Tracking in Reinforcement Learning

Matthieu Geist[1,2,3], Olivier Pietquin[1], and Gabriel Fricout[2]

[1] IMS Research Group, Supélec, Metz, France
[2] MC Cluster, ArcelorMittal Research, Maizières-lès-Metz, France
[3] CORIDA project-team, INRIA Nancy - Grand Est, France

Abstract. Reinforcement learning induces non-stationarity at several levels. Adaptation to non-stationary environments is of course a desired feature of a fair RL algorithm. Yet, even if the environment of the learning agent can be considered as stationary, generalized policy iteration frameworks, because of the interleaving of learning and control, will produce non-stationarity of the evaluated policy and so of its value function. Tracking the optimal solution instead of trying to converge to it is therefore preferable. In this paper, we propose to handle this tracking issue with a Kalman-based temporal difference framework. Complexity and convergence analysis are studied. Empirical investigations of its ability to handle non-stationarity is finally provided.

Keywords: Reinforcement learning, value function approximation, tracking, Kalman filtering.

1 Introduction

Reinforcement learning (RL) [1] is a general paradigm in which an agent learns to control a dynamic system (its *environment*) through examples of real interactions without any model of the physics ruling this system. A feedback signal is observed by this agent after each interaction as a reward information, which is a local hint about the quality of the control. When addressing a reinforcement learning problem, one considers the system as made up of states and accepting actions from the controling agent. The objective of the agent is to learn the mapping from states to actions (a *policy*) that maximizes the expected cumulative reward over the long term, which it locally models as a so-called value or Q-function. Reinforcement learning induces non-stationarity at several levels. First, as in a lot of real-world machine learning applications, adaptation to non-stationary environments is a desired feature of a learning method. Yet most of existing machine learning algorithms assume stationarity of the problem and aim at converging to a fixed solution. Few attempts to handle non-stationarity of the environment in RL can be found in the litterature. Most of them are based on interleaving of RL and planning such as in the Dyna-Q algorithm [1]. Tracking value function is proposed by [2], which can be seen as a specific case of the proposed approach. Second, a large class of RL approaches consists in alternatively learning the value function of a given policy, and then improving the policy according to the learnt values. This is known as *generalized policy iteration* [1]. This scheme suggests to have a value function learner. However, because of the policy improvement phase, the value function changes together with the

C.S. Leung, M. Lee, and J.H. Chan (Eds.): ICONIP 2009, Part I, LNCS 5863, pp. 502–511, 2009.

policy and makes it non-stationary. In both cases, tracking the value function rather than converging to it seems preferable. Other arguments can be discussed on the advantages of tracking *vs* converging even in stationary environments [3]. To address this issue, we propose a statistical approach to value function approximation in RL based on Kalman filtering, namely the *Kalman Temporal Difference* framework. Kalman filtering is indeed an efficient solution to tracking problems and is shown here to apply positively to the problem at sight. Readers are invited to refer to [4] for a deaper theoretical description. Contributions of this paper are the analysis of this framework (computational complexity, bias caused by stochastic transitions and convergence) and a set of experimental results which show its ability to handle non-stationary (for a non-stationary system and in the case of interlacing of control and learning), as well as sample efficiency.

2 Background

Originally, Kalman filtering [5] aims at online tracking the hidden state of a non-stationary dynamic system through indirect observations of this state. The idea behind KTD is to express the problem of value function approximation in RL as a filtering problem. Considering a parametric value function approximator, the parameters are the hidden state to be tracked, the observation being the reward linked to the parameters through a Bellman equation.

2.1 Reinforcement Learning

This paper is placed in the framework of Markov decision process (MDP). An MDP is a tuple $\{S, A, P, R, \gamma\}$, where S is the state space, A the action space, $P : s, a \in S \times A \rightarrow p(.|s, a) \in \mathcal{P}(S)$ a family of transition probabilities, $R : S \times A \times S \rightarrow \mathbb{R}$ the bounded reward function, and γ the discount factor. A policy π associates to each state a probability over actions, $\pi : s \in S \rightarrow \pi(.|s) \in \mathcal{P}(A)$. The value function of a given policy is defined as $V^\pi(s) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi]$ where r_i is the immediate reward observed at time step i, and the expectation is done over all possible trajectories starting in s given the system dynamics and the followed policy. The Q-function allows a supplementary degree of freedom for the first action and is defined as $Q^\pi(s, a) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi]$. RL aims at finding (through interactions) the policy π^* which maximises the value function for every state: $\pi^* = \text{argmax}_\pi(V^\pi)$. Two schemes among others can lead to the optimal policy. First, *policy iteration* implies learning the value function of a given policy and then improving the policy, the new one being greedy respectively to the learned value function. It requires solving the *Bellman evaluation equation*, which is given here for the value and Q-functions:

$$V^\pi(s) = E_{s',a|\pi,s}[R(s, a, s') + \gamma V^\pi(s')], \forall s \tag{1}$$

$$Q^\pi(s, a) = E_{s',a'|\pi,s,a}[R(s, a, s') + \gamma Q^\pi(s', a')], \forall s, a \tag{2}$$

The second scheme, *value iteration*, aims directly at finding the optimal policy. It requires solving the *Bellman optimality equation*:

$$Q^*(s, a) = E_{s'|s,a}[R(s, a, s') + \gamma \max_{b \in A} Q^*(s', b)], \forall s, a \tag{3}$$

2.2 Kalman Temporal Differences

For the sake of generality, the following notations are adopted, given that the aim is respectively the value or the Q-function evaluation or the Q-function optimization:

$$t_i = \begin{cases} (s_i, s_{i+1}) \\ (s_i, a_i, s_{i+1}, a_{i+1}) \\ (s_i, a_i, s_{i+1}) \end{cases} \quad \text{and } g_{t_i}(\theta_i) = \begin{cases} \hat{V}_{\theta_i}(s_i) - \gamma \hat{V}_{\theta_i}(s_{i+1}) \\ \hat{Q}_{\theta_i}(s_i, a_i) - \gamma \hat{Q}_{\theta_i}(s_{i+1}, a_{i+1}) \\ \hat{Q}_{\theta_i}(s_i, a_i) - \gamma \max_b \hat{Q}_{\theta_i}(s_{i+1}, b) \end{cases} \tag{4}$$

where $\hat{V}_\theta$ (resp. $\hat{Q}_\theta$) is a parametric representation of the value (resp. Q-) function and θ is the parameter vector. A statistical point of view is adopted and the problem at sight is stated in a so-called *state-space formulation* (that is the value function approximation problem is cast into the Kalman filtering paradigm):

$$\begin{cases} \theta_i = \theta_{i-1} + v_i & \text{(evolution equation)} \\ r_i = g_{t_i}(\theta_i) + n_i & \text{(observation equation)} \end{cases} \tag{5}$$

The first equation (*evolution equation*), is the key of non-stationarity handling. It specifies that the parameter vector evolves with time according to a random walk. The random walk model is chosen for its simplicity. Expectation of θ_i corresponds to the optimal estimation of the value function at time step i. The evolution noise v_i is centered, white, independent and of variance matrix P_{v_i} (to be chosen by the practitioner). The second equation (*observation equation*) links the observed transition to the value (or Q-) function through one of the Bellman equations. The observation noise n_i is supposed centered, white, independent and of variance P_{n_i} (also to be chosen by the practitioner). Notice that this white noise assumption does not hold for stochastic MDP (see Sec. 3.2). KTD is a second order algorithm (and thus sample efficient): it updates the mean parameter vector, but also the associated variance matrix. It breaks down in three steps (see also algorithm 1). First, the *prediction* step (i) consists in predicting the parameters mean and covariance at time step i according to the evolution equation and using previous estimates. Then some *statistics of interest* are computed (ii). Third, the *correction* step (iii) consists in correcting first and second order moments of the parameter random vector according to the Kalman gain K_i (obtained thanks to statistics computed at step (ii)), the predicted reward $\hat{r}_{i|i-1}$ and the observed reward r_i (the difference between the two being a form of temporal difference error). The statistics of interest are generally not analytically computable, except in the linear case, which does not hold for nonlinear parameterization and for the Bellman optimality equation, because of the $\max$ operator. Yet, a derivative-free approximation scheme, the *unscented transform* (UT) [6], is used to estimate first and second order moments of nonlinearly mapped random vectors. Let X be a random vector of size n and $Y = f(X)$ its nonlinear mapping. A set of $2n+1$ so-called sigma-points is computed as follows:

$$\begin{cases} x^{(0)} = \bar{X} & w_0 = \frac{\kappa}{n+\kappa}, \quad j = 0 \\ x^{(j)} = \bar{X} + (\sqrt{(n+\kappa)P_X})_j & w_j = \frac{1}{2(n+\kappa)}, \; 1 \leq j \leq n \\ x^{(j)} = \bar{X} - (\sqrt{(n+\kappa)P_X})_{n-j} & w_j = \frac{1}{2(n+\kappa)}, \; n+1 \leq j \leq 2n \end{cases} \tag{6}$$

where $\bar{X}$ is the mean of X, P_X is its variance matrix, κ is a scaling factor which controls the accuracy [6], and $(\sqrt{P_X})_j$ is the j^{th} column of the Cholesky decomposition of P_X. Then the image through the mapping f is computed for each of these sigma-points:

$$y^{(j)} = f(x^{(j)}), \quad 0 \leq j \leq 2n \tag{7}$$

The set of sigma-points and their images can then be used to compute the following approximations:

$$\begin{cases} \bar{Y} \approx \bar{y} = \sum_{j=0}^{2n} w_j y^{(j)} \\ P_Y \approx \sum_{j=0}^{2n} w_j (y^{(j)} - \bar{y})(y^{(j)} - \bar{y})^T \\ P_{XY} \approx \sum_{j=0}^{2n} w_j (x^{(j)} - \bar{X})(y^{(j)} - \bar{y})^T \end{cases} \tag{8}$$

Using the UT practical algorithms can be derived. At time-step i, a set of sigma-points is computed from predicted random parameters characterized by mean $\hat{\theta}_{i|i-1}$ and variance $P_{i|i-1}$. Predicted rewards are then computed as images of these sigma-points using one of the observation functions (4). Then sigma-points and their images are used to compute statistics of interest. This gives rise to three algorithms, namely KTD-V, KTD-SARSA and KTD-Q, given that the aim is to evaluate the value or Q-function of a given policy or directly the optimal Q-function. They are summarized in Algorithm 1, p being the number of parameters.

3 KTD Analysis

3.1 Computational Cost

The UT involves a Cholesky decomposition, which can be perfomed in $O(p^2)$ instead of $O(p^3)$ when done with a square-root approach [7]. The different algorithms imply to evaluate $2p + 1$ times the g_{t_i} function at each time-step. For KTD-V or KTD-SARSA and a general parameterization, each evaluation is bounded by $O(p)$. For KTD-Q, the maximum over actions has to be computed. Let $\mathcal{A}$ be the cardinality of action space if finite, the computational complexity of the algorithm used to search the maximum otherwise (*e.g.*, the number of samples for Monte Carlo). Then each evaluation is bounded by $O(p\mathcal{A})$. The rest of operations is basic linear algebra, and is bounded by $O(p^2)$. Thus the global computational complexity (per iteration) of KTD-V and KTD-SARSA is $O(p^2)$, and KTD-Q is in $O(\mathcal{A}p^2)$. This is comparable to approaches such as LSTD [8] (nevertheless with the additional ability to handle nonlinear parameterization).

3.2 Stochastic MDP

The KTD framework assumes a white observation noise. In the case of deterministic MDP, this observation noise only models the inductive bias introduced by function approximation. But in stochastic MDP, this noise includes the stochasticity of transitions as well and cannot be considered white anymore. Similarly to other second

Algorithm 1. KTD-V, KTD-SARSA and KTD-Q

Initialization;
priors $\hat{\theta}_{0|0}$ and $P_{0|0}$;
for $i \quad 1, 2, \ldots$ **do**

Observe transition $t_i = \begin{cases} (s_i, s_{i+1}) \text{ (KTD-V)} \\ (s_i, a_i, s_{i+1}, a_{i+1}) \text{ (KTD-SARSA)} \\ (s_i, a_i, s_{i+1}) \text{ (KTD-Q)} \end{cases}$ and reward r_i ;

Prediction Step;
$\hat{\theta}_{i|i-1} = \hat{\theta}_{i-1|i-1}$;
$P_{i|i-1} = P_{i-1|i-1} + P_{v_i}$;

Sigma-points computation ;
$\Theta_{i|i-1} = \left\{ \hat{\theta}^{(j)}_{i|i-1}, \quad 0 \quad j \quad 2p \right\}$ (using the UT, from $\hat{\theta}_{i|i-1}$ and $P_{i|i-1}$);
$\mathcal{W} = \quad w_j, \quad 0 \quad j \quad 2p \quad ;$
$\mathcal{R}_{i|i-1} =$
$\begin{cases} \left\{ \hat{r}^{(j)}_{i|i-1} = \hat{V}_{\hat{\theta}^{(j)}_{i|i-1}}(s_i) \quad \gamma \hat{V}_{\hat{\theta}^{(j)}_{i|i-1}}(s_{i+1}), \, 0 \quad j \quad 2p \right\} \text{ (KTD-V)} \\ \left\{ \hat{r}^{(j)}_{i|i-1} = \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_i, a_i) \quad \gamma \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_{i+1}, a_{i+1}), \, 0 \quad j \quad 2p \right\} \text{ (KTD-SARSA)} \quad ; \\ \left\{ \hat{r}^{(j)}_{i|i-1} = \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_i, a_i) \quad \gamma \max_b \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_{i+1}, b), \, 0 \quad j \quad 2p \right\} \text{ (KTD-Q)} \end{cases}$

Compute statistics of interest;
$\hat{r}_{i|i-1} = \sum_{j=0}^{2p} w_j \hat{r}^{(j)}_{i|i-1}$;
$P_{\theta r_i} = \sum_{j=0}^{2p} w_j (\hat{\theta}^{(j)}_{i|i-1} \quad \hat{\theta}_{i|i-1})(\hat{r}^{(j)}_{i|i-1} \quad \hat{r}_{i|i-1})$;
$P_{r_i} = \sum_{j=0}^{2p} w_j \left(\hat{r}^{(j)}_{i|i-1} \quad \hat{r}_{i|i-1} \right)^2 + P_{n_i}$;

Correction step;
$K_i = P_{\theta r_i} P_{r_i}^{-1}$;
$\hat{\theta}_{i|i} = \hat{\theta}_{i|i-1} + K_i \left(r_i \quad \hat{r}_{i|i-1} \right)$;
$P_{i|i} = P_{i|i-1} \quad K_i P_{r_i} K_i^T$;

order approaches, such as residual algorithms [9], the cost function minimized by KTD is thus biased. For the value function evaluation (extension to other cases is straightforward), the bias is:

$$\|K_i\|^2 E[\underset{s'|s_i}{\text{cov}}(r_i + \gamma V_\theta(s'))|r_{1:i-1}] \tag{9}$$

where K_i is the Kalman gain, the covariance depends on transition probabilities and the expectation is over parameters conditioned on past observed rewards. Proof, although not tricky, is not given here due to a lack of space. This bias, which is zero for deterministic transitions, is similar to the one arising from the minimization of a square Bellman residual. It favorises smooth value functions [10] and acts as a regularization effet, but cannot be controlled.

3.3 Convergence Analysis

Theorem 1. *Assume that posterior and noise distributions are Gaussian, and that the prior is flat (uniform distribution). Then:*

$$\hat{\theta}_{i|i} = \underset{\theta}{\operatorname{argmin}} \sum_{j=1}^{i} \frac{1}{P_{n_j}} (r_j - g_{t_j}(\theta))^2 \tag{10}$$

Proof. KTD is a special form of a Sigma-Point Kalman Filter (SPKF) with a random walk evolution model. It is shown in [7, Ch. 4.5] that under the hypothesis of Gaussian posterior and noises, such a filter produces a *maximum a posteriori* (MAP) estimator. Thus, for KTD, $\hat{\theta}_{i|i} = \hat{\theta}_i^{\text{MAP}} = \operatorname{argmax}_\theta p(\theta|r_{1:i})$. Using the Bayes rule, the posterior can be rewritten as the normalized product of likelihood and prior: $p(\theta|r_{1:i}) = \frac{p(r_{1:i}|\theta)p(\theta)}{p(r_{1:i})}$. The prior is assumed flat, and the denominator does not depend on parameters, so MAP resumes to maximum likelihood. Moreover, the noise being white, the joint likelihood is the product of local likelihoods: $\hat{\theta}_{i|i} = \operatorname{argmax}_\theta p(r_{1:i}|\theta) = \operatorname{argmax}_\theta \prod_{j=1}^{i} p(r_j|\theta)$. As the noise is assumed Gaussian, $r_j|\theta \sim \mathcal{N}(g_{t_j}(\theta), P_{n_j})$, and maximizing a product of likelihood is equivalent to minimizing the sum of their negative logarithms: $\hat{\theta}_{i|i} = -\operatorname{argmin}_\theta \sum_{j=1}^{i} \ln(p(r_j|\theta)) = \operatorname{argmin}_\theta \sum_{j=1}^{i} \frac{1}{P_{n_j}} (r_j - g_{t_j}(\theta))^2$. □

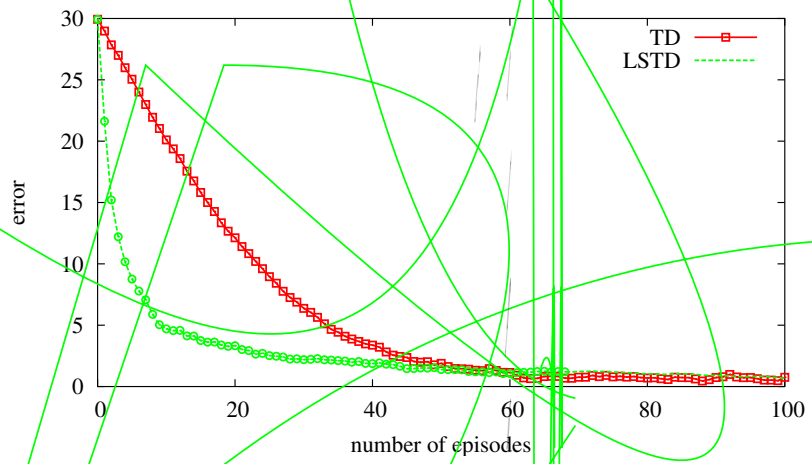
The form of the minimized cost function strengthen the parallel drawn in Sec. 3.2 between KTD and square Bellman residual minimization. It can also be shown (see again [7, Ch. 4.5.]) that a SPKF (and thus KTD) update is actually an online form of a modified Gauss-Newton method, which is a variant of natural gradient descent. In this case, the Fisher information matrix is $P_{i|i}^{-1}$. Natural gradient approach has been shown to be quite efficient for direct policy search [11] and actor-critic [12], so it lets envision good empirical results for KTD. This may be considered as the first RL value (and Q-) function approximation algorithm (in a pure critic sense) involving natural gradient.

4 Experiments

4.1 Boyan Chain

The first experiment is the Boyan chain [13]. The aim is to illustrate the bias caused by stochastic transitions and to show sample-efficiency and tracking ability of KTD-V on a deterministic version of this experiment.

Stochastic Case. The Boyan chain is a 13-state Markov chain where state s^0 is an absorbing state, s^1 transits to s^0 with probability 1 and a reward of -2, and s^i transits to either s^{i-1} or s^{i-2}, $2 \leq i \leq 12$, each with probability 0.5 and reward -3. In this experiment, KTD-V is compared to TD [1] and LSTD [8]. The feature vectors $\phi(s)$ for states s^{12}, s^8, s^4 and s^0 are respectively $[1,0,0,0]^T$, $[0,1,0,0]^T$, $[0,0,1,0]^T$ and $[0,0,0,1]^T$. The feature vectors for other states are obtained by linear interpolation. The approximated value function is thus $\hat{V}_\theta(s) = \theta^T \phi(s)$. The optimal value function is linear in these features, and $\theta^* = [-24, -16, -8, 0]^T$. The error measure is $\|\theta - \theta^*\|$.


30
25
20
15
10
5
0
error
0
20
40
60
80
100
number of episodes
TD
LSTD

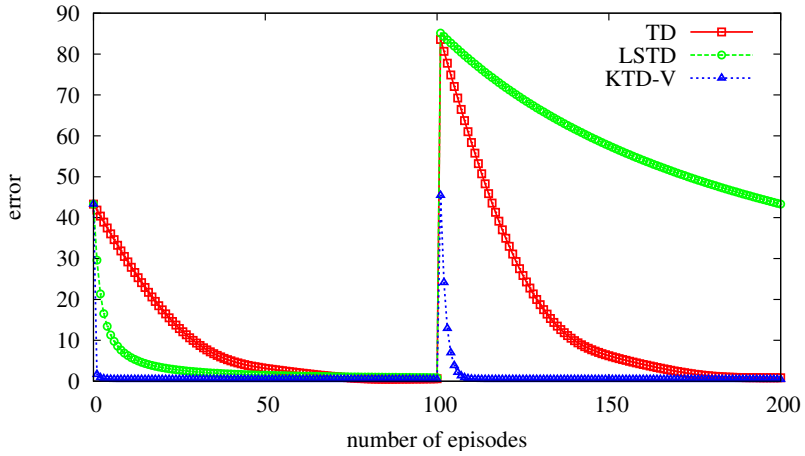


Fig. 2. Boyan Chain: deterministic and non-stationary case

learning while controlling induces non-stationary value dynamics. This task consists in driving an underpowered car up a steep mountain road, the gravity being stronger than the car engine. The discount factor is set to 0.95. State is normalized, and the parameterization is composed of a constant term and a set of 9 equispaced Gaussian kernels (centered in $\{0, 0.5, 1\} \times \{0, 0.5, 1\}$ and with a standard deviation of 0.5) for each action. This experiment compares SARSA with Q-function approximation, LSTD and KTD-SARSA within an optimistic policy iteration scheme. The followed policy is ε-greedy, with $\varepsilon = 0.1$. For SARSA, the learning rate is set to $\alpha = 0.1$. For LSTD the prior is set to $P_{0|0} = 10I$. For KTD-SARSA, the same prior is used, and the noise variances are set to $P_{n_i} = 1$ and $P_{v_i} = 0.05I$. For all algorithms the initial parameter vector is set to zero. Each episode starts in a random position and velocity uniformly sampled from the given bounds. A maximum of 1500 steps per episode is allowed. For each trial, learning is done for 200 episodes, and Fig. 3 shows the length of each learning episode averaged over 300 trials. KTD-SARSA performs better than LSTD, which performs better than SARSA with Q-function approximation. Better results have perhaps been reported for SARSA with tile-coding parameterization in the literature, however the chosen parameterization is rather crude and involves much less parameters. Moreover, even with tile-coding and optimized parameters, SARSA with function approximation is reported to take about 100 episodes to reach an optimal policy [1, Ch. 8.4.], which is about an order of magnitude higher than KTD. The optimistic policy iteration scheme used in this experiment implies non-stationarity for the learned Q-function, which explains that LSTD fails to learn a near-optimal policy. This is confirmed by [2]. LSTD has been extended to LSPI [14], which allows searching an optimal control more efficiently, however it is a batch algorithm which does not imply to learn while controlling, so it is not considered here. KTD-SARSA performs well, and learns a near-optimal policy after only a few tens of steps. Learning is also more stable with it.

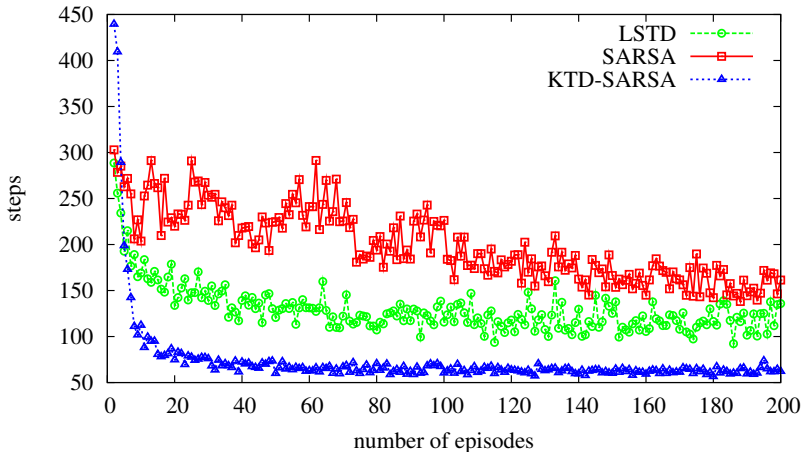


Fig. 3. Mountain car

5 Conclusion

In this paper we proposed the use of a stochastic framework (the Kalman Temporal Differences framework) for value function approximation to handle non-stationartity in RL. Its ability to handle non-stationarity has been shown experimentally (both for non-stationary system and for the case of interlaced learning and control) as well as its sample efficiency. KTD minimizes a square Bellman residual in a natural-gradient descent-like scheme which links it to other promising approaches. Computational (and memory) cost is quadratic. The KTD framework compares favorably to state-of-the-art algorithms, however there is still room for improvement. First, the resulting estimator is biased for stochastic transitions. It would be a great advantage to handle stochastic MDP, and [10,15,16] are interesting leads. When using KTD, the practitioner has to choose a prior, a process noise and an observation noise, which are domain-dependent; there exists a vast literature on adaptive filtering for traditional Kalman filtering, and adaptation to KTD is possible. Using a KTD-based function evaluation in an actor-critic architecture can also be envisioned. For exemple, incremental natural actor-critic algorithms are presented in [17]. TD is used as the actor part instead of LSTD, mostly because of the inability of the latter one to handle non-stationarity. In this case, we argue that KTD is an interesting alternative for the critic part. Finally, as this framework can handle nonlinear parameterization, it can be of interest to combine it with neural networks or basis adaptation schemes. A kernel-based nonlinear parameterization is used in [18].

Acknowledgements

Olivier Pietquin thanks the European Community (FP7/2007-2013, grant agreement 216594, CLASSiC project : www.classic-project.org) and the Région Lorraine for financial support.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1996)
2. Phua, C.W., Fitch, R.: Tracking Value Function Dynamics to Improve Reinforcement Learning with Piecewise Linear Function Approximation. In: International Conference on Machine Learning, ICML 2007 (2007)
3. Sutton, R.S., Koop, A., Silver, D.: On the role of tracking in stationary environments. In: Proceedings of the 24th international conference on Machine learning, pp. 871–878 (2007)
4. Geist, M., Pietquin, O., Fricout, G.: Kalman Temporal Differences: the deterministic case. In: Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2009), Nashville, TN, USA (April 2009)
5. Kalman, R.E.: A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME–Journal of Basic Engineering 82(Series D), 35–45 (1960)
6. Julier, S.J., Uhlmann, J.K.: Unscented filtering and nonlinear estimation. Proceedings of the IEEE 92(3), 401–422 (2004)
7. van der Merwe, R.: Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models. PhD thesis, Oregon Health&Science University, Portland, USA (2004)
8. Bradtke, S.J., Barto, A.G.: Linear Least-Squares Algorithms for Temporal Difference Learning. Machine Learning 22(1-3), 33–57 (1996)
9. Baird, L.C.: Residual Algorithms: Reinforcement Learning with Function Approximation. In: Proceedings of the International Conference on Machine Learning, pp. 30–37 (1995)
10. Antos, A., Szepesvári, C., Munos, R.: Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. Machine Learning 71(1), 89–129 (2008)
11. Kakade, S.: A natural policy gradient. In: Advances in Neural Information Processing Systems 14 (NIPS 2001), Vancouver, British Columbia, Canada, pp. 1531–1538 (2001)
12. Peters, J., Vijayakumar, S., Schaal, S.: Natural actor-critic. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 280–291. Springer, Heidelberg (2005)
13. Boyan, J.A.: Technical Update: Least-Squares Temporal Difference Learning. Machine Learning 49(2-3), 233–246 (1999)
14. Lagoudakis, M.G., Parr, R.: Least-Squares Policy Iteration. Journal of Machine Learning Research 4, 1107–1149 (2003)
15. Jo, S., Kim, S.W.: Consistent Normalized Least Mean Square Filtering with Noisy Data Matrix. IEEE Transactions on Signal Processing 53(6), 2112–2123 (2005)
16. Engel, Y., Mannor, S., Meir, R.: Reinforcement Learning with Gaussian Processes. In: Proceedings of Internation Conference on Machine Learning, ICML 2005 (2005)
17. Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Incremental Natural Actor-Critic Algorithms. In: Advances in Neural Information Processing Systems, Vancouver, vol. 21 (2008)
18. Geist, M., Pietquin, O., Fricout, G.: Bayesian Reward Filtering. In: Girgin, S., Loth, M., Munos, R., Preux, P., Ryabko, D. (eds.) EWRL 2008. LNCS (LNAI), vol. 5323, pp. 96–109. Springer, Heidelberg (2008)