CLASSiC

1.2: New frameworks for generalization, with implementation in DIPPER ISU DM system

Olivier Pietquin, Hervé Frezza-Buet, Paul Crook

Distribution: Public

CLASSiC
Computational Learning in Adaptive Systems for Spoken Conversation
216594 Deliverable 1.2

October 2009

Project funded by the European Community
under the Seventh Framework Programme for
Research and Technological Development

The deliverable identification sheet is to be found on the reverse of this page.

Project ref. no.	216594
Project acronym	CLASSiC
Project full title	Computational Learning in Adaptive Systems for Spoken Conversation
Instrument	STREP
Thematic Priority	Cognitive Systems, Interaction, and Robotics
Start date / duration	01 March 2008 / 36 Months

Security	Public
Contractual date of delivery	M18 = August 2009
Actual date of delivery	October 2009
Deliverable number	1.2
Deliverable title	1.2: New frameworks for generalization, with implementation in DIPPER ISU DM system
Type	Prototype
Status & version	final 1.0
Number of pages	34 (excluding front matter)
Contributing WP	1
WP/Task responsible	SUPELEC
Other contributors	EDIN, HWU
Author(s)	Olivier Pietquin, Hervé Frezza-Buet, Paul Crook
EC Project Officer	Philippe Gelin
Keywords	Reinforcement Learning, generalization

The partners in CLASSiC are:

Heriot-Watt University	HWU
University of Cambridge	UCAM
University of Geneva	GENE
Ecole Superieure d'Electricite	SUPELEC
France Telecom/ Orange Labs	FT
University of Edinburgh HCRC	EDIN

For copies of reports, updates on project activities and other CLASSIC-related information, contact:

The CLASSIC Project Co-ordinator:
Dr. Oliver Lemon
School of Mathematical and Computer Sciences (MACS)
Heriot-Watt University
Edinburgh
EH14 4AS
United Kingdom
O.Lemon@hw.ac.uk
Phone +44 (131) 451 3782 - Fax +44 (0)131 451 3327

Copies of reports and other material can also be accessed via the project's administration homepage, `http://www.classic-project.org`

Contents

Executive summary

This document explains the work done in the framework of task 1.2 and the insights of the deliverable D1.2. This deliverable is a prototype of a new generalization framework for reinforcement learning. A new class of reinforcement learning algorithms based on Kalman filtering has been developed. One goal of the CLASSiC project was to improve state-of-the-art machine learning algorithms and this deliverable contributes to the achievement of this goal. The theory is first explained. Some experimental results are then described. The prototype is developed in C++ and has been interfaced with the DIPPER dialogue manager designed by the University of Edinburgh and written in JAVA.
Publications arising form this work are :

- Matthieu Geist, Olivier Pietquin, and Gabriel Fricout. From supervised to reinforcement learning: a kernel-based bayesian filtering framework. *International Journal On Advances in Software*, 2(1), 2009. accepted for publication.
- Matthieu Geist, Olivier Pietquin, and Gabriel Fricout. Tracking in reinforcement learning. In *Proceedings of the 16th International Conference on Neural Information Processing (ICONIP 2009)*, page 8 pages, Bangkok (Thailand), December 2009. Springer. to appear.
- Matthieu Geist, Olivier Pietquin, and Gabriel Fricout. Kalman temporal differences: the deterministic case. In *IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2009)*, pages 185–192, Nashville, TN, USA, April 2009.
- Matthieu Geist, Olivier Pietquin, and Gabriel Fricout. Différences temporelles de kalman. In *actes des Journées Francophones de Planification, Décision et Apprentissage pour la conduite de systèmes (JFPDA 2009)*, page 20 pages, Paris, June 2009.
- Matthieu Geist, Olivier Pietquin, and Gabriel Fricout. Différences temporelles de kalman : le cas stochastique. In *actes des Journées Francophones de Planification, Décision et Apprentissage pour la conduite de systèmes (JFPDA 2009)*, Paris, 2009.

Suplec organized a French national event on machine learning and artificial intelligence for optimal control and planning in 2008 (JFPDA : http://www.metz.supelec.fr/JFPDA2008/) where an earlier version of this work was presented.

Chapter 1

Theory

1.1 Abstract

One main drawback of Reinforcement Learning (RL) is that they don't easily scale up to real world problems. Particularly, in the field of spoken dialogue systems, the state space can be really huge and especially when POMDP versions are considered. Generalization schemes have therefore to be investigated because the state space can not be explored completely during simulation (and of course in a reasonable number of real interactions). This contribution deals with value function and Q-function approximation in Markovian decision processes. A general statistical framework based on the Kalman filtering paradigm is introduced. Its principle is to adopt a parametric representation of the value function, to model the associated parameter vector as a random variable and to minimize the mean-squared error of the parameters conditioned on past observed transitions. From this general framework, which will be called *Kalman Temporal Differences* (KTD), and using an approximation scheme called the unscented transform, a family of algorithms is derived, namely KTD-V, KTD-SARSA and KTD-Q, which aim respectively at estimating the value function of a given policy, the Q-function of a given policy and the optimal Q-function. The proposed approach holds for linear and nonlinear parameterization. This framework is discussed and experimentally tested on standard RL algorithms. It has been implemented in a C++ library freely available in an open source format that has been interfaced with the DIPPER dialogue management system (written in JAVA).

1.2 Introduction

This contribution deals with value function and Q-function approximation in deterministic Markovian Decision Process (MDP). An MDP is a tuple $\{S, A, T, R, \gamma\}$, where S is the state space, A the action space, $T : S \times A \rightarrow S$ the deterministic transition function, $R : S \times A \times S \rightarrow \mathbb{R}$ the bounded reward function, and γ the discount factor. A policy π is a (here deterministic) mapping from states to actions, $\pi : S \rightarrow A$. The value function of a given policy is classically defined as:

$$V^{\pi}(s) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi] \tag{1.1}$$

where r_i is the reward observed at time i. The Q-function is defined as:

$$Q^{\pi}(s,a) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi] \tag{1.2}$$

Reinforcement learning (RL) [1] aims at finding (through interaction) the policy π^* which maximises the value function for every state:

$$\pi^* = arg\max_{\pi}(V^{\pi}) \tag{1.3}$$

Two schemes (among others) can lead to the solution. First, policy iteration involves learning the value function of a given policy, and then improve the policy, the new one being greedy respectively to the learned value function. It involves solving the *Bellman evaluation equation*, which is given here for the value function and the Q-function, respectively:

$$V^{\pi}(s) = R(s,\pi(s),s') + \gamma V^{\pi}(s'), \forall s \tag{1.4}$$

$$Q^{\pi}(s,a) = R(s,a,s') + \gamma Q^{\pi}(s',\pi(s')), \forall s,a \tag{1.5}$$

Here and through the rest of the chapter, s' denotes the transiting state, that is $s' = T(s,\pi(s))$ or $s' = T(s,a)$, depending on the context. The second scheme, called value iteration, aims directly at finding the optimal policy. It involves solving the *Bellman optimality equation*, which is given here for the Q-function:

$$Q^*(s,a) = R(s,a,s') + \gamma \max_{b \in A} Q^*(s',b), \forall s,a \tag{1.6}$$

The aim of this contribution is to find an approximate solution of the Bellman evaluation or optimality equations, for the value function or the Q-function, when the state or action spaces are too large for classical dynamic programming or reinforcement learning algorithms [2]. Moreover the proposed algorithm is online, so as to keep this important RL feature.

To do so, Temporal Differences (TD) algorithms will be considered. They form a class of methods which consist in correcting the representation of the value (or Q-) function according to the TD error made on it. Most of them can be generically written as:

$$\theta_{i+1} = \theta_i + K_i \delta_i \tag{1.7}$$

In this expression, θ_i is the current representation of the value function, θ_{i+1} is an updated representation given an observed transition, δ_i is the so-called temporal difference error, and K_i is a gain indicating in which direction the representation of the value function should be corrected. Each of these terms will now be discussed.

If the state space S and the action space A are finite and small enough, an exact description of the value function is possible, and θ will be a vector with as many components as the state (-action) space (tabular representation). If these spaces are too large, approximation is necessary. A classical choice in RL is the linear parameterization, that is the value function is approximated by:

$$\hat{V}_{\theta}(s) = \sum_{j=1}^{p} w_j \phi_j(s) \tag{1.8}$$

where $(\phi_j)_{1 \leq j \leq p}$ is a set of basis functions, which should be defined beforehand, and the weights w_j are the parameters:

$$\theta = [w_1, \ldots, w_p]^T \tag{1.9}$$

Many function approximation algorithms require such a representation to ensure convergence [3], or even to be applicable [4]. Other representations are possible such as neural networks where θ contains the set of associated synaptic weights. The proposed KTD framework is applicable to any representation of the value (or Q-) function, as long as it can be fully described by a finite set of p parameters.

In (1.7), the term δ_i is the TD error. Suppose that at step i a transition (s_i, a_i, s_{i+1}, r_i) is observed. For TD-like RL algorithms, that is algorithms which aim at evaluating the value function of a given policy π, the TD error is:

$$\delta_i = r_i + \gamma \hat{V}_{\theta_i}(s_{i+1}) - \hat{V}_{\theta_i}(s_i) \tag{1.10}$$

For SARSA-like algorithms, that is algorithms which aim at evaluating the Q-function of a given policy π, the TD error is:

$$\delta_i = r_i + \gamma \hat{Q}_{\theta_i}(s_{i+1}, a_{i+1}) - \hat{Q}_{\theta_i}(s_i, a_i) \tag{1.11}$$

Finally, for Q-learning-like algorithms, that is algorithms which aim at computing the optimal Q-function, the TD error is:

$$\delta_i = r_i + \gamma \max_{b \in A} \hat{Q}_{\theta_i}(s_{i+1}, b) - \hat{Q}_{\theta_i}(s_i, a_i) \tag{1.12}$$

The type of temporal difference which is used determines which Bellman equation is to be solved (evaluation equation for (1.10) and (1.11), optimality equation for (1.12)), and thus if the algorithm belongs to the policy iteration or value iteration family.

The term K_i is a gain which is specific to each algorithm. The most common are reviewed here. For TD, SARSA and Q-learning (see [1] for example), the gain can be written as

$$K_i = \alpha_i e_i \tag{1.13}$$

where α_i is a classical learning rate in stochastic approximation theory, and should verify:

$$\sum_{i=0}^{\infty} \alpha_i = \infty \text{ and } \sum_{i=0}^{\infty} \alpha_i^2 < \infty \tag{1.14}$$

and e_i is a unitary vector which is zero everywhere except in the component corresponding to the state s_i (or to the state-action (s_i, a_i)) which is equal to one (Kronecker function). These algorithms have been modified to consider so-called eligibility traces (see [1]), and the gain can then be written as

$$K_i = \alpha_i \sum_{j=1}^{i} \lambda^{i-j} e_j \tag{1.15}$$

where λ is the eligibility factor. These algorithms have also been extended to take into account approximate representation of the value function. According to [5] they are called direct algorithms. Without eligibility traces, the gain can be written as

$$K_i = \alpha_i \nabla_{\theta_i} \hat{V}_{\theta_i}(s_i) \tag{1.16}$$

where $\nabla_{\theta_i} \hat{V}_{\theta_i}(s_i)$ is the derivation following the parameter vector of the parameterized value function in the current state. The value function can be replaced straightforwardly by the Q-function in this gain. The direct algorithms have also been extended to take into account eligibility traces, which leads to the following gain:

$$K_i = \alpha_i \sum_{j=1}^{i} \lambda^{i-j} \nabla_{\theta_i} \hat{V}_{\theta_i}(s_j) \tag{1.17}$$

Another well known approach is the set of residual algorithms [5], for which the gain is obtained through the minimization of the L_2-norm of the Bellman residual (*i.e.* the difference between the left side and the right side of the Bellman equation):

$$K_i = \alpha_i \nabla_{\theta_i} \left(\hat{V}_{\theta_i}(s_i) - \gamma \hat{V}_{\theta_i}(s_{i+1}) \right) \tag{1.18}$$

The last approach we review is the Least-Squares Temporal Differences (LSTD) algorithm [4], which is only defined for linear parameterization (1.8) and for which the gain is defined recursively:

$$K_i = \frac{C_{i-1}\phi(s_i)}{1 + (\phi(s_i) - \gamma\phi(s_{i+1}))^T C_{i-1}\phi(s_i)} \tag{1.19}$$

$$C_i = C_{i-1} - \frac{C_{i-1}\phi(s_i)(\phi(s_i) - \gamma\phi(s_{i+1}))^T C_{i-1}}{1 + (\phi(s_i) - \gamma\phi(s_{i+1}))^T C_{i-1}\phi(s_i)} \tag{1.20}$$

where $\phi(s)$ is defined in (1.47). This algorithm has also been extended to eligibility traces, see [6] for details.

The problem addressed in this work can be stated as: given a representation of the value function (or of the Q-function) summarized by the parameter vector θ and given a temporal difference scheme (or equivalently given a Bellman equation to be solved), what is the best gain K ? To answer this question, a statistical point of view is adopted here and the Kalman filtering framework [7] is followed. The proposed approach can be linked to papers based on Gaussian processes [8], least-squares [4] or Kalman filtering [9, 10]. This will be further discussed in Section 1.9. In the next section the general *Kalman Temporal Differences* (KTD) framework is presented. The following sections specialize it to derive a family of algorithms for the value function and the Q-function evaluation (KTD-V and KTD-SARSA), and for the Q-function optimization (KTD-Q). An approximating scheme, the so-called *unscented transform* [11], which is necessary to handle nonlinear parameterization and the Bellman optimality equation, is also presented. Eventually some points are discussed and perspectives are presented.

1.3 KTD: the general case

In this section a very general point of view is adopted, and practical algorithms will be derived later. For now, a transition is generically noted as:

$$t_i = \begin{cases} (s_i, s_{i+1}) \\ (s_i, a_i, s_{i+1}, a_{i+1}) \\ (s_i, a_i, s_{i+1}) \end{cases} \tag{1.21}$$

given that the aim is the value function evaluation, the Q-function evaluation or the Q-function optimization. Similarly, for the same cases, the following shortcuts hold:

$$g_{t_i}(\theta_i) = \begin{cases} \hat{V}_{\theta_i}(s_i) - \gamma \hat{V}_{\theta_i}(s_{i+1}) \\ \hat{Q}_{\theta_i}(s_i, a_i) - \gamma \hat{Q}_{\theta_i}(s_{i+1}, a_{i+1}) \\ \hat{Q}_{\theta_i}(s_i, a_i) - \gamma \max_{b \in A} \hat{Q}_{\theta_i}(s_{i+1}, b) \end{cases} \tag{1.22}$$

Thus all TD schemes can be written generically as

$$\delta_i = r_i - g_{t_i}(\theta_i) \tag{1.23}$$

As said before, a statistical point of view is adopted. The parameter vector θ_i is modeled as a random variable following a random walk. The problem at sight can then be stated in a so-called *state-space formulation*:

$$\begin{cases} \theta_{i+1} = \theta_i + v_i \\ r_i = g_{t_i}(\theta_i) + n_i \end{cases} \tag{1.24}$$

This expression is fundamental for the proposed framework. The first equation is the evolution equation, it specifies that the parameter vector follows a random walk which expectation corresponds to the optimal estimation of the value function. The evolution noise v_i is centered (its mean is 0), white (equally distributed in terms of frequencies) and independent (independence between to consecutive samples). Notice that this allows handling non-stationary MDPs. The second equation is the observation equation, it links the observed transition to the value (or Q-) function through a Bellman equation. The observation noise n_i is supposed centered, white and independent. Notice that this necessary assumption does not hold for stochastic MDPs (see [12] for an expression of this noise in the stochastic case), that is why deterministic transitions are supposed here. This model noise arises from the fact that the solution of the Bellman equation does not necessarily exist in the functional space spanned by the set of parameter vectors.

The objective could be to estimate the parameter vector which minimizes the expectation of the mean-squared error conditioned on past observed rewards. The associated cost can be written as:

$$J(\hat{\theta}_i) = E\left[\|\theta_i - \hat{\theta}_i\|^2 | r_{1:i}\right] \text{ with } r_{1:i} = r_1, \ldots, r_i \tag{1.25}$$

Generally speaking, the minimum mean square error (MMSE) estimator is the conditional expectation:

$$\underset{\hat{\theta}_i}{\operatorname{argmin}} J(\hat{\theta}_i) = \hat{\theta}_{i|i} = E\left[\theta_i | r_{1:i}\right] \tag{1.26}$$

However, except in specific cases, this estimator is not computable. Instead, the aim is here to find the best *linear* estimator. It can be written in a form quite similar to equation (1.7):

$$\hat{\theta}_{i|i} = \hat{\theta}_{i|i-1} + K_i \tilde{r}_i \tag{1.27}$$

In equation (1.27), $\hat{\theta}_{i|i}$ is the estimate at time i, $\hat{\theta}_{i|i-1} = E[\theta_i | r_{1:i-1}]$ is the prediction of this estimate according to past observed rewards $r_{1:i-1}$, and for a random walk model the following equality holds:

$$\hat{\theta}_{i|i-1} = \hat{\theta}_{i-1|i-1} \tag{1.28}$$

The innovation

$$\tilde{r}_i = r_i - \hat{r}_{i|i-1} \tag{1.29}$$

is the difference between the observed reward r_i and its prediction based on the previous estimate of the parameter vector:

$$\hat{r}_{i|i-1} = E\left[g_{t_i}(\theta_i) | r_{1:i-1}\right] \tag{1.30}$$

Note that the innovation $\tilde{r}_i$ is not exactly the temporal difference defined in equation (1.23), which is a random variable through its dependency to the random vector θ_i. It is its expectation conditioned on past observed data.

Using classical equalities, the cost function can be rewritten as:

$$\begin{aligned} J(\hat{\theta}_i) &= E\left[\|\theta_i - \hat{\theta}_i\|^2 | r_{1:i}\right] \\ &= E\left[(\theta_i - \hat{\theta}_i)^T (\theta_i - \hat{\theta}_i) | r_{1:i}\right] \\ &= \text{trace}\left(E\left[(\theta_i - \hat{\theta}_i)(\theta_i - \hat{\theta}_i)^T | r_{1:i}\right]\right) \\ &= \text{trace}\left(\text{cov}\left(\theta_i - \hat{\theta}_i | r_{1:i}\right)\right) \end{aligned} \tag{1.31}$$

where the *trace* function indicates that the trace of the matrix is considered and the *cov* function indicates the covariance matrix of the random variable. A first step to the computation of the optimal gain is to express the conditioned covariance over parameters as a function of the gain K_i. A few more notations are first introduced (recall also the definition of the innovation (1.29)):

$$\tilde{\theta}_{i|i} = \theta_i - \hat{\theta}_{i|i} \text{ and } \tilde{\theta}_{i|i-1} = \theta_i - \hat{\theta}_{i|i-1} \tag{1.32}$$

$$P_{i|i} = \text{cov}\left(\tilde{\theta}_{i|i} | r_{1:i}\right) \tag{1.33}$$

$$P_{i|i-1} = \text{cov}\left(\tilde{\theta}_{i|i-1} | r_{1:i-1}\right) \tag{1.34}$$

$$P_{r_i} = \text{cov}\left(\tilde{r}_i | r_{i:i-1}\right) \tag{1.35}$$

$$P_{\theta r_i} = E\left[\tilde{\theta}_{i|i-1} \tilde{r}_i | r_{1:i-1}\right] \tag{1.36}$$

Using the postulated update of equation (1.27), and the various estimators being unbiased, the covariance can be expanded:

$$\begin{aligned} P_{i|i} &= \text{cov}\left(\theta_i - \hat{\theta}_{i|i} | r_{1:i}\right) \\ &= \text{cov}\left(\theta_i - \left(\hat{\theta}_{i|i-1} + K_i \tilde{r}_i\right) | r_{1:i-1}\right) \\ &= \text{cov}\left(\tilde{\theta}_{i|i-1} - K_i \tilde{r}_i | r_{1:i-1}\right) \\ P_{i|i} &= P_{i|i-1} - P_{\theta r_i} K_i^T - K_i P_{\theta r_i}^T + K_i P_{r_i} K_i^T \end{aligned} \tag{1.37}$$

The optimal gain can thus be obtained by deriving the trace of this matrix.
First note that the gradient being linear, for three matrices of *ad hoc* dimensions A, B and C, B being symmetric, the following algebraic identities hold:

$$\nabla_A \left(\text{trace}\left(ABA^T\right)\right) = 2AB \tag{1.38}$$

$$\nabla_A \left(\text{trace}\left(AC^T\right)\right) = \nabla_A \left(\text{trace}\left(CA^T\right)\right) = C \tag{1.39}$$

and thus using equation (1.37) and previous identities:

$$\begin{aligned} & \nabla_{K_i} \left(\text{trace}\left(P_{i|i}\right)\right) = 0 \\ \Leftrightarrow \quad & 2K_i P_{r_i} - 2P_{\theta r_i} = 0 \\ \Leftrightarrow \quad & K_i = P_{\theta r_i} P_{r_i}^{-1} \end{aligned} \tag{1.40}$$

Using (1.37) and (1.40), the covariance matrix $P_{i|i}$ is

$$P_{i|i} = P_{i|i-1} - K_i P_{r_i} K_i^T \tag{1.41}$$

Notice that no Gaussian assumption has been made to derive these equations.

The most general KTD algorithm, which breaks down in three stages, can now be derived. The first step consists in computing predicted quantities $\hat{\theta}_{i|i-1}$ and $P_{i|i-1}$. Recall that for a random walk model, equation (1.28) holds, and the predicted covariance can also be computed analytically:

$$\begin{aligned} P_{i|i-1} &= \mathrm{cov}\left(\tilde{\theta}_{i|i-1}|r_{1:i-1}\right) \\ &= \mathrm{cov}\left(\tilde{\theta}_{i-1|i-1}+v_{i-1}|r_{1:i-1}\right) \\ &= P_{i-1|i-1}+P_{v_{i-1}} \end{aligned} \tag{1.42}$$

where $P_{v_{i-1}}$ is the variance matrix of the evolution noise (which is given).

The second step is to compute some statistics of interest. It will be specialized for each algorithm of the next sections. The first statistic to compute is the prediction $\hat{r}_{i|i-1}$ (1.30). The second statistic to compute is the covariance between the parameter vector and the innovation:

$$P_{\theta r_i} = E\left[(\theta_i - \hat{\theta}_{i|i-1})(r_i - \hat{r}_{i|i-1})|r_{1:i-1}\right] \tag{1.43}$$

However, from the state-space model (1.24), $r_i = g_{t_i}(\theta_i) + n_i$, and the observation noise is centered and independent, so

$$P_{\theta r_i} = E\left[(\theta_i - \hat{\theta}_{i|i-1})(g_{t_i}(\theta_i) - \hat{r}_{i|i-1})|r_{1:i-1}\right] \tag{1.44}$$

The last statistic to compute is the covariance of the innovation, which can be written (using again the characteristics of the observation noise):

$$\begin{aligned} P_{r_i} &= E\left[(r_i - \hat{r}_{i|i-1})^2|r_{1:i-1}\right] \\ &= E\left[(g_{t_i}(\theta_i) - \hat{r}_{i|i-1} + n_i)^2|r_{1:i-1}\right] \\ &= E\left[(g_{t_i}(\theta_i) - \hat{r}_{i|i-1})^2|r_{1:i-1}\right] + P_{n_i} \end{aligned} \tag{1.45}$$

where P_{n_i} is the variance of the observation noise (which is also known).

The last step of the algorithm is the correction step. It consists in computing the gain (1.40), correcting the parameter vector (1.27) and updating the associated covariance matrix (1.41) accordingly. Notice that as the proposed method is online, it must thus be initialized with some priors $\hat{\theta}_{0|0}$ and $P_{0|0}$. The proposed general framework is summarized in Algorithm 1. The main difficulty in applying the KTD is to compute the statistics of interest $\hat{r}_{i|i-1}$, $P_{\theta r_i}$ and P_{r_i} (for which statistics $\hat{\theta}_{i|i-1}$ and $P_{i|i-1}$ are necessary), which will be the subject of the next three sections.

1.4 KTD-V

This section focuses on evaluating the value function, that is finding an approximate solution of the Bellman evaluation equation (1.4). First the linear case is considered, which allows deriving an analytical solution to the statistics of interest computation. An approximation scheme, the unscented transform [11], is then introduced, which proves itself to be useful for the nonlinear parameterization case.

1.4.1 Linear parameterization

In this section the linear parameterization of equation (1.8) is adopted, which is shortened as:

$$V_\theta(s) = \phi(s)^T\theta \tag{1.46}$$

Algorithm 1: General KTD algorithm

Initialization: priors $\hat{\theta}_{0|0}$ and $P_{0|0}$;

for $i \leftarrow 1,2,\ldots$ **do**

Observe transition t_i and reward r_i ;

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

Compute statistics of interest;
$\hat{r}_{i|i-1} = E[g_{t_i}(\theta_i)|r_{1:i-1}]$;
$P_{\theta r_i} = E\left[(\theta_i - \hat{\theta}_{i|i-1})(g_{t_i}(\theta_i) - \hat{r}_i)|r_{1:i-1}\right]$;
$P_{r_i} = E\left[(g_{t_i}(\theta_i) - \hat{r}_{i|i-1})^2|r_{1:i-1}\right] + P_{n_i}$;

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

where the parameter vector is given in equation (1.9) and where $\phi(s)$ is a vector defined as:

$$\phi(s) = [\phi_1(s), \ldots, \phi_p(s)]^T \tag{1.47}$$

The state-space formulation (1.24) can thus be rewritten as:

$$\begin{cases} \theta_{i+1} = \theta_i + v_i \\ r_i = \left(\phi(s_i) - \gamma\phi(s_{i+1})\right)^T \theta_i + n_i \end{cases} \tag{1.48}$$

To shorten the equations, H_i is defined as:

$$H_i = \phi(s_i) - \gamma\phi(s_{i+1}) \tag{1.49}$$

As the observation equation is linear, the statistics of interest can be derived analytically. The prediction is

$$\begin{aligned} \hat{r}_{i|i-1} &= E\left[g_{t_i}(\theta_i)|r_{1:i-1}\right] \\ &= E\left[H_i^T \theta_i|r_{1:i-1}\right] \\ &= H_i^T E\left[\theta_i|r_{1:i-1}\right] \\ &= H_i^T \hat{\theta}_{i|i-1} \end{aligned} \tag{1.50}$$

The covariance between the parameter vector and the innovation can also be computed analytically:

$$\begin{aligned} P_{\theta r_i} &= E\left[\tilde{\theta}_{i|i-1}\left(g_{t_i}(\theta_i) - \hat{r}_{i|i-1}\right)|r_{1:i-1}\right] \\ &= E\left[\tilde{\theta}_{i|i-1}H_i^T\tilde{\theta}_{i|i-1}|r_{1:i-1}\right] \\ &= E\left[\tilde{\theta}_{i|i-1}\tilde{\theta}_{i|i-1}^T|r_{1:i-1}\right]H_i \\ &= P_{i|i-1}H_i \end{aligned} \tag{1.51}$$

The covariance of the innovation is derived analytically too:

$$\begin{aligned} P_{r_i} &= E\left[\left(g_{t_i}(\theta_i) - \hat{r}_{i|i-1}\right)^2|r_{1:i-1}\right] + P_{n_i} \\ &= E\left[\left(H_i^T\tilde{\theta}_{i|i-1}\right)^2|r_{1:i-1}\right] + P_{n_i} \\ &= H_i^T P_{i|i-1}H_i + P_{n_i} \end{aligned} \tag{1.52}$$

The gain can thus be defined algebraically and recursively:

$$K_i = \frac{P_{i|i-1}H_i}{H_i^T P_{i|i-1}H_i + P_{n_i}} \tag{1.53}$$

This gain shares similarities with the gain of the LSTD algorithm [4] (equation (1.19)), which is not a surprise. The LSTD is based on a least-squares minimization (however with the introduction of instrumental variables in order to handle stochastic transitions), and the Kalman filter can be seen as a generalization of the least-squares method. This gain also shares similarities with the one arising from a parametric Gaussian process modelling of the value function evaluation problem [8]. The KTD-V approach for linear parameterization is summarized in Algorithm 2.

The next case to be addressed is the nonlinear parameterization of the value function. Basically, the issue of computing the statistics of interest for the KTD can be stated as the following problem: given the mean and covariance of a random variable, how can the mean and covariance of a nonlinear (and perhaps non derivable) mapping of this random variable be computed ? The following section presents the unscented transform, which is an approximation scheme designed to handle such a problem.

1.4.2 The Unscented Transform

Let's abstract a little bit from reinforcement learning and Kalman filtering. Let X be a random vector, and let Y be a mapping of X. The problem is to compute the mean and covariance of Y knowing the mapping and first and second order moments of X. If the mapping is linear, the relation between X and Y can be written as $Y = AX$ where A is a matrix of *ad hoc* dimension. In this case, required mean and covariance can be analytically computed: $E[Y] = AE[X]$ and $E[YY^T] = AE[XX^T]A^T$. This result has been used to derive the KTD-V of Section 1.4.1.

If the mapping is nonlinear, the relation between X and Y can be generically written as:

$$Y = f(X) \tag{1.54}$$

A first solution would be to approximate the nonlinear mapping, that is to linearize it around the mean of the random vector X. This leads to the following approximations of the mean and covariance of Y:

$$E[Y] \approx f\left(E[X]\right) \tag{1.55}$$

$$E[YY^T] \approx \left(\nabla f\left(E[X]\right)\right)E[XX^T]\left(\nabla f\left(E[X]\right)\right)^T \tag{1.56}$$

Algorithm 2: KTD-V: linear parameterization

Initialization: priors $\hat{\theta}_{0|0}$ and $P_{0|0}$;

for $i \leftarrow 1,2,\ldots$ **do**

Observe transition (s_i, s_{i+1}) and reward r_i ;

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

Compute statistics of interest;
$\hat{r}_{i|i-1} = H_i^T \hat{\theta}_{i|i-1}$;
$P_{\theta r_i} = P_{i|i-1} H_i$;
$P_{r_i} = H_i^T P_{i|i-1} H_i + P_{n_i}$;
/* where $H_i = \phi(s_i) - \gamma\phi(s_{i+1})$ */

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

This approach is the base of Extended Kalman Filtering (EKF) [13], which has been extensively studied and used in past decades. However it has some limitations. First it cannot handle non-derivable nonlinearities, and thus cannot handle the Bellman optimality equation (1.6) because of the max operator. It requires to compute the gradient of the mapping f, which can be quite difficult even if possible. It also supposes that the nonlinear mapping is locally linearizable, which is unfortunately not always the case and can lead to quite bad approximation, as exemplified in [11].

The basic idea of the unscented transform is that it is easier to approximate an arbitrary random vector than an arbitrary nonlinear function. Its principle is to sample *deterministically* a set of so-called sigma-points from the expectation and the covariance of X. The images of these points through the nonlinear mapping f are then computed, and they are used to approximate statistics of interest. It shares similarities with Monte-Carlo methods, however here the sampling is deterministic and requires fewer samples to be drawn, nonetheless allowing a given accuracy [11].

The original unscented transform is now described more formally (some variants have been introduced since, but the basic principle is the same). Let n be the dimension of X. A set of $2n+1$ sigma-points is computed as follows:

$$x_0 = \bar{X} \qquad j = 0 \tag{1.57}$$

$$x_j = \bar{X} + \left(\sqrt{(n+\kappa)P_X}\right)_j \qquad 1 \leq j \leq n \tag{1.58}$$

$$x_j = \bar{X} - \left(\sqrt{(n+\kappa)P_X}\right)_{n-j} \qquad n+1 \leq j \leq 2n \tag{1.59}$$

as well as associated weights:

$$w_0 = \frac{\kappa}{n+\kappa} \text{ and } w_j = \frac{1}{2\,(n+\kappa)} \forall j > 0 \tag{1.60}$$

where $\bar{X}$ is the mean of X, P_X is its variance matrix, κ is a scaling factor which controls the accuracy of the unscented transform [11], and $(\sqrt{(n+\kappa)P_X})_j$ is the j^{th} column of the Cholesky decomposition of the matrix $(n+\kappa)P_X$. Then the image through the mapping f is computed for each of these sigma-points:

$$y_j = f(x_j), \quad 0 \leq j \leq 2n \tag{1.61}$$

The set of sigma-points and their images can finally be used to compute first and second order moments of Y, and even P_{XY}, the covariance matrix between X and Y:

$$\bar{Y} \approx \bar{y} = \sum_{j=0}^{2n} w_j y_j \tag{1.62}$$

$$P_Y \approx \sum_{j=0}^{2n} w_j \,(y_j - \bar{y})\,(y_j - \bar{y})^T \tag{1.63}$$

$$P_{XY} \approx \sum_{j=0}^{2n} w_j \,(x_j - \bar{x})\,(y_j - \bar{y})^T \tag{1.64}$$

where $\bar{x} = x_0 = \bar{X}$. The unscented transform having been presented, it is now possible to address the value function evaluation problem with nonlinear parameterization.

1.4.3 Nonlinear parameterization

In this section a generic parameterization of the value function $\hat{V}_\theta$ is considered: it can be a neural network [14], a parametric kernel representation [12], or any function representation of interest, as long as it can be described by a finite set of p parameters. The general state-space formulation (1.24) can thus be rewritten as:

$$\begin{cases} \theta_{i+1} = \theta_i + v_i \\ r_i = \hat{V}_{\theta_i}(s_i) - \gamma \hat{V}_{\theta_i}(s_{i+1}) + n_i \end{cases} \tag{1.65}$$

The problem is still to compute the statistics of interest, which becomes tractable with the unscented transform. The first thing to compute is the set of sigma-points from known statistic $\hat{\theta}_{i|i-1}$ and $P_{i|i-1}$ as described in Section 1.4.2, as well as the associated weights:

$$\Theta_{i|i-1} = \left\{ \hat{\theta}^{(j)}_{i|i-1},\, 0 \leq j \leq 2p \right\} \tag{1.66}$$

$$\mathcal{W} = \left\{ w_j,\, 0 \leq j \leq 2p \right\} \tag{1.67}$$

Then the images of the sigma-points are computed, using the observation function of state-space model (1.65), which is linked to the Bellman evaluation equation (1.4):

$$\mathcal{R}_{i|i-1} = \left\{ \hat{r}^{(j)}_{i|i-1} = \hat{V}_{\hat{\theta}^{(j)}_{i|i-1}}(s_i) - \gamma \hat{V}_{\hat{\theta}^{(j)}_{i|i-1}}(s_{i+1}),\, 0 \leq j \leq 2p \right\} \tag{1.68}$$

The sigma-points and their images being computed, the statistics of interest can be approximated by:

$$\hat{r}_{i|i-1} \approx \sum_{j=0}^{2p} w_j \hat{r}_{i|i-1}^{(j)} \tag{1.69}$$

$$P_{r_i} \approx \sum_{j=0}^{2p} w_j \left(\hat{r}_{i|i-1}^{(j)} - \hat{r}_{i|i-1}\right)^2 + P_{n_i} \tag{1.70}$$

$$P_{\theta r_i} \approx \sum_{j=0}^{2p} w_j \left(\hat{\theta}_{i|i-1}^{(j)} - \hat{\theta}_{i|i-1}\right)\left(\hat{r}_{i|i-1}^{(j)} - \hat{r}_{i|i-1}\right) \tag{1.71}$$

As the unscented transform is no longer an approximation for linear mapping, this formulation is still valid for value function evaluation with linear function approximation. The KTD-V with nonlinear function approximation is summarized in Algorithm 3.

Algorithm 3: KTD-V: nonlinear parameterization

Initialization: priors $\hat{\theta}_{0|0}$ and $P_{0|0}$;

for $i \leftarrow 1, 2, \ldots$ **do**

Observe transition (s_i, s_{i+1}) and reward r_i ;

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

Sigma-points computation ;
$\Theta_{i|i-1} = \left\{\hat{\theta}_{i|i-1}^{(j)}, \quad 0 \le j \le 2p\right\}$;
$\mathcal{W} = \left\{w_j, \quad 0 \le j \le 2p \quad \right\}$;
$\mathcal{R}_{i|i-1} = \left\{\hat{r}_{i|i-1}^{(j)} = \hat{V}_{\hat{\theta}_{i|i-1}^{(j)}}(s_i) - \gamma \hat{V}_{\hat{\theta}_{i|i-1}^{(j)}}(s_{i+1}), \quad 0 \le j \le 2p\right\}$;

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

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

1.5 KTD-SARSA

This section focuses on the Q-function evaluation of a given policy. The associated algorithm is called KTD-SARSA, which can be misleading. Indeed, generally speaking, SARSA is a Q-function evaluation algorithm associated with an optimistic policy iteration scheme. Here the focus is on the Q-function evaluation problem, and the control part is left apart. For a general parameterization $\hat{Q}_\theta$, and considering the Bellman evaluation equation (1.5), the state-space model (1.24) can be rewritten as:

$$\begin{cases} \theta_{i+1} = \theta_i + v_i \\ r_i = \hat{Q}_{\theta_i}(s_i, a_i) - \gamma \hat{Q}_{\theta_i}(s_{i+1}, a_{i+1}) + n_i \end{cases} \tag{1.72}$$

For a fixed policy, the value function evaluation on the state space induced Markov chain is quite similar to the Q-function evaluation on the state-action space induced Markov chain. It is thus straightforward to extend Algorithms 2 and 3 to Q-function evaluation. Recall that for a linear parameterization, the unscented transform leads to an exact computation of statistics of interest, and thus in this case Algorithm 3 is equivalent to Algorithm 2. That is why only the sigma-point formulation of KTD-SARSA is derived (Algorithm 4).

Algorithm 4: KTD-SARSA

Initialization: priors $\hat{\theta}_{0|0}$ and $P_{0|0}$;

for $i \leftarrow 1, 2, \ldots$ **do**

Observe transition $(s_i, a_i, s_{i+1}, a_{i+1})$ and reward r_i;

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

Sigma-points computation ;
$\Theta_{i|i-1} = \left\{ \hat{\theta}^{(j)}_{i|i-1}, \quad 0 \leq j \leq 2p \right\}$;
$\mathcal{W} = \left\{ w_j, \quad 0 \leq j \leq 2p \quad \right\}$;
$\mathcal{R}_{i|i-1} = \left\{ \hat{r}^{(j)}_{i|i-1} = \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_i, a_i) - \gamma \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_{i+1}, a_{i+1}), \quad 0 \leq j \leq 2p \right\}$;

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

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

1.6 KTD-Q

This section focuses on the Q-function optimization, that is on finding an approximate solution of the Bellman optimality equation (1.6). A general parameterization $\hat{Q}_\theta$ is adopted. The state-space model (1.24) can be specialized as follows:

$$\begin{cases} \theta_{i+1} = \theta_i + v_i \\ r_i = \hat{Q}_{\theta_i}(s_i, a_i) - \gamma \max_{b \in A} \hat{Q}_{\theta_i}(s_{i+1}, b) + n_i \end{cases} \tag{1.73}$$

Here linear and nonlinear parameterization are not distinguished, because of the max operator, which is inherent to the Bellman optimality equation, and because of which the observation equation of state-space model (1.73) is nonlinear. This max operator is difficult to handle, especially because of its non-derivability.

Hopefully, as it approximates the random variable rather than the mapping, the unscented transform is a derivative free approximation. Given the general KTD algorithm introduced in Section 1.3 and the unscented transform described in Section 1.4.2, it is possible to derive KTD-Q, the KTD algorithm for Q-function direct optimization. One has first to compute the set of sigma-points associated with the parameter vector, as in equations (1.66-1.67). Then the mapping of these sigma-points through the observation equation of state-space model (1.73), which contains the max operator, is computed:

$$\begin{aligned} \mathcal{R}_{i|i-1} = \Big\{ & \hat{r}^{(j)}_{i|i-1} = \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_i, a_i) \\ & - \gamma \max_{b \in A} \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_{i+1}, b),\ 0 \le j \le 2p \Big\} \end{aligned} \tag{1.74}$$

Then, as usual, the sigma-points and their images are used to compute the statistics of interest, as in equations (1.69-1.71). The proposed KTD-Q is summarized in Algorithm 5.

1.7 Colored noise model and eXtended KTD

The white observation noise assumption does not hold for stochastic transitions. Indeed, under this hypothesis, KTD minimizes a square Bellman residual : $\hat{\theta}_{i|i} = \text{argmin}_\theta \sum_{j=1}^{i} \frac{1}{P_{n_j}} (r_j + \gamma \hat{V}_\theta(s_{j+1}) - \hat{V}_\theta(s_j))^2$. KTD therefore produces biased estimates of the value function. We first present the colored noise model initially introduced by [15] and then show how it can be used to extend the KTD framework.

The value function can be defined as the expectation (over all possible trajectories) of the following discount return random process: $D(s) = \sum_{i=0}^{\infty} \gamma^i R(s_i, a_i, s_{i+1}) | s_0 = s$ with $a_i \sim \pi(.|s_i)$ and $s_{i+1} \sim p(.|s_i, a_i)$. This equation naturally leads to a Bellman-like equation: $D(s) = R(s,a,s') + \gamma D^\pi(s')$ with again $a \sim \pi(.|s)$ and $s' \sim p(.|s,a)$. This random process can be broken down into the value function plus a random zero-mean residual: $D(s) = V(s) + \Delta V(s)$ with $V(s) = E[D(s)]$ and $\Delta V^\pi(s) = D^\pi(s) - V^\pi(s)$. Manipulating the two previous equations, the reward can be expressed as a function of the value plus a noise: $R(s,a,s') = V(s) - \gamma V(s') + N(s,s')$ with $N(s,s') = \Delta V(s) - \gamma \Delta V(s')$. As in [15], the residuals are supposed to be independent, which leads to a colored noise model. This assumption can seem rather strong. However, despite this, a convergence of the resulting algorithm to the same solution as LSTD(1) can be obtained.

Algorithm 5: KTD-Q

Initialization: priors $\hat{\theta}_{0|0}$ and $P_{0|0}$;

for $i \leftarrow 1,2,\ldots$ **do**

Observe transition (s_i, a_i, s_{i+1}) and reward r_i ;

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

Sigma-points computation ;
$\Theta_{i|i-1} = \left\{ \hat{\theta}^{(j)}_{i|i-1}, \quad 0 \leq j \leq 2p \right\}$;
$\mathcal{W} = \left\{ w_j, \quad 0 \leq j \leq 2p \quad \right\}$;
$\mathcal{R}_{i|i-1} = \left\{ \hat{r}^{(j)}_{i|i-1} = \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_i, a_i) - \gamma \max_{b \in A} \hat{Q}_{\hat{\theta}^{(j)}_{i|i-1}}(s_{i+1}, b), \quad 0 \leq j \leq 2p \right\}$;

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

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

The residuals being centered and supposed independent, they can be modeled as a centered white noise u_i of theoretical variance $E[u_i^2] = \mathrm{var}(D(s_{i+1}))$. This leads to the following moving average (MA) observation noise model:

$$n_i = -\gamma u_i + u_{i-1}, \quad u_i \sim (0, \sigma_i^2) \tag{1.75}$$

The XKTD model we propose uses the state-space model (1.24) with observation noise (1.75). Taking MA noise into account within the Kalman paradigm is a key contribution of this deliverable. As far as we know, the case of MA observation noise has never been addressed before in the Kalman filtering literature. In light of this, we believe that naively re-deriving KTD for the case with MA noise would be quite complicated.

Fortunately, we can express the scalar MA noise n_i as a vectorial auto-regressive (AR) noise, and it is quite easy to introduce AR process noise in a Kalman filter by extending the evolution equation (e.g., see [13]). This is done by extending the state-space model (1.24) to a new one for which Algorithm 1 applies rather directly. Let ω_i be an auxiliary random variable. Scalar MA noise (1.75) is equivalent to the following vectorial AR noise:

$$\begin{pmatrix} \omega_i \\ n_i \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \omega_{i-1} \\ n_{i-1} \end{pmatrix} + \begin{pmatrix} 1 \\ -\gamma \end{pmatrix} u_i \tag{1.76}$$

This new noise formulation having been defined, it is now possible to define a state-space formulation equivalent to (1.24) with a MA observation noise:

$$\begin{cases} \mathbf{x}_i = F\mathbf{x}_{i-1} + v'_i \\ r_i = g_i(\mathbf{x}_i) \end{cases} \Leftrightarrow \tag{1.77}$$

$$\begin{cases} \begin{pmatrix} \theta_i \\ \omega_i \\ n_i \end{pmatrix} = \begin{pmatrix} I_p & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^T & 0 & 0 \\ \mathbf{0}^T & 1 & 0 \end{pmatrix} \begin{pmatrix} \theta_{i-1} \\ \omega_{i-1} \\ n_{i-1} \end{pmatrix} + \begin{pmatrix} v_i \\ u_i \\ -\gamma u_i \end{pmatrix} \\ r_i = \hat{V}_{\theta_i}(s_i) - \gamma \hat{V}_{\theta_i}(s_{i+1}) + n_i \end{cases} \tag{1.78}$$

The parameters are now extended with the vectorial AR noise: $\mathbf{x}_i^T = \begin{pmatrix} \theta_i^T & \omega_i & n_i \end{pmatrix}$. The evolution matrix F takes into account the structure of the MA observation noise (p is the number of parameters, I_p the identity matrix of size p, and $\mathbf{0}$ a zero $p \times 1$ column vector). The process noise is also extended to take into account the MA observation noise. The new noise v'_i is centered with variance $P_{v'_i}$:

$$v'_i = \begin{pmatrix} v_i \\ u_i \\ -\gamma u_i \end{pmatrix} \sim \left(\begin{pmatrix} \mathbf{0} \\ 0 \\ 0 \end{pmatrix}, P_{v'_i} = \begin{pmatrix} P_{v_i} & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^T & \sigma_i^2 & -\gamma\sigma_i^2 \\ \mathbf{0}^T & -\gamma\sigma_i^2 & \gamma^2\sigma_i^2 \end{pmatrix} \right) \tag{1.79}$$

The observation equation remains the same: $r_i = g_i(\mathbf{x}_i) = \hat{V}_{\theta_i}(s_i) - \gamma \hat{V}_{\theta_i}(s_{i+1}) + n_i$. However now the observation noise is a part of the evolution equation. Algorithm 1 applies to this equivalent state-space formulation, however with a slight modification: the evolution equation being no longer the identity , it has to be taken into account in the prediction step, as depicted in Algorithm 6.

1.8 Uncertainty Evaluation and Active Learning

The parameters being modeled as random variables, and the value (or Q-) function being a function of these parameters, it is a random variable for a given state (or state-action pair). We first show how to compute its expectation and the associated uncertainty thanks to the unscented transform. The treatment of the dilemma between exploration and exploitation should benefit from such uncertainty information. Few approaches in the literature allow handling the value function approximation problem as well as computing uncertainty over values at the same time. The work in [8] is such an approach, however the effective use of the obtained uncertainty information is left as perspective. We propose a form of active learning which is a sort of totally explorative policy in the context of KTD-Q. This contribution is shown to effectively speed up learning in Chapter 2.

Algorithm 6: XKTD: practical algorithm

Initialization: priors $\hat{\mathbf{x}}_{0|0} = \begin{pmatrix} \hat{\theta}_{0|0}^T & 0 & 0 \end{pmatrix}^T$ and $P_{\mathbf{x}_{0|0}}$;
for $i \leftarrow 1, 2, \ldots$ **do**

Observe transition (s_i, s_{i+1}) and reward r_i ;

Prediction Step;
$\hat{\mathbf{x}}_{i|i-1} = F\hat{\mathbf{x}}_{i-1|i-1}$;
$P_{\mathbf{x}_{i|i-1}} = FP_{\mathbf{x}_{i-1|i-1}}F^T + P_{v'_{i-1}}$;
Sigma-points computation ;
$\mathbf{X}_{i|i-1} = \{\hat{\mathbf{x}}_{i|i-1}^{(j)} = [(\hat{\theta}_{i|i-1}^{(j)})^T, \hat{\omega}_{i|i-1}^{(j)}, \hat{n}_{i|i-1}^{(j)}]^T, \quad 0 \leq j \leq 2(p+2)\}$ (from $\hat{\mathbf{x}}_{i|i-1}$ and $P_{\mathbf{x}_{i|i-1}}$);
$\mathcal{W} = \{w_j, \quad 0 \leq j \leq 2(p+2)\}$;
$\mathcal{R}_{i|i-1} = \{\hat{r}_{i|i-1}^{(j)} = \hat{V}_{\hat{\theta}_{i|i-1}^{(j)}}(s_i) - \gamma\hat{V}_{\hat{\theta}_{i|i-1}^{(j)}}(s_{i+1}) + \hat{n}_{i|i-1}^{(j)}, 0 \leq j \leq 2(p+2)\}$;
Compute statistics of interest;
$\hat{r}_{i|i-1} = \sum_{j=0}^{2(p+2)} w_j \hat{r}_{i|i-1}^{(j)}$;
$P_{\mathbf{x}r_i} = \sum_{j=0}^{2(p+2)} w_j (\hat{\mathbf{x}}_{i|i-1}^{(j)} - \hat{\mathbf{x}}_{i|i-1})(\hat{r}_{i|i-1}^{(j)} - \hat{r}_{i|i-1})$;
$P_{r_i} = \sum_{j=0}^{2(p+2)} w_j (\hat{r}_{i|i-1}^{(j)} - \hat{r}_{i|i-1})^2$;
Correction step;
$K_i = P_{\mathbf{x}r_i} P_{r_i}^{-1}$;
$\hat{\mathbf{x}}_{i|i} = \hat{\mathbf{x}}_{i|i-1} + K_i(r_i - \hat{r}_{i|i-1})$;
$P_{i|i} = P_{i|i-1} - K_i P_{r_i} K_i^T$;
end

1.8.1 Computing Uncertainty over Values

Let $\hat{V}_\theta$ be the approximated value function parameterized by the random vector θ of mean $\bar{\theta}$ and variance matrix P_θ. Let $\bar{V}_\theta(s)$ and $\hat{\sigma}^2_{V_\theta}(s)$ be the associated mean and variance for a given state s. To propagate the uncertainty from the parameters to the value function a first step is to compute the sigma-points associated to the parameter vector $\Theta = \{\theta^{(j)}, 0 \leq j \leq 2p\}$ as well as corresponding weights $\mathcal{W} = \{w_j, 0 \leq j \leq 2p\}$ from $\bar{\theta}$ and P_θ. Then the images of this sigma-points are computed for the given state s using the parameterized value function :

$$\mathcal{V}_\theta(s) = \left\{ \hat{V}^{(j)}_\theta(s) = \hat{V}_{\theta^{(j)}}(s), \quad 0 \leq j \leq 2p \right\} \tag{1.80}$$

Knowing these images and corresponding weights, it is possible to compute the statistics of interest, namely mean and variance of the approximated value function:

$$\bar{V}_\theta(s) = \sum_{j=0}^{2p} w_j \hat{V}^{(j)}_\theta(s) \quad \text{and} \quad \hat{\sigma}^2_{V_\theta}(s) = \sum_{j=0}^{2p} w_j \left(\hat{V}^{(j)}_\theta(s) - \bar{V}_\theta(s) \right)^2 \tag{1.81}$$

Thus, for a given representation of the value function and a random parameter vector, the uncertainty can be propagated to the value of a state. Extension to Q-function is straightforward. The complexity is here again quadratic. So, as at each time-step i an estimate $\hat{\theta}_{i|i}$ and the associated variance $P_{i|i}$ are known, uncertainty information can be computed in the KTD framework.

1.8.2 A Form of Active Learning

It is shown here how this available uncertainty information can be used in a form of active learning. KTD-Q is an off-policy algorithm: it learns the optimal policy π^* while following a different behaviorial policy b. A natural question is to know what behaviorial policy to choose in order to speed up learning. A possible answer is given here.

Let i be the current temporal index. The system is in a state s_i, and the agent has to choose an action a_i. The considered algorithm being KTD-Q, the estimates $\hat{\theta}_{i-1|i-1}$ and $P_{i-1|i-1}$ are available. They can be used to approximate the uncertainty of the Q-function parameterized by θ_{i-1} in the state s_i and for any action a. Let $\sigma^2_{Q_{\theta_{i-1}}}(s_i, a)$ be the corresponding variance. The action a_i is chosen according to the following random behaviorial policy:

$$b(a_i|s_i) = \frac{\sigma_{Q_{\theta_{i-1}}}(s_i, a_i)}{\sum_{a \in A} \sigma_{Q_{\theta_{i-1}}}(s_i, a)} \tag{1.82}$$

A totally explorative policy is obtained, in the sense that it favours less certain actions. This is a way among others to use the available uncertainty information, nevertheless it is shown in the experiments section to be quite efficient compared to a uniformly random behaviorial policy.

1.9 Discussion and Perspectives

A general Kalman-based function approximation scheme for reinforcement learning in deterministic Markovian decision processes has been introduced, and algorithms for value function and Q-function evaluation (policy iteration scheme) and for Q-function direct optimization (value iteration scheme) have been derived from it.

As mentioned in Section 1.2, related approaches have been proposed previously. In [8] a Gaussian process modelling of reinforcement learning function approximation is proposed. In the deterministic and parametric case, an algorithm which is almost the same as the KTD-V with linear parameterization is obtained (the only difference resides in the absence of process noise). However it is derived from a different point of view. As Kalman filtering is strongly linked to least-squares minimization, our approach shares similarities with the LSTD [4], however without taking into account an instrumental variables concept. In [9] a Kalman filter designed to handle fixed-point approximation in the case of linear parameterization is introduced. It can be roughly seen as a bootstrapping version of the proposed KTD-V. Instead of the observation equation of state-space model (1.48), the following observation equation is used:

$$r_i + \gamma \phi(s_{i+1})^T \hat{\theta}_{i-1|i-1} = \phi(s_i)^T \theta_i + n_i \tag{1.83}$$

In other words, the reward is not considered as the observation, but an approximation of the value function is used to compute a "pseudo"-observation, and the update of the parameters is made so as to match the value function of the current state to this pseudo-observation. In [10], a bank of classical Kalman filters is used to learn the parameters of a piecewise linear parameterization of the value function. It can be roughly seen as a special case of the proposed approach, however differences exist: not one filter but a bank is used and the parameterization is piecewise linear, which is exploited to develop specificities of the algorithm.

The proposed framework has some potential advantages. First it does not suppose stationarity. An immediate application is to handle non-stationary environments. But an even more interesting one is the control case. The algorithm LSTD is known to fail when combined with optimistic policy iteration, because of the induced non-stationarities of this specific learning and control scheme. Kalman filtering and thus the proposed framework is designed to be robust to non-stationarities (random walk model of the parameter vector). This can be quite interesting for the control case, which has not been treated here (the focus was on learning the value function or the Q-function, given observed transitions, and not on how to choose action for a given state). Second, the parameter vector is modelled as a random vector. As a consequence, at each time step, the covariance of this random vector is available. It can be propagated to the value function (as it is clearly a function of the parameters), using the unscented transform if necessary, in order to provide *local* uncertainty information for the value at a given state. This is exemplified in [16] for a simple regression problem. This uncertainty propagation can be useful to handle the well known dilemma between exploration and exploitation.

To finish with, a new Kalman-based function approximation scheme for reinforcement learning has been introduced. Most interesting perspectives are to extend this framework to the control case, for which the non-stationarity hypothesis and the uncertainty propagation should be useful,

and to handle more rigorously stochastic transitions. It is also planned to conduct more comparisons, theoretically and experimentally, of the KTD to other related function approximation schemes for reinforcement learning.

Chapter 2

Experiments

2.1 Standard RL Benchmarks

In this section we use a set of classical reinforcement learning benchmarks in order to compare KTD with state-of-the-art algorithms and to highlight its different aspects, namely the bias caused by stochastic transitions (Boyan chain), non-stationarity handling (Boyan chain and mountain car), value uncertainty used in a form of active learning (inverted pendulum), and sample efficiency (all benchmarks). Compared algorithms are TD, SARSA, Q-learning and (online) LSTD. We do not consider their extensions to eligibility traces, as LSTD performs better than TD(λ) and varying λ has small effect on LSTD(λ) performances [6].

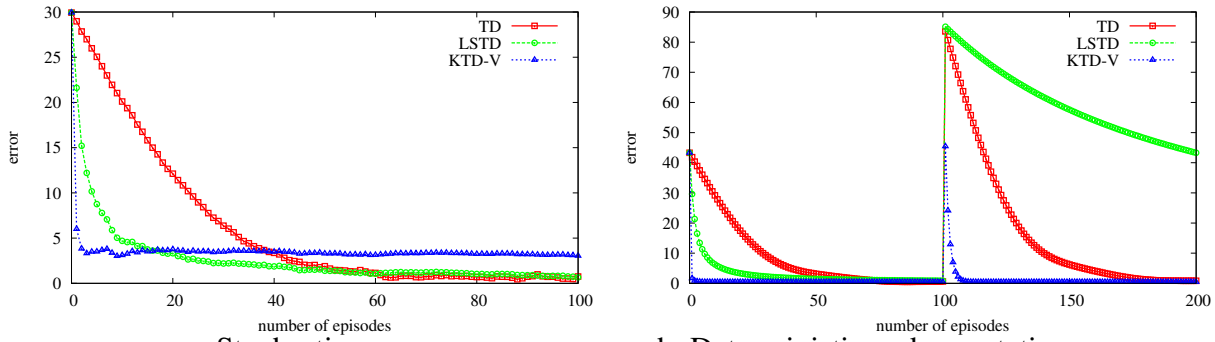
2.1.1 Boyan Chain

The first experiment is the Boyan chain [6]. The aim is to illustrate the bias caused by stochastic transitions and to show sample-efficiency and non-stationarity handling of the 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, the KTD-V is compared to TD [1] and LSTD [4]. The feature vector $\phi(s)$ 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 exactly linear in these features, and the optimal parameter vector is $\theta^* = [-24,-16,-8,0]^T$. To measure the quality of each algorithm the Euclidian distance between the current parameter vector and the optimal one $\|\theta - \theta^*\|$ is computed.

The discount factor γ is set to 1 in this episodic task. For TD, the learning rate is set to $\alpha = 0.1$. For LSTD, the prior is set to $P_{0|0} = I$ where I is the identity matrix. For KTD-V, the same prior



a. Stochastic case. b. Deterministic and non-stationary case.

Figure 2.1: Boyan chain.

is used, the observation noise is set to $P_{n_i} = 10^{-3}$ and the process noise covariance to $P_{v_i} = 0I$. Choosing these parameters requires some practice, but no more than choosing a learning rate for other algorithms. Moreover, online LSTD also needs a prior. For all algorithms the initial parameter vector is set to zero. Fig. 2.1.a shows results.

LSTD converges faster than TD, as expected, and KTD-V converges even faster than LSTD. However it does not converge to the optimal parameter vector, which is explained by the fact that the minimized cost-function is biased. This bias was identified in the thoretical study and a solution was proposed resulting in the XKTD algorihtm. This will be discussed later.

Deterministic and Non-Stationary Case

The Boyan chain is thus made deterministic by setting the probability of transiting from s^i to s^{i-1} to 1 (and from s^i to s^{i-2} to 0). The KTD-V algorithm is again compared to LSTD and TD. Moreover, to simulate a change in the MDP, and thus non-stationarity, the sign of the reward is switched from the 100^{th} episode. The optimal value function is still linear in the feature vectors, and the optimal parameter vector is $\theta^*_{(-)} = [-35, -23, -11, 0]^T$ before the MDP change, and $\theta^*_{(+)} = -\theta^*_{(-)}$ after. Algorithms parameters are the same, except the procces noise covariance which is set to $P_{v_i} = 10^{-3}I$. Results are presented in Fig. 2.1.b.

Here again KTD-V converges much faster than LSTD and TD, however now to the correct optimal parameter vector. After the change in reward, LSTD is very slow to converge, because of the induced non-stationarity. TD can track the correct parameter vector faster, the learning rate being constant. However, KTD converges again faster.

Thus, KTD-V fails to handle the stochastic case as expected, however it converges much faster than LSTD or TD in the deterministic one. Moreover, it handles well non-stationarity. This can be useful if the MDP is non-stationary, but also to track the dynamics of the value function when optimistic policy iteration or actor-critic schemes are used, even in a stationary MDP.

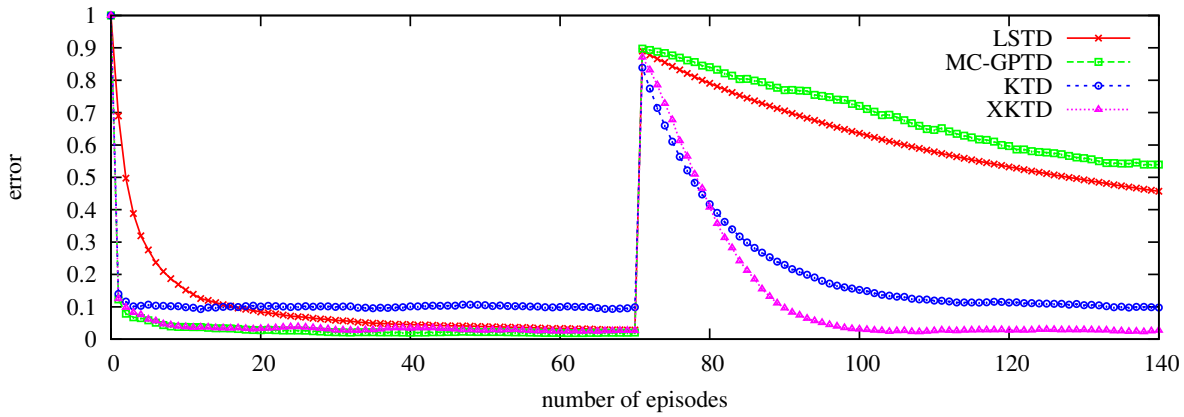


Figure 2.2: Boyan chain.

Stochastic and Non-Stationary Case

An extended version of KTD (XKTD) has been described in the theoretical part of this contribution so as to handle stochastic environments. We compare XKTD to three other second order value function approximation algorithms, namely KTD, LSTD [4] and parametric MC-GPTD [8] (using Gaussian processes). The objective is threefold: showing sample efficiency, demonstrating the bias removal (which is the main purpose of this contribution) and showing non-stationarity handling which remains in this algorithm.

Before the MDP change, KTD variations and MC-GPTD converge faster than LSTD (and equally well). XKTD, as well as LSTD and MC-GPTD, is unbiased, contrary to KTD. Thus XKTD does the job it has been designed for, that is removing the bias due to stochastic transitions. After the MDP change, both LSTD and MC-GPTD fail to track the value function. KTD manages to do it, but it is still biased. XKTD tracks the value function without being biased. Notice that after the MDP change, KTD adaptation is a little bit faster than KTD one. Actually, using a colored observation noise induces a memory effect, like using eligibility traces in classical TD methods. For KTD, the innovation $r_i - \hat{r}_{i|i-1}$ is the TD error, for XKTD it is a combination of such TD errors along the followed trajectory (this can be demonstrated by expanding Kalman equations). This memory effect explains the faster adaptation. This experiments shows that XKTD performs as well as KTD, however without the bias problem, which was the motivation for introducing this new algorithm. It is sample efficient (at least more than LSTD) and it tracks the value function rather than converging to it, which allows handling non-stationarity. Another advantage of XKTD over LSTD and MC-GPTD is its ability to handle nonlinear parameterization. It is not demonstrated empirically here, however it is demonstrated in the white noise case (KTD) by [17], where the handled nonlinearity is the max operator induced by the Bellman optimality equation.

2.1.2 Inverted Pendulum

The second experiment is the inverted pendulum as described in [18]. The goal is here to compare two value-iteration-like algorithms, namely KTD-Q and Q-learning, which aim at learning directly the optimal policy. As far as we know, KTD-Q is the only second order algorithm for Q-function approximation in a value iteration scheme (the major difficulty being to handle the max operator). An active learning-like scheme is also experimented: it uses the uncertainty computed by KTD to speed up convergence.

This task requires balancing a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to. Three actions are allowed: left force (-1), right force ($+1$), or no force (0). The associated state space consists in vertical angle ω and angular velocity $\dot{\omega}$ of the pendulum. Deterministic transitions are computed according to physical dynamics of the system, and depends on action a:

$$\ddot{\omega} = \frac{g\sin(\omega) - \beta ml\dot{\omega}^2\sin(2\omega)/2 - 50\beta\cos(\omega)a}{4l/3 - \beta ml\cos^2(\omega)} \tag{2.1}$$

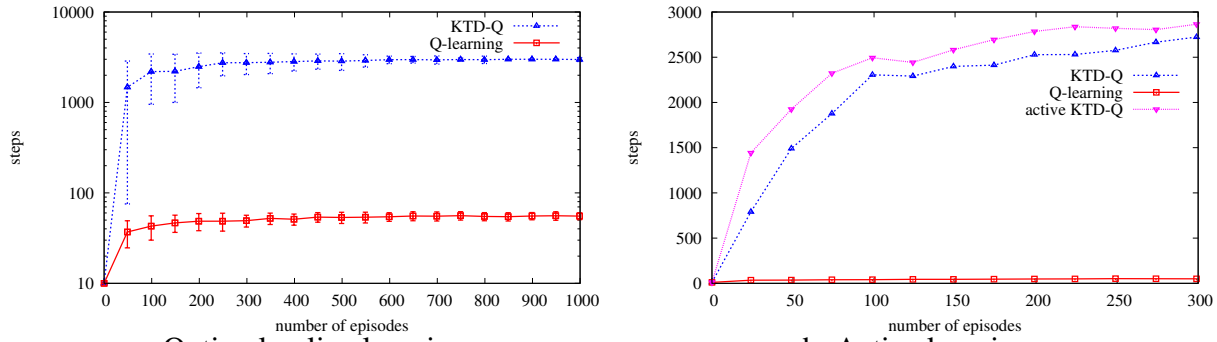
where g is the gravity constant, m and l the mass and the length of the pendulum, M the mass of the cart, and $\beta = \frac{1}{m+M}$. A zero reward is given as long as the angular position is in $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Otherwise, the episode ends and a reward of -1 is given. The parameterization is composed of a constant term and a set of 9 equispaced Gaussian kernels (centered in $\{-\frac{\pi}{4}, 0, \frac{\pi}{4}\} \times \{-1, 0, 1\}$ and with a standard deviation of 1) for each action. Thus there is a set of 30 basis functions. The discount factor γ is set to 0.95.

Learning the Optimal Policy

First, we compare algorithms ability to learn an optimal policy. For Q-learning, the learning rate is set to $\alpha_i = \alpha_0 \frac{n_0+1}{n_0+i}$ with $\alpha_0 = 0.5$ and $n_0 = 200$, according to [18]. For KTD-Q, the parameters are set to $P_{0|0} = 10I$, $P_{n_i} = 1$ and $P_{v_i} = 0I$. For all algorithms the initial parameter vector is set to zero. Training samples are collected online with random episodes. The agent starts in a randomly perturbed state close to the equilibrium $(0,0)$ and then follows a policy that selects actions uniformly at random. The average length of such episodes was about 10 steps, and both algorithms learned from the same trajectories. Results are summarized in Fig. 2.3.a.

For each trial, learning is done over 1000 episodes. Every 50 episodes, learning is frozen and the current policy is evaluated. For this, the agent is randomly initialized in a state close to the equilibrium and the greedy policy is followed until the end of episode; this is repeated 100 times and averaged. Performance is measured as the number of steps in an episode. Maximum number of steps for one episode is bounded by 3000 steps, which corresponds to 5 minutes of balancing the pole without failure. Results in Fig. 2.3.a are averaged over 100 trials and presented in semi-log scale.

KTD-Q learns an optimal policy (that is balancing the pole for the maximum number of steps) asymptotically and near-optimal policies are learned after only a few tens of episodes. With the same number of learning episodes, Q-learning with the same linear parameterization fails to



a. Optimal policy learning. b. Active learning.

Figure 2.3: Inverted pendulum.

learn a policy which balances the pole for more than a few tens of time steps. Similar results for Q-learning are obtained in [18].

A Form of Active Learning

Here we experiment with the active learning scheme proposed in Sec. 1.8.2. The environment is initialized randomly as before. When the system is in a given state, the standard deviation of the Q-function is computed for each action. These deviations are normalized, and the new action is sampled randomly according to the probabilities weighted by the deviations. Thus, an uncertain action will be more likely sampled. The average length of such episodes was about 11 steps, which does not differ much from uniformly random transitions. Consequently this can only slightly help to improve speed of convergence (at most 10%, much less than the real improvement which is about 100%). Results are summarized in Fig. 2.3.b.

For each trial, learning is done over 300 episodes. Fewer episodes are considered to show the speed up of convergence, however both versions of KTD perform as well asymptotically. Every 25 episodes, learning is frozen and the current policy is evaluated as before. Performance is measured as the number of steps of an episode, again for a maximum of 3000 steps. Results in Fig. 2.3.b are averaged over 100 trials. Notice that the scale is no longer logarithmic. It compares KTD-Q with informed transitions ("active" KTD-Q) to KTD-Q with uniformly random learning policy and Q-learning. When comparing the two versions of KTD-Q, it is clear that sampling actions according to uncertainty speeds up convergence. It is almost doubled in the first 100 episodes: for example, a performance of 1500 is obtained after only 25 episodes with active-KTD, whereas it needs about 50 episodes for the classic KTD. Thus the uncertainty information available thanks to the KTD framework can be quite useful for reinforcement learning.

2.1.3 Mountain Car

The last experiment is the mountain car task as described in [1]. The objective here is to illustrate behavior of algorithms in an optimistic policy iteration scheme: learning while controlling

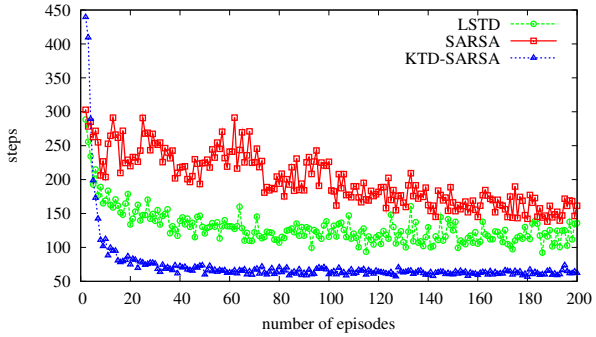


Figure 2.4: Mountain car.

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 state space consists in position and velocity $(x,\dot{x}) \in [-1.2, 0.5] \times [-0.07, 0.07]$. The three possible actions are left (-1) or right (+1) moves or do nothing (0). The dynamics of the system are given by:

$$\begin{cases} \dot{x}_{i+1} = \text{bound}\left[\dot{x}_i + 10^{-3}(a_i - 2.5\cos(3x_i))\right] \\ x_{i+1} = \text{bound}\left[x_i + \dot{x}_{i+1}\right] \end{cases} \tag{2.2}$$

where the bound operator enforces position and velocity bounds. When the position reaches the left bound, the velocity is set to zero. When it reaches the right bound, the reward is 0 and the episode ends. The reward is -1 otherwise. 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. Thus there is a set of 30 basis functions.

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. 2.4 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 fewer 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, Chapt. 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 can explain that LSTD fails to learn a near-optimal policy. LSTD has been extended to LSPI [18], 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 KTD-SARSA.

2.1.4 Conclusion

The KTD framework has been extensively experimented on standard reinforcement learning algorithms. The bias caused by stochastic transitions has been illustrated, and advantages of the KTD concerning non-stationarity and uncertainty handling as well as sample efficiency and ability to control a system have been experimentally demonstrated. The XKTD algorithm has been introduced to handle stochastic MDPs which is a requirement for spoken dialogue systems and it has been demonstrated that this algorithm is actually unbiased. The KTD framework thus compares favorably to state-of-the-art algorithms

Chapter 3

Implementation

3.1 Reinforcement Learning Library

Reinforcement Learning (RL) is a set of techniques and paradigms that are formalized with concepts like observations, states, actions, reward, transitions, etc. As resolution techniques rely on this formalism, they are expressed on rather abtract problems, independently from any real instantiation. The purpose of the RL library is actually to address two objectives. First, it keeps the RL concepts at their abstract level, and second, it allows a dedicated instantiation of any problem resolution. To cope with these two apparently opposite objectives, but also in order to have a high performance, the RL library has been written in C++, using templates rather than inheritance, since the latter are some high-level macros that rewrite a dedicated code.

With the library, for example, a RL designer can take a ready-to-use resolution algorithm involving the regression of the Q-value, being free to use any representation of the Q-function, any regression technique, and any kind of problem to solve, since the requirements for the different elements are reduced to a few type and method definitions. The design is thus more flexible, and oriented to the inclusion of pre-existing simulators, pre-existing regression tools, etc....

Currently, Q-Learning, Sarsa, Fitted-Q, Value Iteration, eligibility traces and KTD-Q are available in the library, as well as a mountain car simulator. Other algorithms are at work, as well as other benchmark problems.

Jointly to the core RL library, we have provided an interface to the DIPPER dialogue manager, as well as a KTD library. The first allows to apply the RL available algorithms to DIPPER, and the second allows any ”naive” designers to take benefits from using the KTD regression method developed in the IMS team. Multilayer neural networks regressors, whose learning is driven by KTD, are also provided.

The RL and KTD libraries are licenced under Lesser-GPL, their sources and documentations are available on the software pages of the IMS team web site kttp://ims.metz.supelec.fr.

3.2 JNI Interface for DIPPER

DIPPER is a modular dialogue manager developed in EDIN and based on OAA (Open Agent Architecture) [19]. J-DIPPER (Java DIPPER) implements a functional programming language (consisting of Prolog like ”u-rules”) which carries out updates over sets of Information State (IS) objects. It is typically controlled from a Graphical User Interface (GUI) which allows control over program execution, i.e. execution of single update steps or continuous running of update steps.

In its standard implementation, J-DIPPER is designed to request actions to perform to a policy learner. Yet, the RL lib has been developed to learn policies by interacting with an environment as in the standard representation of a reinforcement learning problem. From a programming point of view, this makes the environment a client to which actions are submitted and providing states and rewards in return. J-DIPPER had to be modified so as to create a client version of it.

To create a client version of J-DIPPER, control from the GUI was disabled and the step update method ”step()” was re-factorised to allow execution to be ceded to a parent process at various points. A DipperClient class was then created which wraps the newly factorised J-DIPPER methods and implements the agreed interface, see appendix A.

Implementation of the RL lib interface for DIPPER can be obtained on Supelec’s Foundry : http://foundry.supelec.fr. This interface allows the RL lib interacting with the DIPPER dialogue manager developed in JAVA by the University of Edinburgh.

Appendix A - DIPPER Client Interface

The client version of DIPPER implements the following methods.

Definition of RL - DIPPER (Client) / OAA User Simulation Interfaces

DIPPER (Client) interface for Reinforcement Learning (RL)
version 0.1, 1st July 2009

DIPPER (Client) to expose the following methods for accessing data about the state and executing actions:

method name	*arguments*	*return value*	*blocking/non-blocking*	*comments*
DipperDmClient	String isFileName, String uruleFileName, boolean startGUI	void	blocking until initialization complete and ready for first "executeAction"	constructor
setStateSummarizationMethod	String classNameOfMethod	boolean classFound	blocking until summarization class instantiated	select the summarization method (default is max slots)
getSummarizedStateObjectStructure	-	String objectStructure	blocking	get the structure used to report the summarized states, e.g. an array of N integers.
getSummarizedState	-	Object (with structure as indicated by getSummarizedStateObjectStructure)	blocking	get the current state
setActionSetMapping	String classNameOfMethod	boolean classFound	blocking	select the method which maps RL actions to concrete dialogue acts
getActionSet	-	String[] actionSet	blocking	list the set of possible actions afforded by the selected action mapping set. (Could vary from state to state - i.e. be limited to those which can executed in this state.)
executeAction	String act	boolean updateSucceeded	blocking until new state computed	execute action "act" selected from set of actions as provided by getActionSet
getCurrentNumberOfISNodes	-	int numberOfISNodes	blocking	get the current number of IS nodes (leaves of the belief state tree)

getISNodeAttributeNames	-	List<String> listOfAttributeNames	blocking	get attribute list for IS nodes
getISNodeAttributeValue	int nodeNumber, String attributeName	String value	blocking	get the value for a specified IS node and attribute. For a list of IS attributes which form part of this interface see below.
statusOkay	-	boolean statusOkay	blocking	get DipperDmClient status. This is the same as returned by executeAction. This method is useful to check initilisation went okay.
close	-	void	blocking	closes DipperDmClient (closes the OAA connection and the GUI if it was started)

IS attributes which are potentially useful for the getISNodeAttributeValue interface:

- "is:simulatedUserGoal" - Simulated user's goal. Populated after the first response of the simulated user at the start of each dialogue. Format is list of slots and associated slot values, e.g. [[slot_1,slot_2,slot_3],[french,cheap,central]]. NB slot's and values can be listed in any order but always match each other [[slot_3,slot_2,slot_1],[central,cheap,french]].
- "is:dialogueLength" - Number of *system* dialogue acts in this dialogue.
- "is:dialogueCycle" - Dialogue number (1 more than the number of completed dialogues).
- "is:presentedSlotValues" - The values presented to the user. When running with the simulated user this attribute is populated after the "closingDialogue" move. Format is same as "is:simulatedUserGoal", e.g. [[slot_3,slot_2,slot_1],[central,cheap,french]], though ordering may well differ.

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 3rd edition, March 1998.

[2] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, May 1996.

[3] Ralph Schoknecht. Optimality of Reinforcement Learning Algorithms with Linear Function Approximation. In *Conference on Neural Information Processing Systems (NIPS 15)*, 2002.

[4] Steven J. Bradtke and Andrew G. Barto. Linear Least-Squares Algorithms for Temporal Difference Learning. *Machine Learning*, 22(1-3):33–57, 1996.

[5] Leemon C. Baird III. Residual Algorithms: Reinforcement Learning with Function Approximation. In *International Conference on Machine Learning (ICML 95)*, pages 30–37, 1995.

[6] Justin A. Boyan. Technical Update: Least-Squares Temporal Difference Learning. *Machine Learning*, 49(2-3):233–246, 1999.

[7] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[8] Yaakov Engel. *Algorithms and Representations for Reinforcement Learning*. PhD thesis, Hebrew University, April 2005.

[9] David Choi and Benjamin Van Roy. A Generalized Kalman Filter for Fixed Point Approximation and Efficient Temporal-Difference Learning. *Discrete Event Dynamic Systems*, 16:207–239, 2006.

[10] Chee Wee Phua and Robert Fitch. Tracking Value Function Dynamics to Improve Reinforcement Learning with Piecewise Linear Function Approximation. In *International Conference on Machine Learning (ICML 07)*, 2007.

[11] Simon J. Julier and Jeffrey K. Uhlmann. Unscented Filtering and Nonlinear Estimation. In *Proceedings of the IEEE*, volume 92, pages 401–422, March 2004.

[12] Matthieu Geist, Olivier Pietquin, and Gabriel Fricout. Bayesian Reward Filtering. In S. Girgin et al., editor, *Proceedings of the European Workshop on Reinforcement Learning (EWRL 2008)*, volume 5323 of *Lecture Notes in Artificial Intelligence*, pages 96–109. Springer Verlag, Lille (France), June 2008.

[13] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley & Sons, 1. auflage edition, August 2006.

[14] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, New York, USA, 1995.

[15] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement Learning with Gaussian Processes. In *Proceedings of International Conference on Machine Learning (ICML-05)*, 2005.

[16] Matthieu Geist, Olivier Pietquin, and Gabriel Fricout. A Sparse Nonlinear Bayesian Online Kernel Regression. In *Proceedings of the Second IEEE International Conference on Advanced Engineering Computing and Applications in Sciences (AdvComp 2008)*, pages 199–204, Valencia (Spain), October 2008.

[17] Matthieu Geist, Olivier Pietquin, and Gabriel Fricout. Tracking in Reinforcement Learning. In *Proceedings of the 16th International Conference on Neural Information Processing (ICONIP 2009)*, Bangkok (Thailande), 2009. Springer LNCS.

[18] Michail G. Lagoudakis and Ronald Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[19] A. Cheyer and D. Martin. The open agent architecture. In *Journal of Autonomous Agents and Multi-agent Systems*, number 4, pages 143–148, 2001.