Preprint accepted for publication in Neural Computation, 2016

Online Reinforcement Learning using a Probability Density Estimation

Alejandro Agostini
aagosti@gwdg.de
Bernstein Center for Computational Neuroscience, 37077 Göttingen, Germany.

Enric Celaya
celaya@iri.upc.edu
Institut de Robòtica i Informàtica Industrial (CSIC-UPC), 08028 Barcelona, Spain.

Keywords: online reinforcement learning, biased sampling problem, non-stationarity problem, Gaussian mixture model, online expectation-maximization

Abstract

Function approximation in online, incremental, reinforcement learning needs to deal with two fundamental problems: biased sampling and non-stationarity. In this kind of tasks, biased sampling occurs because samples are obtained from specific trajectories dictated by the dynamics of the environment and are usually concentrated in particular convergence regions, which in the long term tend to dominate the approximation in the less sampled regions. The non-stationarity comes from the recursive nature of the estimations typical of temporal difference methods. This non-stationarity has a local profile, not only varying along the learning process but also along different regions of the state space. We propose to deal with these problems using an estimation of the probability density of samples represented with a Gaussian mixture model. To deal with the non-stationarity problem we use the common approach of introducing a forgetting factor in the updating formula. However, instead of using the same forgetting factor for the whole domain, we make it to depend on the local density of samples, which we use to estimate the non-stationarity of the function at any given input point. On the other hand, to address the biased sampling problem, the forgetting factor applied to each mixture component is modulated according to the new information provided in the updating, rather than forgetting only depending on time, thus avoiding undesired distortions of the approximation in less sampled regions.

1 Introduction

In this work we approach the problem of function approximation (FA) when applied to real situations of online, incremental reinforcement learning (RL). Addressing this problem is important because it avoids the introduction of assumptions seldom justifiable, like, just to name a few, the availability of a model of the environment, the determinism of the environment, the continuity of the transition and reward functions, or the availability "a priori" of a set of relevant transitions which is sufficient to learn a good controller (the usual assumption in the *fitted value iteration* approaches (Gordon, 1995)). Even though such assumptions are often used to make algorithms more efficient or to allow proofs of convergence, algorithms which do not make such assumptions are necessary since only these address the full reinforcement learning problem in general.

FA methods applied to incremental RL needs to deal with two fundamental problems: biased sampling and non-stationarity. During the interaction with the environment, the action policy and the dynamics of the environment guide the agent through specific trajectories of the state space, usually concentrated in particular convergence regions, making the sampling highly biased. Biased sampling induces regions that are more frequently sampled to have better estimations than regions sparsely explored. This is an inconvenience since, for a good control strategy, the reward function should be well represented even in those regions sparsely visited, or visited long time ago, in which a wrong decision may imply a failure in the control task. The non-stationarity problem, on the other hand, is caused by the recursive nature of the estimation of the expected return and the evolution of the action policy during learning. These two factors make the values associated to particular states and actions to be highly variable, not only along the learning process but also along the different regions of the state space (Gordon, 1995).

One of the most relevant approaches able to cope with the incremental RL problems is the work by Sato and Ishii (1999, 2000). They propose an incremental non-parametric FA approach for approximating the value and policy functions of an actor-critic architecture in a continuous state-action space. For the approximation, they use a normalized Gaussian network (NGnet), a network of local lineal regression units that softly partitions the input space by normalized Gaussian functions. Each unit linearly approximates the target function within its partition, hence mitigating the distortion in the approximation in regions far away from the sample. The updating of the parameters is carried out by an incremental, low complexity version of the Expectation-Maximization (EM) algorithm that trains an NGnet for regression. The updating approach permits adaptations to non-stationary changes by incorporating a forgetting factor to progressively replace old outdated estimations with every new experience. We identified two aspects of the approach to be improved. One of them is the time-dependent approach used for the updating of the parameters. In their approach, the estimations of unvisited regions are forgotten even though no new information is provided there. This provokes that regions far away from the visited one to get their approximation distorted as a consequence of biased sampling. The second aspect is the forgetting factor used for the updating. As in most of the FA approaches that incorporate a forgetting in the updating, they apply the same forgetting to all the regions of the domain. This prevents regulating the forgetting according to the different local changes due to non-stationarity, as

required in RL.

Inspired by the NGnet approach, we propose a new incremental non-parametric method for FA in continuous state-action space RL. Instead of using a NGnet, we use for the approximation a Gaussian mixture model (GMM), a model closely related to the NGnet from which we can easily obtain not only the value of the approximated function at any given input using simple conditional probabilities but also the density of samples in the input-output space. Based on this information, we propose a formula that updates past estimations in a region only when new information is provided there, rather than updating depending on time. In this manner, sparsely visited regions keep their approximations unaltered, no matter how long ago the last updating occurred, avoiding the distortions produced by biased sampling. We also use the density information to regulate the forgetting according to the local non-stationary changes. This is done by estimating the number of samples with similar output values in the vicinity of the new experience as an indicator of the non-stationarity of the function at that point. The FA approach is made non-parametric by generating new Gaussians to improve the approximation to any given precision. This is done by using the density information to let the new Gaussian quickly improve the approximation at a given input, with a reduced distortion in the rest of the regions.

The next sections are organized as follows. Section 2 introduces the elements for function approximation using a GMM. Section 2.1 specifies how the parameters of the GMM are updated using the EM algorithm and presents our online EM approach. Later, in Section 2.2, we present the approach to deal with the local non-stationary variations of the target function using the density of samples to regulate the forgetting. Section 2.3 describes the Gaussian generation process and Section 2.4 presents the algorithmic description the function approximation method. In Section 3, our reinforcement learning approach using a GMM for function approximation is described. The experimental evaluation is presented in Section 4. Finally, Section 5 concludes the paper. Part of this work is an extension of our previous contribution Agostini and Celaya (2010).

2 Function Approximation using a GMM (GMMFA)

A Gaussian mixture model (Bishop, 2006) 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 $\mathbf{z}$ is:

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

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; $\mathcal{N}(\mathbf{z};\mu_i,\boldsymbol{\Sigma}_i)$ is the multidimensional Gaussian function with mean vector μ_i and covariance matrix $\boldsymbol{\Sigma}_i$; and $\boldsymbol{\Theta} = \{\{\alpha_1,\mu_1,\boldsymbol{\Sigma}_1\},...,\{\alpha_K,\mu_K,\boldsymbol{\Sigma}_K\}\}$ is the whole set of parameters of the

mixture. By allowing the adaptation of the number K of Gaussians in the mixture, any smooth density distribution can be approximated arbitrarily close (Figueiredo, 2000).

Using the density model (1), it is possible to approximate any arbitrary stochastic function $g(\mathbf{x})$ defined in a D-dimensional continuous domain $X \subset \Re^D$, from which we can only observe input-output pairs obtained at particular points, $\mathbf{z}_t = (\mathbf{x}_t, y_t)$, where the output $y_t = g(\mathbf{x}_t)$ is supposed to be uni-dimensional [1] , and t accounts for the t-th time step. By using samples $\mathbf{z}_t = (\mathbf{x}_t, y_t)$ we can estimate the parameters of the joint probability distribution (1) from which we can easily obtain the full probability of the output variable y for any given input $\mathbf{x}$,

$$p(y|\mathbf{x}) = \sum_{i=1}^{K} \beta_i(\mathbf{x}) \mathcal{N}\left(y; \mu_i(y|\mathbf{x}), \sigma_i(y)\right), \tag{2}$$

where

$$\mu_i(y|\mathbf{x}) = \mu_i^y + \Sigma_i^{yx} \left(\Sigma_i^{xx}\right)^{-1} \left(\mathbf{x} - \mu_i^x\right), \tag{3}$$

$$\sigma_i^2(y) = \Sigma_i^{yy} - \Sigma_i^{yx} \left(\Sigma_i^{xx}\right)^{-1} \Sigma_i^{xy}, \tag{4}$$

and

$$\beta_i(\mathbf{x}) = \frac{\alpha_i \mathcal{N}(\mathbf{x}; \mu_i^x, \Sigma_i^{xx})}{\sum_{j=1}^{K} \alpha_j \mathcal{N}(\mathbf{x}; \mu_j^x, \Sigma_j^{xx})}, \tag{5}$$

which are calculated from values obtained from decomposing the covariances Σ_i and means μ_i in the following way:

$$\mu_i = \begin{pmatrix} \mu_i^x \\ \mu_i^y \end{pmatrix}, \tag{6}$$

$$\Sigma_i = \begin{pmatrix} \Sigma_i^{xx} & \Sigma_i^{xy} \\ \Sigma_i^{yx} & \Sigma_i^{yy} \end{pmatrix}. \tag{7}$$

The probability (2) provides very useful information for the function approximation of a stochastic function $g(\mathbf{x})$, as illustrated in Fig. 1. On the one hand, we can obtain an estimation of the expected value of $g(\mathbf{x})$ as the conditional mean, $\mu(y|\mathbf{x})$, of the mixture at a point $\mathbf{x}$,

$$g(\mathbf{x}) \approx \mu(y|\mathbf{x}) = \sum_{i=1}^{K} \beta_i(\mathbf{x}) \mu_i(y|\mathbf{x}). \tag{8}$$

Probability (2) also permits obtaining a point-wise variance of the outputs,

$$\sigma^2(y|\mathbf{x}) = \sum_{i=1}^{K} \beta_i(\mathbf{x}) (\sigma_i^2(y) + (\mu_i(y|\mathbf{x}) - \mu(y|\mathbf{x}))^2). \tag{9}$$

[1] We suppose a uni-dimensional output since it suffices for the type of functions we will use. However, all the formulations that follow can be easily extended to the multidimensional case.

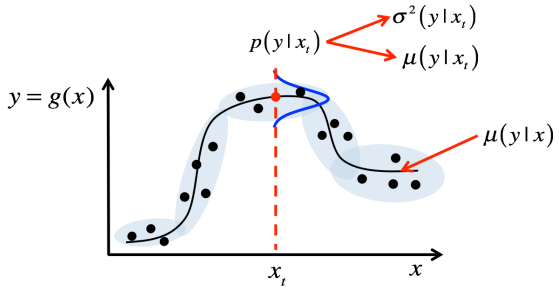


Figure 1: Example of a stochastic function approximation using a GMM. Each ellipse represents a multi-dimensional Gaussian that approximates the probability density of samples in its domain. At a query point x_t, a unidimensional Gaussian (blue line) represents the probability distribution of y at this point, $p(y|x_t)$, with expected value $\mu(y|x_t)$ and variance $\sigma^2(y|x_t)$.

2.1 Online Expectation-Maximization

The parameters of the model (1) can be estimated using a maximum-likelihood estimator (MLE). Given a set of samples $\mathbf{Z} = \{\mathbf{z}_t; t = 1, \ldots, N\}$, the likelihood function is given by

$$\mathcal{L}[\mathbf{Z}; \mathbf{\Theta}] = \prod_{t=1}^{N} p(\mathbf{z}_t; \mathbf{\Theta}). \tag{10}$$

The maximum-likelihood estimation of the model parameters is the $\mathbf{\Theta}$ that maximizes the likelihood (10) for the data set $\mathbf{Z}$. Direct computation of the MLE requires complete information about which mixture component generated which instance. Since this information is missing, the EM algorithm is often used. The Expectation-Maximization (EM) algorithm (Dempster et al., 1977) is a general tool that permits estimating the parameters that maximize the likelihood function (10) for a broad 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.

For the estimation of the parameters $\mathbf{\Theta}$, the process starts with an initialization of the mean vectors and covariance matrices of the Gaussians. The E step consists in obtaining the probability $P(i|\mathbf{z}_t)$ for each component i of generating instance $\mathbf{z}_t$, that we denote by $w_{t,i}$,

$$w_{t,i} = P(i|\mathbf{z}_t) = \frac{P(i)p(\mathbf{z}_t|i)}{\sum_{j=1}^{K} P(j)p(\mathbf{z}_t|j)} = \frac{\alpha_i \mathcal{N}(\mathbf{z}_t; \mu_i, \mathbf{\Sigma}_i)}{\sum_{j=1}^{K} \alpha_j \mathcal{N}(\mathbf{z}_t; \mu_j, \mathbf{\Sigma}_j)}, \tag{11}$$

where $t = 1, .., N$ and $i = 1, .., K$. The value of $w_{t,i}$ can be seen as the proportion of the sample corresponding to Gaussian i that will be used to update the Gaussian

parameters in the maximization step. This step consists in computing the MLE using the estimated $w_{t,i}$. It can be shown (Duda et al., 2001) that, for the case of a GMM, the mixing parameters, means, and covariances are given by

$$\alpha_i = \frac{1}{N}\sum_{t=1}^{N} w_{t,i}, \tag{12}$$

$$\mu_i = \frac{\sum_{t=1}^{N} w_{t,i}\mathbf{z}_t}{\sum_{t=1}^{N} w_{t,i}}, \tag{13}$$

and

$$\mathbf{\Sigma}_i = \frac{\sum_{t=1}^{N} w_{t,i}(\mathbf{z}_t - \mu_i)(\mathbf{z}_t - \mu_i)^\mathsf{T}}{\sum_{t=1}^{N} w_{t,i}}, \tag{14}$$

respectively.

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 incremental FA, sample data are not all available at once: they arrive sequentially and must be used online to improve the approximation. This prevents the use of the offline EM algorithm, and requires an online, incremental version of it. We present an online, low complexity version of the EM algorithm, which is a modified version of that proposed in Sato and Ishii (2000). In the online 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 (11), except that it is only computed for the new sample $\mathbf{z}$. For the M step, the parameters of all mixture components are updated as

$$\alpha_i(t) = \frac{\ll 1 \gg_i^{[t]}}{\sum_{j=1}^{K} \ll 1 \gg_j^{[t]}}, \tag{15}$$

$$\mu_i(t) = \frac{\ll \mathbf{z} \gg_i^{[t]}}{\ll 1 \gg_i^{[t]}}, \tag{16}$$

and

$$\mathbf{\Sigma}_i(t) = \frac{\ll \mathbf{z}\mathbf{z}^\mathsf{T} \gg_i^{[t]}}{\ll 1 \gg_i^{[t]}} - \mu_i(t)\mu_i(t)^\mathsf{T}, \tag{17}$$

where $\ll f(x) \gg_i^{[t]}$ are time-discounted weighted sums calculated as [2]

[2] In Sato and Ishii (2000) time-discounted weighted means, instead of sums, are defined by normalizing the sums (18) with a factor $\eta_T = \left(\sum_{t=1}^{T}\prod_{s=t+1}^{T}\lambda(s)\right)^{-1}$. However, in the expressions of the estimators, all these η_T cancel out, and we can skip its calculation.

$$\ll f(x) \gg_i^{[t]} = \sum_{\tau=1}^{t} \left(\prod_{s=\tau+1}^{t} \lambda(s) \right) f(x_\tau) w_{\tau,i}, \tag{18}$$

where $\lambda(t) \in [0, 1]$ is a forgetting factor introduced to progressively decrease the influence of old, possibly outdated values. The value of $\lambda(t)$ is made to approach 1 as the number of experiences increases, as will be discussed later.

The sum $\ll 1 \gg_i^{[t]}$ in (15)-(17) represents the accumulated proportion of samples $w_{t,i}$, with forgetting determined by $\lambda(t)$, attributed to unit i along time. Similarly, $\ll \mathbf{z} \gg_i^{[t]}$ corresponds to the weighted sum with forgetting of sample vectors $\mathbf{z}_\tau$, and $\ll \mathbf{z}\mathbf{z}^\mathsf{T} \gg_i^{[t]}$ is the weighted sum with forgetting of the matrices obtained as the products $\mathbf{z}_\tau \mathbf{z}_\tau^\mathsf{T}$.

From (18), we obtain the recursive *time-dependent* updating formula for the incremental updating of the parameters,

$$\ll f(x) \gg_i^{[t]} = \lambda(t) \ll f(x) \gg_i^{[t-1]} + f(x_t) w_{t,i}. \tag{19}$$

But this approach has a drawback. The forgetting effect of $\lambda(t)$ is clearly seen in the incremental formula (19), which shows how, at each time step, all past data are multiplied by $\lambda(t)$, 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 (19) 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 $\lambda(t)$. It can be seen that this is exactly the case when setting $w_{t,i} = 0$ in Eq. (19), what yields:

$$\ll f(x) \gg_i^{[t]} = \lambda(t) \ll f(x) \gg_i^{[t-1]}, \tag{20}$$

showing that the accumulators of units that are seldom activated will progressively decay to 0. This situation is particularly annoying in the case of biased sampling, specially when some regions of the domain are sampled much more frequently than others (as in RL), so that in the long term, units covering sparsely sampled regions will get their statistics lost.

To avoid the undesired forgetting effect we need a formula that updates the parameters according to the new information provided, i.e., in the same proportion as the Gaussian activation $w_{t,i}$, rather than on time. For instance, when no new information is provided to a Gaussian, i.e., when $w_{t,i} = 0$, the cumulative sums should remain unmodified,

$$\ll f(x) \gg_i^{[t]} = \ll f(x) \gg_i^{[t-1]} . \tag{21}$$

On the other hand, if the full proportion of sample is provided to the Gaussian ($w_{t,i} = 1$), the updating of the cumulative sum should be the same as in Eq. (19),

$$\ll f(x) \gg_i^{[t]} = \lambda(t) \ll f(x) \gg_i^{[t-1]} + f(x_t). \tag{22}$$

It can be demonstrated (Celaya and Agostini, 2015) that the updating formula that regulates the updating according to the proportion of sample $w_{t,i}$ is

$$\ll f(x) \gg_i^{[t]} = \lambda(t)^{w_{t,i}} \ll f(x) \gg_i^{[t-1]} + \frac{1-\lambda(t)^{w_{t,i}}}{1-\lambda(t)} f(x_t). \tag{23}$$

With the *weight-dependent* updating formula (23), 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 Eq. (23), what we get is Eq. (21) 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 Eq. (19).

2.2 Local Adjustment of the Forgetting Factor $\lambda(\mathbf{z})$

In Sato and Ishii (1999), the evolution of the forgetting factor is modelled as

$$\lambda(t) = 1 - \frac{1-a}{a\,t\, + b}, \tag{24}$$

where a and b are parameters regulating the forgetting rate and t is the total number of experienced samples. Eq. (24) allows forgetting old possibly outdated estimations and better adapting the parameters to non-stationary changes as learning proceeds. In this case, the forgetting factor takes the same value for all the regions of the domain, preventing the regulation of the forgetting to the different local variations due to non-stationarity, typical of RL.

In our approach we use the same formula (24) for $\lambda(t)$ but, instead of using the total number of experienced samples, we use an estimation of the local number of samples $n(\mathbf{z})$, calculated from the density of samples in the vicinity of $\mathbf{z} = (\mathbf{x}, y)$. If the sample density near $\mathbf{z}$ is high, this indicates that the approximation is locally converging and the forgetting factor should approach 1. On the contrary, if the density near $\mathbf{z}$ is low, this may be because the input region has been still little sampled, or because the output of the points in this input region has changed, in which cases the region is far from convergence and the forgetting factor must be kept low.

The number of samples in the vicinity of the experienced point can be estimated from the density model as

$$n(\mathbf{z}) = n_T \int_{\hat{Z}} p(\mathbf{z}; \mathbf{\Theta}) d\mathbf{z}, \tag{25}$$

where n_T is the total number of samples represented in the model,

$$n_T = \sum_{j=1}^{K} \ll 1 \gg_j, \tag{26}$$

$\hat{Z}$ is a region surrounding $\mathbf{z}$, and $p(\mathbf{z})$ is the probability density function at $\mathbf{z}$ estimated with the GMM (Eq. (1)). In order to simplify the calculation of the integral in (25) we can assume that $\hat{Z}$ is small enough for the probability density function $p(\mathbf{z})$ to be nearly constant in this region, which permits estimating the number of points as

$$n(\mathbf{z}) \approx V_z \, n_T \, p(\mathbf{z}), \tag{27}$$

where V_z is the volume of $\hat{Z}$, defined empirically.

By replacing (27) in (24) we get the point-dependent forgetting factor

$$\lambda(\mathbf{z}) = 1 - \frac{1-a}{a \, n_T \, V_z \, p(\mathbf{z}) \, + b}. \tag{28}$$

2.3 Gaussian Generation

To approximate the target function to any desired precision we make the function approximation non-parametric by permitting Gaussian generation on demand for a better approximation. A new Gaussian is generated each time the prediction error is too large and there is a lack of nearby Gaussians to locally improve the approximation. A large prediction error is determined by

$$(g(\mathbf{x}) - \mu(y|\mathbf{x}))^2 \geq \mathrm{thr}_{\mathrm{error}}, \tag{29}$$

where $\mathrm{thr}_{\mathrm{error}}$ is the threshold for the error. The lack of nearby Gaussians is reflected by a low density, which means that the influence of the existing Gaussians at the experienced point is weak. We say that there is a low density at the experienced point whenever it goes below a threshold,

$$p(\mathbf{x}, y) \leq \mathrm{thr}_{\mathrm{density}}, \tag{30}$$

where $\mathrm{thr}_{\mathrm{density}}$ is the threshold for the density.

Initialization of New Gaussians

Once the criteria (29) and (30) are fulfilled, a new Gaussian is generated centred in the experienced point,

$$\mu_{K+1}(\mathbf{x}, y) = (\mathbf{x}_t, y_t), \tag{31}$$

with initial number of accumulated samples equal to 1,

$$\ll 1 \gg_{K+1} = 1. \tag{32}$$

To allow for a rapid improvement of the approximation with the addition of the new Gaussian, the initial weight of the added Gaussian $w_{t,K+1}$ (see Eq. (11)) at the experienced point $\mathbf{z}_t = (\mathbf{x}_t, y_t)$ should be large enough to produce a significant change in the cumulative values (see Eq. (23)). This is implemented by regulating the initialization of the covariance of the new Gaussian, $\mathbf{\Sigma}_{K+1}$, so that the weight of the new Gaussian at $\mathbf{z}_t$ is equal to a predefined value $w_{t,K+1} = w_{new}$.

The covariance of the new Gaussian is defined as

$$\mathbf{\Sigma}_{K+1} = diag\{(C \, d_1)^2, ..., (C \, d_D)^2\}, \tag{33}$$

where d_i is the total range size of the variable i; D is the dimension of the joint space; and C is a factor calculated so that the weight at the experienced point is equal to w_{new}. To determine the value of C we use the D-dimensional multivariate Gaussian formula

$$\mathcal{N}(\mathbf{z}; \mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D \, |\mathbf{\Sigma}|}} \, e^{-\frac{1}{2}(\mathbf{z}-\mu)' \mathbf{\Sigma}^{-1} (\mathbf{z}-\mu)}, \tag{34}$$

which, for the case of the new added unit evaluated at $\mathbf{z}_t$, results in

$$\mathcal{N}(\mathbf{z}_t; \mu_{K+1}, \mathbf{\Sigma}_{K+1}) = \frac{1}{\sqrt{(2\pi)^D \, |\mathbf{\Sigma}_{K+1}|}}. \tag{35}$$

By replacing (33) in (35) and computing the determinant we get

$$\mathcal{N}(\mathbf{z}_t; \mu_{K+1}, \mathbf{\Sigma}_{K+1}) = \frac{1}{\sqrt{(2\pi)^D \, C^{2D} \, \prod_{j=1}^{D} d_j^2}}, \tag{36}$$

from which we can clear C,

$$C = \frac{1}{\sqrt{(2\pi)}} \left(\mathcal{N}^2(\mathbf{z}_t; \mu_{K+1}, \mathbf{\Sigma}_{K+1}) \, \prod_{j=1}^{D} d_j^2 \right)^{-\frac{1}{2D}}. \tag{37}$$

To complete the calculation of C we need to define the value of $\mathcal{N}(\mathbf{z}_t; \mu_{K+1}, \mathbf{\Sigma}_{K+1})$. This is done by including $\mathcal{N}(\mathbf{z}_t; \mu_{K+1}, \mathbf{\Sigma}_{K+1})$ in Eq. (11) and making it equal to w_{new} [3],

$$w_{new} = w_{t,K+1} = \frac{\ll 1 \gg_{K+1} \mathcal{N}(\mathbf{z}_t; \mu_{K+1}, \mathbf{\Sigma}_{K+1})}{\sum_{j=1}^{K+1} \ll 1 \gg_j \mathcal{N}(\mathbf{z}_t; \mu_j, \mathbf{\Sigma}_j)}. \tag{38}$$

Then, resolving for $\mathcal{N}(\mathbf{z}_t; \mu_{K+1}, \Sigma_{K+1})$, we get

$$\mathcal{N}(\mathbf{z}_t; \mu_{K+1}, \mathbf{\Sigma}_{K+1}) = \frac{w_{new}}{1 - w_{new}} \frac{\sum_{i=1}^{K} \ll 1 \gg_i \mathcal{N}(\mathbf{z}_t; \mu_i, \mathbf{\Sigma}_i)}{\ll 1 \gg_{K+1}}. \tag{39}$$

By replacing (39) in (37) we obtain the final formula for C,

$$C(\mathbf{z}) = \frac{1}{\sqrt{2\pi}} \left[\left(\frac{w_{new}}{1 - w_{new}} \frac{\sum_{i=1}^{K} \ll 1 \gg_i \mathcal{N}(\mathbf{z}; \mu_i, \mathbf{\Sigma}_i)}{\ll 1 \gg_{K+1}} \right)^2 \prod_{j=1}^{D} d_j^2 \right]^{-\frac{1}{2D}}. \tag{40}$$

[3] In Eq. (38), we have replaced the mixing parameters α_i by their estimations using the accumulated samples $\alpha_i = \frac{\ll 1 \gg_i}{\sum_{j=1}^{K+1} \ll 1 \gg_j}$ (see Eq. (15)), which permits cancelling out the term $\sum_{j=1}^{K+1} \ll 1 \gg_j$ from the equation.

Algorithm 1 The GMMFA Algorithm

```
Initialize the GMM with 1 Gaussian.
loop
  Get observation (x_t, y_t)
  Calculate the activation w_{t,i} of each Gaussian in (x_t, y_t) (11) (E step)
  Update the parameters of the GMM, Θ = {α_i, μ_i, Σ_i}, i = 1, ..., K (23) (M step)
  Calculate μ(y|x_t) (8)
  Calculate the approximation error e = (y_t − μ(y|x_t))^2
  if e ≥ thr_error then
    Get density of samples p(x, y)
    if p(x, y) ≤ thr_density then
      Generate new Gaussian (31-33)
    end if
  end if
end loop
```

To complete the initialization process we use the initial number of accumulated samples $\ll 1 \gg_{K+1}$ to derive the cumulative sums $\ll \mathbf{z} \gg_{K+1}$ in Eq. (16) and $\ll \mathbf{z}\mathbf{z}^{\mathsf{T}} \gg_{K+1}$ in Eq. (17), from the initial mean μ_{K+1} (31) and covariance $\mathbf{\Sigma}_{K+1}$ (33), respectively.

2.4 Algorithm for the GMMFA

The complete algorithm for the incremental function approximation using a GMM is presented in Alg. 1.

3 Reinforcement Learning using a GMM (GMMRL)

Reinforcement Learning is a paradigm in which an agent has to learn an optimal action policy by interacting with its environment (Sutton and Barto, 1998). 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

$$R^{\pi}(s_t) = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{41}$$

where $\gamma \in [0, 1]$ is the discount rate that regulates the importance of future rewards with respect to immediate ones. In stochastic environments the actual return (41) may vary at different runs depending on the probability of transitions and the reward obtained after each execution. Under these circumstances, a good prediction of the cumulative sum of rewards starting at state s and then following policy π is the expected value of the return,

$$V^{\pi}(s) = \mathrm{E}[R^{\pi}(s)], \tag{42}$$

known as the *value function* for policy π. In a similar way, the expected return can also be associated to an action a in state s,

$$Q^{\pi}(s,a) = \mathrm{E}[R^{\pi}(s,a)], \tag{43}$$

where a is any of the possible actions in s. The function $Q^{\pi}(s,a)$ is known as the *action-value function*, or simply the *Q-function*, and is the expected return that would be obtained after executing action a in state s and then following policy π. An optimal policy is one that maximizes the expected return.

The optimal value function or the optimal action-value function is that corresponding to an optimal policy, and can be estimated using different RL approaches (Sutton and Barto, 1998). One of the most popular is the actor/critic approach, where a policy function (called the actor) is learned and explicitly stored, so that actions are directly decided by the actor. The critic, on the other hand, learns the value (or action-value) function associated to the policy coded in the actor. Its main role is to evaluate which other actions would have resulted in a higher cumulative reward than the one selected by the actor. This information is used by the actor to improve the action policy. In this work we use the actor/critic architecture as in Sato and Ishii (1999). This architecture consists of a critic, which approximates the Q-function for a particular policy, $Q^{\pi}(s,a)$, and an actor that codes this policy, $a = \pi(s)$. In our approach, both, the action-value function $Q^{\pi}(s,a)$ and the policy $a = \pi(s)$, are approximated incrementally using the GMMFA algorithm described in Alg. 1.

3.1 Action Selection

The action policy is approximated using a density model defined in the joint space of states and actions,

$$p(s,a) = \sum_{i=1}^{K_{\pi}} \alpha_{\pi,i} \mathcal{N}(s,a;\mu_{\pi,i},\mathbf{\Sigma}_{\pi,i}), \tag{44}$$

where $\{\alpha_{\pi,i}, \mu_{\pi,i}, \mathbf{\Sigma}_{\pi,i}\}$, $i = 1, ..., K_{\pi}$, are the parameters for the density model of the action policy π. The value of the action policy $\pi(s_t)$ is taken as the expected value of a in state s_t, and is obtained from the density model of the actor as the conditional probability (8),

$$\pi(s) = \mu(a|s) = \sum_{i=1}^{K_{\pi}} \beta_{\pi,i}(s) \mu_{\pi,i}(a|s), \tag{45}$$

where $\beta_{\pi,i}(s)$ and $\mu_{\pi,i}(a|s)$ are calculated as in Eqs. (5) and (3), respectively. However, action selection in RL must address the exploration/exploitation trade-off to explore alternative, possibly better policies. This trade-off is approached by generating a stochastic action for a visited state using the probability density function

$$p(a|s) = \sum_{i=1}^{K_\pi} \beta_{\pi,i}(s)\mathcal{N}(a; \mu_{\pi,i}(a|s), \sigma_{\pi,i}(a)), \tag{46}$$

where $\sigma_{\pi,i}(a)$ is calculated as in Eq. (4). A stochastic action is obtained using the generative approach consisting of, first, selecting a Gaussian randomly according to the probability $\beta_{\pi,i}$, and, then, drawing an action randomly from the corresponding Gaussian distribution $\mathcal{N}(s, a; \mu_{\pi,i}(a|s), \sigma_{\pi,i}(a))$. Optionally, exploration can be increased by adding some extra random noise to the action.

3.2 Critic Update

The Q-function in the critic is approximated using a density model defined in the space of states, actions, and q-values,

$$p(s, a, q) = \sum_{i=1}^{K_Q} \alpha_{Q,i}\mathcal{N}(s, a, q; \mu_{Q,i}, \mathbf{\Sigma}_{Q,i}), \tag{47}$$

where $\{\alpha_{Q,i}, \mu_{Q,i}, \mathbf{\Sigma}_{Q,i}\}$, $i = 1, ..., K_Q$, are the parameters for the density model of the Q-function. The samples used for the updating of the density model are of the form $(s, a, q) = (s_t, a_t, q(s_t, a_t))$, corresponding to the visited state s_t, the executed action a_t, and the estimated value of $q(s_t, a_t)$ given by

$$q(s_t, a_t) = r(s_t, a_t) + \gamma Q^\pi(s_{t+1}, a_{t+1}). \tag{48}$$

To obtain this estimation we need to evaluate $Q^\pi(s_{t+1}, a_{t+1})$ using Eq. (8),

$$Q^\pi(s_{t+1}, a_{t+1}) = \mu(q|s_{t+1}, a_{t+1}), \tag{49}$$

where $a_{t+1} = \pi(s_{t+1})$ is obtained from Eq. (45).

3.3 Actor Update (Policy Improvement)

The samples used for the updating of the density model of the actor are of the form $(s, a) = (s_t, a_{target})$, corresponding to the visited state s_t and the target action a_{target}, determined so as to converge to the maximum of the function $Q^\pi(s_t, a)$. The traditional approach to determine the target action is to use the gradient of the Q-function (Sato and Ishii, 1999),

$$a_{target} = a_0 + \epsilon \nabla Q^\pi(s_t, a_0), \tag{50}$$

where $a_0 = \pi(s_t)$ is the value of the action policy at s_t, ϵ is a small constant, and $\nabla Q^\pi(s_t, a_0)$ is the gradient of $Q^\pi(s_t, a)$ with respect to the action variable a at a_0. The gradient locally adjusts the policy in the direction of the maximum growth of the function $Q^\pi(s_t, a)$. However, if the function is nonlinear, this local adjustment may lead to a local maximum preventing the optimal policy to be found. On the other hand, since the ϵ should be kept small to favour convergence, the gradient method may require several iterations to reach the global maximum if it lies far away from a_0. Another drawback of using the gradient approach to determine the target action is that the just

Algorithm 2 The GMMRL algorithm

Initialize the GMM of the action policy with 1 Gaussian.
Initialize the GMM of the Q-function with 1 Gaussian.
Observe current state s_t
loop
  Select an action a_t randomly for state s_t (Eq. (46))
  Execute a_t, get $r(s_t, a_t)$, and observe new state s_{t+1}
  Generate $q(s_t, a_t) = r(s_t, a_t) + \gamma Q^{\pi}(s_{t+1}, a_{t+1})$ (Eq. (48))
  Update the GMM of the Q-function using sample $(s_t, a_t, q(s_t, a_t))$ (Alg. 1)
  Generate a_{target} (Sec. 3.3)
  Update the GMM of the policy π using sample (s_t, a_{target}) (Alg. 1)
  $s_t \leftarrow s_{t+1}$
end loop

acquired experience of executing the exploration action a_t, consisting of the resulting reward and next state, is not immediately used to improve the policy. The actor will be able to use this information only through the critic, after it has incorporated such information following a number of similar observations.

Thus, we improve the selection of the target action by first evaluating the executed action using the observed q-value (48). Similarly, we use the critic to evaluate the deterministic action a_0 provided by the actor in state s_t to obtain $Q^{\pi}(s_t, a_0)$. Then, we compare both results, and if the executed action a_t yields a better value than a_0, we take a_t as the target action. In the case that the action provided by the actor a_0 appears to be better than the executed action a_t, we rely on the gradient approach to gradually improve the policy. In this case, we also allow for a local search by evaluating a number of actions at fixed small intervals around a_0, to facilitate a faster progress towards the maximum, or use the gradient (50) if a_0 is better than the evaluated actions. Note that this strategy for policy improvement not only increases the chances of finding the global maximum but also performs a fine adjustment of the action policy when the maximum lies close to a_0.

3.4 Algorithm for the GMMRL

The compilation of all the processes taking place in the method of GMMRL in an actor/critic architecture is shown in Alg. 2.

4 Experiments

We evaluated the performance of the GMMRL on two scenarios with different complexities: the standard benchmark of an inverted pendulum with limited torque (Doya, 2000; Sato and Ishii, 1999) and the cart-pole benchmark (Deisenroth, 2010). In the first benchmark we compare the results obtained by using the traditional time-dependent updating (19), proposed in Sato and Ishii (1999), with the results obtained with our weight-dependent updating (23) for the updating of the parameters of the GMMs. In the latter case, we ran experiments using the global forgetting factor $\lambda(t)$ (24) and the point-wise

forgetting factor $\lambda(\mathbf{z})$ (28). We compare the results obtained with those of the state-of-the-art approach PILCO (Deisenroth and Rasmussen, 2011). In the inverted pendulum benchmark we also evaluate the improvements obtained by the point-dependent initialization of the covariance of a new Gaussian using the density information (Sec. 2.3) and we carried out a specific experiment for function approximation where the non-stationarity problem of the RL was avoided to clearly visualize the effects of biased sampling in function approximation with the two types of updating: time- and weight-dependent (see Sec.4.4).

The scalability of the approach is assessed using the more complex benchmark of the cart-pole. We tested the performance of the GMMRL method in the cart-pole scenario as presented in Deisenroth (2010), which represents a higher dimensional, more complex control problem than the inverted pendulum. As done in the inverted pendulum scenario, we also compare the performance of the GMMRL with that of the PILCO approach.

4.1 The Inverted Pendulum

The inverted pendulum with limited torque task consists of swinging a 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 the inverted pendulum problem is two-dimensional and comprises the angular position θ and angular velocity $\dot{\theta}$: $s = (\theta, \dot{\theta})$, where θ takes values in the interval $[-\pi, \pi]$. As the reward signal we simply use the absolute value of the distance to the goal position at $\theta = 0$, $r(s, a) = -|\theta|$. The discount coefficient γ used to update the Q-values of the critic (Eq. (48)) is set to 0.85.

The Gaussians of the mixture model for the actor are three-dimensional, providing an estimation of the probability densities in the joint space $(\theta, \dot{\theta}, a)$. On the other hand, the Gaussians of the mixture model for the critic is four-dimensional, defined in the joint space $(\theta, \dot{\theta}, a, q)$ [4].

The threshold for the approximation error to evaluate the necessity of a Gaussian generation (Sec. 2.3) was set to $\mathrm{thr}_{\mathrm{error}} = (0.0667\ \mathrm{rang}_a)^2$ for the actor and $\mathrm{thr}_{\mathrm{error}} = (0.1\ \mathrm{rang}_c)^2$ for the critic, where rang_a and rang_c are the ranges of the output variables of the actor and the critic, respectively. The initial influence of the new Gaussians at the mean vector was set to $w_{new} = 0.95$ (see Eq.(40)). Finally, the volume V_z for calculating $n(\mathbf{z})$ in Eq. (27), was set to an arbitrary small proportion of the total volume of the domain V_T, $V_z = 0.0001\ V_T$.

In all the experiments the evaluation is implemented by averaging the results of 20 independent runs of 300 episodes each. Each episode consists of 5 seconds of simulated time with an actuation interval of 0.1 seconds. At the beginning of each episode the pendulum is placed in the downward position. After each training episode, a test

[4] In the function approximation we use the spatial symmetry of the inverted pendulum problem with respect to $\theta = 0$ to have a more compact representation of the state space.

episode exploiting the policy learned so far is done and the total accumulated reward is computed. We plot the average values and the confidence interval for these averages for a confidence level of 95 %.

4.2 Time-dependent vs. Weight-dependent Updating

Fig. 2(a) presents the results of the comparison between using the time-dependent updating and the weight-dependent updating for the best combination of parameters a and b found for each case. It can be observed that the time-dependent updating presents the poorest performance (black line), presumably due to the undesired forgetting of past estimations when no new samples are provided. The weight-dependent updating, which prevents this undesired forgetting, significantly improves the performance with respect to the time-dependent updating (blue and red lines). The best result was obtained by using the point-dependent forgetting factor $\lambda(\mathbf{z})$, which permits a better adaptation to the local non-stationary changes of the target function (red line). In this case, the average simulated time required to swing up and stabilize the pendulum in the upright position (corresponding to a cumulative reward of around -55) is of 1100 seconds with an average number of Gaussians of 56 for the actor and 75 for the critic. We would like to remark that our best experiment reached convergence in 510 seconds of simulated time. An example of the control achieved after convergence can be seen in the video available at https://dl.dropboxusercontent.com/u/19473422/NECOinvpend.wmv.

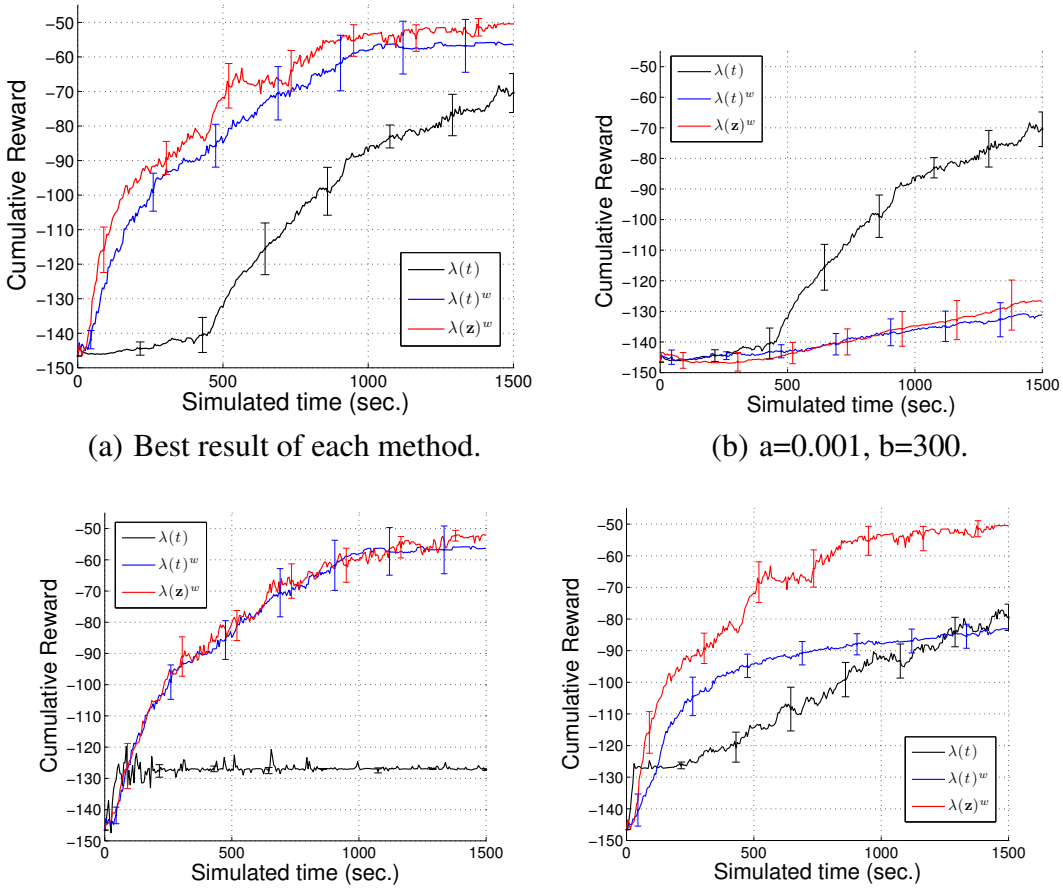
The results presented in Fig. 2(a) were obtained from an exhaustive evaluation of the sensitivity to the parameters a and b of the forgetting factor. To better see the effects of changing a and b in the forgetting factor, Figures 2(b), 2(c), and 2(d) present a comparison of the results between the time- and weight-dependent forgetting using the best set of parameters found for each case. To ease the interpretation of the results, we rewrite the formula used for the forgetting factor (24):

$$\lambda(t) = 1 - \frac{1-a}{a\,t\ + b}.$$

We can observe that, for the case of the time-dependent updating, the best results were obtained for higher values of b (Fig. 2(b)). This can be explained by the fact that higher values of b establish a higher lower bound of the forgetting factor, reducing the forgetting rate at all stages of the learning, which mitigates the undesired forgetting effect when a time-dependent updating is used. The best results were obtained for $a = 0.01$ and $b = 300$ (see Fig. 2(b)).

For the case of the weight-dependent updating, the best results were obtained for lower values of b, rather than higher values as in the time-dependent updating case. This can be caused by the regulation of the forgetting with the new information provided rather than with time, allowing for a smaller lower bound of the forgetting factor (smaller b) that permits replacing outdated estimations faster but consistently with the new information provided, avoiding forgetting values in regions far away from the sample.

The improvements in the performance with the values of b is similar for both cases of the weight-dependent updating. However, the behaviour of these cases under the variation of a presents some remarkable differences. For instance, for the case of using



(a) Best result of each method.

(b) a=0.001, b=300.

(c) a=0.001, b=5.

(d) a=0.01, b=5.

Figure 2: Comparison of the 3 updating methods for different combinations of a and b. Method 1: time-dependent updating ($\lambda(t)$). Method 2: weight-dependent updating with global forgetting factor ($\lambda(t)^w$). Method 3: weight-dependent updating with point-wise forgetting factor ($\lambda(\mathbf{z})^w$).

the global forgetting factor ($\lambda(t)^w$), the higher value of $a = 0.01$ (Fig. 2(d)) presents a poorer performance than the lower one, $a = 0.001$ (Fig. 2(c)). The reason behind this might be that the total number of experiences t quickly increases as learning proceeds, which, in turn, quickly increases the forgetting factor if the value of a is relatively large. This may prevent the system to properly adapt to the non-stationary changes taking place during the learning process. Interestingly, this is not the case when using $\lambda(\mathbf{z})^w$, which depends on the local number of samples $n(\mathbf{z})$, which is in general much smaller than the total number of experiences. In this case, a higher value of a would permit better considering the influence of $n(\mathbf{z})$ in the forgetting, which improves the capability of the system to regulate the forgetting according to the non-stationary changes (Fig. 2(d)). The best results were obtained with $a = 0.001$ and $b = 5$, for the case of $\lambda(t)^w$, and with $a = 0.01$ and $b = 5$, for the case of $\lambda(\mathbf{z})^w$.

4.3 Gaussian Generation with Point-dependent Initialization of the Covariance

In this section we evaluate the improvements obtained by initializing the covariance of the new Gaussian (33) with a point-dependent dispersion $C(\mathbf{z})$ (see Sec. 2.3). Recall that a new Gaussian is generated with mean value given by the experienced point to use the fresh information provided by this point to improve the approximation. To make a more efficient use of this information we increase the influence of this point in the approximation (w_{new}) to any desired value by regulating the dispersion of the covariance as in Eq. (40). To show the improvement in the performance obtained by allowing this regulation, we compare the results obtained by considering a *fixed* dispersion $C = 0.2$ to initialize the covariance (without regulation) with those obtained by considering a point-dependent dispersion $C(\mathbf{z})$ (with regulation).

The results are visualized in Fig. 3. We can observe an improvement in the performance when using a point-dependent dispersion $C(\mathbf{z})$, which demonstrates the advantage of using the density information to regulate the influence of the new Gaussians in the approximation.

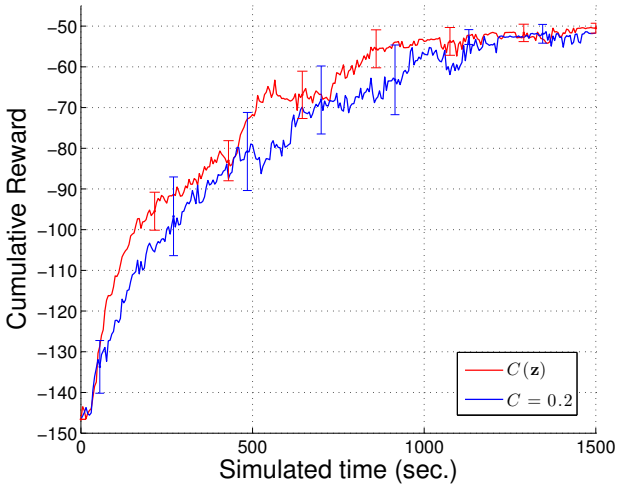


Figure 3: Comparison of the performance of the Gaussian generation using a fixed initial dispersion, $C = 0.2$, and a variable initial dispersion, $C(\mathbf{z})$.

4.4 Effects of Biased Sampling

The reason why we introduced the weight-dependent updating was to attenuate the wrong effects that biased sampling produces in the online function approximation. To show how effective is our approach in this aspect, we compared the performance of the time- and the weight-dependent updating in the task of approximating a given function when the samples present a biased distribution. To test this on a setting related with our learning problem, we take as the function to be approximated the $Q(s, a)$ function learned by the critic in one of the experiments, and we apply Alg. 1 to approximate it by providing as training samples the whole set of samples recorded from one execution

of the learning process, presented in the same order they were collected. Fig. 4 shows the distribution of these samples, which, as can be appreciated, is strongly biased. For better accuracy in the comparison, we initialized each GMM with 64 Gaussians equally distributed in the state-action space and no further Gaussians were generated along the process. The evaluation was carried out for different fixed values of the forgetting factor $\lambda(t)$.

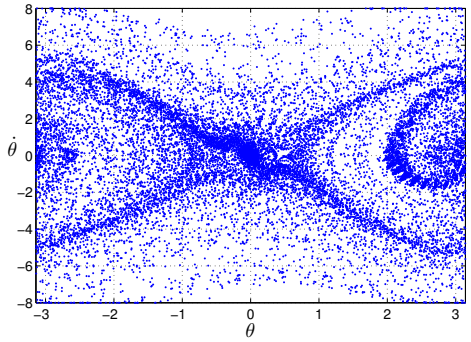


Figure 4: Example of sample distribution from the trajectories followed by the inverted pendulum.

Fig. 5 presents the average MSE of the last 10000 iterations of 10 runs. We can observe a significant difference between the performances of the time-dependent updating with respect to the weight-dependent updating. The forgetting caused by lower values of $\lambda(t)$ increases the undesired forgetting effect when the time-dependent updating is used. On the contrary, the weight-dependent updating presents a better performance for all the values of $\lambda(t)$, with the best results around $\lambda(t) = 0.995$. As expected, both updating formulas performed identically for $\lambda(t) = 1$, in which case Eq. (19) is equal to Eq. (23).

4.5 Comparison with PILCO

To provide a reference for the performance, we compare our results with those reported for the state-of-the-art approach PILCO (Deisenroth and Rasmussen, 2011; Deisenroth, 2010). PILCO is a model-based policy search method that copes with the model bias problem, typical of model-based approaches when learning from scratch, by using a probabilistic dynamics model to consider model uncertainty.

PILCO is a *batch* approach, i.e. it can collect data and make use of it as desired, so that the total time of experience with the plant (or obtained by simulation) can be reduced to a very minimum. Batch algorithms like PILCO are appealing for problems where the cost of getting new samples is high, like, for instance, real-robot applications with intensive wear out. On the contrary, our GMMRL approach is model-free and works *online* without storing any sample. Online model-free approaches like ours avoid the computational burden of storing and processing samples offline and are more suitable for problems in which the data are continuously produced by the plant at no

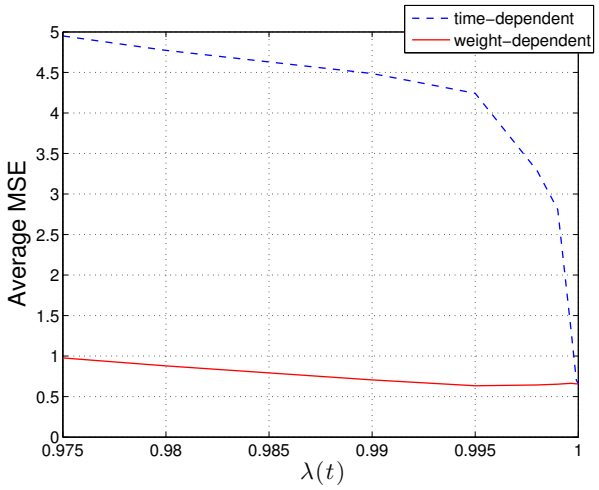


Figure 5: Average of the MSE of the approximation for different values of $\lambda(t)$.

extra cost such as flight stabilization, online price control, control of robotic prostheses, control of wind-power generators, gait generation in legged robots, etc.

Because of these important differences, comparing the simulation time needed by both systems would not be fair. Therefore, to provide an idea of the performance of the GMMRL approach with respect to PILCO we compare the *computation* time consumed by both methods to converge in the inverted pendulum problem (and later in the cart-pole problem as explained in Sec. 4.6). To do this, we downloaded the PILCO implementation available at http://mlg.eng.cam.ac.uk/pilco/ and used an implementation of the GMMRL in the same language as PILCO (Matlab).

The comparison is made considering, on the one hand, the CPU time, i.e. the sums across all threads of the processing time, calculated using the `cputime` function of Matlab (Mathworks, 2016). This measure provides an idea of the actual processing resources used by each method. On the other hand, we compare the wall-clock time, computed using the `tic` and `toc` functions of Matlab, which indicates the time elapsed while the experiments are run.

The CPU time consumed by the GMMRL approach was 562 seconds which contrasts with the 5184 seconds consumed by PILCO[5]. On the other hand, the time elapsed until convergence was 581 seconds for the GMMRL approach and 1433 seconds for PILCO. The results showed that the GMMRL needed less wall-clock time and much less CPU time to converge than the PILCO approach. This indicates that the GMMRL is able to achieve a comparable performance to PILCO using significantly less processing effort.

[5]We ran the implementations in a MacBook Pro, 2.9 Ghz Intel Core i7 with 8 GB of memory.

4.6 The Cart-pole Swing-up

In order to assess the scalability of the approach, we also tested the GMMRL method in the cart-pole scenario. The scenario consists of a pole mounted on a cart that has to be stabilized in the upright position by the motions of the cart. This control problem is more demanding than the inverted pendulum since the state space has two more dimensions and the external actions are now applied to a moving cart rather than directly to the pendulum. For the evaluation, we used the same experimental set-up than in Deisenroth (2010), where the complexity of the task is increased by additionally requiring that the cart is at a specific target position when the pendulum is stabilized.

The state space of the cart-pole problem is four-dimensional and comprises the position x and velocity $\dot{x}$ of the cart as well as the angular position θ and angular velocity $\dot{\theta}$ of the pole: $s = (x, \dot{x}, \theta, \dot{\theta})$, where x takes values in the interval $[-6, 6]$ and θ in the interval $[-\pi, \pi]$. As the reward signal we use the negative of the cost function used in Deisenroth (2010) for the cart-pole scenario:

$$r(s, a) = -(1 - exp(-0.5\,(j - j_{target})\,T^{-1}\,(j - j_{target})')); \tag{51}$$

where $j = (x, cos(\theta), sin(\theta))$ is the current position of the cart and the pole, $j_{target} = (0, 0, 1)$ is the target position, and

$$T^{-1} := A^2 \begin{bmatrix} 1 & l & 0 \\ l & l^2 & 0 \\ 0 & 0 & l^2 \end{bmatrix} \tag{52}$$

is the precision matrix, where A^2 is a scalar parameter controlling the width of the cost parabola and l is the length the pole. In our experiments we set A^2 to 1. The discount coefficient γ to update the Q-values of the critic is set to 0.85.

The Gaussians of the mixture model for the actor have five dimensions, the same as state representation plus the action: $(x, \dot{x}, \theta, \dot{\theta}, a)$. On the other hand, the Gaussians for the critic are six-dimensional: $(x, \dot{x}, \theta, \dot{\theta}, a, q)$.

The learning parameters used for the experiments were defined empirically: $\mathrm{thr}_{\mathrm{error}} = (0.05\ \mathrm{rang}_a)^2$ for the actor and $\mathrm{thr}_{\mathrm{error}} = (0.1\ \mathrm{rang}_c)^2$ for the critic; $a = 0.01$, $b = 10$; and $V_z = 0.0001\ V_T$ for $\lambda(\mathbf{z})$ (Eq. 28), where V_T is the total volume of the domain. Finally, the initial weight of the new Gaussians to initialize the covariances (Eq.(40)) is set to $w_{new} = 0.25$.

The experiments are carried out using episodes of 4 seconds of simulated time with an action interval of 0.1 seconds. After some training episodes a test episode of 4 seconds is performed and the total accumulated reward is computed. At the beginning of each episode the cart is placed at the center of the track and the pendulum is placed in the downward position.

Fig. 6 presents the average of 20 experiments. The average number of Gaussians generated after learning was 92 for the actor and 172 for the critic. The average simulated time for convergence, i.e. the time required to swing-up and stabilize the pole in the upright position with the cart at the center (achieved with around -4.5 of accumulated reward) is 7100 seconds. It is important to point out that the estimated simulated time for convergence is extracted from the average of 20 experiments and that many experiments reached the convergence value much earlier than 7100 seconds of simulated

time. For instance, in our best experiment, the simulated time for convergence was 4200 seconds. An example of the control achieved after convergence can be seen in the video available at https://dl.dropboxusercontent.com/u/19473422/NECOcartpole.wmv.

As carried out in the inverted pendulum case, to provide a reference of the performance we compare the CPU time as well as the wall-clock time required to converge of our approach with that of the PILCO method (see Sec. 4.5 for more information about the set-up for this comparison). In this scenario, the time elapsed until convergence for the GMMRL was 6892 seconds and 7041 seconds for PILCO. As for the inverted pendulum case, the CPU time difference is significantly higher. The CPU time consumed by GMMRL was 6740 seconds, while the one for PILCO was 28173 seconds, showing that the GMMRL uses much less processing effort to achieve a comparable performance.

Figure 6: Performance of the GMMRL in the cart-pole scenario.

5 Conclusions

In this work we approached the problem of function approximation in incremental reinforcement learning. This problem entails two main challenges for the FA method: avoid large distortions in the approximation at regions far away from the current sample and properly adapt the approximation to local non-stationary changes.

Our approach was based on the previous contributions by Sato and Ishii (1999, 2000). They proposed an incremental FA method using a normalized Gaussian Network (NGnet) to deal with the incremental RL problems. We extended the capabilities of the NGnet approach by using a Gaussian mixture model (GMM) for function representation. The GMM provides a straightforward representation of the probability density function in the joint input-output space. We have shown that this additional density information provides useful tools for dealing with the challenges of incremental RL. In particular, the probability density function captures all the information available to the RL agent: In the first place, it provides a function approximation as the mean of

the sample values; in the second place, it provides a full probability distribution of the possible values of the output variable.

By using the weight-dependent updating formula (23) we updated the parameters of the Gaussians only when new information is provided and in an exact proportion of this information. This prevented the undesired distortions in the approximation at regions far away of the sample that takes place when the traditional time-dependent updating is used. The improvements achieved can be observed from the comparisons between the time-dependent updating proposed by Sato and Ishii (1999, 2000) and our weight-dependent updating (Fig. 2(a)).

The density information also permitted a local regulation of the forgetting of past estimations when new information was provided. This local forgetting permits specifically tracking the local non-stationary changes and rapidly adapting the parameters according to the local approximation demands. The local forgetting was implemented by using the number of samples in the vicinity of the experienced point, $n(\mathbf{z})$, to regulate the forgetting factor (28). This contrasts with the traditional approach of using the total number of samples in the forgetting factor, which does not permit a local regulation of the forgetting. The improvement achieved by using the local number of samples $n(\mathbf{z})$ can be seen in Fig. 2(a).

Our approach is non-parametric as it generates new Gaussians on demand for a better approximation. We propose a Gaussian generation approach that uses the density information to regulate the influence of a new Gaussian in the approximation (see Sec. 2.3). This increases the speed at which the new Gaussian improves the approximation in its main region of influence. The improvements achieved by using this generation strategy are observed in Fig. 3.

There are additional advantages of using a GMM for function approximation in incremental RL. Although our GMM provides more information than the NGnet used in Sato and Ishii (2000), it requires less parameters for storing this information, hence providing a more compact representation. In addition, from the probability distribution of output values, it is possible to estimate a point-wise variance of the samples (9), which has been argued to be an important feature of Gaussian process methods (GPs) (Engel et al., 2005; Deisenroth et al., 2009). The point-wise variance estimation can be used, for example, to enrich the repertory of exploration-exploitation strategies, a possibility suggested in Engel et al. (2003, 2005). This variance estimation was also used to evaluate the quality in the approximation in a competitive strategy (Agostini and Celaya, 2011), where several parallel function approximators, each of them consisting of a GMM, compete to provide the inference of the output value at a specific input. Finally, by using an online version of the EM algorithm, the training of the GMM can be done incrementally and, thanks to the simplicity of the GMM, the updating process is computationally efficient.

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

References

Agostini, A. and Celaya, E. (2010). Reinforcement Learning with a Gaussian Mixture Model. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'10)*, pages 3485–3492.

Agostini, A. and Celaya, E. (2011). A Competitive Strategy for Function Approximation in Q-learning. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 1146–1151. AAAI Press.

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

Celaya, E. and Agostini, A. (2015). Online EM with Weight-Based Forgetting. *Neural Computation*, 27(5):1142–1157.

Deisenroth, M., Rasmussen, C., and Peters, J. (2009). Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524.

Deisenroth, M. P. (2010). *Efficient reinforcement learning using Gaussian processes*. KIT Scientific Publishing.

Deisenroth, M. P. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 465–472.

Dempster, A., Laird, N., Rubin, D., et al. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245.

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

Engel, Y., Mannor, S., and Meir, R. (2003). Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 154–161.

Engel, Y., Mannor, S., and Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd international conference on machine learning (ICML'05)*, pages 201–208. ACM.

Figueiredo, M. (2000). On Gaussian radial basis function approximations: Interpretation, extensions, and learning strategies. *Proceedings of the International Conference on Pattern Recognition*, 2:618–621.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In *Proceedings of the international conference on machine learning (ICML)*, pages 261–268.

Mathworks (2016). http://www.mathworks.com.

Sato, M.-A. and Ishii, S. (1999). Reinforcement learning based on on-line em algorithm. In *Proceedings of the conference on Advances in neural information processing systems (NIPS'99)*, pages 1052–1058, Cambridge, MA, USA. MIT Press.

Sato, M.-A. and Ishii, S. (2000). On-line em algorithm for the normalized Gaussian network. *Neural Computation*, 12(2):407–432.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.