Bayesian Reward Filtering

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

[1] Supélec
IMS Research Group, Metz, France
{matthieu.geist,olivier.pietquin}@supelec.fr
[2] ArcelorMittal Research,
MCE Department, Maizières-lès-Metz, France
gabriel.fricout@arcelormittal.com

Abstract. A wide variety of function approximation schemes have been applied to reinforcement learning. However, Bayesian filtering approaches, which have been shown efficient in other fields such as neural network training, have been little studied. We propose a general Bayesian filtering framework for reinforcement learning, as well as a specific implementation based on sigma point Kalman filtering and kernel machines. This allows us to derive an efficient off-policy model-free approximate temporal differences algorithm which will be demonstrated on two simple benchmarks.

1 Introduction

Reinforcement learning [1] is a general paradigm in which an agent learns to control a dynamic system only through interactions. A feedback signal is provided to it as a reward information, which is a hint on the quality of the control. Markov Decision Processes (MDP) are a common framework to solve this problem. A MDP is fully described by a tuple $\{S, A, T, R\}$ where S is the state space that can be explored, A is the action set that can be chosen by the agent, T is a family of transition probabilities between states conditioned by the actions and R is a set of expected rewards associated to transitions. This is further explained in section 2.1. In this framework, at each time step k, the system adopts a state s_k. According to this, the agent can chose an action a_k included in a subset of A. This action leads to a transition to state s_{k+1} and to the obtention of a reward r_k, the agent's objective being to maximize the future expected cumulative rewards. Actions can be of two kinds: exploitative or explorative. Exploitative actions are optimal according to the agent's knowledge about the system. Explorative actions aim at increasing the agent's knowledge about the system. This is known as the exploitation vs exploration dilemma. In this paper the knowledge of the environment will be modelled as a Q-function which maps state action pairs to the expected cumulative rewards when following a given associated policy after the first transition. The proposed approach is model-free, no model of transitions and reward distributions is learned or known.

Solutions exist for the reinforcement learning problem with discrete state and action spaces. However they generally do not scale very well and cannot handle

continuous state and/or action spaces. A wide variety of function approximation schemes have thus been applied to reinforcement learning (see [1] as a starting point). This is known as the generalization problem, and we propose to handle it with a Bayesian filtering approach.

The objective of Bayesian filtering [2] is to infer a hidden state sequentially from observations. The state evolution is driven by a possibly nonlinear mapping combined with a random process noise. Observations are linked to hidden states through another possibly nonlinear mapping combined with a random observation noise. Thus hidden states and observations are random variables. Under classical assumptions, the analytical solution to this problem is given by the recursive Bayes equations. As they are usually intractable, specific solutions have been proposed. This is further developed in section 2.2.

In this paper are proposed the premises of a Bayesian filtering framework for reinforcement learning. The general idea is to parameterize the Q-function, and to consider the associated parameter vector as the hidden state of a Bayesian filter. The associated observation equation links the reward to the parameters through the Bellman equation [3]. We propose a specific implementation of this general Bayesian reward filter: the Q-function is expressed as a weighted sum of Gaussian kernels, the prior on these kernels is chosen with a dictionary method [4], and the parameter vector evolution is computed with a sigma point Kalman filter [5].

The idea to use Bayesian filtering for reinforcement learning is not novel, but it has been surprisingly little studied. In [6] a modification of the linear quadratic Gaussian Kalman filter model is proposed, which allows the on-line estimation of optimal control (which is off-line for the classical one). In [4] Gaussian processes are used for reinforcement learning. This method can be understood as an extension of the Kalman filter to an infinite dimensional hidden state (the Gaussian process), but it can only handle SARSA-like update rules (because of the necessary linearity of the observation equation), contrarily to the proposed contribution, which can be seen as a nonlinear extension of the parametric case developed in [4]. In [7] a Kalman filter bank is used to find the parameters of a piecewise linear approximation of the value function.

The rest of this paper is organized as follows. First the necessary background is briefly presented. Then the proposed general framework and a specific implementation are explained. It is followed by the first results on two benchmarks: the wet-chicken and the mountain car problems. Eventually we conclude and sketch our future works.

2 Background

In this section will be presented the classical reinforcement learning formalism and the Q-learning algorithm, as well as the general Bayesian filtering paradigm and more specific solutions. Through the rest of the paper, a variable x will denote a column vector or a scalar, which should be clear from the context.

2.1 Reinforcement Learning

A Markov Decision Process (MDP) consists of a state space S, an action space A, a Markovian transition probability $p : S \times A \rightarrow \mathcal{P}(S)$ and a bounded reward function $r : S \times A \times S \rightarrow \mathbb{R}$. A policy is a mapping from state to action space: $\pi : S \rightarrow A$. At time step k, the system is in a state s_k, the agent chooses an action $a_k = \pi(s_k)$, and the system is then driven in a state s_{k+1} following the conditional probability distribution $p(.|s_k, a_k)$. The agent receives the associated reward $r_k = r(s_k, a_k, s_{k+1})$. Its goal is to find the policy which maximizes the expected cumulative rewards, that is the quantity $E_\pi[\sum_{k\in\mathbb{N}} \gamma^k r(S_k, A_k, S_{k+1})|S_0 = s_0]$ for every possible starting state s_0, the expectation being over the state transitions taken upon executing π, where $\gamma \in [0, 1[$ is a discount factor.

A classical approach to solve this optimization problem is to introduce the Q-function defined as:

$$Q^\pi(s, a) = \int_S p(z|s, a)\Big(r(s, a, z) + \gamma Q^\pi(z, \pi(z))\Big)dz$$

It is the expected cumulative rewards by taking action a in state s and then following the policy π. The optimality criterion is to find the policy π^* (and associated Q^*) such that for every state s and for every policy π, $\max_{a\in A} Q^*(s, a) \geq \max_{a\in A} Q^\pi(s, a)$. The optimal Q-function Q^* satisfies Bellman's equation:

$$Q^*(s, a) = \int_S p(z|s, a)\left(r(s, a, z) + \gamma \max_{b\in A} Q^*(z, b)\right)dz$$

In the case of discrete and finite action and state spaces, the Q-learning algorithm provides a solution to this problem. Its principle is to update a tabular approximation of the optimal Q-function after each transition (s, a, r, s'):

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha\left(r + \gamma \max_{b\in A} \hat{Q}(s', b) - \hat{Q}(s, a)\right)$$

where α is a learning rate. An interesting fact is that the Q-learning is an off-policy algorithm, that is it allows to learn the optimal policy (from the learned optimal Q-function) while following a suboptimal one, given that it is sufficiently explorative. The proposed contribution can be seen as an extension of this algorithm to a Bayesian filtering framework (however with other advantages). See [1] for a comprehensive introduction to reinforcement learning, or [8] for a more formal treatment.

2.2 Bayesian Filtering

The problem of Bayesian filtering is to sequentially infer a hidden state x_k given past observations $y_{1:k} = \{y_1, y_2, \ldots, y_k\}$. It can be expressed in its state-space formulation:

$$\begin{aligned} x_{k+1} &= f_k(x_k, v_k) \\ y_k &= g_k(x_k, n_k). \end{aligned}$$

The state evolution is driven by the mapping f_k and the process noise v_k. The observation y_k is a function of the state x_k, corrupted by an observation noise n_k. Some prior knowledge about the system evolution is necessary (such as the possibly nonlinear and non stationary mappings). The general principle is to predict the new state x_k given the previous observations $y_{1:k-1}$, and to correct it given the new observation y_k, according to prediction and correction equations:

$$p(X_k|Y_{1:k-1}) = \int_{\mathcal{X}} p(X_k|X_{k-1})p(X_{k-1}|Y_{1:k-1})dX_{k-1} \text{ (prediction)},$$
$$p(X_k|Y_{1:k}) = \frac{p(Y_k|X_k)p(X_k|Y_{1:k-1})}{\int_{\mathcal{X}} p(Y_k|X_k)p(X_k|Y_{1:k-1})dX_k} \text{ (correction)}.$$

These equations are generally intractable. If the mappings are linear and if the noises n_k and v_k are Gaussian, the optimal solution is given by the Kalman filter: quantities of interest are random variables, and inference (that is prediction of these quantities and correction of them given a new observation) is done online by propagating sufficient statistics through linear transformations. If the mappings are nonlinear (but the noises are still Gaussian), a first solution is to linearize them around the state: it is the principle of the Extended Kalman Filter (EKF), and sufficient statistics are still propagated through linear transformations. Another approach is the Sigma Point Kalman Filter (SPKF) framework [5]. The basic idea is that it is easier to approximate a probability distribution than an arbitrary nonlinear function. A set of so-called sigma points are deterministically computed from the hidden state sufficient statistics. These points are representative of the distribution of interest. The nonlinear mappings of these points are then computed and used to compute sufficient statistics of interest for prediction and correction equations. Algorithm 1 sketches a SPKF update in the case of additive noise, based on the state-space formulation, and using the standard Kalman notations: $x_{k|k-1}$ denotes a prediction, $x_{k|k}$ an estimate (or a correction), $P_{x,y}$ a covariance matrix, $\bar{n}_k$ a mean and k is the discrete time index. The reader can refer to [5] for details. This filter will be used in the specific implementation of the proposed framework. A last method is the particle filter (or sequential Monte Carlo). It is a numerical approach designed for nonlinear mappings and nongaussian noises. See [2] for a survey on Bayesian filtering.

3 The general framework

Formulation of value function estimation as a Bayesian filtering problem has been already proposed by Engel [4], however with a linear observation model: it allows nonparametric representation of the value function (Gaussian processes) but is mainly dedicated to the evaluation of the followed policy value (on-policy aspect). Our contribution can be seen as a nonlinear extension of Engel's work, however with a parametric representation constraint.

Algorithm 1 SPKF Update

Inputs: $x_{k-1|k-1}$, $P_{k-1|k-1}$
Outputs: $x_{k|k}$, $P_{k|k}$

Sigma points computation:
Compute deterministically the sigma point set $X_{k-1|k-1}$ from $x_{k-1|k-1}$ and $P_{k-1|k-1}$;

Prediction step:
Compute $X_{k|k-1} = f_k(X_{k-1|k-1}, \bar{v}_k)$;
Compute $x_{k|k-1}$ and $P_{k|k-1}$ from $X_{k|k-1}$ and the process noise covariance;

Correction step:
Observe y_k;
$Y_{k|k-1} = g_k(X_{k|k-1}, \bar{n}_k)$;
Compute $y_{k|k-1}$, $P_{y_{k|k-1}}$ and $P_{x_{k|k-1},y_{k|k-1}}$ from $X_{k|k-1}$, $Y_{k|k-1}$ and the observation noise covariance;
$K_k = P_{x_{k|k-1},y_{k|k-1}} P_{y_{k|k-1}}^{-1}$; $\{Kalman\ gain\}$
$x_{k|k} = x_{k|k-1} + K_k(y_k - y_{k|k-1})$;
$P_{x_{k|k}} = P_{x_{k|k-1}} - K_k P_{y_{k|k-1}} K_k^T$;

The Bellman equation can be written as:

$$Q^*(s,a) = r(s,a,s') + \gamma \max_{b \in A} Q^*(s',b) - n_{s,a}(s')$$

$$\text{with } n_{s,a}(s') = \int_S p(z|s,a)\Big\{r(s,a,s') - r(s,a,z) + \gamma\left(\max_{b\in A} Q^*(s',b) - \max_{b\in A} Q^*(z,b)\right)\Big\}dz.$$

Being the nonlinear mapping of the random variable $(S'|S = s, A = a)$ of law $p(.|s,a)$, $n_{s,a}$ is indeed a random variable. It can be easily shown that this random variable is centered with finite variance, that is, $r_{\max}$ being the bound on the reward function:

$$\int_S n_{s,a}(z)p(z|s,a)dz = 0 \quad \text{and} \quad \int_S n_{s,a}^2(z)p(z|s,a)dz \leq \left(\frac{r_{\max}}{1-\gamma}\right)^2$$

For an observed transition (s,a,r,s'), the Bellman equation can be rewritten:

$$r(s,a,s') = Q^*(s,a) - \gamma \max_{b\in A} Q^*(s',b) + n_{s,a}(s')$$

This form is very "Q-learning like" and is of primary importance for the proposed framework.

Suppose that the Q-function is parameterized (either linearly or nonlinearly) by a vector θ. The aim is to find a good approximation $\hat{Q}_\theta$ of the optimal Q-function Q^* by observing transitions (s,a,r,s'). This reward regression problem is cast into a state-space representation. For an observed transition

(s_k, a_k, r_k, s'_k), it is written as:

$$\begin{aligned}\theta_{k+1} &= \theta_k + v_k \\ r_k &= \hat{Q}_{\theta_k}(s_k, a_k) - \gamma \max_{a \in A} \hat{Q}_{\theta_k}(s'_k, a) + n_k.\end{aligned}$$

Here v_k is an artificial process noise and n_k a centered observation noise including all the stochasticity of the MDP. The framework is thus posed, but is far from being solved. The observation equation is nonlinear (because of the max operator), that's why classical methods such as the Kalman filter cannot be used. Formally, the process noise is null, nevertheless introducing an artificial noise can improve the stability and convergence performances of the filter. A (possibly adaptive) observation noise has to be chosen also, as well as the parameterization for the Q-function. Last but not least, as for each Bayesian approach, a prior on parameters has to be set. A specific implementation of this Bayesian filtering framework is proposed in the next section.

4 Practical solution

A kernel parameterization is chosen for the Q-function, because of its expressiveness given by the Mercer theorem [9]. A kernel is a continuous, symmetric and positive definite function. Each kernel is a dot product in a (generally higher) dimensional space. More precisely, for each kernel K, there exists a mapping φ from the working space $\mathcal{X}$ to a so-called feature space $\mathcal{F}$ such that $\forall x, y \in \mathcal{X}$, $K(x, y) = \langle \varphi(x), \varphi(y) \rangle$. This fact is important for the initialisation of the proposed algorithm.

More precisely Gaussian kernels are chosen, and their mean and deviation are considered as parameters:

$$\hat{Q}_\theta(s, a) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i^s}(s, s_i) K_{\sigma_i^a}(a, a_i)$$

$$\text{with } K_{\sigma_i^x}(x, x_i) = \exp\left(-(x - x_i)^T (\Sigma_i^x)^{-1} (x - x_i)\right),$$

$$\text{where } x = s, a,\ \Sigma_i^x = \operatorname{diag}(\sigma_i^x)^2,$$

$$\text{and } \theta = [(\alpha_i)_{i=1}^p, (s_i^T)_{i=1}^p, (a_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p, ((\sigma_i^a)^T)_{i=1}^p]^T$$

The operator diag applied on a column vector gives a diagonal square matrix. Note that $K_{(\sigma_i^s, \sigma_i^a)}([s^T, a^T]^T, [s_i^T, a_i^T]^T) = K_{\sigma_i^s}(s, s_i) K_{\sigma_i^a}(a, a_i)$ is a product of Gaussian kernels, thus it is a (still Gaussian) kernel.

4.1 Dictionary computation

A first problem is to choose the number p of kernel functions and the associated prior. To do this, a prior is first put on the Gaussian widths $\sigma_0^T = [(\sigma_0^s)^T, (\sigma_0^a)^T]$. Then a dictionary method proposed by Engel [4] is used to determine the number of kernels and their prior centers. Consider a kernel K, the associated mapping

φ and a set of points $X = \{x_1, x_2, \dots\}$. The aim of the dictionary method is to compute a set of p points $\mathcal{D} = \{\tilde{x}_1, \dots, \tilde{x}_p\}$ such that $\varphi(\mathcal{D})$ is an approximate basis of $\varphi(X)$.

This procedure is iterative. Suppose that samples $x_1, x_2, \dots$ are sequentially observed. At time k, a dictionary $\mathcal{D}_{k-1} = (\tilde{x}_j)_{j=1}^{m_{k-1}} \subset (x_j)_{j=1}^{k-1}$ is available where by construction feature vectors $\varphi(\tilde{x}_j)$ are approximately linearly independent in $\mathcal{F}$. Sample x_k is then observed, and is added if $\varphi(x_k)$ is linearly independent on $\mathcal{D}_{k-1}$. To test this, weights $w = (w_1, \dots, w_{m_{k-1}})^T$ have to be computed so as to satisfy

$$\delta_k = \min_w \left\| \sum_{j=1}^{m_{k-1}} w_j \varphi(\tilde{x}_j) - \varphi(x_k) \right\|^2$$

A predefined threshold ν determining the quality of the approximation (and consequently the sparsity of the dictionary) is used. If $\delta_k > \nu$, $x_k = \tilde{x}_{m_k}$ is added to the dictionary, otherwise $\varphi(x_k)$ can be written as:

$$\varphi(x_k) = \sum_{i=1}^{m_{k-1}} w_i \varphi(\tilde{x}_i) + \varphi_{res} \text{ with } \|\varphi_{res}\|^2 \leq \nu$$

Defining the $m_{k-1} \times m_{k-1}$ matrix $\tilde{K}_{k-1}$ and the $m_{k-1} \times 1$ vector $\tilde{k}_{k-1}(x)$ as

$$\left(\tilde{K}_{k-1}\right)_{i,j} = K(\tilde{x}_i, \tilde{x}_j) \text{ and } \left(\tilde{k}_{k-1}(x)\right)_i = K(x, \tilde{x}_i)$$

and by using the bilinearity of the dot product and the kernel trick, δ_k can be written as:

$$\delta_k = \min_w \left\{ w^T \tilde{K}_{k-1} w - 2 w^T \tilde{k}_{k-1}^T(x_k) + K(x_k, x_k) \right\}$$

whereof solution is $\delta_k = K(x_k, x_k) - \tilde{k}_{k-1}(x_k) w_k$ with $w_k = \tilde{K}_{k-1}^{-1} \tilde{k}_{k-1}(x_k)$. Moreover there exists a computationally efficient algorithm using the partitioned matrix inversion formula to construct this dictionary. See [4] for details. Practically it is supposed that S and A are compact sets and that their bounds are known. Then the corresponding dictionary is computed in a preprocessing step from N samples uniformly sampled from $S \times A$.

4.2 Gaussian prior

Recall that as for any Bayesian approach, a prior parameter distribution has to be chosen. We state that $\theta_0 \sim \mathcal{N}(\bar{\theta}_0, \Sigma_{\theta_0})$ where

$$\bar{\theta}_0 = [\alpha_0, \dots, \mathcal{D}_s, \mathcal{D}_a, (\sigma_0^s)^T, \dots, (\sigma_0^a)^T, \dots]^T$$
$$\Sigma_{\theta_0} = \text{diag}(\sigma^2_{\alpha_0}, \dots, \sigma^2_{\mu_0^s}, \dots, \sigma^2_{\mu_0^a}, \dots, \sigma^2_{\sigma_0^s}, \dots, \sigma^2_{\sigma_0^a}, \dots)$$

In these expressions, α_0 is the prior mean on kernel weights, $\mathcal{D} = \mathcal{D}_s \times \mathcal{D}_a$ is the set of prior means on kernel centers computed in a preprocessing step with

Algorithm 2 A Bayesian reward filtering algorithm

inputs: ν, N, α_0, σ_0, σ_{α_0}, $\sigma_{\mu_0^x}$, $\sigma_{\sigma_0^x}$, σ_{v_0}, σ_{n_0}
outputs: $\bar{\theta}$, Σ_θ

Compute Engel's dictionary:
$\forall i \in \{1 \dots N\}$, $[s_i^T, a_i^T]^T \sim \mathcal{U}_{\mathcal{S}\times\mathcal{A}}$;
Set $X = \{[s_1^T, a_1^T]^T, \dots, [s_N^T, a_N^T]^T\}$;
$\mathcal{D}$ =`Compute-Dictionary`(X,ν,σ_0)

Initialization:
Initialize $\bar{\theta}_0$, Σ_{θ_0}, R_{n_0}, R_{v_0};

for $k = 1, 2, \dots$ **do**
  Observe $t_k = (s_k, a_k, r_k, s'_k)$;

  SR-CDKF update:
  $[\bar{\theta}_k, \Sigma_{\theta_k}, K_k]$ = `SR-CDKF-Update`$(\bar{\theta}_{k-1},\Sigma_{\theta_{k-1}},t_k,R_{v_{k-1}},R_{n_{k-1}})$;

  Artificial process noise update:
  $R_{v_k} = (\lambda^{-1} - 1)\Sigma_{\theta_k}$;
end for

the dictionary from the prior means on kernel deviations $\sigma_0^T = [(\sigma_0^s)^T, (\sigma_0^a)^T]$, and $\sigma_{\alpha_0}^2$, $\sigma_{\mu_0^x}^2$, $\sigma_{\sigma_0^x}^2$ are respectively the prior variances on kernel weights, centers and deviations, for $x = s, a$. All these parameters (except the dictionary) have to be initialized beforehand, using the problem's prior knowledge. Let $q = (2(n_a + n_s) + 1)\, p$, with n_s (resp. n_a) being the dimension of the state space (resp. the action space). Note that $\bar{\theta}_0 \in \mathbb{R}^q$ and $\Sigma_{\theta_0} \in \mathbb{R}^{q\times q}$. A prior on noises also has to be put: more precisely, $v_0 \sim \mathcal{N}(0, R_{v_0})$ and $n_0 \sim \mathcal{N}(0, R_{n_0})$, where $R_{n_0} = \sigma_{n_0}^2$.

4.3 Parameters update

Once the parameters are initialized, the parameter vector has still to be updated as new observations (s_k, a_k, r_k, s'_k) are available. A Square-Root Central Difference Kalman Filter (SR-CDKF) is used, which is a specific implementation of the SPKF sketched before. See [5] for further information. The last problem is to choose the artificial process noise. Formally, since the target function is stationary, there is no process noise. However, following [5], an artificial process noise is introduced.

Its covariance is set to a fraction of the parameter covariance, that is

$$R_{v_k} = (\lambda^{-1} - 1)\Sigma_{\theta_k}$$

where $\lambda \in]0, 1]$ $(1 - \lambda \ll 1)$ is a forgetting factor similar to the one from the recursive least-squares (RLS) algorithm. In this paper, the observation noise is constant, that is $R_{n_k} = R_{n_{k-1}}$. The proposed Bayesian reward filtering algorithm is summarized in Algorithm 2.

4.4 Maximum over action space

Notice that a technical difficulty is to compute the maximum over the actions for the parameterized Q-function. This computation is necessary for the filter update. A first solution is to sample the action space and to compute the maximum over the obtained samples. However this is especially computationally inefficient. The used method is closed to one proposed in [10].

The maximum over action kernel centers is computed: $\mu^a = \text{argmax}_{a_i} \hat{Q}_\theta(s, a_i)$. It serves then as the initialization for the Newton-Raphson method used to find the maximum :

$$a_m \leftarrow a_m - \Big((\nabla_a \nabla_a^T)\hat{Q}_\theta(s,a)\Big)^{-1}_{a=a_m} \nabla_a \hat{Q}_\theta(s,a)\Big|_{a=a_m}$$

If the Hessian matrix is singular, a gradient ascent/fixed point scheme is used:

$$a_m \leftarrow a_m + \nabla_a \hat{Q}_\theta(s,a)\Big|_{a=a_m}$$

The obtained action a_m is considered as the action which maximizes the parameterized Q-function.

5 Preliminary results

The proposed approach is demonstrated on two problems. First, the "wet-chicken" task is a continuous state and action space and stochastic problem. Second, the "mountain car" problem is a continuous state, discrete action and deterministic problem. The latter will require an hybrid parametrization. Let's first discuss the choice of parameters.

5.1 Choice of parameters

For both tasks the reinforcement learning discount factor was set to $\gamma = 0.9$. The dictionary's sparsity factor was set to $\nu = 0.9$. Similarly to the recursive least-squares algorithm, the adaptive process noise covariance was set to a high value, such that $\lambda^{-1} - 1 \simeq 10^{-6}$.

The initial choice of kernel deviations is problem dependant. However a practical good choice seems to take a fraction of the quantity $x(j)_{\max} - x(j)_{\min}$ for the kernel deviation associated to $x(j)$, the j^{th} component of the column vector x, $x(j)_{\max}$ and $x(j)_{\min}$ being the bounds on the values taken by the variable $x(j)$. We supposed the prior kernel weights to be centered, and we set the associated standard deviation to a little fraction of the theoretical bound on Q-function, that is $\frac{r_{\max}}{1-\gamma}$. Because of geometry of Gaussian distributions, we suppose that centers provided by the dictionary are approximately uniformly distributed, and we set the prior deviation of the j^{th} component of the vector x to a little fraction of $(x(j)_{\max} - x(j)_{\min})p^{-\frac{1}{ns+na}}$, with the convention that for discrete spaces $n = 0$. Finally we set up the deviation of the prior kernel deviations to a little

fraction of them. Otherwise speaking, we set $\sigma_{\sigma_0^{x(j)}}$ to a little fraction of $\sigma_0^{x(j)}$ for the j^{th} component of x.

To sum up, the Gaussian prior on parameterization is chosen such that:

$$\sigma_0^{x(j)} \propto x(j)_{\max} - x(j)_{\min}$$
$$\mu_{\alpha_0} = 0 \text{ and } \sigma_{\alpha_0} \propto \frac{r_{\max}}{1-\gamma}$$
$$\sigma_{\mu_0^{x(j)}} \propto \frac{x(j)_{\max} - x(j)_{\min}}{\sqrt[(n_s+n_a)]{p}}$$
$$\sigma_{\sigma_0^{x(j)}} \propto \sigma_0^{x(j)}$$

5.2 Wet-chicken

In the wet-chicken problem (inspired by [11]), a canoeist has to paddle on a river until reaching a waterfall. It restarts if it falls down. Rewards increase linearly with the proximity of the waterfall, and drop off for falling. Turbulences make the transition probabilistic. More formally, the state space is $S = [0, 10]$ (10 being the waterfall position), the action space $A = [-1, 1]$ (continuously from full backward padding to full forward padding), the transition is $s' = s + a + c$ with $c \sim \mathcal{N}(0, \sigma_c)$, $\sigma_c = 0.3$ and the associated reward is equal to $r = \frac{s'}{10}$. If $s' \geq 10$ the canoeist falls, the associated reward is $r = -1$ and the episode ends.

To test the proposed framework, random transitions are uniformly sampled and used to feed the filter: at each time step k, a state s_k and an action a_k are uniformly sampled from $S \times A$, and used to generate a (random) transition to s'_k, with associated reward r_k, and the transition (s_k, a_k, s'_k, r_k) is the input of the algorithm. The results are shown on Fig. 1. For each run of the algorithm and every 250 samples, the expected cumulative rewards for the current policy has been computed as an average of cumulative rewards over 1000 episodes which were randomly (uniform distribution) initiated (thus the average is done over starting states and stochasticity of transitions). Notice that the lifetime of the agent (the duration of an episode) was bounded to 100 interactions with its environment. We then computed a two dimensional histogram of those averaged cumulative rewards over 100 different runs of the algorithm. In other words, we show the distribution of cumulative rewards over different runs of the algorithm as a function of the number of observed transitions. The bar on the right shows the percentages associated to the histogram.

The optimal policy has an averaged cumulative rewards of 6 (computed with value iteration over a very finely sampled state-action space). One can see on Fig. 1 that the proposed algorithm can learn near optimal policies. After 1000 samples some of policies can achieve a score of 5 (84% of the optimal policy), which is achieved by a majority of the policies after 3000 samples. After 7000, very close to optimal policies were found in almost all runs of the algorithm (the mode of the associated distribution is at 5.85, that is 98% of the optimal policy). To represent the approximated Q-function, 7.7 ± 0.7 kernel functions were used, which is relatively few for such a problem (from a regression perspective).

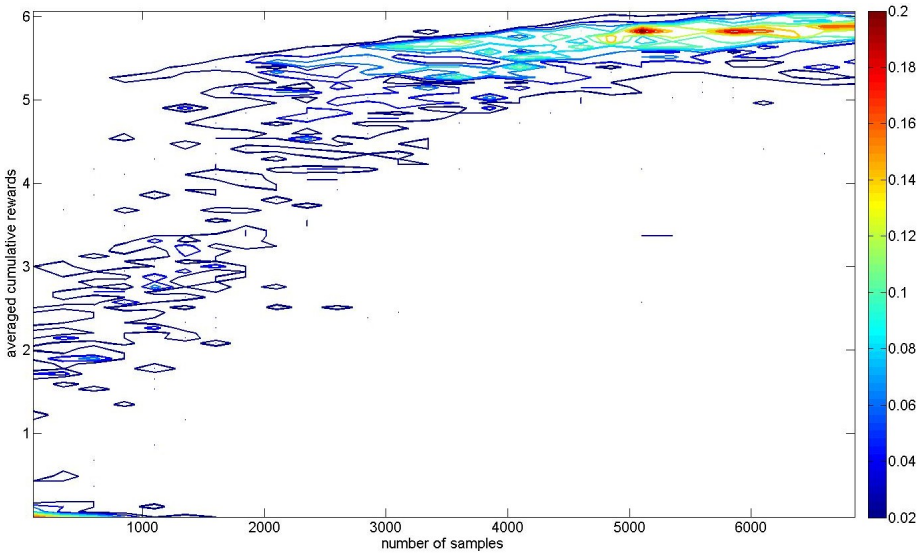

Fig. 1. Two-dimensional histogram of averaged cumulative rewards for the wet-chicken problem (100 runs).

Two remarks of interest have to be made on this benchmark. First, the observation noise is input-dependant, as it models the stochasticity of the MDP. Recall that here we have chosen a constant observation noise. Secondly, the noise can be far to Gaussianity. For example, in the proximity of the waterfall it is bimodal because of the shift of reward. Recall that the proposed filter assumes Gaussianity of noises. Thus we can conclude that the proposed approach is quite robust, and that it achieves good performance considering that the observations were totally random (off-policy aspect).

5.3 Mountain car

The second problem is the mountain-car task. A underpowered car has to go up a steep mountain road. The state is 2-dimensional (position and velocity) and continuous, and there are 3 actions (forward, backward and zero throttle). The problem full description is given in [1]. A null reward is given at each time step, and $r = 1$ is given when the agent reaches the goal.

A first problem is to find a parameterization for this task. The proposed one is adapted for continuous problems, not hybrid ones. But this approach can be easily extended to continuous states and discrete actions tasks. A simple solution consists in having a parameterization for each discrete action, that is a parametrization of the form $\theta = [\theta^{a_1}, \theta^{a_2}, \theta^{a_3}]$ and an associated Q-function $Q_\theta(s, a) = Q_{\theta^a}(s)$. But one can notice that for a fixed state and different actions

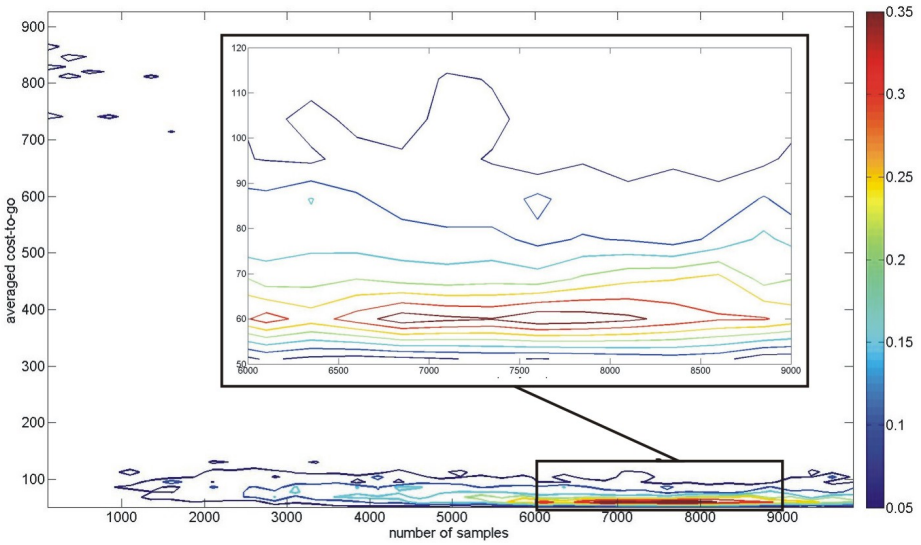

Fig. 2. Two-dimensional histogram of the averaged cost-to-go for the mountain-car problem (100 runs).

the Q-values will be very close. In other words $Q^*(s, a_1)$, $Q^*(s, a_2)$ and $Q^*(s, a_3)$ will have similar shapes, as functions over the state space. Thus consider that the weights will be specific to each action, but the kernel centers and deviations will be shared over actions. More formally the parameter vector is

$$\theta = [(\alpha_i^{a_1})_{i=1}^p, (\alpha_i^{a_2})_{i=1}^p, (\alpha_i^{a_3})_{i=1}^p, (s_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p]^T$$

the notation being the same as in the previous sections.

As for the wet-chicken problem, the filter has been fed with random transitions. Results are shown on Fig. 2, which is a two-dimensional histogram similar to the previous one. The slight difference is that the performance measure is now the "cost-to-go" (the number of steps needed to reach the goal). It can be linked directly to the averaged cumulative rewards, however we think it is more meaningful. For each run of the algorithm and every 250 samples, the expected cost-to-go for the current policy has been computed as an average of 1000 episodes which were randomly initiated (average is only done over starting states here, as transitions are deterministic). The lifetime of the agent was bounded to 1000 interactions with its environment. The histogram is computed over 100 runs.

The optimal policy has an averaged cost-to-go of 55 (computed with value iteration over a very finely sampled state space). One can see on Fig. 2 that the proposed algorithm can find near optimal policies. After 1500 samples most of policies achieve a cost-to-go smaller than 120. After 6000 samples, policies very

close to the optimal one were found in almost all runs of the algorithm (the mode of the associated distribution is at 60). To represent the approximated Q-function, 7.5 ± 0.8 kernel functions were used, which is relatively few for such a problem (from a regression perspective).

This problem is not stochastic, but informative rewards are very sparse (which can cause the Kalman gain to converge too quickly), transitions are nonlinear and rewards are binary. Despite this, the proposed filter exhibits good convergence. Once again we can conclude that the proposed approach achieves good results considering the task at hand.

5.4 Comparison to other methods

For now the proposed algorithm treats the different control tasks as regression problems (learning from random transitions), thus it is ill comparable to state-of-the-art reinforcement learning algorithms which learns from trajectories. Nevertheless we argue that the quality of learned policy is comparable to state-of-the art methods. Measuring this quality depends on the problem settings and on the measure of performance, however the Bayesian reward filter finds very close to optimal policies. See [11] for example.

In most approaches, the system is controlled while learning, or for batch methods observed samples come from a suboptimal policy. In the proposed experiments, totally random transitions are observed. However for the mountain-car problem it is often reported that at least a few hundreds of episodes are required to learn a near-optimal policy (see for example [1]), and each episode may contain from a few tens to hundreds steps (this depends on the quality of the current control). In the proposed approach a few thousands of transitions have to be observed in order to obtain a near optimal policy. This is roughly the same order of magnitude for convergence speed.

We have demonstrated the algorithm on two simple benchmarks, notwithstanding it is planned to conduct more extensive comparison to other approaches when a control organ is added to the proposed framework.

6 Conclusion and future works

We have introduced a general Bayesian filtering framework for reinforcement learning. By observing rewards (and associated transitions) the filter is able to infer a near-optimal policy (through the parameterized Q-function). A specific implementation, based on sigma point Kalman filtering, on kernel machines and on a sparsification method has been described. It has been tested on two reinforcement learning benchmarks, each one exhibiting specific difficulties for the algorithm. This off-policy Bayesian reward filter has been shown to be efficient on this two continuous tasks.

However, this paper did not demonstrate all the potentialities of the proposed framework. The Bayesian filtering approach allows to derive uncertainty information over estimated Q-function which can be used to handle the exploration-exploitation dilemma, in the spirit of [12] or [13]. This could allow to speed-up

and to enhance learning. Moreover, we think that the partial observability problem can be quite naturally embedded in such a Bayesian filtering framework, as the Q-function can be considered as a function over probability densities.

In our future works we aim to treat the two aforementioned problems, our main goal being to handle what we consider to be the three major reinforcement learning problems (generalization, partial observability and exploration-exploitation trade-off) at once. However there are other points of interest. A more efficient adaptive process noise could speed up and improve the convergence's robustness of the filter, and adaptive observation noise could be necessary for some more complex tasks, as it is formally input-dependent. Other parametrization for the Q-function can also be considered. Moreover, there are a few technical issues, as the search of maxima over actions. Last but not least theoretical convergence may be a problem and should be studied.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning). 3rd edn. The MIT Press (March 1998)
2. Chen, Z.: Bayesian Filtering : From Kalman Filters to Particle Filters, and Beyond. Technical report, Adaptive Systems Lab, McMaster University (2003)
3. Bellman, R.: Dynamic Programming. sixth edn. Dover Publications (1957)
4. Engel, Y.: Algorithms and Representations for Reinforcement Learning. PhD thesis, Hebrew University (April 2005)
5. van der Merwe, R.: Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models. PhD thesis, OGI School of Science & Engineering, Oregon Health & Science University, Portland, OR, USA (April 2004)
6. Szita, I., Lőrincz, A.: Kalman filter control embedded into the reinforcement learning framework. Neural Comput. **16**(3) (2004) 491–499
7. Phua, C.W., Fitch, R.: Tracking Value Function Dynamics to Improve Reinforcement Learning with Piecewise Linear Function Approximation. In: ICML 07. (2007)
8. Bertsekas, D.P.: Dynamic Programming and Optimal Control. 3rd edn. Athena Scientific (1995)
9. Vapnik, V.N.: Statisical Learning Theory. John Wiley & Sons, Inc. (1998)
10. Carreira-Perpinan, M.A.: Mode-Finding for Mixtures of Gaussian Distributions. IEEE Transsactions on Pattern Analalysis and Machine Intelligence **22**(11) (2000) 1318–1323
11. Schneegass, D., Udluft, S., Martinetz, T.: Kernel rewards regression: an information efficient batch policy iteration approach. In: AIA'06: Proceedings of the 24th IASTED international conference on Artificial intelligence and applications, Anaheim, CA, USA, ACTA Press (2006) 428–433
12. Dearden, R., Friedman, N., Russell, S.J.: Bayesian Q-learning. In: AAAI/IAAI. (1998) 761–768
13. Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC Model-Free Reinforcement Learning. In: 23rd International Conference on Machine Learning (ICML 2006), Pittsburgh, PA, USA (2006) 881–888