

An extended policy gradient algorithm for robot task learning

A. Cherubini F. Giannone L. Iocchi P. F. Palamara

Dipartimento di Informatica e Sistemistica

Sapienza University of Roma

Via Ariosto 25, 00185 Roma, Italy

{cherubini, iocchi}@dis.uniroma1.it

Abstract—In real-world robotic applications, many factors, both at low-level (e.g., vision and motion control parameters) and at high-level (e.g., the behaviors) determine the quality of the robot performance. Thus, for many tasks, robots require fine tuning of the parameters, in the implementation of behaviors and basic control actions, as well as in strategic decisional processes. In recent years, machine learning techniques have been used to find optimal parameter sets for different behaviors. However, a drawback of learning techniques is time consumption: in practical applications, methods designed for physical robots must be effective with small amounts of data. In this paper, we present a method for concurrent learning of best strategy and optimal parameters, by extending the policy gradient reinforcement learning algorithm. The results of our experimental work in a simulated environment and on a real robot show a very high convergence rate.

I. INTRODUCTION

In order for robots to be useful for real-world applications, they must adapt to novel and changing environments. In most domains, the robot should be able to respond to changes in its surroundings by adapting both its low-level skills (e.g., vision or motion control parameters) and the higher-level skills (e.g., the behaviors) which depend on them.

Recently, machine learning techniques have been used in many robotic applications, both for finding optimal parameter sets of specific behaviors and for determining the best choice of behaviors required to accomplish a task in a given situation. In fact, machine learning approaches generate solutions with little human interaction, and explore the search space of possible solutions in a systematic way, whereas humans are often biased towards exploring a small part of the space.

Despite this growing interest, considerable work remains, due to the difficulties associated with machine learning in the real world. Compared to other machine learning scenarios such as classification or action learning in simulation, learning on physical robots presents several formidable challenges. In particular, learning methods designed for physical robots must be effective with small amounts of data, and should converge in short time. Indeed, it is often prohibitively difficult to generate large amounts of data due to the maintenance required on robots, such as battery changes, hardware repairs, and, usually, constant human supervision. Thus, learning methods designed for physical robots must be effective with small amounts of data. Other practical issues that make robot learning extremely challenging are the time and material constraints: to be of practical use for robotics, the learning algorithms must converge in a relatively small number of training epochs [1].

Following up on all these considerations, and on our previous work [2], we have developed an extended version of the policy gradient learning technique introduced in [3]. In [2], we have shown the effectiveness of the incremental learning approach in complex robot learning scenarios. We have also studied how to use a 3D simulator for speeding up

robot learning. Finally, we have shown that the learned low-level parameters are strongly related to the desired behavior. Consequently, the extended policy gradient algorithm that we propose in this work must learn the best strategy (i.e., composition of simple behaviors), and correspondingly the best parameters for accomplishing a given task. Our approach guarantees faster training than the classic policy gradient [3], by exploiting additional information on the system properties, such as the *contiguities* between strategies, and the *relevance* of the behavior parameters.

The method has been tested in the application example of an attacker robot in the RoboCup Four-Legged League. However, in our opinion, the algorithm is flexible, and can be extended in order to optimize tasks different from soccer attack on robot platforms different from Sony AIBO.

II. RELATED WORK

Robot learning is a growing area of research at the intersection of robotics and machine learning. It is used both at low level, for sensing and control issues, and at high level, for cognitive and behavior issues. We include in the first class – *parameter learning* – all problems where learning is aimed at fine-tuning of the parameters used by the low level algorithms, and in the second class – *behavior learning* – problems where learning aims at finding the optimal composition of behaviors for accomplishing a task.

In the first class of problems, one of the primary application areas is robot motion control, in all cases where the complete mathematical model is not known, and traditional optimization methods cannot be used. Gait optimization for legged robots is one of such fields [1]. Parameter learning has proved very effective for improving other motion control tasks, such as robot grasping. In [4], this is achieved by applying the *layered learning* paradigm [5]: grasping parameters rely on previously learned walk parameters. Another interesting application area for parameter learning is robot vision. In [6], for instance, biologically inspired perceptual learning mechanisms are used to improve object identification in real-world environments.

On the other hand, researchers have proved the utility of behavior learning for improving high level robot tasks, e.g. navigation, exploration, path-planning. In the *behavioral decomposition* approach, the desired behavior is broken down into a set of simpler behaviors, which are activated or suppressed through a coordination mechanism. The structure of the simpler behavioral modules can emerge from a learning process, instead of being pre-designed. Nolfi [7] showed how a good evolutionary approach can produce simpler and more robust solutions, than those designed by hand. The robot can evolve by adapting its behaviors to various constraints on the environment, on the desired task, and on the user satisfaction. Genetic Programming (GP, [8]) and Reinforcement Learning (RL, [9]) are useful tools for learning behaviors at the cognitive layer. The adaptation mechanism of GP is similar

to that of Genetic Algorithms, except that it is based on symbolic, rather than bit string, representation.

To our knowledge, the aforementioned classes of learning (parameter and behavior learning) have rarely been joined in a single framework. One reason is the long time required to solve the optimization problem in the large search space including both parameters and behaviors. Here, we present a fast method based on the policy gradient reinforcement learning technique [10] useful in all cases where behavior and parameter learning should be integrated for optimal task execution. Our method can be considered an extended version of the policy gradient technique introduced in [3], which takes into account behavior similarities and parameter relevance during the learning process, to speed up convergence. Experimental comparison between the policy gradient from [3] and the extended version that we propose is carried out in this work, based on an incremental learning approach.

III. PROBLEM DEFINITION

In this paper we consider learning a complex task as a composition of different behaviors. More specifically, we consider situations in which a task \mathcal{T} can be accomplished by applying different strategies, where each strategy is a composition of different behaviors. Each behavior B is characterized by a set of parameters $\Theta_B = \{\theta_1, \dots, \theta_{k_B}\}$. Notice that a behavior can be present in different strategies and possibly requires the definition of different parameters depending on the situation in which it is used.

The learning problem is thus twofold: behavior learning is needed to select the best strategy (i.e., the best composition of basic behaviors) and meanwhile each behavior requires fine tuning of each parameter. As in any robot learning problem, the proposed algorithm should minimize time consumption, which is often related to 'hardware consumption' in real robot applications. Hence, the algorithm must converge to the best solution after few iterations.

The learning problem we are interested in is the following. Given a set \mathcal{S} of different *strategies* for accomplishing a certain task \mathcal{T} , we want to learn the best choice of the strategy, as well as the parameters Θ_B that ensure the best performance on the execution of the best strategy. Two main difficulties arise when dealing with this problem: 1) the choice of an optimal strategy, 2) an optimization problem in the behavior parameters space.

In general, the choice about which strategy must be selected to perform a task is not easy. This is firstly due to the fact that in complex scenarios, the winning strategy depends on the context (i.e., the current situation of the environment around the robot). Secondly, some strategies can perform similarly in a given context: this similarity (*contiguity*) can be used to speed up the learning process. Thirdly, a coarse initial design of the learning algorithm can lead to overestimation of the search space dimensions. In particular, this occurs when parameters which have little *relevance* on the performance are learned. Based on these considerations, we will show that the speed of a robot task learning algorithm can be improved by: 1) detecting and exploiting similarities between learned strategies, and 2) updating the search space dimensions, by removing irrelevant parameters during learning.

Furthermore, in the optimization of behavior parameters, the optimization function is not known ('black box') and analytical computation of its derivatives is impossible. Besides, the parameters are box-constrained due to the physical characteristics of the system. Hence, conventional derivative-based optimization methods (e.g., Gradient or Newton methods) cannot be utilized. The selected approach must handle

CLASSIC POLICY GRADIENT

```

INPUT:  $\mathcal{S}, {}^v \underline{X}^0, N_{iter}$ 
OUTPUT:  $S_{v^*}, {}^{v^*} \underline{X}^*$ 
1 initialize parameter set for all strategies:  ${}^v \underline{X} \leftarrow {}^v \underline{X}^0$ 
2 for each strategy  $v = 1$  to  $N_{str}$ 
3    ${}^v \underline{X}^* = \text{classic policy gradient}({}^v \underline{X}^0, N_{iter})$ 
4 endfor
5  $v^* \leftarrow \arg \max F({}^v \underline{X}^*)$ 
6 return  $(S_{v^*}, {}^{v^*} \underline{X}^*)$ 
7 END

```

Fig. 1. Pseudocode for classic policy gradient task learning.

non differentiable search space, have high convergence rate and be resistant to noise.

Many algorithms possess these characteristics. In particular, we will utilize the **Policy Gradient learning algorithm**, which has been successfully used for robot learning [3], [2].

In the next section we present a general learning methodology that integrates behavior and parameter learning for executing a complex robot task in a small amount of experiments. The approach is based on an extension of the policy gradient algorithm that allows for significantly increased convergence rate for complex robot tasks.

IV. COMPLEX TASK LEARNING

The method for task learning proposed in this paper aims at determining at the same time the best strategy and the optimal parameters to be used in the behaviors used in such a strategy. We will refer in the following to a set of N_{str} different strategies for accomplishing the task \mathcal{T} : $\mathcal{S} = \{S_1, \dots, S_{N_{str}}\}$. Each strategy is characterized by a set of behaviors, for which we want to optimize the characteristic parameters. Let ${}^v \underline{X} \in \mathbf{R}^n$ be a representation of the parameters used by behaviors in the strategy S_v . Although, in general, different strategies have different parameter spaces, we will assume that the parameters of the behaviors for all the strategies can be represented as a vector in \mathbf{R}^n . Furthermore, we will denote with F the fitness function used to evaluate strategies and parameters and thus with $F({}^v \underline{X})$ the evaluation of the parameters \underline{X} within the strategy S_v .

The objective of the leaning process is to learn the best choice of the strategy S_{v^*} , along with the optimal parameters ${}^{v^*} \underline{X}^*$ that ensure the best performance on the execution of that strategy, i.e., that maximize $F({}^v \underline{X})$. To achieve this goal, we first present a straightforward application of policy gradient approach, and then describe a novel extended method that substantially increases convergence rate. The first method simply consists of solving N_{str} independent optimization problems (one for each strategy) in \mathbf{R}^n , and choose the pair $S_{v^*}, {}^{v^*} \underline{X}^*$ that returns the highest fitness value for task \mathcal{T} .

A. Classic policy gradient method

The policy gradient algorithm [3] estimates the gradient of the objective function in the parameter space and follows it towards a local optimum. The parameter optimization approach starts from an initial parameter set \underline{X}^0 and proceeds to estimate the partial derivative of $F(\underline{X}^0)$ with respect to each parameter. From the initial set \underline{X}^0 , p randomly generated *policies* ${}_m \underline{X}$ ($m = 1, \dots, p$), near \underline{X}^0 , are evaluated, such that: ${}_m \underline{X} = \underline{X}^0 + [\rho_1, \dots, \rho_k]^T$, and each ρ_j is chosen randomly in the set $\{-\epsilon_j, 0, +\epsilon_j\}$. Each ${}_m \underline{X}$ is grouped into one of three sets for each j : $G_{-\epsilon_j}, G_{0,j}$ or $G_{+\epsilon_j}$ depending whether its j^{th} parameter was obtained

by adding $-\epsilon_j$, 0 or $+\epsilon_j$. After evaluating the objective function at each $m\bar{X}$, average scores $\bar{F}_{-\epsilon,j}$, $\bar{F}_{0,j}$, and $\bar{F}_{+\epsilon,j}$ are computed for $G_{+\epsilon,j}$, $G_{0,j}$ and $G_{-\epsilon,j}$, respectively. These scores are used to construct an estimate of the *gradient*, which is then normalized, multiplied by a scalar *step-size* η and added to \bar{X}^0 , to begin the next iteration i ($i = 1, \dots, N_{iter}$). Pseudocode for learning the best strategy S_{v^*} and the corresponding best parameters $v^* \bar{X}^*$ for a given task \mathcal{T} , using the policy gradient method, is shown in Fig. 1. The function *classic policy gradient* ($v \bar{X}^0, N_{iter}$), summarizes the algorithm described in [3].

B. Extended policy gradient method

Although effective, the method described above does not exploit two important factors of this learning process: similarities between strategies and relevance of parameters.

In this section, we present an extension of the method, which additionally takes into account: *contiguities* between strategies, and *relevance* of parameters within each strategy, to fasten the learning process. Let us define, at every iteration step $\bar{i} \neq 0$, the following metrics:

- *Contiguity* between strategies S_v and S_w :

$$C^{\bar{i}}(v, w) = \frac{1}{d(v \bar{X}^{\bar{i}}, w \bar{X}^{\bar{i}}) + 1}$$

where:

$$d(v \bar{X}^{\bar{i}}, w \bar{X}^{\bar{i}}) = \sqrt{\sum_{j=1}^k (v X_j^{\bar{i}} - w X_j^{\bar{i}})^2}$$

is the euclidean distance between $v \bar{X}^{\bar{i}}$ and $w \bar{X}^{\bar{i}}$. We also note $v P^i$ the set ('cluster') containing all points $w \bar{X}^i$ which are 'near' $v \bar{X}^i$.

- *Relevance* of parameter j for a given strategy v : it is the norm of the weighted average of the j^{th} gradient component:

$$R^{\bar{i}}(v, j) = \frac{\sum_{i=1}^{\bar{i}} \lambda^{\bar{i}-i} \nabla^v X_j^i}{\sum_{i=1}^{\bar{i}} \lambda^{\bar{i}-i}}$$

where $0 < \lambda < 1$ is a *forgetting factor* that operates as a weight diminishing for the more remote data. Small values of the *relevance* imply that the estimated gradient varies 'slightly' along the j^{th} component during the learning process: hence, we assume that the corresponding parameter has little relevance on the system performance. We denote with $v J^i \subseteq \{1, \dots, v k^i\}$ the subset of *relevant* parameters (i.e., parameters j with 'high' $R^i(v, j)$) for learning strategy v at iteration i . $v J^i$ is updated at each iteration step i during the learning process, depending on the values of $R^i(v, j)$.

The extended algorithm is shown in Fig. 2. This algorithm has a similar structure than the one described in the previous section, but it contains two major improvements. Firstly, for each strategy, the subset $v J^i$ is updated, and at each iteration, policies are generated by perturbing only *relevant* parameters; hence, the size of the search space is reduced, and the algorithm convergence is faster. Secondly, for *contiguous* strategies, points reached at each learning iteration and belonging to $v P^i$ are evaluated, and the best point is chosen as the initial point for the next iteration for each of the contiguous strategies.

EXTENDED POLICY GRADIENT

```

INPUT:  $\mathcal{S}, v \bar{X}^0, N_{iter}$ 
OUTPUT:  $S_{v^*}, v^* \bar{X}^*$ 
1 initialize for all  $v$ :  $v \bar{X} \leftarrow v \bar{X}^0, v J^0 \equiv \{1, \dots, v k^0\}$ 
2 for each iteration  $i = 1$  to  $N_{iter}$ 
3   for each strategy  $v = 1$  to  $N_{str}$ 
4     generate  $v_m \bar{X}^i$  perturbations of  $v \bar{X}^{i-1}$  on  $v J^{i-1}$ 
5     evaluate  $F(v \bar{X}^i)$  at all  $p$  policies  $v_m \bar{X}^i$  for task  $\mathcal{T}$ 
6      $v J^i \leftarrow \emptyset$ 
7     for each parameter  $j \in v J^{i-1}$ 
8       evaluate  $\bar{F}_{-\epsilon,j}, \bar{F}_{0,j}$ , and  $\bar{F}_{+\epsilon,j}$ 
9       if  $\bar{F}_{0,j} > \bar{F}_{-\epsilon,j}$  and  $\bar{F}_{0,j} > \bar{F}_{+\epsilon,j}$ 
10         $\nabla^v X_j^i \leftarrow 0$ 
11      else
12         $\nabla^v X_j^i \leftarrow \bar{F}_{+\epsilon,j} - \bar{F}_{-\epsilon,j}$ 
13      endif
14      evaluate  $R^i(v, j)$ 
15      if  $R^i(v, j) > T_r$  ( $T_r$  a given threshold)
16         $v J^i \leftarrow v J^i \cup j$ 
17      endif
18    endfor %parameters
19     $\nabla^v \bar{X}^i \leftarrow \eta \times \frac{\nabla^v \bar{X}^i}{|\nabla^v \bar{X}^i|}$ 
20     $v \bar{X}^i \leftarrow v \bar{X}^{i-1} + \nabla^v \bar{X}^i$ 
21  endfor %strategies
22  for each strategy  $v = 1$  to  $N_{str}$ 
23    for each strategy  $w = 1$  to  $N_{str}$ 
24      evaluate  $C^i(v, w)$ 
25      if  $C^i(v, w) > T_c$  ( $T_c$  a given threshold)
26         $v P^i \leftarrow v P^i \cup w \bar{X}^i$ 
27      endif
28    endfor %strategies w
29    evaluate  $F(\bar{X})$  at all  $\bar{X} \in v P^i$ 
30     $v \bar{X}^i \leftarrow \arg \max_{\bar{X} \in v P^i} F(\bar{X})$ 
31  endfor %strategies v
32 endfor %iterations
33 for each strategy  $v = 1$  to  $N_{str}$ 
34    $v \bar{X}^* \leftarrow \arg \max_i F(v \bar{X}^i)$ 
35 endfor %strategies v
36  $v^* \leftarrow \arg \max F(v \bar{X}^*)$ 
37 return ( $S_{v^*}, v^* \bar{X}^*$ )
38 END

```

Fig. 2. Pseudocode for extended policy gradient task learning.

V. SOCCER ROBOT APPLICATION

Concurrent behavior and parameter learning is useful in many complex high dimensional systems. Here we present a detailed example on which the proposed method has been applied and experimented (results are commented in Section VI). We consider a robot playing soccer within the RoboCup Four-Legged league competitions. One of the main tasks to be accomplished in this scenario is to approach the ball and kick it to the opponent goal. This is a complex task that requires the integration of different behaviors in different ways. Many strategies can be defined to accomplish this task, but a winning strategy is difficult to identify since it depends on many factors: position of the robot and of the ball in the field, position of the other robots, etc. Besides, each behavior has several parameters to be tuned: walking gait parameters, kick parameters, etc. Also, these values should depend on

the situation at hand: for example, we may prefer a fast but imprecise walk when the robot is far from the ball and an opponent robot is closer, while a slower but precise walk may be better when the robot is close to the ball and no other robots are around. Learning such a complex task requires to define a strategy (as a combination of behaviors) and tune the parameters of the behaviors involved in the strategy.

Learning the *attacking* task for a soccer robot consists in learning to approach the ball and kick it to the opponent goal in the ‘best’ way. In the rest of this section, we will describe our task learning process, by describing behaviors and parameters of the attacking robot and some implementation details of the proposed algorithm.

A. Behaviors

For learning the *attacking* task, a set of seven behaviors $\mathcal{B}_P = \{B_1, \dots, B_7\}$ has been considered. These behaviors (see [2] for further details) can be classified in three subsets:

- *Ball approaching* behaviors:
 - B_1 : *fast ball approach* – the path length is minimized by an omnidirectional walk;
 - B_2 : *aligning ball approach* – while the robot approaches the ball, its heading is concurrently oriented towards the opponent goal;
- *Ball controlling* behaviors:
 - B_3 : *rotational ball carrying* – the robot grasps the ball with its chin, and subsequently orients its heading towards the opponent goal;
 - B_4 : *gradual ball reaching* – the robot gradually decreases its translational and rotational velocities in order to reach the ball without knocking it;
- *Ball kicking* behaviors:
 - B_5 : *head straight kick* – the robot ‘dives’ on the ball and hits it with the head - the head pan angle is kept null during this kick;
 - B_6 : *head spanning kick* – the robot ‘dives’ on the ball and hits it with varying head pan: the kick is more powerful but less precise than B_5 ;
 - B_7 : *lateral head kick* – the robot hits the ball laterally with a varying head pan angle.

A strategy is a combination of such behaviors. In this paper we consider only a simple form of behavior composition: chaining between a ball approaching, a ball controlling, and a ball kicking behavior. Thus we consider *strategies* as sequences of three behaviors. For example, $\{B_1; B_3; B_5\}$ is a possible strategy for the attacking task.

B. Parameters

Each behavior B is characterized by a set of parameters Θ_B . The parameters are box-constrained due to the physical characteristics of the system, and we note Δ_j the range size for each θ_j .

Speed and stability of the ball approach are mainly characterized by the *walking gait parameters*, which define the kinematic characteristics of the walk. The trot gait is used, with a half-ellipsoidal locus shape for foot motion, and contact between the base of the locus and the ground generates body motion. The gait is characterized by the eleven walking gait parameters: $\Theta_{WG} = \{\theta_{WG,1}, \dots, \theta_{WG,11}\}$.

Walking gait parameters have little influence on the quality of ball control, which relies mainly on the way the robot slows down and stops near the ball (i.e. on the *ball controlling parameters*). We use translational and rotational slowing down velocities that are respectively proportional to the ball distance and to the relative robot-ball angle.

Thus, ball control is influenced by the four parameters: $\Theta_{BC} = \{\theta_{BC,1}, \dots, \theta_{BC,4}\}$.

The robot kicks are generated by a sequence of fixed joint positions (for AIBO’s 15 joints), at three stages of the kick, related to a timing information, stating for how long the joint values must be kept (i.e. the kick ‘speed’). Motion of the back leg joints is fixed (since it has little influence on the kick quality), whereas motion of the head and front legs should be learned. Hence, twenty three *ball kicking parameters* $\Theta_{BK} = \{\theta_{BK,1}, \dots, \theta_{BK,23}\}$ are used.

The behaviors are not influenced by *all* of the parameters: in practice, each behavior is characterized by a subset of such parameters. For instance, the ball distance at which the robot stops translating ($\theta_{BC,2}$), may be fundamental for behavior B_4 and irrelevant for B_3 . In particular, we are interested in learning the seven sets of parameters Θ_B :

$$\begin{aligned} \Theta_{B1} &\subseteq \Theta_{WG} & \Theta_{B2} &\subseteq \Theta_{WG} \\ \Theta_{B3} &\subseteq \Theta_{BC} & \Theta_{B4} &\subseteq \Theta_{BC} \\ \Theta_{B5} &\subseteq \Theta_{BK} & \Theta_{B6} &\subseteq \Theta_{BK} & \Theta_{B7} &\subseteq \Theta_{BK} \end{aligned}$$

C. Learning configuration

The learning process consists in choosing the best strategy and optimizing the parameter sets: Θ_{WG} , Θ_{BC} , and Θ_{BK} . These sets are represented as a vector:

$$\underline{X} = [\theta_{WG,1} \dots \theta_{WG,11} \theta_{BC,1} \dots \theta_{BC,4} \theta_{BK,1} \dots \theta_{BK,23}]^T \in \mathbf{R}^{38}$$

that also represents a candidate solution of the optimization problem. As already mentioned, these sets might include parameters which are irrelevant for the strategy, and the learning algorithm will deal with them.

The very large dimensions of the search space, and the system characteristics, suggest the use of the *layered learning* paradigm [5] for optimizing this problem. In fact, given a hierarchical task decomposition, layered learning allows for learning at each level of the hierarchy, with learning at each level directly affecting learning at the next higher level. The *incremental learning* approach that we use is inspired by the layered learning paradigm; however, in contrast with classic layered learning, we utilize the same learning method for each layer of the hierarchy. Specifically, we decompose optimization of the strategy parameters in the following three optimization subtasks (layers):

- L_1 : find $\underline{X}_{1,opt} = [\theta_{WG,1} \dots \theta_{WG,11}]_{opt}^T \in \mathbf{R}^{11}$ for ‘best’ walk;
- L_2 : find $\underline{X}_{2,opt} = [\theta_{BC,1} \dots \theta_{BC,4}]_{opt}^T \in \mathbf{R}^4$ for ‘best’ ball control, given $\underline{X}_{1,opt}$ found by L_1 ;
- L_3 : find $\underline{X}_{3,opt} = [\theta_{BK,1} \dots \theta_{BK,23}]_{opt}^T \in \mathbf{R}^{23}$ for ‘best’ kicking, given $\underline{X}_{1,opt}$ and $\underline{X}_{2,opt}$ found by L_1 and L_2 .

The solution of the attacker strategy optimization problem can then be obtained as:

$$\underline{X}_{opt} = [\underline{X}_{1,opt} \ \underline{X}_{2,opt} \ \underline{X}_{3,opt}]^T$$

The appropriate choice of the objective function for optimization is fundamental. The function for evaluating the quality of an attacking strategy – *the fitness function* – must take into account: the quality (speed and precision) of the robot motion for approaching the ball, the quality of ball control, and the quality (power and precision) of the kick. Hence, we adopt the following objective function:

$$F(\underline{X}) = K_{WG} f_{WG}(\underline{X}) + K_{BC} f_{BC}(\underline{X}) + K_{BK} f_{BK}(\underline{X})$$

where f_{WG} , f_{BC} , f_{BK} indicate respectively the quality of the ball approaching, ball controlling, and ball kicking

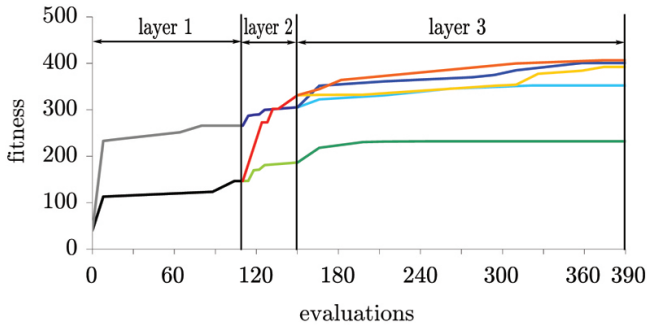


Fig. 3. The best fitness values obtained from each evaluation during USARSim training with *classic policy gradient* at layers: L_1 (strategies S_1 , S_2 and S_3 in black, S_4 and S_5 in grey), L_2 (strategies S_1 , S_2 in red, S_3 in green, S_4 and S_5 in blue) and L_3 : S_1 (orange), S_2 (red), S_3 (green) S_4 (blue) and S_5 (cyan).

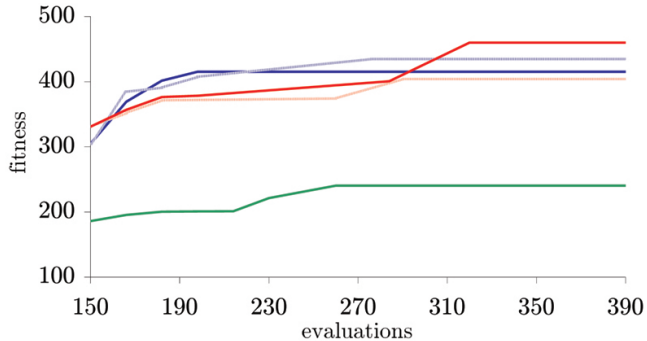


Fig. 4. The fitness values obtained during USARSim training with *extended policy gradient* (considering *parameter relevance*) at L_3 : S_1 (pink), S_2 (red), S_3 (green) S_4 (cyan) and S_5 (blue).

behavior used. These functions are derived with heuristics on the robot and ball positions at various stages of the attacking strategy execution. The functions are derived identically for behaviors in the same subset (e.g., f_{BC} is derived in the same way for B_3 and B_4). K_{WG} , K_{BC} , and K_{BK} are positive weights indicating the significance desired for each of the three aspects in the learning process. $F(\underline{X})$ is thus identical (i.e., same functions f_{WG} , f_{BC} and f_{BK} and same weights) for all the strategies used to execute the attacking task.

VI. EXPERIMENTAL RESULTS

In this section, we report the experimental results obtained by applying the two proposed policy gradient methods for task learning in the described robot soccer scenario. We take advantage of two results from our previous work [2]: the utility of using a 3D simulator for speeding up the learning process, and the effectiveness of the incremental approach for robot task learning. The major contribution of the experiments is the comparison between the classic and extended policy gradient methods presented in Section IV.

The objective of the learning algorithms is to find the most suitable strategy S_{v^*} and the corresponding optimal parameters $v^* \underline{X}^*$ for a given task. In practice, we focus on the following task T : *ball 50 cm in front of robot, and direction from ball to goal orthogonal to the robot sagittal axis, with the goal on the right of the robot*. The five strategies chosen to achieve the task are:

$$\begin{aligned} S_1 &= \{B_1 B_3 B_5\} & S_2 &= \{B_1 B_3 B_6\} \\ S_3 &= \{B_1 B_4 B_7\} & S_4 &= \{B_2 B_4 B_5\} & S_5 &= \{B_2 B_4 B_6\} \end{aligned}$$

The performance of these strategies can be compared, since the objective function is the same for all the strategies.

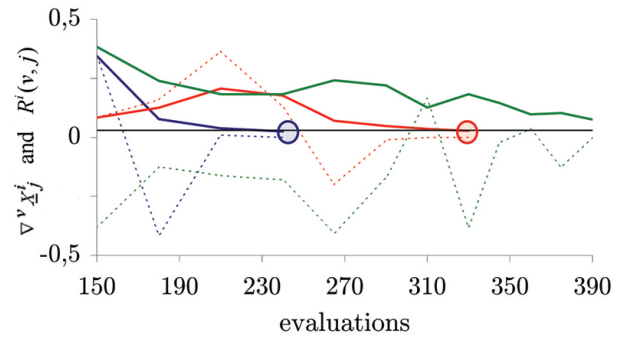


Fig. 5. Evolution of *gradient component* (dashed line) and *relevance* (continuous line) of: $\theta_{BK,3}$ (green), $\theta_{BK,13}$ (red) and $\theta_{BK,24}$ (blue) during USARSim training of S_2 at L_3 : circles indicate the evaluation step where the parameter becomes irrelevant (the black line indicates T_r).

Learning this task is initially implemented with the 3D AIBO simulator [11] embedded in USARSim¹ before experimentation on the real robot. In fact, in [2], we have shown that the learning configurations tuned in the simulated environment can be successfully ported on the real robot.

We utilize the incremental learning approach presented in the previous section. For the 3 layers, we choose the policy gradient configuration: $\epsilon_j = 0.1 \Delta_j$, $\eta = 3$. In the classic version, the number of policies p at the 3 layers is set to: $p_1 = 8$, $p_2 = 4$, $p_3 = 16$. Instead, in the extended version, since the number of parameters decreases during the learning process (depending on their relevance), p is set to two thirds of the number of parameters to be learned. The same initial parameter set ${}^0 \underline{X}$ is used for starting learning on each strategy v . Since there is significant noise in the experiments, each set of parameters is evaluated twice, and the resulting fitness *evaluated* for that set, is computed by averaging over the two experiments. In USARSim, since 'hardware consumption' is not an issue, for each layer, we compare the learning algorithms after 10 k_i evaluations have been performed (e.g. for L_3 , 240 evaluations, i.e., 480 experiments). Comparison between extended and classic policy gradient has been carried out at layer L_3 . For the first two layers, classic policy gradient learning has been used to derive $\underline{X}_{1,opt}$ and $\underline{X}_{2,opt}$. The results of incremental learning at L_1 and L_2 are shown in Fig. 3, where the best fitness value found by classic policy gradient is plotted over the number of evaluations, for each of the 5 strategies.

Afterwards, we proceed by learning layer L_3 with the classic and extended policy gradient. To emphasize the capability of the extended version to filter out irrelevant parameters, we add a 24th *irrelevant* parameter $\theta_{BK,24}$ (which is not used by the robot for kicking the ball) to the set Θ_{BK} . The results of learning L_3 with the classic policy gradient are shown in Fig. 3. The extended policy gradient is initially experimented by considering solely **parameter relevance**. We use: $T_r = 0.03$ and $\lambda = 0.8$. At the end of the learning process, for all five strategies, approximately 50% of the parameters are filtered out (e.g., 11 parameters for S_1 , 13 parameters for S_4). For all strategies, $\theta_{BK,24}$ is filtered out during the learning process. The fitness values during learning with parameter reduction are shown in Fig. 4: for all strategies, the final fitness outperforms the fitness obtained with the classic algorithm. The best improvements are for strategies S_2 (the fitness increases by 18%) and S_5 (14%). For strategy S_2 (which prevails in the experiments), the gradient component and relevance of parameters: $\theta_{BK,3}$, $\theta_{BK,13}$ and $\theta_{BK,24}$ are plotted in Fig. 5. Note that, as expected, $\theta_{BK,24}$ is filtered out

¹<http://usarsim.sourceforge.net>

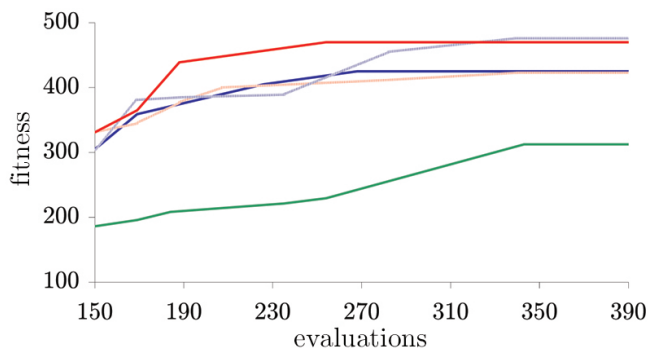


Fig. 6. The fitness values obtained during USARSim training with *extended policy gradient* (considering *parameter relevance* and *strategy contiguity*) at L_3 : S_1 (pink), S_2 (red), S_3 (green) S_4 (cyan) and S_5 (blue)

very early in the learning process. Most of the experiments reveal the irrelevance of the leg positions at the second (i.e., intermediate) stage of the kick. This result implies that it could be sufficient to learn the leg positions at the first and final stage of the kick.

In the next experiment, **strategy contiguity** is also utilized in the extended algorithm, along with parameter relevance; we set $T_c = 0.002$. For all strategies, the final fitness (plotted in Fig. 6) outperforms the fitnesses obtained with the classic algorithm and with the extended version that considers only parameter relevance. The best improvements are for strategies S_3 (the fitness increases by 35% with respect to classic policy gradient) and S_5 (21%). During the learning process, the algorithm takes advantage of the contiguities between strategies S_1 , S_2 and S_3 and also between strategies S_1 , S_4 and S_5 . With this complete version of the extended policy gradient, strategy S_4 prevails.

Finally, both algorithms are ported on the real robot, for comparison at learning $X_{3,opt}$ with fixed: $X_{1,opt}$, $X_{2,opt}$. The strategies that performed better in USARSim (S_2 and S_4) are used. For each strategy, three experiments are carried out. Firstly, classic policy gradient is applied for learning *all* 24 parameters (dotted curves in Fig. 7). Then (dashed curves), classic policy gradient is applied for learning only the *relevant* parameters (12 parameters in S_2 , 13 in S_4) derived in the simulated experiments. Finally (solid curves), the extended policy gradient is used, by *reducing* the parameter set initialized with the relevant parameters derived in USARSim. The initial parameter sets are the final sets obtained from USARSim training. Each set of parameters is evaluated twice, and the resulting fitness is obtained by averaging. This time, we terminate learning after 120 *evaluations* have been performed. For both strategies, the utility of reducing the parameter set in the simulated environment before porting on the real robot is visible by comparison of the dotted and dashed curves in Fig. 7. Further improvement, as expected, is obtained by reducing the parameter set throughout the learning process (solid curves). At the end of the experiments, 10 parameters are preserved for S_2 , 9 for S_4 . Hence, 2 parameters for S_2 and 4 for S_4 have not required further adjustment on the real robot after USARSim training.

VII. CONCLUSIONS

In this paper, we presented a method for concurrent learning of best strategy and optimal parameters, by extending the policy gradient reinforcement learning algorithm. The proposed method guarantees fast convergence by exploiting information on the system properties, while learning. In particular, the contiguities between strategies, and the pa-

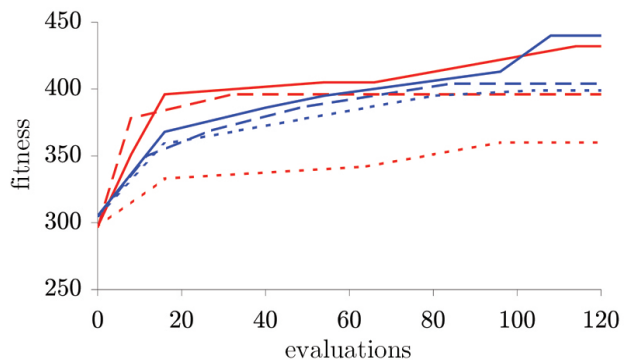


Fig. 7. Fitnesses during AIBO training with start point derived from USARSim training: S_2 in red, S_4 in blue.

parameter relevance, are estimated and utilized by the algorithm during training. Parameter relevance enables reduction of the search space size during learning, while contiguities improve optimization by search for maxima among similar strategies.

The proposed learning technique has been applied to the soccer attacking task. It has been implemented in USARSim and extensively experimented in laboratory on AIBO, showing a notable improvement in the learned robot performance with respect to robot performance learned with classic policy gradient. Firstly, our technique outperforms classic policy gradient when the comparison is carried out over the same number of evaluations. Moreover, the convergence rate of our technique is higher than that of classic policy gradient: the same fitness value (e.g., in USARSim, 400) is attained much earlier (145 evaluations in advance). These results are fundamental in robot applications, where hardware consumption is a major issue. Development of a robot task learning algorithm that identifies the system major characteristics during training could be the object of further work.

ACKNOWLEDGMENT

The authors thank Maria Sciannelli for her support in the USARSim experiments.

REFERENCES

- [1] S. Chalup, C. Murch, and M. Quinlan, "Machine learning with AIBO robots in the four-legged league of robocup," *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 2006.
- [2] A. Cherubini, F. Giannone, and L. Iocchi, "Layered learning for a soccer legged robot helped with a 3D simulator," in *Proceedings of 11th International Robocup 2007 Symposium*, 2007.
- [3] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proceedings of International Conference on Robotics and Automation*, 2004.
- [4] P. Fiedelman and P. Stone, "The chin pinch: A case study in skill learning on a legged robot," in *Proceedings of 10th International Robocup 2006 Symposium*, 2006.
- [5] P. Stone and M. Veloso, "Layered learning," in *Machine Learning: ECML 2000 (Proceedings of the Eleventh European Conference on Machine Learning)*, R. L. D. Mántaras and E. Plaza, Eds. Springer Verlag, Barcelona, Spain, May/June 2000, pp. 369–381.
- [6] N. Bredeche, Z. Shi, and J.-D. Zucker, "Perceptual learning and abstraction in machine learning: an application to autonomous robots," *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, vol. 36, no. 2, pp. 172–181, March 2006.
- [7] S. Nolfi, "Evolutionary robotics: Exploiting the full power of self-organization," *Connection Science*, vol. 10, pp. 167–183, 1998.
- [8] J. Koza, *Genetic Programming: on the programming of computers by means of natural selection*, MIT Press, Cambridge, Mass., 1992.
- [9] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass., 1998.
- [10] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057–1063, 2000.
- [11] M. Zaratti, M. Fratarcangeli, and L. Iocchi, "A 3D simulator of multiple legged robots based on USARSim," in *Proceedings of 10th International Robocup 2006 Symposium*, 2006.