Bayesian Nonparametric Multi-Optima Policy Search in Reinforcement Learning [*]

Danilo Bruno and **Sylvain Calinon** and **Darwin G. Caldwell**
Department of Advanced Robotics - Istituto Italiano di Tecnologia (IIT) - Via Morego, 30 - 16163 Genova

Abstract

Skills can often be performed in many different ways. In order to provide robots with human-like adaptation capabilities, it is of great interest to learn several ways of achieving the same skills in parallel, since eventual changes in the environment or in the robot can make some solutions unfeasible. In this case, the knowledge of multiple solutions can avoid relearning the task. This problem is addressed in this paper within the framework of Reinforcement Learning, as the automatic determination of multiple optimal parameterized policies. For this purpose, a model handling a variable number of policies is built using a Bayesian non-parametric approach. The algorithm is first compared to single policy algorithms on known benchmarks. It is then applied to a typical robotic problem presenting multiple solutions.

Introduction

Robots interact with the surrounding unpredictable environment when performing tasks and their behaviour should be able to rapidly adapt to external events: when acquiring skills, the robot should be able to change the way it is performing the task, whenever an environmental change happens. This should be achieved not only with fine refinement, but also with possibly very different policies producing the same end-result.

In general, a robot has several different ways of performing a given task. It is sufficient to think of the different ways in which a target can be reached or the different joint configurations that a kinematically redundant robot can use to achieve a desired aim. The concurrent learning of multiple ways of performing a task could help the robot adapt to new circumstances, when some solutions become unavailable, because of environmental changes (Neumann 2011; Ganesh and Burdet 2013).

We propose a model that can be thought as a way of transferring multiple policies to the robot related to a given task, with the advantage of being able to exploit the one that best suits the circumstances. This paper focusses on the learning problem, without addressing the task-specific problem of selecting one of the policy once a set of policies have been learned.

The problem is tackled within the framework of Reinforcement Learning: the aim is to determine by trial and error a set of policies describing the actions an agent has to take to perform a given task in an optimal way.

The existing algorithms use Gaussian Mixture Models (GMM) to encode a fixed number of multiple policies optimized using Expectation Maximization (Dempster, Laird, and Rubin 1977; Dayan and Hinton 1997) or Cross Entropy method (Kobilarov 2012). There exist also other approaches, using Bayesian nonparametric models to allow for a variable number of policies (Li, Liao, and Carin 2009; Doshi-Velez et al. 2010; Grollman and Jenkins 2010) on discrete state-action spaces. In this paper, the two aspects are merged to work on continuous state-action spaces. This is made possible by using parameterized policies, allowing us to treat episodic learning as immediate reward learning problems (Peters and Schaal 2007; Rueckstiess et al. 2010; Stulp and Sigaud 2012).

Another recent approach (Daniel, Neumann, and Peters 2012) uses hierarchical policy learning to treat the different policies as multiple possible options for the given task. In order to be effective, the method needs to start from a high number of initial options and discard the less promising ones.

Our previous work proposed the use of a GMM to encode multiple policies and allows a variable number of policies by splitting and merging the components if required (Calinon, Pervez, and Caldwell 2012; Calinon, Kormushev, and Caldwell 2013).

In this paper, multi-optima policy search is performed by building a mathematical model encoding a possibly variable number of policies, using Bayesian non-parametric techniques. In particular an Infinite Gaussian Mixture Model (Rasmussen 2000) will be used to describe the mapping between policy parameters and rewards. Policy search is taken as an incremental density estimation problem to fit a set of policy parameters augmented by their respective rewards.

An advantage of the proposed approach is that the formulation embeds the already existing policy search algorithms. The model is built so that any such algorithm can be bond to the multi-optima modelling tool and perform multi-optima search.

[*]This work was partially supported by the STIFF-FLOP European project under contract FP7-ICT-287728.

The algorithm is tested on the Cart-Pole and Mountain Car benchmarks. Then, an experiment consisting in passing through a set of viapoints for the Barrett WAM arm robot is provided, where the multi-optima policy search is performed by internal simulation and reproduced on the real robot after learning.

Multi-optima search with Dirichlet Processes

Reinforcement Learning (RL) is usually studied within the framework provided by Markov Decision Processes (MDP), i.e. a pair (X, A) representing respectively the state and action space, possessing a set of rules governing the transition between states, once an action has been taken. In the present paper, continuous state and action spaces will be considered.

In the learning phase, a reward is assigned to each pair $(x, a) \in (X, A)$ along the task: the aim is to assign to each state the best action to take in order to maximize the reward, represented as a mapping $\pi : X \to A$ (optimal policy).

Depending on the problem, this maximization process can be performed instantaneously, when the reward is evaluated at every time step, or episodically, if it is evaluated at the end of the task.

In this paper, the same approach will be used to tackle both situations, the unification being provided by the use of parameterized policies, described by

$$F_{\boldsymbol{\theta}} : X \to A \quad s.t. \quad a = F(x; \boldsymbol{\theta}) \quad . \tag{1}$$

The aim is to find the best policy parameters $\boldsymbol{\theta}$ maximizing the reward function. There exist many policy search algorithms. The ones that will be used in this paper work on the principle of stochastic exploration. The system starts from a given initial policy and uses it to obtain the actions corresponding to the given state. For instantaneous scenarios, a reward will be given after the action is performed; for the episodic case, a reward will be given at the end of the episode. Once the reward is obtained, it can be used to update the parameters of the policy.

The parameters of the policy are assumed being drawn from an assigned probability distribution $\boldsymbol{\theta} \sim f(\boldsymbol{\theta}|\alpha)$, where α are known as the policy hyper-parameters.

At each episode, new parameters are drawn from the probability distribution and the corresponding reward of the policy is evaluated. This corresponds to exploring a set of stochastically drawn parameters around the current ones. After the exploration is performed, RL algorithms update the hyper-parameters α.

The distribution of parameters is assumed to be Gaussian and the parameters α are the mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$; the results can be eventually extended to more general distribution with a little work.

Setting of the problem

Multi-optima policy search investigates the possibility that at each episode, there exists more than one locally optimal policy. This amounts to say that, if we think of the reward as a function of the policy parameters, the algorithm will track the promising local optima.

There are 3 possible scenarios. In the first, the maximum consists of a non-convex area in the parameters space and a GMM fits this shape in the policy parameters space. In the second, there exist multiple global optima, representing clearly distinct ways to perform the same task. Finally, the solution could consist of a global optimum and local optima with lower reward: in this case, the lower values are modelled and used only if needed.

A simple way to perform multi-policy search is to find a local optimum and then restart the search randomly (around the original position) to search for other possible policies. But in this case, the algorithm does not prevent the system to converge to the same original optimum and a lot of restarts will be needed.

This can be avoided by encoding the found policies into a GMM, increasing its size as the problem grows: in this setup, each component represents a different policy that is being explored. As soon as an optimum is found, other search processes will be taken into account by another component and the already explored area is avoided. Nevertheless, it easily happens that other trials converge to non-optimal areas, forcing a number of restarts that is higher than the number of local optima. Moreover, a check is needed to stop exploration when a maximum is reached.

Another option would be to run the explorations in parallel since the beginning, describing the multiple policies with the GMM, sampling a new point for exploration from each component and updating all of them in parallel. The drawback is that the number of policies has to be known in advance. One possible approach is to start with a high number of policies and discard the less promising ones (Daniel, Neumann, and Peters 2012), resulting into a high number of useless reward evaluations to be performed.

To allow for an unknown number of policies, while keeping the number of trials low, we propose to use the Dirichlet Process as a prior for the GMM encoding the policies.

Dirichlet Processes

The Dirichlet process (DP) is a stochastic process used in Bayesian nonparametric modelling. It can be thought as a distribution over probability distributions: its samples are probability distributions whose marginals are a Dirichlet distribution (Ferguson 1973; Antoniak 1974).

Formally, given a distribution H on a probability space Θ, we say that G is distributed according to a DP with parameter α and base distribution H (written $G \sim DP(\alpha, H)$) if, for any finite measurable partition $A_1, \ldots A_N$ of Θ, we have

$$(G(A_1), \ldots, G(A_N)) \sim Dir(\alpha H(A_1), \ldots, \alpha H(A_N)) \quad .$$

H plays the role of the average of the DP, since $E[G(A)] = H(A)$ for every measurable set $A \subset \Theta$.

The draws of a DP are discrete (Blackwell and MacQueen 1973) and exhibit a clustering behaviour: if $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ are sampled from a DP, the posterior probability for $\boldsymbol{\theta}_{n+1}$ is

$$P(\boldsymbol{\theta}_{n+1}|\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n) = \frac{\alpha H + \sum_{i=1}^{n} \delta_{\boldsymbol{\theta}_i}}{\alpha + n} \ , \tag{2}$$

where $\delta_{\boldsymbol{\theta}_i}$ is the unit mass measure over Θ. The probability of ending up with an already sampled value is proportional to the number of times it has already been sampled, while the probability of sampling a new value is ruled by the distribution H and the parameter α.

For modelling purposes, DP is used as a prior for a hierarchical process, known as Dirichlet Process Mixture Model (DPMM) (Antoniak 1974; Escobar and West 1995). It consists in adding a further probability distribution $F_{\boldsymbol{\theta}}$ to the model, whose parameters $\boldsymbol{\theta}$ are drawn from the DP.

From an algorithmic viewpoint, a DPMM is an infinite model and can be generated as the limit of a finite process, i.e. the limit for $K \to \infty$ of a Hierarchical Mixture Model with K-components. We choose to use the limit of the following one (Rasmussen 2000)

$$\begin{aligned} \pi|\alpha &\sim \mathrm{Dir}(\frac{\alpha}{K},\dots,\frac{\alpha}{K}), & \boldsymbol{\theta}_k|H &\sim H, \\ c_i|\pi &\sim \mathrm{Mult}(\pi), & x_i|c_i,\boldsymbol{\theta}_k &\sim F_{\boldsymbol{\theta}_{z_i}} . \end{aligned} \tag{3}$$

In the above process, a class is associated with each observation $\boldsymbol{y}_i$ and is assigned a label c_i (the number identifying the component); a parameter $\boldsymbol{\theta}_{c_i}$ is sampled from H and is associated to that class. This parameter determines the distribution $F_{\boldsymbol{\theta}_{c_i}}$ whence the observation is drawn. The class labels c_i are drawn from a multinomial distribution, whose priors are given according to a Dirichlet distribution with parameter $\frac{\alpha}{K}$. Once K approaches infinity, the process behaves like a DP with parameter α and prior H.

If a set of n points $\boldsymbol{y}_1,\dots,\boldsymbol{y}_n$ is distributed according to a DPMM, the probability that any point $\boldsymbol{y}_i$ belongs to the cluster labelled by c_i, given the current configuration, is described by:

$$\mathcal{P}(c_i = c|c_{-i},\boldsymbol{y}_i,\boldsymbol{\theta}) = b\frac{n_{-i,c}}{n-1+\alpha}F_{\boldsymbol{\theta}_c}(\boldsymbol{y}_i), \tag{4}$$
$$\mathcal{P}(c_i \neq c_j|c_{-i},\boldsymbol{y}_i,\boldsymbol{\theta}) = b\frac{\alpha}{n-1+\alpha}\int F_{\boldsymbol{\theta}}(\boldsymbol{y}_i)dH(\boldsymbol{\theta})\ (\forall j \neq i)$$

where c_{-i} are the cluster labels without the i^{th} element, $\boldsymbol{\theta}$ the parameters' vector of the DPMM and b a normalization coefficient. The first equation represents the probability of being in an already known cluster, while the second is the probability of generating a new cluster.

The estimate of the posterior probability modelling a given set of data can be obtained using Gibbs sampling. At each iteration, a label c_i is assigned to every point $\boldsymbol{y}_i$, sampling it from the distribution (4). After all the points have been reassigned, the new parameters for the surviving components are estimated from the posterior distribution of the parameters. The procedure is iterated until convergence is reached.

A review of several Gibbs sampling algorithms for DP mixtures can be found in Neal (2000). We choose to use Algorithm 8 of Neal's paper, as reported in Alg. 1. This choice is motivated by the fact that we will need to handle the case when F and H are not conjugate priors.

We also choose F to be a Gaussian distribution: in this case, the process is known as Infinite Gaussian Mixture Model (IGMM).

```
Data: $n$ points $\boldsymbol{y}_1,\dots,\boldsymbol{y}_n$
Result: GMM parameters and labels $c_1 \dots c_n$
Initialize: GMM with 1 component. Mean and
covariance from data
repeat
    Input: $n$ points $\boldsymbol{y}_1,\dots,\boldsymbol{y}_n$ and $n$ labels $c_1 \dots c_n$
    for $i = 1$ to $n$ do
        if $c_i$ contains only $\boldsymbol{y}_i$ then
            relabel it as $c_i = K$ and set $K_- = K - 1$
        else
            set $K_- = K$
        set $h = K_- + m$ where $m$ is arbitrary
        sample parameters for the $m$ additional clusters
        from the prior distribution $H$
        draw a new value $c_i$ from the probability
        distribution $\mathcal{P}(c_i = c|c_{-i},\boldsymbol{y}_i,\boldsymbol{\theta}) = p$, where
            $p = b\frac{n_{-i,c}}{n-1+\alpha}F_{\boldsymbol{\theta}_c}(\boldsymbol{y}_i)$ for $1 \le c \le k_-$
            $p = b\frac{\alpha/m}{n-1+\alpha}F_{\boldsymbol{\theta}_c}(\boldsymbol{y}_i)$ for $k_- < c \le h$        (5)
    Calculate the new values for the clusters
    parameters from posterior distribution of the
    parameters, given the points belonging to it
    Update priors for the DP using posterior
    information on the components parameters
until convergence
```

Algorithm 1: DP Clustering (Neal Algorithm 8)

The DIPOLE algorithm

Dirichlet Processes are used to encode the policies into an IGMM. This both allows us to encode a variable number of components and to use the better modelling capabilities of DP for describing the posterior distribution more precisely.

In order to create a model that is sensible to the values taken by the reward function, the IGMM will not be encoded in the policy parameter space, but in the augmented space built with the reward as an additional variable in the form $[\boldsymbol{\alpha}, r]^\top$.

Multiple parallel exploration is made possible by the policy search approach. For each episode, new parameters are sampled from each component of the IGMM, and their rewards are evaluated. Since each evaluation of the policy is independent of the others, very different values of the parameters can be tested sequentially.

Finally, the parameters of the IGMM are updated, using an adapted version of any RL algorithm performing policy optimization. In this way, the model of the IGMM is built using the data sampled during the search. Since the RL algorithm moves the points towards the optimal value, the IGMM will end up modelling the local maxima of the reward function in parameters space.

The proposed DIrichlet process POlicy LEarning (DIPOLE) algorithm alternatively combines the actions of RL and of DP clustering. At each step, K samples are taken from the IGMM encoding the policies, one for each component. The policy is evaluated and updated using an RL algorithm performing policy optimization. After a given number of RL steps, a DP clustering is run on the data. The interaction between the two algorithms is sequential: the RL algorithm improves the policy and DP reorganizes the updated policies.

```
Input: Initial parameter $\boldsymbol{\theta}$ and exploration noise $\boldsymbol{\Sigma}_{start}$
Result: $K$ optimal policies (IGMM)
begin
    Sample $n_0$ points from $\mathcal{N}(\boldsymbol{\theta}, \boldsymbol{\Sigma}_{Start})$
    Evaluate rewards for the sampled parameters
    Run DP-clustering on the augmented dataset
    repeat
        Sample a new value for each parameter from
        each component of the IGMM
        Evaluate the rewards for each value
        Update the parameters for each policy (RL
        algorithm, PoWER for experiments)
        Run (every $M$ steps) a weighted DP
        clustering on all data
    until convergence
```

Algorithm 2: DIPOLE

The IGMM is evaluated using as priors for the DP a Multivariate Normal-Inverse Wishart distribution, without linking the parameters as required by the conjugate prior of the Normal distribution. This is because exploration noise and the spread across the parameters space of component means are highly unrelated. This is the reason why we chose to use Algorithm 8 of Neal (2000). The starting value is chosen as the mean and covariance of the starting exploration noise.

DIPOLE is able to provide an automatic tuning of the exploration covariance, since the exploration noise of each component is fit to the sampled points at every DP run. This results into a better modelling capability and a finer exploration strategy.

Finally, in order to reject components with low reward, an additional weighting is added to the posterior probabilities of the components. So, probabilities (5) are replaced by

$$\mathcal{P}(c_i = c | c_{-i}, \boldsymbol{y}_i, \boldsymbol{\theta}) = b \frac{r_c \, n_{-i,c}}{n-1+\alpha} F_{\boldsymbol{\theta}_c}(\boldsymbol{y}_i) \; [1 \le c \le k_-],$$
$$\mathcal{P}(c_i = c | c_{-i}, \boldsymbol{y}_i, \boldsymbol{\theta}) = b \frac{r_c \, \alpha/m}{n-1+\alpha} F_{\boldsymbol{\theta}_c}(\boldsymbol{y}_i) \; [k_- < c \le h]. \tag{6}$$

where r_c is the reward corresponding to the mean of the c-th component (evaluated by the IGMM). To avoid degeneracies of the GMM, a regularization term for the covariance is added, calculated on the basis of the initial exploration noise.

In this paper we will use the PoWER (Policy learning by Weighting Exploration with the Returns) algorithm to update the policy (Kober and Peters 2011).

It uses importance sampling on the policy parameters to perform the update, so that only the sampled parameters leading to a higher reward are used to perform the update. This is done based on the idea that the searched policy lies in the convex combination of the best policies obtained so far, each weighted by its reward. Then, the update of the policy is performed by moving the mean in the direction evaluated, making the reward weighted combination of the vectors joining the current mean with the best points.

This is summarized in Algorithm 3.

```
Input: A policy $\boldsymbol{\theta}^{(0)}$ with exploration covariance $\boldsymbol{\Sigma}$
Result: A new policy $\boldsymbol{\theta}^{(n)}$ after $n$ steps
begin
    Sample $M$ points from the policy and reward
    repeat
        Sample 1 new point from the policy and
        evaluate reward
        Select the best $M$ points
        Update the policy with the following rules
            $$\boldsymbol{\theta}^{(n)} = \boldsymbol{\theta}^{(n-1)} + \frac{\sum_{m=1}^{M} r(\boldsymbol{\theta}_m)\left[\boldsymbol{\theta}_m - \boldsymbol{\theta}^{(n-1)}\right]}{\sum_{m=1}^{M} r(\boldsymbol{\theta}_m)}. \tag{7}$$
    until convergence
```

Algorithm 3: PoWER

At each iteration, new samples are added to the previous ones and importance sampling is performed by keeping a fixed number of elements $K \cdot M$ at every iteration, where K is the number of policies and M a fixed number for importance sampling.

Since only the best points are kept during this process, the IGMM will make a statistical model of the distribution of the best policies.

Experiments

We privileged standard benchmarks to introduce the approach and a final experiment showing the multi-policy capabilities on a real robotic platform. The comparison with PoWER on the benchmarks allows to single out the contribution of the DP approach, since PoWER is used by DIPOLE in the optimization step.

In all the experiments the following choices for the parameters were made: $\alpha = 1$ and $m = 10$. The number of points for importance sampling was heuristically set to 5.

Cart-pole and Mountain car benchmarks

In these experiments, we chose problems where a single optimum exists. There would be no advantage of using a multi-policy model in this case, but it is rare to know in advance that a single optimum policy exists. This experiment is thus not aimed at showing that the model performs better, but that the performance does not degrade even if a multi-policy prior assumption is considered.

The aim is to balance the pole for 1000 time steps, giving a reward of is 1 when the pole is vertical (within an error of $\frac{\pi}{60}$), -1 when it is falling down or the cart moves far from the origin, 0 otherwise. The cumulated reward is given by the sum of intermediate rewards collected at each time step.

The starting position is with the pole staying at random small angle from the vertical.

The state of the system is defined by a 4-dimensional vector $\boldsymbol{x} = [x, \dot{x}, \varphi, \dot{\varphi}]$, respectively representing the position and the velocity of the cart and the angle and angular velocity of the pole. The control u is scalar and represents the horizontal force applied to the cart. Both states and controls are assumed continuous. The policy is assumed linear

$$u = \boldsymbol{\theta}^{\top} \boldsymbol{x} \; . \qquad (8)$$

A comparison between PoWER and DIPOLE is provided in Fig.1. The DIPOLE algorithm was optimized using the same parameters as PoWER, in order to provide a fair comparison. Each trial represents a reward evaluations: if K policies have been evaluated at a given iteration, K trials are considered.

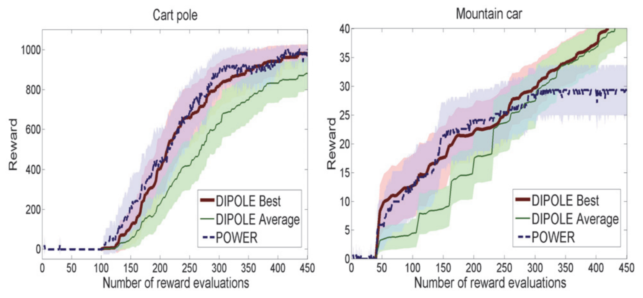

Figure 1: Cart Pole and Mountain car problems: comparison between DIPOLE and PoWER algorithms. All plots present results averaged over 50 learning processes. The shaded areas represent the standard deviation.

The behaviour of the best policy of DIPOLE algorithm is identical to PoWER alone. But to obtain this, the exploration covariance of PoWER was manually reduced when the optimum was reached, while DIPOLE did this automatically. The average of the policies in DIPOLE is slightly lower than PoWER alone, because at each run, more than one policy is evaluated.

As a conclusion, DIPOLE is comparable with the PoWER algorithm stopped after convergence, even though it models a wider number of policies in the starting phase. It has the advantage that is able to automatically learn the correct covariance behaviour when it converges towards the optimum.

We performed a similar experiment with the mountain car problem (Sutton and Barto 1998). The variables are the horizontal position x of the car and the horizontal velocity $\dot{x}$. The control is a horizontal force u on the car, that can be full forward or full backward. The possible limits are $x \in (-1.5, 1.5)$, $\dot{x} \in (-0.07, 0.07)$, $u \in \{-4, 4\}$. The dynamics of the car is $\ddot{x} = \alpha u - \beta \cos(3x)$. In our experiment, we used $\alpha = 0.001$ and $\beta = 0.0055$. Moreover, the variables are reset when they exit their range.

The cumulated reward is set an initial value of 101 at the start of the episode. At every time step, the car is given a reward of -1, unless it reaches the goal, where a reward of 100 is given. The maximum time provided for the experiment is 100 time steps. At the end of the episode, the cumulated reward is evaluated for the update of the policy: it ranges from 0 to 100.

The policy function is chosen as a linear combination of functions of the input variables, as in Strens and Moore (2002). Let $\boldsymbol{\theta}$ be a 4-dimensional vector of parameters and $\boldsymbol{x} = [x, x^2, x\,\dot{x}, x\,\dot{x}^2]^{\top}$ the vector of function primitives. The policy is represented by eq. (8).

We again compared the results with PoWER, but, this time, it is not possible to manually tune it, since no maximum value for the reward is known in advance. In this case, DIPOLE performs better than PoWER both in the best and in the average policies. This is due to the better covariance adaptation that DIPOLE provides.

At every run, it starts exploring many possible optimal policies: when one of them starts converging, the others are automatically discarded and the covariance tuning allows for a more efficient convergence strategy. As a result, DIPOLE only has a lower average reward in the starting phase, where multiple policies are searched in parallel.

Simulation of a viapoint task

The task consists in moving between two points in the workspace of the Barrett WAM 7 DOFs arm robot. Several viapoints are defined between them in the 3D space and the robot is requested to pass to viapoints at given time steps from the start of the motion. For each time step, multiple viapoints are defined, so that the robot has different alternative paths to perform the task. The search process is performed in simulation using the MatLab Robotics Toolbox (Corke 2011).

The trajectories are encoded in the task space using the statistical DS-GMR dynamical system approach, see Calinon et al. (2012) for details. In short, a virtual spring-damper system is pulling the tip of the robot, where the path of the spring's equilibrium point is learned in the form of a Gaussian mixture model (GMM). So, instead of defining the trajectory of the end-effector, the position and the activation weights of the attractors are learnt. The paths of the attractors are encoded within the GMM, whose centers represents the most likely position and the covariance the possible variability. The trajectory generated by the learnt dynamical system is retrieved by using Gaussian mixture regression (GMR) (Ghahramani and Jordan 1994).

DIPOLE is used to move the position of the attractors in the workspace of the robot to generate a trajectory maximizing the reward. Each policy is parametrized by the position of the attractors, collected into a $D \times N$ dimensional vector, where D is the dimension of the task space and N is the number of attractors. The system starts from the policy encoding a straight line from the start to the target point, having zero velocity in both of them. After sampling 30 initial points, the IGMM in the policy-reward space is learnt from the samples. The initial covariance is set so that the trajectory stays within the workspace of the robot. Before evaluating the reward, the desired trajectory is run on the robot simulator and the reward of the resulting trajectory in task space is calculated.

Then, at each iteration, a new position for the attractor is sampled from each component of the IGMM, amounting to sampling new policies in parallel. The policies are run on the robot simulator one after the other.

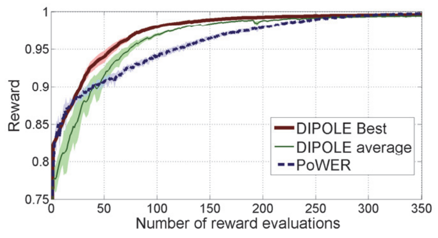

Figure 2: Viapoint task. Reward averaged over 10 runs for the best and average policy of DIPOLE and comparison with PoWER algorithm

The reward depends on the distance of the trajectory from the nearest viapoint at the given pass times. The reward is defined as

$$r = \frac{1}{N_v} \sum_{i \in V} \exp\left[-\alpha\left(\boldsymbol{x}(t(i)) - \boldsymbol{x}_i^{(v)}\right)\right], \tag{9}$$

where N_v is the number of distinct viapoints, V is the set of the nearest viapoints to the trajectory at the given pass times and $\boldsymbol{x}^{(v)}$ are the task coordinates of viapoints. The maximum value of the reward is 1 and is achieved when all the viapoints are reached.

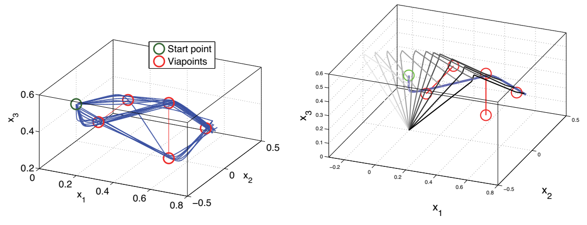

Figure 3: Viapoint task. Left: discovered options for trajectories discovered in 10 runs. Right: one of the final optimal trajectories run on the Barrett WAM robot.

A number of components equal to the number of (time-distinct) viapoints is used. The resulting task encodes the centers of 3 attractors in 3D, corresponding to a 9-dimensional problem. The results of the experiment over 10 runs are shown in Fig. 2. The algorithm is able to determine different optimal trajectories with a very low number of reward evaluations. The algorithm learning curve has a very low covariance, and converges in 300 trials. A comparison with PoWER shows that DIPOLE is slower at the beginning (due to the higher number of policies searched) but converges faster to an optimal policy.

In every single run, an average of 2.5 different policies are found, encoding the different trajectories passing through the viapoints. By executing multiple runs, all the possible trajectories are found (Fig. 3).

Finally, one of the resulting trajectory was run on the real robot as shown in Fig. 3.

Discussion and future work

Being a multi-optima policy search algorithm, the performance of DIPOLE in the case of single peaked solution space is not expected to compete with methods making the inherent assumption of a single optimum. Nevertheless, the performance on the best policy proved to be comparable and in some cases even better than single policy methods. The only performance downgrade is placed in the presence of multiple bad policies at the beginning of the exploration, affecting the average reward among the detected policies (Fig.1). Moreover, the addition of DP-clustering makes the update of the policies slightly slower. In robotic tasks, this is negligible with respect to the time needed to perform the task and is comparable with the computational time required by the RL algorithm. The computational effort is expected to grow with the dimension of the problems and the number of policies and will be further investigated in future papers.

DIPOLE seems promising in performing policy search in continuous state-action spaces. Though a proper comparison was not made, it is competitive with other concurring methods (Daniel, Neumann, and Peters 2012), by using a fewer number of exploration trials on a similar viapoint task.

Further work will explore how the additional information provided by multiple policies can be used to rapidly adapt to the environment, automatizing the choice of the policy, depending on the present situation. Moreover, a possible exploitation of the covariance information will be investigated, to identify more stable policies actively rejecting noise disturbance (Müller and Sternad 2004). In fact, it is reasonable to assume that skilled actions can be aligned with the directions where noise and variability have lower effect on the end result (Scholz and Schoener 1999; Latash, Scholz, and Schoener 2002; Sternad et al. 2010).

Conclusions

A new algorithm combining Dirichlet Process clustering and Reinforcement Learning policy search algorithms was introduced to perform multi-optima policy learning. The non-parametric Bayesian approach allows the system to cope with a variable number of policies at the same time. In this way, different solutions of the RL problem can be encoded into a GMM. The algorithm was implemented with PoWER, but it can be interfaced with any parametrized policy search algorithm. The results of the experiments show that the algorithm finds solutions with similar accuracy as the single policy search version. The slight loss in performance is largely compensated by the benefit of the approach in the form of the information it provides about the learned skill, represented by a higher number of available policies and a more accurate covariance adaptation.

References

Antoniak, C. E. 1974. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *Ann. Statist.* 2(6):1152–1174.

Blackwell, D., and MacQueen, J. B. 1973. Ferguson distributions via Pólya urn schemes. *Ann. Statist.* 1(2):353–355.

Calinon, S.; Li, Z.; Alizadeh, T.; Tsagarakis, N. G.; and Caldwell, D. G. 2012. Statistical dynamical systems for skills acquisition in humanoids. In *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, 323–329.

Calinon, S.; Kormushev, P.; and Caldwell, D. G. 2013. Compliant skills acquisition and multi-optima policy search with em-based reinforcement learning. *Robotics and Autonomous Systems* 61(4):369–379.

Calinon, S.; Pervez, A.; and Caldwell, D. G. 2012. Multi-optima exploration with adaptive Gaussian mixture model. In *Proc. Intl Conf. on Development and Learning (ICDL-EpiRob)*.

Corke, P. I. 2011. *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer.

Daniel, C.; Neumann, G.; and Peters, J. 2012. Learning concurrent motor skills in versatile solution spaces. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, 3591–3597.

Dayan, P., and Hinton, G. E. 1997. Using expectation-maximization for reinforcement learning. *Neural Comput.* 9(2):271–278.

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

Doshi-Velez, F.; Wingate, D.; Roy, N.; and Tenenbaum, J. 2010. Nonparametric bayesian policy priors for reinforcement learning. In *Neural Information Processing Systems (NIPS)*.

Escobar, M. D., and West, M. 1995. Bayesian density estimation and inference using mixtures. *J. Amer. Statist. Assoc.* 90(430):577–588.

Ferguson, T. S. 1973. A Bayesian analysis of some nonparametric problems. *Ann. Statist.* 1:209–230.

Ganesh, G., and Burdet, E. 2013. Motor planning explains human behaviour in tasks with multiple solutions. *Robotics and Autonomous Systems* 61(4):362–368.

Ghahramani, Z., and Jordan, M. I. 1994. Supervised learning from incomplete data via an EM approach. In Cowan, J. D.; Tesauro, G.; and Alspector, J., eds., *Advances in Neural Information Processing Systems*, volume 6, 120–127. Morgan Kaufmann Publishers, Inc.

Grollman, D. H., and Jenkins, O. C. 2010. Incremental learning of subtasks from unsegmented demonstration. In *International Conference on Intelligent Robots and Systems (IROS 2010)*, 261–266.

Kober, J., and Peters, J. 2011. Policy search for motor primitives in robotics. *Machine Learning* 84(1):171–203.

Kobilarov, M. 2012. Cross-entropy motion planning. *The International Journal of Robotics Research* 31(7):855–871.

Latash, M. L.; Scholz, J. P.; and Schoener, G. 2002. Motor control strategies revealed in the structure of motor variability. *Exerc. Sport Sci. Rev.* 30(1):26–31.

Li, H.; Liao, X.; and Carin, L. 2009. Multi-task reinforcement learning in partially observable stochastic environments. *The Journal of Machine Learning Research* 10(May):1131–1186.

Müller, H., and Sternad, D. 2004. Decomposition of variability in the execution of goal-oriented tasks: three components of skill improvement. *Journal of Experimental Psychology: Human Perception and Performance* 30(1):212.

Neal, R. M. 2000. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics* 9(2):249–265.

Neumann, G. 2011. Variational inference for policy search in changing situations. In *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, 817–824.

Peters, J., and Schaal, S. 2007. Using reward-weighted regression for reinforcement learning of task space control. In *Proc. IEEE Intl Symp. on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 262–267.

Rasmussen, C. E. 2000. The infinite Gaussian mixture model. In *Advances in Neural Information Processing Systems (NIPS) 12*, 554–560. MIT Press.

Rueckstiess, T.; Sehnke, F.; Schaul, T.; Wierstra, D.; Sun, Y.; and Schmidhuber, J. 2010. Exploring parameter space in reinforcement learning. *Paladyn. Journal of Behavioral Robotics* 1(1):14–24.

Scholz, J. P., and Schoener, G. 1999. The uncontrolled manifold concept: identifying control variables for a functional task. *Experimental Brain Research* 126(3):289–306.

Sternad, D.; Park, S.-W.; Mueller, H.; and Hogan, N. 2010. Coordinate dependence of variability analysis. *PLoS Computational Biology* 6(4):1–16.

Strens, M., and Moore, A. 2002. Policy search using paired comparisons. *Journal of Machine Learning Research* 3(1):921–950.

Stulp, F., and Sigaud, O. 2012. Path integral policy improvement with covariance matrix adaptation. In *Proc. Intl Conf. on Machine Learning (ICML)*.

Sutton, R., and Barto, A. 1998. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press.