Reinforcement Learning with a Gaussian Mixture Model

Alejandro Agostini, *Member, IEEE* and Enric Celaya

***Abstract*— Recent approaches to Reinforcement Learning (RL) with function approximation include *Neural Fitted Q Iteration* and the use of *Gaussian Processes*. They belong to the class of *fitted value iteration* algorithms, which use a set of support points to fit the value-function in a *batch* iterative process. These techniques make efficient use of a reduced number of samples by reusing them as needed, and are appropriate for applications where the cost of experiencing a new sample is higher than storing and reusing it, but this is at the expense of increasing the computational effort, since these algorithms are not incremental. On the other hand, non-parametric models for function approximation, like Gaussian Processes, are preferred against parametric ones, due to their greater flexibility. A further advantage of using Gaussian Processes for function approximation is that they allow to quantify the uncertainty of the estimation at each point. In this paper, we propose a new approach for RL in continuous domains based on Probability Density Estimations. Our method combines the best features of the previous methods: it is non-parametric and provides an estimation of the variance of the approximated function at any point of the domain. In addition, our method is simple, incremental, and computationally efficient. All these features make this approach more appealing than Gaussian Processes and fitted value iteration algorithms in general.**

I. Introduction

A crucial issue in Reinforcement Learning (RL) is how to deal with problems whose state and action spaces are continuous, or discrete but very large. In these cases, the application of classical tabular methods to store the Q-value for each possible state-action pair (or the value of each state, in model-based approaches) becomes infeasible. On the other hand, if the number of states is too large, learning about them by visiting them all turns out to be impossible, so that it is necessary to infer the value of a state from the values of similar ones for which experiences have been collected. To achieve this, RL must be used with some form of function approximation providing the necessary compactness in its representation and appropriate generalization on states and actions.

In general, function approximation methods can be classified as parametric and non-parametric [1]. Parametric methods include neural nets, polynomials, and combinations of radial basis functions, among others. They define parameterized families of functions with a *finite* number of parameters (which, in the discrete case, is much lesser than the number of states), and try to find the values of the parameters for which the function best represents the available data. Parametric methods have been extensively used since they allow the application of gradient techniques for parameter optimization. One difficulty with parametric models resides in the selection of the parameterized family: if it is too restrictive, it could not be able to model the data with the necessary accuracy; if it is too general, there is a risk of overfitting the data and provide poor generalization. Non-parametric function approximators, instead, do not fix in advance the number or the nature of the parameters (despite their name, non-parametric approximators usually *have* parameters, but their number is not upper bounded) so that they can be endowed with unrestricted function approximation capabilities. Some examples are Gaussian Processes, tree-based methods, and Mixtures of Gaussians with variable number of units. Since, in general, in a complex RL problem it is not possible to guess what kind of function representation will work, the more flexible non-parametric methods are preferred.

In the last years, different non-parametric function approximators for RL have been proposed. In [2], Rasmussen and Kuss proposed the use of Gaussian Processes (GP) for RL: Using a model-based approach, a number of GPs (one for each dimension of the state space) is used to model the system dynamics, and a further GP represents the value function. In [3], the approach is extended to online learning using Bayesian active learning. An alternative application of GPs to RL is that of Engel et al. [4], who use a GP to directly represent the Q-function in a model-free approach. One benefit of using GPs for function approximation is that, besides providing the expected value of the function, they also provide its variance, what allows to quantify the uncertainty of the predicted value. As pointed out in [5], this information may be very useful to direct the exploration in RL.

All of these GP-based RL algorithms fall in the class of the so-called *fitted value iteration* algorithms [6], which, in order to approximate the desired function, take a finite number of samples, or *support points*, and try to fit the function to them in a batch iterative process. Fitted value iteration has been used with both, parametric and non-parametric function approximators. For example, Ernst et al. [7], based on the previous work of Ormoneit and Sen [8] on kernel-based RL, proposed the *fitted Q Iteration algorithm* using (non-parametric) randomized trees for function approximation, while Riedmiller [9], [10] proposed *Neural Fitted Q Iteration* using a (parametric) multi-layer neural net. The main idea of fitted value iteration is to reuse the set of samples as much as needed to get all possible information from them. This allows to learn with a minimal number of interactions with the real system, but this does not imply a lesser number of

A. Agostini and E. Celaya are with the Institut de Robòtica i Informàtica Industrial (UPC - CSIC), c./ Llorens i Artigas 4-6, 08028 Barcelona, Spain. emails: agostini@iri.upc.edu and celaya@iri.upc.edu

This work was partially supported by the Spanish Ministry of Science and Innovation under project MIPRCV, Consolider Ingenio 2010 (CSD2007-00018).

function updates. Thus, it is appropriate when acquiring new data is more costly than just storing them for future use. However, assuming that data can be obtained at low cost, as for example by simulation, this advantage disappears, and can even become a disadvantage when the dynamics evolves with time, since old data would be no more valid.

A key issue in fitted value iteration algorithms is the generation of the set of representative samples of the function to be approximated. When the model of the problem is known, they can be obtained by uniform sampling through the state-space as in [7]. When the model is not available, samples can be generated by interacting with the system with random actions, but this strategy may not work when the complexity of the problem is such that reaching the interesting regions of the state space requires a long chain of lucky actions. For this reason, Riedmiller [9] uses a *greedy heuristic*, which consists in exploiting the policy learned in the previous learning stages to generate new samples for the next iterations of the algorithm. Even so, when the problem raises in complexity, he finds necessary to use what he calls the *hint-to-goal heuristic* to provide specific exemplars within the goal region. Note that, in principle, fitted value iteration algorithms are not incremental, in the sense that each time new samples are introduced, the function approximation process must be repeated from scratch for the new dataset, what is computationally inefficient.

In this paper, we propose an approach to RL for continuous state-action spaces with a function approximation based on probability density estimations. The idea is to represent the density distribution of the observed samples in the joint space of states, actions, and q-values. To represent this density distribution we use a Gaussian Mixture Model with variable number of units, so that the function approximation is non-parametric, what makes it general. With this approach, it is possible to obtain, for each given state and action, the probability distribution of $q(s, a)$ as the conditional probability $p(q|s, a)$. From this distribution we can obtain the value of $Q(s, a)$ as the expected value of $q(s, a)$. Furthermore, we can obtain the variance of $q(s, a)$ and estimate its confidence, so that our approach also presents what has been argued to be an important feature of GPs [4], [3]. The Gaussian Mixture Model can be updated with an incremental, low complexity version of the Expectation Maximization algorithm, what makes this approach more appealing than GPs and fitted value iteration algorithms in general. As a further benefit of using density estimations, it is possible, by marginalization on the state-action variables, to obtain the local sampling density in a point (s, a), which, in stochastic problems, may be used to evaluate how reliable is the estimation at this point.

The rest of the paper is organized as follows: Section II briefly resumes the basics of RL. Section III introduces the GMM for multivariate density estimation and the EM algorithm in its batch version. In Section IV we define the on-line EM algorithm for the GMM. In Section V we develop our RL algorithm using a density estimation of the Q-value function. Section VI shows the feasibility of the approach with an example, and Section VII concludes the paper.

II. Reinforcement Learning

Reinforcement Learning is a paradigm in which an agent has to learn an optimal action policy by interacting with its environment [11]. The task is formally modelled as the solution of a Markov decision process in which, at each time step, the agent observes the current state of the environment, s_t, and chooses an allowed action a_t using some action policy, $a_t = \pi(s_t)$. In response to this action, the environment changes to state s_{t+1} and produces an instantaneous reward $r_t = r(s_t, a_t)$. Using the information collected in this way, the agent must find the policy that maximizes the expected sum of discounted rewards, also called *return*, defined as:

$$R = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{1}$$

where γ is the discount rate, with values in [0,1], that regulates the importance of future rewards with respect to immediate ones.

One of the most popular algorithms used in RL is Q-Learning [12], which uses an action-value function $Q(s, a)$ to estimate the maximum expected return that can be obtained by executing action a in situation s and acting optimally thereafter. Q-learning uses the Bellman equation [13] to estimate sample values for $Q(s, a)$ that we denote by $q(s, a)$:

$$q(s_t, a_t) = r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) \tag{2}$$

where $\max_a Q(s_{t+1}, a)$ is the estimated maximum expected return corresponding to the next observed situation s_{t+1}. At a given stage of the learning, the temporary policy can be derived from the estimated Q-function as

$$\pi(s) = \underset{a}{argmax}\, Q(s, a) \tag{3}$$

In actor/critic architectures, a policy function (called the actor) is learned and explicitly stored, so that actions are directly decided by the actor and do not need to be computed through the maximization in (3). Despite this computational advantage, the learning of an actor may slow down convergence, since then the learning of the Q-function must be done on-policy instead of off-policy, and both functions, actor and critic, must adapt to each other to reach convergence. In our implementation we avoid the use of an actor, and thus we must face the problem of maximizing the $Q(s, a)$ function in (3).

The basic formulation of Q-learning assumes discrete state-action spaces and the Q-function is stored in a tabular representation. For continuous domains a function approximation is required to represent the Q-function and generalize between similar situations. In next sections we present our proposal for function approximation using density estimations.

III. Density Estimation with a Gaussian Mixture Model

A Gaussian Mixture Model [14] is a weighted sum of multivariate Gaussian probability density functions, and is used to represent general probability density functions in multidimensional spaces. It is assumed that the samples of the distribution to be represented have been generated through the following process: first, one Gaussian is randomly selected with *a priori* given probabilities, and then, a sample is randomly generated with the probability distribution of the selected Gaussian. According to this, the probability density function of generating sample $\mathbf{x}$ is:

$$p(\mathbf{x};\mathbf{\Theta}) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(\mathbf{x};\mu_i,\mathbf{\Sigma}_i) \tag{4}$$

where K is the number of Gaussians of the mixture; α_i, usually denoted as the mixing parameter, is the prior probability, $P(i)$, of Gaussian i to generate a sample; $\mathcal{N}(\mathbf{x};\mu_i,\mathbf{\Sigma}_i)$ is the multidimensional Gaussian function with mean vector μ_i and covariance matrix $\mathbf{\Sigma}_i$; and $\mathbf{\Theta} = \{\{\alpha_1,\mu_1,\mathbf{\Sigma}_1\},...,\{\alpha_K,\mu_K,\mathbf{\Sigma}_K\}\}$ is the whole set of parameters of the mixture. By allowing the adaption of the number K of Gaussians in the mixture, any smooth density distribution can be approximated arbitrarily close [15]. The parameters of the model can be estimated using a maximum-likelihood estimator (MLE). Given a set of samples $\mathbf{X} = \{\mathbf{x}_t; t = 1,\ldots,N\}$, the likelihood function is given by

$$\mathcal{L}[\mathbf{X};\mathbf{\Theta}] = \prod_{t=1}^{N} p(\mathbf{x}_t;\mathbf{\Theta}). \tag{5}$$

The maximum-likelihood estimation of the model parameters is the $\mathbf{\Theta}$ that maximizes the likelihood (5) for the data set $\mathbf{X}$. Direct computation of the MLE requires complete information about which mixture component generated which instance. Since this information is missing, the EM algorithm, described in the next section, is often used.

A. The Expectation-Maximization algorithm

The Expectation-Maximization (EM) algorithm [16] is a general tool that permits to estimate the parameters that maximize the likelihood function (5) for a board class of problems when there are some missing data. The EM method first produces an estimation of the expected values of the missing data using initial values of the parameters to be estimated (E step), and then computes the MLE of the parameters given the expected values of the missing data (M step). This process is repeated iteratively until a convergence criterion is fulfilled.

In this section we briefly describe how EM is applied to the specific case of a GMM. The process starts with an initialization of the mean vectors and covariance matrices of the Gaussians. The E step consists in obtaining the probability $P(i|\mathbf{x}_t)$ for each component i of generating instance $\mathbf{x}_t$, that we denote by $w_{t,i}$,

$$w_{t,i} = P(i|\mathbf{x}_t) = \frac{P(i)p(\mathbf{x}_t|i)}{\sum_{j=1}^{K} P(j)p(\mathbf{x}_t|j)} = \frac{\alpha_i \mathcal{N}(\mathbf{x}_t;\mu_i,\mathbf{\Sigma}_i)}{\sum_{j=1}^{K} \alpha_j \mathcal{N}(\mathbf{x}_t;\mu_j,\mathbf{\Sigma}_j)} \tag{6}$$

where $t = 1,..,N$ and $i = 1,..,K$. The maximization step consists in computing the MLE using the estimated $w_{t,i}$. It can be shown [17] that, for the case of a GMM, the mixing parameters, means, and covariances are given by

$$\alpha_i = \frac{1}{N}\sum_{t=1}^{N} w_{t,i} \tag{7}$$

$$\mu_i = \frac{\sum_{t=1}^{N} w_{t,i}\mathbf{x}_t}{\sum_{t=1}^{N} w_{t,i}} \tag{8}$$

$$\mathbf{\Sigma}_i = \frac{\sum_{t=1}^{N} w_{t,i}(\mathbf{x}_t-\mu_i)(\mathbf{x}_t-\mu_i)^{\mathsf{T}}}{\sum_{t=1}^{N} w_{t,i}} \tag{9}$$

IV. On-line EM

Estimating a probability density function by means of the EM algorithm involves the iteration of E and M steps on the complete set of available data, that is, the mode of operation of EM is in batch. However, in RL, sample data are not all available at once: they arrive sequentially and must be used online to improve the policy that will allow an efficient exploration-exploitation strategy. This prevents the use of the off-line EM algorithm, and requires an on-line, incremental version of it. Several incremental EM algorithms have been proposed for the Gaussian Mixture Model applied to clustering or classification of stationary data [18], [19].

The approach proposed in [18] in not strictly an on-line EM algorithm. It applies the conventional batch EM algorithm onto separate data streams corresponding to successive episodes. For each new stream, a new GMM model is trained in batch mode and then merged with the previous model. The number of components for each new GMM is defined using the Bayesian Information Criterion, and the merging process involves similarity comparisons between Gaussians. This method involves many computationally expensive processes at each episode and tends to generate more components than actually needed. The applicability of this method to RL seems limited, not only for its computational cost, but also because, due to the non-stationarity of the Q-estimation, old data should not be taken as equally valid during all the process.

The work of [19] performs incremental updating of the density model using no historical data and assuming that consecutive data vary smoothly. The method maintains two GMMs: the current GMM estimation, and a previous GMM of the same complexity after which no model updating (i.e. no change in the number of Gaussians) has been done.

By comparing the current GMM with the historical one, it is determined if new Gaussians are generated or if some Gaussians are merged together. Two observed shortcomings of the algorithm are that the system fails when new data is well explained by the historical GMM, and when consecutive data violate the condition of smooth variation.

In [20], an on-line EM algorithm is presented for the Normalized Gaussian Network (NGnet), a model closely related to the GMM. This algorithm is based on the works of [21], [22]. In [21] a method for the incremental adaptation of the model parameters using a forgetting factor and cumulative statistics is proposed, while in [22] the method in [21] is evaluated and contrasted with an incremental version which performs steps of EM over a fixed set of samples in an incremental way. The method proposed in [20] uses foundations of both works to elaborate an on-line learning algorithm to train a NGnet for regression, where weighted averages of the model parameters are calculated using a learning rate that implicitly incorporates a forgetting factor to deal with non-stationarities. Inspired by this work, we developed an on-line EM algorithm for the GMM. Our approach uses cumulative statistics whose updating involves a forgetting factor explicitly.

A. *On-line EM for the GMM*

In the on-line EM approach, an E step and an M step are performed after the observation of each individual sample. The E step does not differ from the batch version (equation (6)), except that it is only computed for the new sample. For the M step, the parameters of all mixture components are updated with the new sample. For this, we define the following time-discounted weighted sums

$$W_{t,i} = [[1]]_{t,i} \quad (10)$$

$$X_{t,i} = [[\mathbf{x}]]_{t,i} \quad (11)$$

$$(XX)_{t,i} = \left[\left[\mathbf{x}\mathbf{x}^{\mathsf{T}}\right]\right]_{t,i} \quad (12)$$

where we use the notation:

$$[[f(x)]]_{t,i} = \sum_{\tau=1}^{t} \left(\prod_{s=\tau+1}^{t} \lambda_s \right) f(x_\tau) w_{\tau,i} \quad (13)$$

where $\lambda_t \in [0,1]$ is a time dependent discount factor introduced for forgetting the effect of old, possibly outdated values. Observe that for low values of λ_t, the influence of old data decreases progressively, so that they are forgotten along time. This forgetting effect of old data is attenuated when λ_t approaches 1: in this case, old and new data have the same influence in the sum. As learning proceeds and data values become more stable, forgetting them is no more required and λ_t can be made to progressively approach 1 to allow convergence.

The sum $W_{t,i}$ can be interpreted as the accumulated number of samples (composed of weights $w_{t,i}$) attributed to unit i along time, with forgetting. Similarly, $X_{t,i}$ corresponds to the weighted sum with forgetting of sample vectors $\mathbf{x}_\tau$ attributed to unit i, which is used to derive the mean vector μ_i. In the same way, $(XX)_{t,i}$ is the weighted sum with forgetting of the matrices obtained as the products $\mathbf{x}_\tau \mathbf{x}_\tau^{\mathsf{T}}$ of sample vectors attributed to unit i, which will be used to find the covariance matrix $\mathbf{\Sigma}_i$.

From (13), we obtain the recursive formula:

$$[[f(x)]]_{t,i} = \lambda_t \, [[f(x)]]_{t-1,i} + f(x_t) w_{t,i}. \quad (14)$$

When a new sample $\mathbf{x}_t$ arrives, the accumulators (10), (11), and (12) are updated with the incremental formula (14), and new estimators for the GMM parameters are obtained as:

$$\alpha_i(t) = \frac{W_{t,i}}{\sum_{j=1}^{K} W_{t,j}} \quad (15)$$

$$\mu_i(t) = \frac{X_{t,i}}{W_{t,i}} \quad (16)$$

$$\mathbf{\Sigma}_i(t) = \frac{(XX)_{t,i}}{W_{t,i}} - \mu_i(t)\mu_i(t)^{\mathsf{T}} \quad (17)$$

If the number K of Gaussians in the mixture is fixed, the GMM is a parametric function approximation method whose approximation capabilities are determined by K. Since we can not determine the most appropriate K beforehand, we allow the number of Gaussians to be incremented on-line by a process of unit generation, so that the function approximation method as a whole becomes non-parametric. The process of unit generation is explained in Section V-B.

B. *Weight-Dependent Forgetting*

The factors λ_t in (13) were introduced by [20] with the purpose of progressively replace (forget) old data by new, more reliable values. The effect of this is clearly seen in the incremental formula (14), which shows how, at each time step, all past data are multiplied by λ_t, and this is done for all units, no matter how much weight $w_{t,i}$ is attributed to each of them. We observe that, the real effect of applying (14) to units with low activation $w_{t,i}$ is not to replace their past values by the new one but, essentially, to decrease their values by a factor λ_t. It can be seen that this is exactly the case when setting $w_{t,i} = 0$ in equation (14), what yields:

$$[[f(x)]]_{t,i} = \lambda_t \, [[f(x)]]_{t-1,i} \, , \quad (18)$$

showing that the accumulators of units that are seldom activated will systematically decay to 0. This situation is particularly annoying in the case of online RL, for which it is very likely that highly valued regions of the state-action space will be sampled much more frequently than less promising ones, so that in the long term, units covering low valued regions will get their statistics lost. This can be avoided by modifying the updating formula (14) in this way:

$$[[f(x)]]_{t,i} = \lambda_t^{w_{t,i}} \, [[f(x)]]_{t-1,i} + f(x_t) w_{t,i}. \quad (19)$$

With this updating formula, the amount by which old data are forgotten is regulated by the amount $w_{t,i}$ in which a new value is added to the sum, so that data are always *replaced*, instead of simply forgotten. Effectively, if now we make $w_{t,i} = 0$ in (19), what we get is:

$$[[f(x)]]_{t,i} = [[f(x)]]_{t-1,i}\,, \tag{20}$$

so that the values of the statistics of the inactive units remain unchanged. On the other hand, in the case of a full activation of unit i, i.e., if $w_{t,i} = 1$, the effect of the new updating formula is exactly the same as that of (14).

Therefore, we will prefer the updating formula (19) to keep better track of less sampled regions, noting that by doing this, the definition given in (13) does no longer hold.

V. The GMM for Q-Learning

In this Section, we describe how the GMM can be used for function approximation to estimate the expected Q-value as well as its variance at each point of the state-action space by means of a single representation of the probability density function in the joint space of states, actions and Q-values:

$$p(\mathbf{s}, a, q) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(\mathbf{s}, a, q; \mu_i, \mathbf{\Sigma}_i). \tag{21}$$

In online Q-learning, each sample is of the form $\mathbf{x}_t = (\mathbf{s}_t, a_t, q(\mathbf{s}_t, a_t))$, corresponding to the visited state $\mathbf{s}_t$, the executed action a_t, and the estimated value of $q(\mathbf{s}_t, a_t)$ as given by eq. (2). To obtain this estimation we need to evaluate $\max\limits_a Q(\mathbf{s}_{t+1}, a_t)$, where $Q(\mathbf{s}, a)$ is defined as the expected value of q given $\mathbf{s}$ and a for the joint probability distribution (21) provided by the GMM:

$$Q(\mathbf{s}, a) = E\left[q|\mathbf{s}, a\right] = \mu(q|\mathbf{s}, a). \tag{22}$$

To compute this, we must first obtain the distribution $p(q|\mathbf{s}, a)$. Decomposing the covariances $\mathbf{\Sigma_i}$ and means μ_i in the following way:

$$\mu_i = \begin{pmatrix} \mu_i^{(\mathbf{s},a)} \\ \mu_i^{q} \end{pmatrix} \tag{23}$$

$$\mathbf{\Sigma}_i = \begin{pmatrix} \mathbf{\Sigma}_i^{(\mathbf{s},a)(\mathbf{s},a)} & \mathbf{\Sigma}_i^{(\mathbf{s},a),q} \\ \mathbf{\Sigma}_i^{q,(\mathbf{s},a)} & \mathbf{\Sigma}_i^{qq} \end{pmatrix}, \tag{24}$$

the probability distribution of q, for a given state $\mathbf{s}$ and action a, can then be expressed as:

$$p(q|\mathbf{s}, a) = \sum_{i=1}^{K} \beta_i(\mathbf{s}, a) \mathcal{N}\left(q; \mu_i(q|\mathbf{s}, a), \sigma_i(q)\right) \tag{25}$$

where,

$$\mu_i(q|\mathbf{s}, a) = \mu_i^q + \mathbf{\Sigma}_i^{q,(\mathbf{s},a)} \left(\mathbf{\Sigma}_i^{(\mathbf{s},a)(\mathbf{s},a)}\right)^{-1} \left((\mathbf{s}, a) - \mu_i^{(\mathbf{s},a)}\right) \tag{26}$$

$$\sigma_i^2(q) = \mathbf{\Sigma}_i^{qq} - \mathbf{\Sigma}_i^{q,(\mathbf{s},a)} \left(\mathbf{\Sigma}_i^{(\mathbf{s},a)(\mathbf{s},a)}\right)^{-1} \mathbf{\Sigma}_i^{(\mathbf{s},a),q} \tag{27}$$

$$\beta_i(\mathbf{s}, a) = \frac{\alpha_i \mathcal{N}(\mathbf{s}, a; \mu_i^{(\mathbf{s},a)}, \mathbf{\Sigma}_i^{(\mathbf{s},a)(\mathbf{s},a)})}{\sum\limits_{j=1}^{K} \alpha_j \mathcal{N}(\mathbf{s}, a; \mu_j^{(\mathbf{s},a)}, \mathbf{\Sigma}_j^{(\mathbf{s},a)(\mathbf{s},a)})}. \tag{28}$$

From (25) we can obtain the conditional mean and covariance, $\mu(q|\mathbf{s}, a)$ and $\sigma^2(q|\mathbf{s}, a)$, of the mixture at a point $(\mathbf{s}, a)$ as:

$$\mu(q|\mathbf{s}, a) = \sum_{i=1}^{K} \beta_i(\mathbf{s}, a) \mu_i(q|\mathbf{s}, a) \tag{29}$$

$$\sigma^2(q|\mathbf{s}, a) = \sum_{i=1}^{K} \beta_i(\mathbf{s}, a)(\sigma_i^2(q) + (\mu_i(q|\mathbf{s}, a) - \mu(q|\mathbf{s}, a))^2). \tag{30}$$

Equation (29) is the estimated Q value for a given state and action, while (30) is its estimated variance. Our purpose was to find the maximum for all actions and for a given $\mathbf{s}$ of $Q(\mathbf{s}, a)$, in order to compute (2). Unfortunately, this is hard to do analytically, but an approximated value can be obtained by numerical techniques. In our implementation, we take the simple approach of computing $Q(\mathbf{s}, a)$ for a finite number of actions, and then taking the largest Q value as the approximated maximum:

$$\max_a Q(s, a) \approx \max_{a \in A} Q(s, a), \tag{31}$$

where A is the set of actions that we take into consideration to find the approximated maximum.

A. Action Selection with Exploration

Action selection in RL must address the exploration/exploitation tradeoff. If we want just to exploit what has been learnt so far, with no exploration, we must execute the action a_g corresponding to the greedy policy as given by eq. (3), that in our case is computed as:

$$a_g = \pi_g(s) \approx \underset{a \in A}{argmax}\, Q(s, a). \tag{32}$$

However, during learning, an exploration strategy is necessary that guarantees that no action is excluded from execution in any state. Two well-known ways to achieve this are the ε-greedy and the Boltzmann exploration. According to [23], these strategies are in the family of the undirected exploration methods, meaning that exploration is based in randomness, and no exploration-specific knowledge is used for guiding exploration. It is claimed that directed exploration techniques are often more efficient than undirected ones, so we propose a more directed method of exploration that takes into account the prediction error for each action, which is captured in the variance of the Q-values. For this, we define:

$$Q_{rnd}(s, a) = Q(s, a) + \Delta_Q(\sigma^2(q|s, a)), \tag{33}$$

where $\Delta_Q(\sigma^2(q|s, a))$ is a value taken at random from a normal probability distribution with 0 mean and variance $\sigma^2(q|s, a)$. Then, the action selection with exploration is made according to:

$$a_{explr} = \underset{a \in A}{argmax}\, Q_{rnd}(s, a) + a_{rnd}, \tag{34}$$

where a_{rnd} is an appropriately sized random perturbation of the action, introduced to allow the execution of arbitrary actions and not just those contained in A.

With this form of exploration all the actions have always a chance of getting a Q_{rnd} above its competitors, and hence, a probability to be selected. Usually, higher-valued actions will have more chances of getting the highest Q_{rnd}. However, a low-valued action may eventually receive a high Q_{rnd} that surpasses the values of other actions. This provides a balance between exploration and exploitation, that tends to take the greedy action when we are rather certain that it will result in a larger value, and increases the probability of exploring a non-greedy action when its predicted outcome is uncertain.

B. Unit Generation

The GMM is initialized with a small number of units that is selected according to the expected complexity of the problem. However, if during training the model is found to be insufficient to represent the sample distribution with the required accuracy, it may be upgraded by generating new units. Since our main interest is to accurately represent the Q function, the generation of a new Gaussian is determined by the failure of the current GMM to account for an actually observed q value. Thus, a new Gaussian is generated when the two following conditions are satisfied:

1) The estimation error of the observed q value is larger than a predefined value δ:

$$(q(s,a) - \mu(q|\mathbf{s},a))^2 \geq \delta. \tag{35}$$

2) Units close to the experienced point have been sufficiently updated. We consider a unit i is close to a point $\mathbf{x} = (s,a,q)$, if the Mahalanobis distance $D_M^{(i)}$, with covariance matrix Σ_i, between the unit mean and the point is less than 1,

$$I = \{1 \leq i \leq K | D_M^{(i)}(\mathbf{x}, \mu_i) < 1\}, \tag{36}$$

and thus, the criterion can be expressed as:

$$\sum_{\tau=1}^{t} w_{\tau,i} > N_{conf}, \forall i \in I. \tag{37}$$

The purpose of this condition is to avoid the premature generation of new units in a region before the system has had the opportunity to adapt to data in the given region.

Whenever both criteria are fulfilled, a Gaussian is generated with parameters given by:

$$W_{K+1} = 1 \tag{38}$$

$$\mu_{K+1}(\mathbf{s},a,q) = (\mathbf{s}_t, a_t, q(\mathbf{s}_t, a_t)) \tag{39}$$

$$\mathbf{\Sigma}_{K+1} = C\ diag\{d_1, ..., d_D, d_a, d_q\}, \tag{40}$$

where d_i is the total range size of variable i, D is the dimension of the state space, and C is a positive value to size the variances of the new Gaussian.

VI. Experiments

A classical benchmark problem for RL, the control of an inverted pendulum with limited torque [24], has been selected to test our algorithm. We addressed the problem of the swing up and stabilization of the pendulum. The task consists in swinging the pendulum until reaching the upright position and then stay there indefinitely. The optimal policy for this problem is not trivial to find since, due to the limited torques available, the controller has to swing the pendulum several times back and forth until its kinetic energy is large enough to overcome the load torque and reach the upright position, and then stabilize the pendulum there.

The state space of this problem is two-dimensional and is formed by the angular position θ and angular velocity $\dot{\theta}$: $\mathbf{s} = (\theta, \dot{\theta})$, where θ takes values in the interval $[-\pi, \pi]$, and $\dot{\theta}$ is limited to the interval $[-8, 8]s^{-1}$. The Gaussians of the mixture model are four-dimensional and the GMM provides estimations of the probability densities in the joint space $\mathbf{x}$=$(\theta, \dot{\theta}, a, q)$. As the reward signal we simply take the height of the tip of the pendulum $h = cos(\theta)$ which ranges in the interval $[-1, 1]$, and the discount coefficient γ in equation (2) is set to 0.99. We initialize the model with 20 Gaussians with random initial means μ_i for all except the q dimension, that is initialized to the maximum possible Q value to favor exploration of unvisited regions. The initial covariance matrices $\mathbf{\Sigma}_i$ are diagonal and the variance of each variable is set to the range of that variable. The initial number of samples W_i of each Gaussian is set to 0.1. This small value makes the component i to have a small influence in the estimation while there is no, or little, updating.

The discount factor λ_t for the weighted sums with forgetting (section IV) takes values from the equation,

$$\lambda_t = 1 - 1/(at + b) \tag{41}$$

where b regulates the initial value λ_0, and a determines its growth rate toward 1. In our experiments we set $a = 0.001$ and $b = 1000$ in the case of using the updating formula (14), and $b = 10$ in the case of using the updating formula (19) to compensate for the effect of the exponent $w_{t,i} < 1$.

For the experiments, we adopt the set-up of [25]: we run episodes of 7 seconds with actuation intervals of 0.01 seconds. At the beginning of each episode the pendulum is randomly placed inside an arch centered in the upright position. This can be seen as a form of the *hint-to-goal heuristic* used in [9] and also in [3]. The length of the arch is steadily incremented with each episode until covering its whole range, thus allowing any arbitrary initial position. To evaluate the performance of the learning system, we run 10 independent experiments of 120 episodes each. At the end of each episode, a 7 sec. test exploiting the policy learned so far is done, and the total accumulated reward is computed. Figure 1 shows the result of averaging the results of the 10 experiments using the updating formula (14). It can be observed that, despite an acceptable control is reached, there is some instability that persists even in the final episodes. This is caused by sporadic periods in which, after having

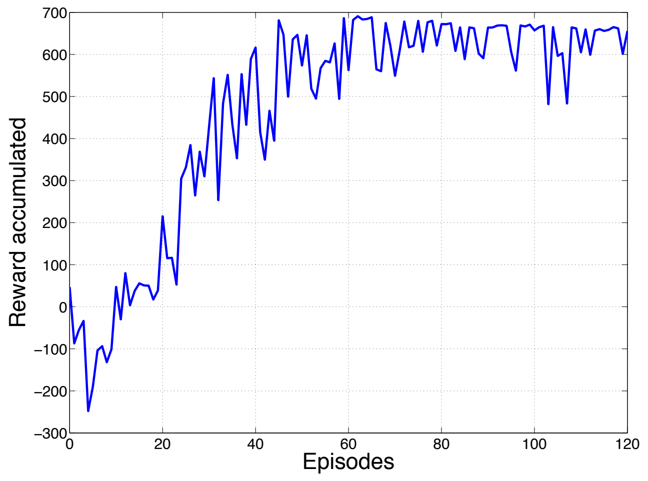

Fig. 1. Average of the accumulated reward per episode performed over 10 experiments, with uniform forgetting (λ_t).

learned to correctly swing-up and stabilize the pendulum, the system "unlearns" it and must re-learn again to recover the right policy. This effect is a consequence of the forgetting of the function approximation in low valued regions caused by the biased sampling that occurs when the system keeps learning after the right policy has been already found. To correct this effect is that we introduced the updating formula (19) for weight-dependent forgetting. The results obtained

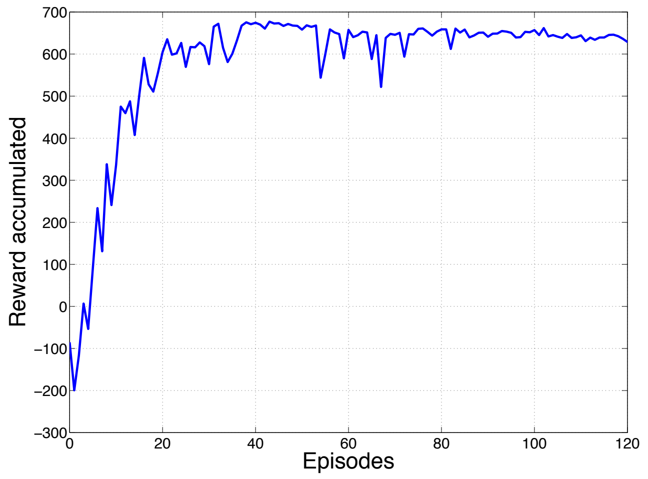

Fig. 2. Average of the accumulated reward per episode performed over 10 experiments, with weight-dependent forgetting ($\lambda_t^{w_{t,i}}$).

using this formula are shown in Figure 2. It can be seen that in this case convergence is much faster and with a much more stable behavior, what demonstrates the effectiveness of the approach.

To illustrate the performance reached, Figure 3 shows a stroboscopic sequence of the pendulum starting from the initial position of the pendulum hanging down. Figures 4 and

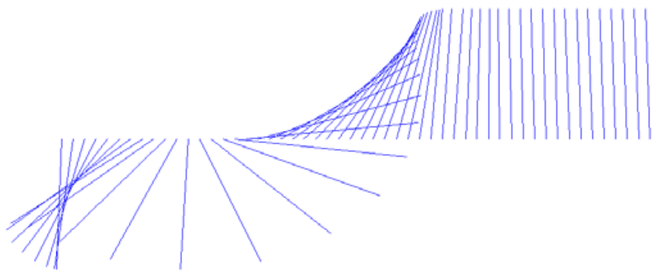

Fig. 3. A stroboscopic sequence obtained from placing the pendulum in the downright position.

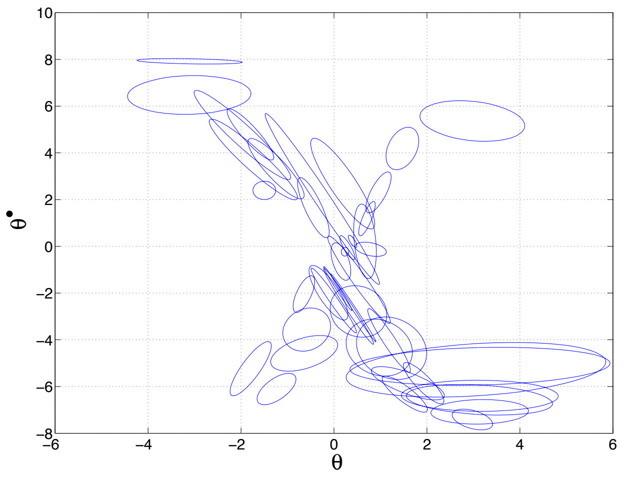

Fig. 4. Projection of the Gaussians of the GMM into the state space.

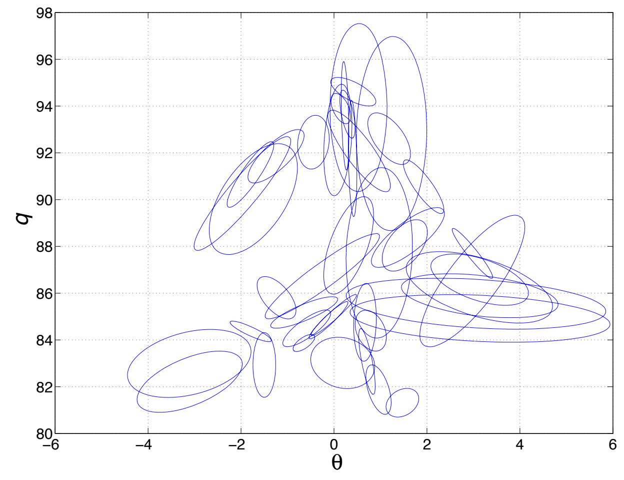

Fig. 5. Projection of the Gaussians of the GMM into the (θ, q) space.

5 show two projections of the Gaussians of a typical GMM obtained for this problem after training. It can be seen that they are not equally distributed on the whole configuration space, but concentrated in the most common trajectories of the system, making an efficient use of resources.

VII. Conclusions

We have shown that estimating a probability density function in the joint space of states, actions and q-values, provides a useful tool for RL in continuous domains. The probability density function captures all the information available to the RL agent: In the first place, it provides a function approximation for the action-value function $Q(s, a)$ as the mean of the sample values $q(s, a)$; in second place, as in the case of using GPs, the probability density function provides not only the mean value of $q(s, a)$, but a full probability distribution of its possible values, and in particular, its variance. We use this information to direct the exploration, a possibility suggested in [5], [4], but that had not been implemented yet. Finally, the probability density in the joint space can be marginalized to obtain the density of samples in the state-action space, which may be used to measure the confidence we may have in the estimation at each point.

To represent the probability density function we use a GMM with variable number of units. This provides a general, non-parametric, function approximation tool. By using an on-line version of the EM algorithm, the training of the GMM can be done incrementally and, thanks to the simplicity of the GMM, the update process is computationally efficient. The feasibility of the method is demonstrated on a standard benchmark for RL, the swing-up and balance of an inverted pendulum with limited torque, with good results.

We believe that the simplicity and expressiveness of this approach makes it a promising alternative for RL in continuous domains.

References

[1] C. Stone, "Optimal global rates of convergence for nonparametric regression," *The Annals of Statistics*, vol. 10, no. 4, pp. 1040–1053, 1982.

[2] C. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 16, pp. 751–759, 2004.

[3] M. Diesenroth, C. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomputing*, vol. 72, no. 7-9, pp. 1508–1524, 2009.

[4] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM, 2005, pp. 201–208.

[5] ——, "Bayes meets Bellman: The Gaussian process approach to temporal difference learning," in *Proc. of the 20th International Conference on Machine Learning*, 2003, pp. 154–161.

[6] G. J. Gordon, "Stable function approximation in dynamic programming," in *ICML*, 1995, pp. 261–268.

[7] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, 2005.

[8] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, no. 2-3, pp. 161–178, 2002.

[9] M. Riedmiller, "Neural Reinforcement Learning to Swing-up and Balance a Real Pole," in *Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, 2005, pp. 3191–3196.

[10] ——, "Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method," *Lecture notes in computer science*, vol. 3720, pp. 317–328, 2005.

[11] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, B. Book, Ed. Cambridge, MA: MIT Press, 1998.

[12] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992. [Online]. Available: http://jmvidal.cse.sc.edu/library/watkins92a.pdf

[13] R. Bellman and S. Dreyfus, *Applied Dynamic Programming*. Princeton, New Jersy: Princeton University Press, 1962.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[15] M. Figueiredo, "On Gaussian radial basis function approximations: Interpretation, extensions, and learning strategies," *Pattern Recognition, International Conference on*, vol. 2, pp. 618–621, 2000.

[16] A. Dempster, N. Laird, D. Rubin, *et al.*, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[17] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. New-York, USA: John Wiley and Sons, Inc, 2001.

[18] M. Song and H. Wang, "Highly efficient incremental estimation of Gaussian mixture models for online data stream clustering," in *Proceedings of SPIE: Intelligent Computing: Theory and Applications III*, Orlando, FL, USA, 2005, pp. 174–183.

[19] O. Arandjelovic and R. Cipolla, "Incremental learning of temporally-coherent Gaussian mixture models," in *Technical Papers - Society of Manufacturing Engineers (SME)*, 2005.

[20] M.-A. Sato and S. Ishii, "On-line em algorithm for the normalized Gaussian network," *Neural Comput.*, vol. 12, no. 2, pp. 407–432, 2000.

[21] S. J. Nowlan, "Soft competitive adaptation: neural network learning algorithms based on fitting statistical mixtures," Ph.D. dissertation, Pittsburgh, PA, USA, 1991.

[22] R. Neal and G. Hinton, "A view of the em algorithm that justifies incremental, sparse, and other variants," in *Proceedings of the NATO Advanced Study Institute on Learning in graphical models*. Norwell, MA, USA: Kluwer Academic Publishers, 1998, pp. 355–368.

[23] S. Thrun, "The role of exploration in learning control," in *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, D. White and D. Sofge, Eds. Florence, Kentucky 41022: Van Nostrand Reinhold, 1992.

[24] K. Doya, "Reinforcement learning in continuous time and space," *Neural Comput.*, vol. 12, no. 1, pp. 219–245, 2000.

[25] M.-a. Sato and S. Ishii, "Reinforcement learning based on on-line em algorithm," in *Proceedings of the 1998 conference on Advances in neural information processing systems (NIPS'99)*. Cambridge, MA, USA: MIT Press, 1999, pp. 1052–1058.