSPI algorithm integrate LSTD method into Policy Iteration, using LSTD method to linearly approximate action-value function Q(s, a) and using Bellman equation to update policy. In comparison to non-linear approaches, it could directly compute closed form solution and provide convergence guarantees. Meanwhile, in comparison to dynamic programming, it does not limit state-space to be discrete, and, in comparison to TD methods, the sample data could come from an arbitrary policy.
Feature selection is a fundamental problem of LSPI. The performance of algorithm highly depends on the quality of manually designed features. Kernel methods is widely used as a non-parametric method to solve feature engineering problem. KLSPI use kernel methods to avoid feature design and use approximate linear dependency(ALD) analysis to decrease the size of dataset for kernel methods.

2 Problem Statement & Notation

2.1 Markov Decision Process

In this paper we consider the problem as discrete-time *Markov Decision Process*(MDP). An MDP could be represented as a tuple, $\{S, A, R, P, \gamma\}$, where S is the state space, A is the action space, P is the probabilistic transition model, where $P(s'_t|s_t, a_t)$ is the probability from state s_t to state s'_t with taking action a_t at time step t, and R is the reward function $R : S \times A \times S \to \mathbb{R}$, such that $R(s, a, s')$ represents the reward obtained when taking action a in state s and ending up in state s'. The $\gamma \in [0, 1]$ denotes a discount factor, for discounting future rewards. If the sequence of states will end in a terminating state or absorbing state, it would be called as episodic. If the terminating state will be reached in a fixed number length, it could be called as finite-horizon task. In infinite-horizon task the length of an episode is infinite without termination.
Significant property of MDP is that the future state only depends on the current state of system [19], $P(s_{t+1}, r_{t+1}|s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_0, a_0) = P(s_{t+1}, r_{t+1}|s_t, a_t)$. Therefore the general case that the probability distribution of future states is a function of all past states could be simplified as function of last state and action.

2.2 Dynamic Programming, policy iteration and value iteration

When we regard the reinforcement learning as a constrained optimization problem, we could derivative state-value function $V^\pi(s)$ and action-value function $Q^\pi(s,a)$[6][16]. Both of them give information about the quality of a state in order to find an optimal policy. They are formally defined as $V^\pi(s) = \mathbb{E}_\pi[R_T|s_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s]$, $Q_\pi(s,a) = \mathbb{E}_\pi[R_T|s_t = s, a_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s, a_t = a]$. Which leads to Monto-Carlo(MC) Methods [1]. V^π denotes a mean expectation of the future reward over all possible actions a, weighted by the probability distribution of action $\pi(a|s)$. Q^π represents an actual reward for a fixed choice of actions. However, more practically we use the incremental form:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V^\pi(s')] \tag{1}$$

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} \sum_{a'} P(s'|s,a)\pi(a'|s')Q^\pi(s',a'). \tag{2}$$

Notice that in this paper we discuss mostly about case of the deterministic policy. Policy Iteration(PI) and Value Iteration (VI) are the the typical algorithms to solve MDP problem. The sketch of policy iteration is shown in Figure 1. PI uses equation (1) in policy evaluation (PE) step until value function convergence. Policy improvement (PI) step use Bellman Equation $\pi(s) = \arg\max_a Q(s,a)$ to update policy. PI and PE would be alternately executed until the convergence. However, for the dynamic programming (both VI and PI) we need the precise transition model and reward model. Dynamic Programming algorithm works only for discrete states and actions, otherwise works only in limited situation such like LQR (linear quadratic regulator) case or we need manual discretization.

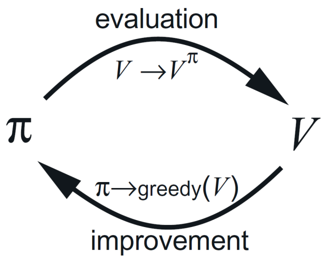

Fig. 1 Policy Iteration algorithm [19].

2.3 LSTD Learning

Temporal Difference(TD) Learning method is more widely used than DP and MC. The model of the environment is not needed and the method could be either *online* or *offline*. TD method aims to minimize the error of estimating value function $MSE(\omega) = \frac{1}{n}\sum_{i=1}^{n}(V_\omega^\pi(s_i) - V^\pi(s_i))^2$. The core idea is that using equation (1) we get approximation of value function as $V^\pi(s_t) \approx E[r_t + \gamma V_\omega^\pi(s_{t+1})]$. Which allows us to compute the approximation error MSE. We use Stochastic Gradient Descent to update the parameter. And the MSE is approximated by using the old approximation to get the target values V^π for a new approximation, this behaviour is called *bootstrapping* (BS). The update rule follows,

$$\begin{aligned}\omega_{t+1} &= \omega_t - \alpha \nabla MSE(\omega_t) \\ &= \omega_t - \alpha [V^{\pi}_{\omega_t}(s_i) - V^{\pi}(s_i)] \nabla_{\omega_t} V_{\omega_t}(s_t) \\ &= \omega_t + \alpha \delta_t \nabla_{\omega_t} V_{\omega_t}(s_t), \end{aligned} \tag{3}$$

where δ_t is the TD error and it is defined as $\delta_t := r_t + \gamma V_{\omega_t}(s_{t+1}) - V_{\omega_t}(s_t)$. To speed up the process, eligibility traces could be used [1]. If we approximate action-value function Q(s, a) instead of state-value function V(s), it leads to Q-Learning and SARSA. The update rule follows,

$$\omega_{t+1} = \omega_t + \alpha \delta_t \nabla_{\omega_t} Q_{\omega_t}(s_t, a_t) \tag{4}$$

$$\delta_t = r_t + \gamma Q_{\omega_t}(s_{t+1}, a_?) - Q_{\omega_t}(s_t, a_t), \tag{5}$$

where $a_?$ could be $\arg\max_a Q(s_{t+1}, a)$ (Q-Learning) or a_{t+1} (SARSA). Those two algorithms are also standard TD learning methods..

2.4 Linear Function Approximation and kernel Methods

For approximation of value function, linear approximations are commonly used. Introducing a feature vector $\phi(s)$, holding linear basis functions, and a parameter vector ω, used to align the functions, the value function can be approximated as $V(s) \approx V_\omega(s) = \omega^T \phi(s)$
Where ω is a set of weights(parameters). We could use batch of data to minimize the TD error to increase data efficiency and least approximation could directly give the closed form solution. Recall the problem $\min_\omega MSE_{BS}(\omega) = \frac{1}{n} \sum_{i=1}^{N} (r(s_i, a_i) + \gamma Q_{\omega_{old}}(s'_i.a'_i) - Q_\omega(s_i, a_i))^2$. We could directly get closed form LSTD solution: $\omega = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}(R + \gamma \mathbf{\Phi}' \omega_{old})$.
Kernel methods is also widely used in linear regression, which supplies a non-parametric methods. LSTD algorithm is actually a linear regression problem. Thus we could use kernel method to solve this problem. According to the Mercer theorem [20], there exists a Hilbert space H and a mapping ϕ from S to H such that $k(s_i, s_j) = \langle \phi(s_i), \phi(s_j) \rangle$. This yield kernel trick, we could directly compute the inner product instead of calculating feature of state. The advantage is obvious, the feature vector $\phi(s)$ could be even infinitely dimensional, but we do not need to know or design feature ϕ.

3 Method

3.1 LSPI Method

We linearly approximate $Q(s, a) = \omega^T \phi(s, a) = \sum_{i=1}^{k} \phi_i(s, a) \omega_i$. Recall that the state-action value function Q^π is generally defined as, $Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} \sum_{a'} P(s'|a, s) \pi(a'|s') Q^\pi(s', a')$. As mentioned, the policy is concerned as deterministic, so we could rewrite it as $Q^\pi = R^\pi + \gamma \mathbf{P}^\pi Q^\pi$, where Q^π and

R^π are vectors of size$|S||A|$ and $\mathbf{P}^\pi$ is the stochastic transition matrix of size $|S||A|\times|S||A|$, which describes the transitions from pair (s, a) to pair $(s', \pi(s'))$. With Linear Approximation, we rewrite the equation of action-value function for a convergence case ($\omega_{old} = \omega$),

$$\mathbf{\Phi}\omega \approx R^\pi + \gamma\mathbf{P}^\pi\mathbf{\Phi}\omega \tag{6}$$

$$(\mathbf{\Phi} - \gamma\mathbf{P}^\pi\mathbf{\Phi})\omega \approx R^\pi, \tag{7}$$

where $\mathbf{\Phi} = (\phi(s_1, a_1), ..., \phi(s, a), ..., \phi(s_{|S|}, a_{|A|}))^T$, $\mathbf{P}^\pi$ and R^π are defined as before. We are interested in a set of weights w that yields a fixed point in value function space, that a value function $Q(s, a) = \omega^T\phi(s, a)$ is invariant under one step of value determination followed by orthogonal projection to the space spanned by the basis functions [12]. Assuming that the columns of $\mathbf{\Phi}$ are linearly independent:

$$\mathbf{\Phi}(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T(R^\pi + \gamma\mathbf{P}^\pi\mathbf{\Phi}\omega) = \mathbf{\Phi}\omega \Rightarrow \underbrace{\mathbf{\Phi}^T(\mathbf{\Phi} - \gamma\mathbf{P}^\pi\mathbf{\Phi})}_{\mathbf{A}}\omega = \underbrace{\mathbf{\Phi}^T R^\pi}_{b} \tag{8}$$

From equation (8) we could construct the form of $\mathbf{A}\omega = b$, where $\mathbf{A} = \mathbf{\Phi}^T(\mathbf{\Phi} - \gamma\mathbf{P}^\pi\mathbf{\Phi}), b = \mathbf{\Phi}^T R^\pi$. The matrix $\mathbf{P}^\pi$ requires the model of system. However, we could use the samples to approximate the term $\mathbf{A}$ and b. $\mathbf{\Phi} = (\phi(s_1, a_1), ..., \phi(s, a), ..., \phi(s_n, a_n))^T, \mathbf{P}\pi\mathbf{\Phi} = (\phi(s'_1, \pi(s'_1)), ..., \phi(s', \pi(s')), ..., \phi(s'_n, \pi(s'_n)))^T, R^\pi = (r(s_1, a_1), ..., r(s, a), ..., r(s_n, a_n)))$. Normally we write it as additive form as is shown in Algorithm 2. This algorithm is called LSTDQ, since it's similar to LSTD. We combined LSTDQ and Policy Iteration, using LSTDQ as policy evaluation and using Equation $\pi(s) = \arg\max_a Q(s, a)$ as policy improvement, we get the algorithm least square policy iteration (LSPI). The algorithm is shown as follows:

Algorithm 1 LSPI
```
Input D, $\phi$, $\gamma$ $\pi_0$
// D: Dataset of samples
// $\phi$: Basis function
// $\pi_0$: initial policy
// $\gamma$: Discount factor
repeat
    update D (optional)
    $\pi \leftarrow \pi'$
    $\pi' \leftarrow$ LSTDQ $(D, \phi, \gamma, \pi)$
until $\pi$ convergence
```

Algorithm 2 LSTDQ
```
Input D, $\phi$, $\gamma$ $\pi$
// input same defined as in Algorithm 1
A ← 0
b ← 0
for each $(s, a, s', r) \in D$ do
    $A \leftarrow A + \phi(s, a)(\phi(s, a) - \gamma\phi(s', \pi(s')))^T$
    b ← b + $\phi(s, a)r$
end for
$\omega^\pi \leftarrow A^{-1}b$
return $\omega^\pi$
```

If there is absorbing states in Markov chains, feature of those states should be all zeros. The update rule of matrix A in algorithm 2 would then be defined as $A_{k+1} := A_k + \phi(s_k, a_k)(\phi(s_k, a_k) - 0)^T$.
The choice of basis function is a fundamental problem. Normally we would use polynomial feature or radial basis function. Recently there are also some successful result on application of random Fourier features into reinforcement

learning problem [17][18].
The numerical stability is also an important issue. In LSTDQ algorithm, we need to compute inverse of matrix A, however, matrix A could not always be full rank. A practical trick is use ridge regression $(A + \lambda I)^{-1}b$, λ is a small number to enhance the numerical stability. It could also be regarded as a regularized regression problem. There are also some other algorithm using regularization to improve LSTD process such like *l2*-regularization, LARS-TD [9][15]. Also we could use some methods to compute the inverse of A such like singular value decomposition(SVD) or using Sherman-Morrison formula to compute inverse matrix implicitly, which yields algorithm LSTDQ-OPT [12].
LSPI gives great flexibility in collection of samples. The sample could be collected from arbitrary policy. But an issue is that the dataset should be sufficiently representative, this is also a general exploration problem of reinforcement learning. LSPI is normally regarded as an *off-line* algorithm, however, on-line version is also possible [13][14], as is shown in Algorithm 1, update the dataset is possible. LSPI is a model free algorithm, but if we have the model, we could use the model to compute the state transition matrix $\mathbf{P}^{\pi}$ instead of approximating $\mathbf{P}^{\pi}\mathbf{\Phi}$, which leads to improvement algorithm LSPI-Model [12].

3.2 KLSPI Method

The design and selection of feature highly influence the performance of learning process. Good basis functions could greatly contribute to the convergence and performance. This is still a fundamental open problem in LSPI. There are many research about improvement of feature selection for LSTD method such like NPDP, KLSPI, OSKTD, GPDP, KBPS, RKRL [2][10]. In this paper we mainly focus on KLSPI algorithm.
If the feature function satisfies Mercer's Condition [20], we can directly compute kernel – inner product of feature. According to Equation (8), we could approximate the action value function as a kernelized form:

$$\begin{aligned} \omega &= A^{-1}b \\ Q(s,a) &= \phi^T(s,a)\omega \\ \Rightarrow Q(x) &= \mathbf{k}^T(x)(\mathbf{K} - \gamma\mathbf{K}_P)R \\ &= \alpha^T\mathbf{k}(x), \end{aligned} \tag{9}$$

where x is the combined feature vector of state-action pair (s, a), $\mathbf{k(x)} = (k(x,x_1)...k(x,x_{|D|}))^T$, $\mathbf{K}_{ij} = k(x_i,x_j) = \langle\phi(x_i),\phi(x_j)\rangle = (\mathbf{\Phi\Phi^T})_{ij}$, $\mathbf{K}_{P,ij} = k(x_i,x'_j)$, note that for this term $\mathbf{K}_p$ is also approximated for deterministic policy case like LSPI. Equation (9) implies that we could directly use the dataset instead of manually design the features and use equation (9) to approximate Q(s, a). Similar to Algorithm 2, we could also derive the incremental form to construct term A and b, $A_{t+1} := A_t + \mathbf{k}_t(x_t)(\mathbf{k}_t(x_t) - \gamma\mathbf{k}_t(x'_t))^T$,

$b_{t+1} := b_t + \mathbf{k}_t(x_t)r_t$.
The problem of this idea is how to decrease the computation cost and keep the sparsity of solutions without the loss of generalization ability, which is caused by the huge size of sample dataset that we could collect thousands of samples or even more. To solve this problem, various methods for sparsifying kernel machines were studied. For instance, support vector regression use structural risk minimization principle and e-intensive cost function [3]. In this paper, we integrate approximate linear dependence (ALD) analysis into LSTDQ algorithm to solve this problem. This idea has been applied in some supervised learning algorithm such like kernel recursive LS (RLS) algorithm [4]. It is easy to extend to RL problem and provides convergence guarantees.
We notice that the core problem is approximate as precise as possible and keeping the size of dataset as small as possible. To solve it, the idea of ALD approach is that, using a data dictionary as subset of whole dataset for kernel calculation, if the feature vector of a new data cannot be approximated with a predefined precision, this data would be added to that dictionary.
Suppose that we have the tested t-1 feature vectors of original dataset and we have already constructed the dictionary $\mathrm{Dic}_{t-1} = \{x_j\}(j = 1...k)$, The ALD condition of a new data x is tested as follows,

$$\delta_t = \min_c \left\| \sum_{j=1}^{|\mathrm{Dic_{t-1}}|} c_j \phi(x_j) - \phi(x_t) \right\|^2 \leq \mu, \tag{10}$$

where μ is a threshold parameter to determine the control accuracy and sparsity level. When μ is appropriately selected, the solution could be guaranteed without sacrificing much in approximation accuracy. We use kernel trick to compute the optimization solution of (10), which is equivalent to,

$$\begin{aligned}\delta_t &= \min_c \left\{ \sum_{i,j} c_i c_j \langle \phi(x_i), \phi(x_j) \rangle - 2 \sum_i c_i \langle \phi(x_i), \phi(x_t) \rangle + \langle \phi(x_t), \phi(x_t) \rangle \right\} \\ &= \min_c \{ c^T \mathbf{K}_{t-1} c - 2c^T \mathbf{k}_{t-1}(x_t) + k(x_t, x_t) \}\end{aligned}$$

Using the kernel trick, it is easily to derive the closed form solution of this optimization problem,

$$c_t = \mathbf{K}_{t-1}^{-1} \mathbf{k}_{t-1}(x_t) \tag{11}$$

$$\delta_t = k(x_t, x_t) - \mathbf{k}_{t-1}^T(x_t) c_t. \tag{12}$$

Equation (11) and (12) give the key step of ALD analysis, if δ_t is smaller than the precision threshold μ, we regard as the data in dictionary is sufficient to represent and we keep the dictionary unchanged. Otherwise we add the new data x_t to dictionary $\mathrm{Dic_t} = \mathrm{Dic_{t-1}} \cup \{x_t\}$.
Based on the previous discussion, we combine ALD analysis and LSPI algorithm to come up KLSPI algorithm. We modifies the LSTDQ algorithm and integrate ALD analysis into LSTDQ framework, we firstly enumerate all the

state-action pair in the dataset and perform ALD analysis to construct the dictionary, then we solve the regression problem $A\alpha = b$ as LSTDQ. The other steps are identical to LSPI algorithm. KLTDQ is described as follows.

Algorithm 3 KLSTDQ

```
Input D, $\mathbf{k}$, $\gamma$ $\pi_0$, $\mu$,
// $\mathbf{k}$, kernel function $\mathbf{k}(\cdot,\cdot)$
// $\mu$ threshold parameter for kernel sparsification
// other input parameters are defined as in Algorithm 1

$\mathrm{Dic}_0 = \emptyset$, t=0
for each $x = (s, a) \in D$ do
    $c_t = K_{t-1}^{-1} k_{t-1}(x)$
    $\delta_t = k(x, x) - k_{t-1}^T(x) c_t$
    if $\delta_t < \mu$ then
        t ← t+1, $\mathrm{Dic}_\mathrm{t} \leftarrow \mathrm{Dic}_{\mathrm{t}-1} \cup \{\mathrm{x}\}$
    end if
end for

for each $(s, a, s', r) \in D, x = (s, a), x' = (s, \pi(s'))$ do
    $A \leftarrow A + k(x)(k(x) - \gamma k(x'))^T$
    b ← b + $k(x)r$
end for
$\alpha^\pi \leftarrow A^{-1} b$
return $\alpha^\pi$
```

Like LSTD, for absorbing states in Markov chain, feature vector should be all zeros and the update rule of matrix A in algorithm 3 should be $A_{k+1} := A_k + \mathbf{k}(x_{k+1})(\mathbf{k}(x_{k+1}) - 0)^T$. And [21] gives the analysis and proof of convergence. KLSPI could converge to a suboptimal policy.
KLSPI is a very efficient non-parametric method among TD mehods. Using ALD analysis, it could significantly decrease the size matrix without loss of generalization ability.

4 Conclusion

LSPI algorithm is a model-free, off-line reinforcement learning algorithm for control problem. LSPI combines the advantages of policy iteration and LSTD methods – the policy search efficiency and data efficiency. It is very practical in control problem. However, there are still some issues to be concerned such like feature selection, numerical stability, data sampling. Feature selection is a fundamental problem among them. There are various research for feature engineering of LSTD. KLSPI uses ALD analysis and kernel method to try to solve the problem of feature selection. The idea is borrowed from sparse kernel machine research. It is a non-parametric, data-efficient approach.

References

1. Boyan, J.A.: Least-squares temporal difference learning. In: ICML, pp. 49–56 (1999)
2. Chen, X., Gao, Y., Wang, R.: Online selective kernel-based temporal difference learning. IEEE transactions on neural networks and learning systems **24**(12), 1944–1956 (2013)
3. Cristianini, N., Shawe-Taylor, J., et al.: An introduction to support vector machines and other kernel-based learning methods. Cambridge university press (2000)
4. Engel, Y., Mannor, S., Meir, R.: The kernel recursive least-squares algorithm. IEEE Transactions on signal processing **52**(8), 2275–2285 (2004)
5. Glimcher, P.W.: Understanding dopamine and reinforcement learning: the dopamine reward prediction error hypothesis. Proceedings of the National Academy of Sciences **108**(Supplement 3), 15647–15654 (2011)
6. Gordon, G.J.: Approximate solutions to markov decision processes. Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE (1999)
7. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of artificial intelligence research **4**, 237–285 (1996)
8. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)
9. Kolter, J.Z., Ng, A.Y.: Regularization and feature selection in least-squares temporal difference learning. In: Proceedings of the 26th annual international conference on machine learning, pp. 521–528. ACM (2009)
10. Kroemer, O.B., Peters, J.R.: A non-parametric approach to dynamic programming. In: Advances in Neural Information Processing Systems, pp. 1719–1727 (2011)
11. Kunz, F.: An introduction to temporal difference learning. In: Seminar on Autonomous Learning Systems (2000)
12. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of machine learning research **4**(Dec), 1107–1149 (2003)
13. Li, L., Littman, M.L., Mansley, C.R.: Online exploration in least-squares policy iteration. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, pp. 733–739. International Foundation for Autonomous Agents and Multiagent Systems (2009)
14. Ma, J., Powell, W.B.: Convergence analysis of on-policy lspi for multi-dimensional continuous state and action-space mdps and extension with orthogonal polynomial approximation. Submitted to SIAM Journal of Control and Optimization (2010)
15. Petrik, M., Taylor, G., Parr, R., Zilberstein, S.: Feature selection using regularization in approximate linear programs for markov decision processes. arXiv preprint arXiv:1005.1860 (2010)
16. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
17. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: Advances in neural information processing systems, pp. 1177–1184 (2008)
18. Rajeswaran, A., Lowrey, K., Todorov, E.V., Kakade, S.M.: Towards generalization and simplicity in continuous control. In: Advances in Neural Information Processing Systems, pp. 6550–6561 (2017)
19. Sutton, R.S., Barto, A.G., et al.: Introduction to reinforcement learning, vol. 135. MIT press Cambridge (1998)
20. Vapnik, V.N.: An overview of statistical learning theory. IEEE transactions on neural networks **10**(5), 988–999 (1999)
21. Xu, X., Hu, D., Lu, X.: Kernel-based least squares policy iteration for reinforcement learning. IEEE Transactions on Neural Networks **18**(4), 973–992 (2007)