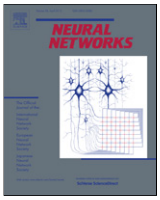
Neural Networks 41 (2013) 156–167

Contents lists available at SciVerse ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



2013 Special Issue

# Autonomous reinforcement learning with experience replay

Paweł Wawrzyński [a,*], Ajay Kumar Tanwani [b]

[a] Warsaw University of Technology, Institute of Control and Computation Engineering, Poland
[b] École Polytechnique Fédérale De Lausanne, Switzerland

ARTICLE INFO

Keywords:
Reinforcement learning
Autonomous learning
Step-size estimation
Actor–critic

ABSTRACT

This paper considers the issues of efficiency and autonomy that are required to make reinforcement learning suitable for real-life control tasks. A real-time reinforcement learning algorithm is presented that repeatedly adjusts the control policy with the use of previously collected samples, and autonomously estimates the appropriate step-sizes for the learning updates. The algorithm is based on the actor–critic with experience replay whose step-sizes are determined on-line by an enhanced fixed point algorithm for on-line neural network training. An experimental study with simulated octopus arm and half-cheetah demonstrates the feasibility of the proposed algorithm to solve difficult learning control problems in an autonomous way within reasonably short time.

## 1. Introduction

Reinforcement learning (RL) is a potentially useful methodology for control optimization. It may be the only feasible approach to many problems. However, its widespread use is limited by the total time many RL algorithms require to solve a given problem. This time encompasses duration of a single run in collecting appropriate number of samples multiplied by the number of runs necessary to tune problem dependent parameters. Selection of appropriate parameter setting requires learning to start over again, thereby, making the previously collected data useless. The cost of collecting data from control process and the elapsed time to handcraft the parameters makes learning infeasible for many real world tasks. Ideally, the learner should interact with the environment in a 'plug-and-play' manner and optimize a control policy in a single trial with minimum number of data samples. The main focus of this research is to make reinforcement learning suitable for real control applications and minimize the associated implementation costs.

A broad class of RL algorithms that encompass Actor–Critics (Bhatnagar, Sutton, Ghavamzadeh, & Lee, 2009; Konda & Tsitsiklis, 2003; Peters, Vijayakumar, & Schaal, 2005) and policy gradient methods (Sutton, McAllester, Singh, & Mansour, 2000; Williams, 1992) optimizes a control policy on the basis of stochastic gradient estimates. In these algorithms, the control policy is parameterized by a vector that is optimized incrementally on the basis of the data coming from the control process. The policy vector adjustment is local in the sense that the algorithm moves a small step in the estimate of the improvement direction; its length is defined by the step-size of the learning process. The appropriate length of the step-size depends on the stage of the learning process and setting it wrong can make the process unstable or prohibitively slow. In this paper, we present a real-time reinforcement learning algorithm that is considerably efficient and autonomous due to: (1) *experience replay*—it stores the agent–environment interaction samples in a database and repeatedly draws previous samples—simultaneous to the interaction—to estimate the improvement direction of policy vector and speed up learning (Wawrzyński, 2009), and (2) *step-size estimation*—it autonomously determines the appropriate step-size to optimize the learning updates and eliminate the overhead of its manual tuning (Wawrzyński, 2010).

Optimization by means of improvement direction estimates is a tool commonly used in RL but it is not unique to this field. It is the main principle of a wide class of stochastic approximation algorithms (Kushner & Yin, 1997). In all these methods the step-size plays a key role in controlling their speed and stability. Despite significance of the problem of step-size estimation, its generally applicable solution has not been found.

### 1.1. Related work

A lot of effort is made in the reinforcement learning community to speed up the learning methods by using the previously collected samples. The concept of reusing samples evolved from recomputing previous experience by means of dynamic programming (Sutton, 1990). Application of the *importance sampling* technique (Rubinstein, 1981) made it possible to optimize a parametric control policy with the use of experience gained with other policies

[*] Corresponding author.
E-mail addresses: pwawrzynski@gmail.com, p.wawrzynski@elka.pw.edu.pl (P. Wawrzyński), ajay.tanwani@epfl.ch (A.K. Tanwani).

0893-6080/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2012.11.007

(Hachiya, Peters, & Sugiyama, 2011; Peshkin & Mukherjee, 2001). Kober and Peters (2011) proposed an algorithm called PoWER—Policy Learning by Weighting Exploration with the Returns—that combines the previous experience by per-decision importance weighting technique. The approach to sample reuse developed here is experience replay (Adam, Busoniu, & Babuska, 2012; Cichosz, 1999) applied to actor–critics with the use of the randomized truncated estimators (Wawrzyński, 2009).

Real world applications of reinforcement learning require a number of parameters to be defined and empirically tuned, e.g., reward function, policy structure, step-size, exploration noise, discount factor, and others specific to the learning algorithm in use. Despite several promising approaches to provide leverage for these overheads such as the use of apprenticeship learning to find reward function (Abbeel & Ng, 2005), dynamic evolution of policy parameters to get its appropriate size and representation (Kormushev, Ugurlu, Calinon, Tsagarakis, & Caldwell, 2011), the parameter that remains difficult to determine autonomously in reinforcement learning is the step-size of the learning process. The work (Sutton, 1992a, 1992b) established *delta-bar-delta* algorithm, introduced earlier by Jacobs (1988) as a generally applicable approach to step-size estimation in RL. The method requires the initial step-size to be specified and collapses if this value is too large, which delimits its autonomous use. Consequently, the solution to the problem of step-size determination in reinforcement learning is still a subject of active research (George & Powell, 2006; Noda, 2009; Schraudolph, Yu, & Aberdeen, 2006).

Many approaches to step-size estimation have been designed as tools for online neural network training. Algorithms of this type include *delta-delta* (Silva & Almeida, 1990), aforementioned *delta-bar-delta*, and stochastic meta-descent (Schraudolph & Giannakopoulos, 2000). Among others, step-sizes can be estimated to ensure Lyapunov-stability of the process (Behera, Kumar, & Patnaik, 2006; Kathirvalavakumar & Subavathi, 2009).

Most methods of autonomous or adaptive step-size estimation are designed as recursions that are updated from one sample to another (George & Powell, 2006). This design is often based on some problem-dependent parameters and may be sensitive to wrong initialization or nonstationarity. In this work we build upon the fixed-point method of step-size estimation (Wawrzyński, 2010) which has a different design to remedy the aforementioned problems. By computing gradient estimates always at two points it captures global characteristics of the process, thereby, obtaining robustness and independence from any parameters. In Wawrzyński and Papis (2011) this approach was specialized to neural-network on-line training, e.g., each weight of the network was assigned a separate step-size. Here the original algorithm is enhanced and adopted to provide autonomy to RL with experience replay.

### 1.2. Contributions

The purpose of this paper is to present a reinforcement learning framework that is fast and autonomous enough for on-line optimization of control of systems with complex dynamics and multidimensional, possibly continuous, state and input spaces. The contribution of this paper may be summarized in the following points:

- An enhanced fixed point method of step-size estimation (Wawrzyński, 2010) is presented.
- The above method of step-size estimation is combined with experience replay for actor–critic class of RL algorithms (Wawrzyński, 2009).
- The proposed approach is implemented on two simulated robot-control problems, both with complex dynamics and rich state and action spaces, namely *octopus arm* and *half-cheetah*.

The rest of the paper is organized as follows: Section 2 formulates the reinforcement learning problem followed by the description of actor–critic algorithm with experience replay in Section 3. In Section 4, we describe the fixed point method of step-size adaptation in a generic stochastic descent procedure. The resulting actor–critic reinforcement learning algorithm with experience replay and step-size estimation is proposed in Section 5. Experimental study is presented in Section 6 on challenging learning control problems. The last section concludes the paper with an outlook to future work.

## 2. Problem formulation

We consider the standard reinforcement learning setup under the Markov Decision Process (MDP) framework (Sutton & Barto, 1998). The problem concerns an agent that observes the state of its environment, $s_t$, in discrete time, $t = 1, 2, 3, \dots$, performs an action, $a_t$, which moves the environment to the next state, $s_{t+1}$, and gives the agent a reward, $r_t \in \Re$. The environment is in general stochastic which means that the consecutive state, $s_{t+1}$, is a result of sampling from the transition distribution conditioned on the preceding state, $s_t$, and the action, $a_t$. Mathematically,

$$s_{t+1} \sim P_s(\cdot|s_t, a_t).$$

The reward may depend deterministically on the current action and the next state, $r_t = r(a_t, s_{t+1})$. A particular MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, P_s, r\rangle$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively; $\{P_s(\cdot|s, a) : s \in \mathcal{S}, a \in \mathcal{A}\}$ is a set of state transition distributions and $r$ is the reward function. The transition distributions, $P_s$, and the reward function, $r$, are initially unknown to the agent. The goal of learning is to determine a stochastic control policy $\pi$ that assigns actions to states such that in each state the agent may expect the highest rewards in the future. Note that the control policy maps each state into a distribution of actions rather than a single action. Performing different actions in each state, the agent is able to differentiate good actions from inferior ones in each state. The control policy $\pi$ is parameterized by the policy vector, $\theta$, e.g., a set of weights in a neural network. It can be represented as

$$a \sim \pi(\cdot; s, \theta).$$

Let us denote by $\pi_\theta$ the policy of the form $\pi$ with the fixed parameter $\theta$. The quality of this policy is measured by the *value-function* $V^{\pi_\theta}$ defined as the expected value of the discounted sum of future rewards the agent may gain in the future, starting from the given state, $s$, at a certain time, $t$, and the policy $\pi_\theta$:

$$V^{\pi_\theta}(s) = E\left(\sum_{i\geq 0} \gamma^i r_{t+i}\Big| s_t = s,\ \text{policy} = \pi_\theta\right). \tag{1}$$

The parameter $\gamma \in [0, 1)$ is the discount factor that defines the weight of distant rewards in relation to those obtained sooner. The quality of a policy may also be evaluated using the *action-value function* $Q^{\pi_\theta}$, defined as

$$Q^{\pi_\theta}(s, a) = E\left(\sum_{i\geq 0} \gamma^i r_{t+i}\Big| s_t = s,\ a_t = a,\ \text{policy} = \pi_\theta\right). \tag{2}$$

It is the expected value of the same sum, but here it also depends on the first action to be performed, that is the action performed in the given state. This function is crucial for the policy optimization as it tells good actions from inferior ones in the same state.

We seek a learning algorithm that can optimize the control policy (i) without parameters determined experimentally, (ii) within short time of agent–environment interaction, but not necessarily after small amount of computation.

Algorithm 1. The classic actor–critic.

```
0:  Set $y_\theta = 0, y_\upsilon = 0, t = 1$. Initialize $\theta$ and $\upsilon$
1:  Choose the action: $a_t \sim \pi(\cdot\,; s_t, \theta)$
2:  Execute $a_t$, evaluate $s_{t+1}$ and $r_t$
3:  Calculate the temporal difference $d_t(\upsilon)$
4:    $d_t(\upsilon) = r_t + \gamma\bar{V}(s_{t+1}; \upsilon) - \bar{V}(s_t; \upsilon)$
5:  Update the actor:
6:    $y_\theta := (\gamma\lambda)y_\theta + \beta_t^\theta \nabla_\theta \ln\pi(a_t; s_t, \theta)$
7:    $\theta := \theta + y_\theta d_t$
8:  Update the critic:
9:    $y_\upsilon := (\gamma\lambda)y_\upsilon + \beta_t^\theta \nabla_\upsilon \bar{V}(s_t; \upsilon)$
10:   $\upsilon := \upsilon + y_\upsilon d_t$
11: Assign $t := t + 1$ and repeat from Point 1
```

## 3. Actor–critic with experience replay

This section presents actor–critic algorithms (Barto, Sutton, & Anderson, 1983; Bhatnagar et al., 2009; Kimura & Kobayashi, 1998; Konda & Tsitsiklis, 2003) and the way of its speeding-up by experience replay (Adam et al., 2012; Cichosz, 1999; Wawrzyński, 2009). Particular attention is paid to the 'classic' actor–critic in the form presented by Kimura and Kobayashi (1998) because this algorithm is conceptually the simplest among its family to which the ideas presented in further sections are directly applicable.

### 3.1. Classic Actor–Critic

The classic actor–critic algorithm employs two data structures. The first one is an actor, $\pi(a; s, \theta)$, that represents a stochastic control policy to generate random actions on the basis of the given state and the policy vector $\theta \in \Re^{n_\theta}$. The second one is a critic, $\bar{V}(s; \upsilon)$, that represents an approximator of the value function (1) parameterized by the critic vector $\upsilon \in \Re^{n_\upsilon}$. The algorithm is based on a special estimator of the action-value function $Q^{\pi_\theta}(s_t, a_t)$ (Eq. (2)). This estimator has the form:

$$R_t^\lambda = r_t + \gamma\bar{V}(s_{t+1}; \upsilon) + \sum_{i\geq 1}(\gamma\lambda)^i \left(r_{t+i} + \gamma\bar{V}(s_{t+i+1}; \upsilon) - \bar{V}(s_{t+i}; \upsilon)\right). \tag{3}$$

The parameter $\lambda \in [0, 1]$ defines how much $R_t^\lambda$ is influenced by the value-function approximator, $\bar{V}$, and the true rewards, $r$. For $\lambda = 1$ the estimator is based on true rewards only and thus unbiased, but its variance may be very large. For $\lambda = 0$ variance of the estimator is the smallest, but it is biased by the inaccuracy of the value-function approximator. Moderate values of $\lambda$ balance the bias and the variance of the estimator.

The classic actor–critic is sketched in Algorithm 1. It can be verified that after a visit in state $s_t$, for $t = 1, 2, \ldots,$ the algorithm modifies the policy vector $\theta$ by the product $\beta_t^\theta \widehat{\phi}_t$ where $\beta_t^\theta$ is a step-size (small positive number) and $\widehat{\phi}_t$ is the policy vector improvement direction estimate given by

$$\widehat{\phi}_t = \left(R_t^\lambda - \bar{V}(s_t; \upsilon)\right)\nabla_\theta \ln\pi(a_t; s_t, \theta), \tag{4}$$

where $\nabla_\theta$ means gradient with respect to $\theta$. (For $\lambda = 0$, the modification takes place in step $t$, for $\lambda > 0$ it is also partially distributed over further steps). Consequently, if the action $a_t$ turns out to bring rewards, $R_t^\lambda$, larger than $\bar{V}(s_t; \upsilon)$, i.e., the rewards expected in state $s_t$, then the probability of action $a_t$ in state $s_t$ is being increased. If, conversely, the action turns out to bring rewards smaller than expected, then its probability is being decreased.

The purpose of the critic, $\bar{V}(s; \upsilon)$ is to approximate the expected value of $R_t^\lambda$. Therefore, a visit in state $s_t$ induces a modification of the critic vector $\upsilon$ by the product $\beta_t^\upsilon \widehat{\psi}_t$ where $\beta_t^\upsilon$ is a step-size and $\widehat{\psi}_t$ is the critic vector improvement direction estimate:

$$\widehat{\psi}_t = \left(R_t^\lambda - \bar{V}(s_t; \upsilon)\right)\nabla_\upsilon \bar{V}(s_t; \upsilon). \tag{5}$$

Algorithm 2. Actor–Critic with Experience Replay. Estimators mentioned in Steps 6 and 7 are based on the samples in a database.

```
1:  $t := 1$, Initialize $\theta$ and $\upsilon$
2:  Choose and execute an action, $a_t \sim \pi(\cdot\,; s_t, \theta)$
3:  Assign $\pi_t = \pi(a_t; s_t, \theta)$
4:  Repeat $\nu(t)$ times, begin
5:    Draw $i \in \{t - N + 1, t - N + 2, \ldots, t\}$
6:    Adjust $\theta$ along an estimator of $\phi(s_i, \theta, \upsilon)$:
7:      $\theta := \theta + \beta_t^\theta \widehat{\phi}_i^r(\theta, \upsilon)$
8:    Adjust $\upsilon$ along an estimator of $\psi(s_i, \theta, \upsilon)$:
9:      $\upsilon := \upsilon + \beta_t^\upsilon \widehat{\psi}_i^r(\theta, \upsilon)$
10: End
11: Register the tuple $\langle s_t, a_t, \pi_t, r_t, s_{t+1}\rangle$ in the database
12: Make sure only $N$ most recent tuples remain in the
    database.
13: Assign $t := t + 1$ and repeat from Line 2
```

### 3.2. Classic actor–critic with experience replay

The idea of experience replay is to repeatedly recall previous pieces of experience and apply them to update the current policy as a sequential RL algorithm (like the classic actor–critic) would if they have just happened. This idea was applied to actor–critics in Wawrzyński (2009). Namely, let us denote by

$$\phi(s, \theta, \upsilon) = E\left(\widehat{\phi}_t \middle| s_t = s, \theta, \upsilon\right) \tag{6}$$

a direction in which $\theta$ is on average modified to increase the rewards expected in state $s$. Also, let us denote by

$$\psi(s, \theta, \upsilon) = E\left(\widehat{\psi}_t \middle| s_t = s, \theta, \upsilon\right) \tag{7}$$

a direction in which $\upsilon$ is on average modified to increase the accuracy of the value function approximation in state $s$.

After each instant $t$, the classic actor–critic estimates $\phi(s_t, \theta, \upsilon)$, i.e., the direction of policy improvement, and adjusts the policy vector $\theta$ along the estimate. Before the next instant $t$, the modified algorithm repeatedly chooses one of the recently visited states at random, $s_i$, estimates $\phi(s_i, \theta, \upsilon)$, and modifies the policy vector along the estimate. Essentially both algorithms achieve the same goal, but the modified one improves the current actor and critic with the use of the whole gathered experience rather than only the present event, like the classic method. Due to more exhaustive exploitation of information experience replay leads to faster learning at the cost of additional computation.

In Wawrzyński (2009), the *randomized-truncated estimators* are introduced for estimating $\phi(s_i, \theta, \upsilon)$ and $\psi(s_i, \theta, \upsilon)$. To define them, let $b$ be the upper limit of the randomized truncated estimators with $b > 1, \theta_{i+j}$ be the policy vector applied to generate $a_{i+j}, \alpha \in [0, 1)$ and $K$ be drawn independently from $Geom(\rho)$ the geometric distribution[1] with parameter $\rho \in [0, 1)$. The randomized-truncated estimators, $\widehat{\phi}_i^r(\theta, \upsilon)$ and $\widehat{\psi}_i^r(\theta, \upsilon)$ have a generic form that in the case of classic actor–critic is reduced to

$$\widehat{\phi}_i^r(\theta, \upsilon) = \nabla_\theta \ln\pi(a_t; s_t, \theta) \times \sum_{k=0}^{K} \alpha^k d_{i+k}(\upsilon) \min\left\{\prod_{j=0}^{k} \frac{\pi(a_{i+j}; s_{i+j}, \theta)}{\pi_{i+j}}, b\right\}, \tag{8}$$

[1] That is, random variable $K$ of values in $\{0, 1, 2, \ldots\}$ has distribution $Geom(\rho)$, iff $P(K = m) = (1 - \rho)\rho^m$ for nonnegative integer $m$.

$$\widehat{\psi}^r_i(\theta, \upsilon) = \nabla_\upsilon \bar{V}(s_t; \upsilon) \times \sum_{k=0}^{K} \alpha^k d_{i+k}(\upsilon) \min \left\{ \prod_{j=0}^{k} \frac{\pi(a_{i+j}; s_{i+j}, \theta)}{\pi_{i+j}}, b \right\} \tag{9}$$

where

$$d_i(\upsilon) = r_i + \gamma \bar{V}(s_{i+1}; \upsilon) - \bar{V}(s_i; \upsilon),$$

is the temporal difference and $\pi_{i+j}$ is the probability of the action taken at instant $i + j$ (or it is the density if $\mathcal{A}$ is continuous). Remembering the definition of $R^\lambda_t$ (Eq. (3)) makes it easy to notice the similarity between $\widehat{\phi}_t$ (Eq. (4)) and $\widehat{\phi}^r_i$ (Eq. (8)), and between $\widehat{\psi}_t$ (Eq. (5)) and $\widehat{\psi}^r_i$ (Eq. (9)). Though, there are two important differences. First, the infinite sum in Eq. (3) is replaced by the finite one with appropriately designed random limit in Eqs. (8) and (9). Second, truncated density ratios are introduced in order to compensate for the difference between the current policy and the ones that generated the actions $a_t, a_{t-1}, \ldots$ present in the database.

The actor–critic with experience replay is presented in Algorithm 2. The number of computation steps after $t$-th control step is defined by the function $\nu(t)$. Since computation is carried out on-line, this function is bounded from above. It should also be limited for small $t$ to prevent overtraining with too few samples in the database. (In Section 5 this function is discussed in more detail.)

### 3.3. Generalized actor–critic with experience replay

Experience replay may be applied to the classic actor–critic to increase its learning speed, but also to a wide class of sequential actor–critic algorithms. As shown in Wawrzyński (2009) the sufficient condition is that each step in the environment prompts the original algorithm to do modifications (possibly distributed in time) of the policy vector and the critic vector with estimates, respectively, $\widehat{\phi}_t$ and $\widehat{\psi}_t$ of the generic form

$$G_t(\theta, \upsilon) \sum_{k \geq 0} (\alpha\rho)^k z_{t,k}(\theta, \upsilon)$$

where $G_t$ is a vector defined by $s_t$ and $a_t$, $\alpha \in [0, 1)$, $\rho \in [0, 1)$, and $z_{t,k} \in \Re$ is defined by $s_{t+k}, a_{t+k}, r_{t+k}, s_{t+k+1}$, and possibly $a_{t+k+1}$.

## 4. Fixed point method of step-size estimation

The actor–critic algorithm with experience replay repeatedly samples data and uses it to compute a certain vector with which it updates parameters $\theta$ and $\upsilon$. This pattern of operation places it in a wide class of optimization algorithms based on stochastic gradient descent (Kushner & Yin, 1997). Each algorithm of that type defines a sequence

$$\theta_{t+1} = \theta_t - \beta_t g(\theta_t; \xi_t), \quad t = 1, 2, \ldots \tag{10}$$

in which $\theta \in \Re^{n_\theta}$ is a parameter to be optimized, $\beta_t$, $t = 1, 2, \ldots$ is a sequence of step-sizes, $\xi_t$, $t = 1, 2, \ldots$ is a sequence of data samples and $g$ is a function such that for random $\xi$ the expected value of $-g(\theta, \xi)$ defines the direction towards the local minimum of a certain quality index, $J : \Re^{n_\theta} \mapsto \Re$. Typically this direction is the opposite of the gradient

$$Eg(\theta, \xi) = \nabla J(\theta).$$

For example, suppose a feedforward neural network is trained to approximate a function from given samples. Then $\theta$ is the vector of neural weights, $J$ is the quality index, $\xi$ represents an input–output pair, and $g$ is the gradient computed by means of e.g., error backpropagation. In order to train the network, the input–output pairs may be sampled repeatedly, and its weights may be adjusted according to Eq. (10), which is exactly how the well established back-propagation algorithm works (Rumelhart, Hinton, & Williams, 1988).

For the sequence (10) to have convergence guarantees, a set of requirements needs to be fulfilled (Kushner & Yin, 1997). Among others, the sequence of step-sizes has to be vanishing at an appropriate pace:

$$\begin{cases} \sum_{t \geq 1} \beta_t = +\infty, \\ \sum_{t \geq 1} \beta_t^2 < +\infty. \end{cases} \tag{11}$$

Unfortunately the practical value of the above conditions is limited. Even if they are satisfied, the step-sizes may still be too large and hence make the convergence process unstable, or too small and hence yield a very slow convergence.

The fixed point method of step-size estimation addresses this issue by autonomously setting the step-sizes (Wawrzyński, 2010). The method is based on the following principles:

1. The process (10) is divided into parts such that within each part the step-size remains constant. The length, $n$, of a part starting at moment $t$ is kept large enough to enable estimation of the expected value of $g(\theta_t, \xi)$ with sufficient accuracy.
2. The sums $G_{t,n}$ and $G^*_{t,n}$ are computed for each part beginning at time $t$ and ending at time $t + n$

$$G_{t,n} = \sum_{i=0}^{n-1} g(\theta_{t+i}, \xi_{t+i}), \qquad G^*_{t,n} = \sum_{i=0}^{n-1} g(\theta_t, \xi_{t+i}). \tag{12}$$

   where the vector $G_{t,n}$ is proportional to the displacement of the point $\theta$ between instant $t$ and $t + n$, and the vector $G^*_{t,n}$ is proportional to the displacement of the point $\theta$ if all the gradient estimators $g$ are calculated using $\theta_t$.
3. At the end of each part the discrepancy between $G^*_{t,n}$ and $G_{t,n}$ is registered. On average, it increases with the ratio of the step-size used and the optimal one giving fastest convergence. Therefore, if the discrepancy is too small, the step-size is being increased. If it is too large, it is decreased. If it is radically too large, then instability of the process is detected, the step-size is radically decreased, and $\theta$ moves back to the point where the part began.

This last principle is illustrated in Fig. 1. For a very small $\beta$, the vector $\theta_{t+i}$ moves through a flat slope of the function $J$. This results in small differences between $g(\theta_t, \xi_{t+i})$ and $g(\theta_{t+i}, \xi_{t+i})$, thereby, making the discrepancy between $G_{t,n}$ and $G^*_{t,n}$ small as well. The speed of the local minimum search can be increased with larger values of $\beta$. However, if $\beta$ is too large, $\theta_{t+i}$ jumps over the minimum of the function to be optimized. This creates large differences between $g(\theta_t, \xi_{t+i})$ and $g(\theta_{t+i}, \xi_{t+i})$ manifested by large discrepancy between $G_{t,n}$ and $G^*_{t,n}$.

The papers (Wawrzyński, 2010; Wawrzyński & Papis, 2011) present a statistical analysis to formalize the above intuitive principles in a simple, unidimensional model. Since each $g(\theta_t, \xi_{t+i})$ in (12) is an unbiased estimate of $\nabla J(\theta_t)$, the value $\frac{1}{n}G^*_{t,n}$ is also an unbiased estimator of $\nabla J(\theta_t)$ and its quality increases with $n$. The model considers $n$ that ensures signal-to-noise ratio of 1 in $\frac{1}{n}G^*_{t,n}$, namely

$$n = \frac{E_{\theta_t} \|g(\theta_t, \xi)\|^2}{\|\nabla J(\theta_t)\|^2}. \tag{13}$$

For $n$ of that size, and constant $\alpha_0 \cong 0.15$ the difference

$$\alpha_0 \|G^*_{t,n}\|^2 - \|G^*_{t,n} - G_{t,n}\|^2 \tag{14}$$

is on average equal to 0 if the step-size is optimal for fast convergence of (10). If the step-size is smaller than optimal, the

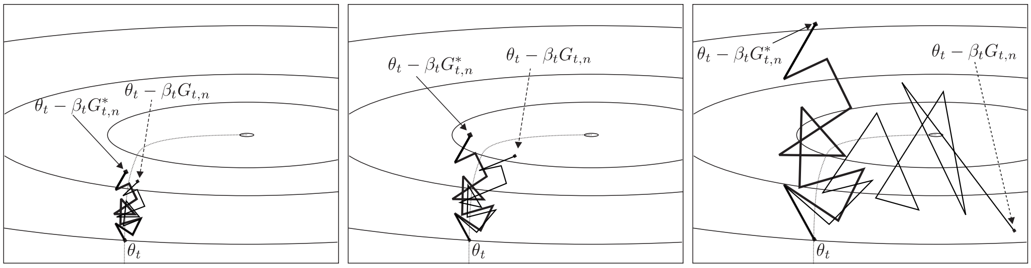

$\theta_t - \beta_t G_{t,n}$
$\theta_t - \beta_t G^*_{t,n}$
$\theta_t$
$\theta_t - \beta_t G^*_{t,n}$
$\theta_t - \beta_t G_{t,n}$
$\theta_t$
$\theta_t - \beta_t G^*_{t,n}$
$\theta_t - \beta_t G_{t,n}$
$\theta_t$

**Fig. 1.** The sketches depict contour lines of three different functions $J$ with equal $\nabla J(\theta_t)$, $\beta$, and $g(\theta_t, \xi_{t+i})$ for $i \geq 0$. In $n$ steps starting from $t$, the parameter in the $J$ domain evolves from $\theta_t$ to $\theta_{t+n} = \theta_t - \beta_t G_{t,n}$. *Left:* Small curvature of $J$ in relation to $\beta$, causes small difference of $-\beta_t G_{t,n}$ and $-\beta_t G^*_{t,n}$. *Middle:* Moderate curvature of $J$ causes moderate difference between $-\beta_t G_{t,n}$ and $-\beta_t G^*_{t,n}$. *Right:* Very large curvature of $J$ relatively to $\beta$ causes instability of the optimization process which is manifested by divergence of $-\beta_t G_{t,n}$ and $-\beta_t G^*_{t,n}$ that grows with $n$.

first term of (14) is larger than the latter, and for too large step-size, the relation is opposite. Therefore, the difference (14) on average indicates the direction in which the step-size should be modified, i.e., increased or decreased. In other words, the step-size is stabilized at the best level when on average the discrepancy $\|G^*_{t,n} - G_{t,n}\|^2$ is stabilized at the value of $\alpha_0 \|G^*_{t,n}\|^2$. Also, if the term (14) with larger $\alpha_0$ is applied to adjust the step-size, then the step-size is being stabilized at a higher level than one that is best for the purpose of static optimization (10). This last property will become useful in the next section.

In order to apply the same principle to multidimensional $\theta$, we will consider only the direction of gradient estimate, that is we will project $G_{t,n}$ on $G^*_{t,n}$ and consider the discrepancy between $G^*_{t,n}$ and the projection. The indicator of step-size adjustment (14) will thus take the form

$$\alpha_0 \|G^*_{t,n}\|^2 - \|G^*_{t,n} - proj(G_{t,n}, G^*_{t,n})\|^2, \quad (15)$$

for

$$proj(G, G^*) = G^* G^T G^* / (\|G^*\|^2 + \epsilon) \quad (16)$$

and $\epsilon$ being a small number that prevents division by zero.

The discussion above leads to Algorithm 3 for autonomously setting the step-sizes. It divides the learning process into parts or *estimation periods* in which the step-size $\beta$ remains fixed. In order to compute $n$ according to (13) the fraction $(L/M)$ estimates

$$E_{\theta_t} \|g(\theta_t, \xi)\|^2$$

such that $L$ adds $\|g(\theta_t, \xi_{t+i})\|^2$ and $M$ counts them. The fraction $(L'/M')$ estimates $\|\nabla J(\theta_t)\|^2$ on the basis of the following property. Let $\xi$ and $\xi'$ be sampled independently from the same distribution. Then we have

$$\begin{aligned} E_{\theta_t} g(\theta_t, \xi)^T g(\theta_t, \xi') &= E_{\theta_t} g(\theta_t, \xi)^T E_{\theta_t} g(\theta_t, \xi') \\ &= \|\nabla J(\theta_t)\|^2. \end{aligned}$$

The estimator of $\|\nabla J(\theta_t)\|^2$ can thus take the form of the statistics

$$\frac{\sum_{i=1}^{n} \sum_{i'=0}^{i-1} g(\theta_t, \xi_{t+i})^T g(\theta_t, \xi_{t+i'})}{\sum_{i=1}^{n} i} = \frac{\sum_{i=1}^{n} g(\theta_t, \xi_{t+i})^T G^*_{t,i}}{\sum_{i=1}^{n} i}, \quad (17)$$

whose numerator and denominator are denoted by $L'$ and $M'$, respectively.

We now explain the algorithm in a step-wise manner for better understanding. In Line 3, it receives a new training example. In Lines 6–8, it initializes variables of the fixed point algorithm. In Lines 9–12, it updates estimators $L/M$ and $L'/M'$. In Lines 13–16 it updates the values $G_{t,n}$, $\theta_{t+n}$, $G^*_{t,n}$, and

$$h = \max_{i=1,\dots,n} \|G^*_{t,i}\|^2,$$

respectively.

In Line 19, the algorithm checks whether the learning process is stable. Its instability is manifested by $\|G_{t,n} - G^*_{t,n}\|^2$ being larger than $h = \max_{1 \leq i \leq n} \|G^*_{t,i}\|^2$. This condition is additionally tightened in the $k_0$ initial periods. If instability is detected, the estimation period is reinitialized with halved step-size.

In Lines 21–22, the algorithm checks if the present estimation period has come to its end. It happens when

$$n \geq \frac{L/M}{L'/M'},$$

that is when the estimator of $Eg(\theta_t, \xi)$ has appropriate quality (13). The period is also finished when

$$n \geq \frac{3}{2} n',$$

where $n'$ is the duration of the previous period. This prevents the algorithm from getting stuck in local minima ($Eg(\theta, \xi) = 0$ in a minimum and appropriate duration of an estimation period becomes infinite).

In the end of an estimation period, the step-size is modified (Line 25) and a new estimation period is started. Namely, the step-size is incremented proportionally to the statistics (15). The increment is scaled with the variable $\bar{h}$, that is determined in Line 26 as the maximal value of previous $\|G^*_{t,n}\|^2/(1 - \alpha_1)$. Line 24 prevents very small values of the scaling factor $\bar{h}$, for instance in the first period.

Every time a new estimation period begins, the estimators $(L/M)$ and $(L'/M')$ become obsolete and the amount of information they carry is decreased. In Lines 27–28, the algorithm decreases proportionally the weight of previous samples in comparison to forthcoming ones.

The algorithm updates the step-size after each estimation period to optimize the process (10). One such period is defined by the number of steps sufficient to estimate the direction of movement of $\theta$ with unity information-noise ratio. While various processes may differ substantially, the algorithm divides them to similar sequences of periods. Coefficients of the algorithm are chosen to make these sequences converge fast for all underlying processes (10) at its all stages.

In comparison to its version presented in Wawrzyński (2010), the algorithm is improved in two directions. First, it estimates $\|Eg(\theta_t, \xi)\|^2$ better by means of the statistics (17). Second, after each of first $k_0$ estimation periods it moves the process back to its origin; at this time only data is collected to estimate $\|Eg(\theta_0, \xi)\|^2$ and $E\|g(\theta_0, \xi)\|^2$. The goal of these modifications is to make the algorithm exploit data better and be robust to too large initial step-size.

## 5. Actor–critic with experience replay and the fixed point method of step-size estimation

This section discuses combination of the actor–critic algorithm with experience replay (ac&er) and the fixed point method of

Algorithm 3. The fixed point method of step-size estimation. The coefficients applied: $\alpha_1 = 0.9$, $\alpha_2 = 0.5$, $k_0 = 10$, initial $\beta = 1$.

```
FIXED_POINT
FP_INITIALIZE($\theta_0$)
Repeat
    Get $\xi_{t+n}$
    FP_LOOP$\big(g(\theta_t, \xi_{t+n}), g(\theta_{t+n}, \xi_{t+n})\big)$
Until some-stopping-condition

FP_INITIALIZE($\theta_0$)
Assign $t = n := 0, k := 1, L = L' = M = M' := 0$
Assign $G_{0,0} = G^*_{0,0} := 0, h = \bar{h} := 0, n' = +\infty$
Initialize $\theta_0$, and $\beta$

FP_LOOP$\big(g(\theta_t, \xi_{t+n}), g(\theta_{t+n}, \xi_{t+n})\big)$
$L := L + \|g(\theta_t, \xi_{t+n})\|^2$
$M := M + 1$
$L' := \max\{L' + g(\theta_t, \xi_{t+n})^T G^*_{t,n}, 0\}$
$M' := M' + n$
$G_{t,n+1} := G_{t,n} + g(\theta_{t+n}, \xi_{t+n})$
$\theta_{t+n+1} := \theta_{t+n} - \beta g(\theta_{t+n}, \xi_{t+n})$
$G^*_{t,n+1} := G^*_{t,n} + g(\theta_t, \xi_{t+n})$
$h := \max\{h, \|G^*_{t,n+1}\|^2\}$
$n := n + 1$
$newperiod := false$
If $\|G_{t,n} - G^*_{t,n}\|^2 > h \vee (k < k_0 \wedge \|G_{t,n} - G^*_{t,n}\|^2 > 0.3h)$
then
    $\beta := \beta/2$, $newperiod := true$
If $\neg newperiod \;\wedge\; n \geq 3 \;\wedge\; L' > 0$
        $\wedge\, n \geq \min\{(L/M)/(L'/M'), \frac{3}{2}n'\}$ then begin
    If $k \geq k_0$ then begin
        $\bar{h} := \max\{\bar{h}, \alpha_0\|G^*_{t,n}\|^2, \|proj(G_{t,n}, G^*_{t,n}) - G^*_{t,n}\|^2\}$
        $\beta := \beta \exp((\alpha_0\|G^*_{t,n}\|^2 - \|proj(G_{t,n}, G^*_{t,n})$
                $- G^*_{t,n}\|^2)/\bar{h})$
        $\bar{h} := \max\{\alpha_1\bar{h}, (L/M)n/(1 - \alpha_1)\}$
        $L := \alpha_1 L, M := \alpha_1 M$
        $L' := \alpha_2 L', M' := \alpha_2 M'$
        $n' := n$
    end
    $t := t + n$
    $k := k + 1$
    $newperiod := true$
end
If $newperiod$ then
    $n := 0, G_{t,0} = G^*_{t,0} := 0, h := h/2$
```

step-size adaptation. The original ac&er repeatedly selects data samples and computes the improvement direction estimators $\widehat{\phi}^r_i$ and $\widehat{\psi}^r_i$ to adjust the parameters of the actor and critic respectively. The combined algorithm optimizes the actor and critic parameters with separate modules that realize the fixed point method of step-size estimation (separate objects, as we would say in terms of object-oriented programming). Below, the generic terms of Section 4 are translated to the components of the ac&er algorithm. For the actor step-size optimization:

- $\xi$ translates into the sequence of data that is required to compute $\widehat{\phi}^r_i$.
- the vector that undergoes optimization is the actor parameter, $\theta$,
- its fixed value will be denoted by $\theta^*$,
- $g(\theta, \xi)$ represents $-\widehat{\phi}^r_i(\theta, \upsilon)$; the minus results from the fact, that originally the fixed point method was meant for minimization while here the direction of optimization is opposite,
- $g(\theta^*, \xi)$ represents $-\widehat{\phi}^r_i(\theta^*, \upsilon)$.

For the critic step-size estimation we have:

- as previously, $\xi$ translates into the sequence of data that is required to compute $\widehat{\psi}^r_i$.
- the vector being optimized is now the critic parameter, $\upsilon$,
- its fixed value will be denoted by $\upsilon^*$,
- $g(\theta, \xi)$ represents $-\widehat{\psi}^r_i(\theta, \upsilon)$; notice that now $\theta$ in the first term corresponds to $\upsilon$ in the second while $\theta$ in the second is only one of the entities that define this random vector,
- $g(\theta^*, \xi)$ represents $-\widehat{\psi}^r_i(\theta, \upsilon^*)$.

The above explanation suffices for straightforward combination of ac&er and the fixed point method of Section 4. This combination may be further enhanced by addressing the issues of overtraining, wrong fixed point initialization, and mutual adjustments of the actor and critic.

*Overtraining.* In general, a model is overtrained when it fits the data very well, but because the data is not sufficiently representative, the model does not describe the actual system that has generated the data. In the case of ac&er, the actor and critic may become overtrained when they are optimized on the basis of a short history of experiences. For instance, suppose the experience contains only several state–action pairs among which there are pairs with the same state, and the ac&er has enough resources for full optimization of the policy on the basis of these pairs. Then, it would readily make the better action in the same state infinitely more probable than the worse one. Consequently, the resulting policy would not try the worse actions at all which could lead to inability to explore certain area of the action space.

A way to overcome the above difficulty is to introduce regularization. Let

$$l : \mathcal{S} \times \mathfrak{R}^{n_\theta} \mapsto \mathfrak{R}_+$$

be a loss function that is continuous and assigns, for each state, zero to a bounded set of policy vectors that ensure exploration in a state, and an increasing positive score ($\in \mathfrak{R}_+$) to the policy vectors with growing distance from this set. Then, the policy vector adjustment takes the form

$$\theta := \theta + \beta^\theta \left(\widehat{\phi}^r_i(\theta, \upsilon) - \nabla l_\theta(s_i, \theta)\right).$$

As a result, even if the unrepresentative data pushes the policy vector outside the feasible exploration region, the loss function pulls it back.

*Wrong fixed point initialization.* In the initial period of the fixed point method, the algorithm only collects data and estimates $\|Eg(\theta, \xi)\|^2$ and $E\|g(\theta, \xi)\|^2$. It is assumed then that the $g$ vectors are independently sampled from the appropriate distribution. If there are very few samples in the database, then this assumption is obviously violated and the estimator of $E\|g(\theta, \xi)\|^2$ becomes positively biased. As a result, the length of the estimation periods gets smaller. In order to stabilize the discrepancy between $G$ and $G^*$ for too short estimation periods, the step-sizes are being increased. This may lead to oscillations in the early stages of learning process.

A simple way to overcome this problem is to delay learning until a certain number of samples are collected in the database. To this end, we use the $\nu$ function to regulate the intensity of replaying experience, of the form

$$\nu(t) = \begin{cases} 0 & \iff t \leq min\_capacity \\ comp\_steps & \iff t > min\_capacity \end{cases} \tag{18}$$

for $min\_capacity > comp\_steps > 0$, i.e., $\upsilon(t)$ is a piecewise constant function that triggers the intensity of replaying experience given by *comp_steps* only after a certain time has elapsed denoted

Algorithm 4. Actor–Critic with experience replay and step-size estimation based on the fixed point method.

```
1:   $t := 1$, Initialize $\theta$ and $\upsilon$
2:   Actor.FP_INITIALIZE($\theta_0$)
3:   Critic.FP_INITIALIZE($\upsilon_0$)
4:   Draw and execute an action, $a_t \sim \pi(\cdot; s_t, \theta)$
5:   Assign $\pi_t = \pi(a_t; s_t, \theta)$
6:   Repeat $\nu(t)$ times, begin
7:      Draw $i \in \{t-N, t-N+2, \dots, t-1\}$
8:      Adjust $\theta$ along an estimator of $\phi(s_i, \theta, \upsilon)$:
9:         Actor.FP_LOOP$\big(-\widehat{\phi}_i^r(\theta^*, \upsilon) + \nabla_{\theta^*} l(s_i, \theta^*),$
             $-\widehat{\phi}_i^r(\theta, \upsilon) + \nabla_\theta l(s_i, \theta)\big)$
10:     Adjust $\upsilon$ along an estimator of $\psi(s_i, \theta, \upsilon)$:
11:        Critic.FP_LOOP$\big(-\widehat{\psi}_i^r(\theta, \upsilon^*), -\widehat{\psi}_i^r(\theta, \upsilon)\big)$
12:  End
13:  Register the tuple $\langle s_t, a_t, \pi_t, r_t, s_{t+1}\rangle$ in the database
14:  Make sure only $N$ most recent tuples remain in the
     database
15:  Assign $t := t + 1$ and repeat from Line 4
```

by *min_capacity*. While *comp_steps* is defined by available computation power, *min_capacity* should be set to exclude the use of too large computational power to optimize the actor and critic vectors on the basis of too few data samples.

*Mutual adjustments of the actor and critic.* The fixed point method of step-size estimation is originally designed as a tool for stochastic optimization. Actor–critic with experience replay may be understood as an algorithm of that type in which the actor parameter, $\theta$, is optimized to adapt it to the current critic, and the critic parameter, $\upsilon$, is optimized to adapt it to the current actor. In both the cases, the ultimate goal of adaptation is constantly changing or running away. Consequently, the step-sizes should be estimated taking into account larger distance to the goal than currently perceived. The discrepancy between $G$ and $G^*$ in the fixed point method should be thus stabilized at a relatively higher level in ac&er as compared to stationary stochastic optimization. It is also inherent in Actor–Critics that the critic should adapt to the actor faster than *vice versa* (see e.g., Bhatnagar et al., 2009; Konda & Tsitsiklis, 2003).

The above remarks translate into the following guideline for the use of the fixed point method in ac&er: the discrepancies between $G$ and $G^*$, regulated by the $\alpha_0$ parameter, should be stabilized at relatively higher level for the critic than for the actor, and in both cases should be larger than optimal for stationary stochastic optimization.

The combined algorithm is presented in Algorithm 4. It defines the agent–environment interaction and the improvement directions for the actor and the critic. Their step-sizes and parameters are adjusted with the two fixed-point modules.

## 6. Experimental study

In the previous section, a reinforcement learning algorithm is introduced that combines experience replay with autonomous step-size estimation. In this section, it is confronted with two simulated control problems in multidimensional continuous spaces. The problems are selected to have general characteristics and the level of complexity corresponding to real-life robotic applications. The tasks analyzed are: (1) point reaching movement of octopus arm, and (2) cyclic running motion of half-cheetah.

### 6.1. Octopus arm

Octopus is well-known for exhibiting a high level of flexibility in controlling arm movements. The highly developed limbs of octopus make it capable of bending, stretching, shortening and twisting its arm at any point and in any direction with virtually unlimited degrees of freedom. Artificial octopus arm has found its niche in continuum robotics to replace conventional serial manipulators with smooth, continuous and flexible links (McMahan et al., 2006). However, their use is severely limited by existing motion planning and control algorithms. Classical approaches of controlling redundant manipulators by applying functional constraints to the additional degrees of freedom with null-space projection do not scale to continuum robots due to added computational complexity (Chiaverini, Oriolo, & Walker, 2008). While the theoretical foundation of continuum kinematics is in its nascent stages, learning based approaches provide an interesting alternative to control such robots. To this end, Engel, Szabó, and Volkinshtein (2005) used the Gaussian process temporal difference based reinforcement learning algorithm to successfully demonstrate various learning tasks of a simulated octopus arm model. Because the original state and action space of an octopus arm has huge number of dimensions, it often serves as a test bed for solutions to the problem of dimensionality in RL (Koutnik, Gomez, & Schmidhuber, 2010; Woolley & Stanley, 2010). Here the proposed reinforcement learning algorithm is tested for solving the task of reaching a given goal by an octopus arm; the state and action spaces are reduced but still rich enough to represent current position and dynamics of the arm.

#### 6.1.1. Control problem

We are interested in learning a reactive control policy for the octopus arm to reach an arbitrary goal point starting from some random position. The octopus arm is represented by a planar simulator model composed of 10 quadrilateral compartments with each segment containing a longitudinal or a transverse muscle, as described by Yekutieli et al. (2005). The mass of each compartment is distributed in its four corners as point masses which are connected by linear damped springs representing muscles. Each compartment follows the muscular hydrostat principle of maintaining a constant volume (area in this case) assuming the muscles to be incompressible. The activation pattern in muscles acts as input to the model and the resulting position of point masses is obtained on the basis of the net force acting on the arm. The forces modeled in the simulation are of four types: (1) internal force generated by muscles, (2) vertical force due to gravity and buoyancy of arm in seawater, (3) external force produced by water drag, and (4) constraint force induced by change in pressure to keep the area of each compartment constant. Moreover, the base is non-rotating in the model, i.e., the first spring is fixed to the ground. The planar dynamic model of the octopus arm provides a trade-off between its fidelity and real-time processing overheads. The model is simulated with the use of software made available for 2006 RL competition (Octopus-sources, 2006) originally written in Java and rewritten for the experiments presented here to C#. The simulator worked in its standard setting with 10 compartments, fixed base, each muscle 1 unit long, and action duration 0.1 s.

We now define the state and action spaces along with the associated reward function to formulate the control problem of octopus arm as MDP for reinforcement learning.

*State space.* In this study, we define three moving frames in polar coordinates that are used to localize the octopus arm movement towards its goal. The first frame $(r_1, \phi_1)$ is located at the center of the point masses of all the quadrilateral compartments, the second one $(r_2, \phi_2)$ is located at the center of the point masses of last 5 quadrilateral compartments, and the third one $(r_3, \phi_3)$ is located at the center of mass of the last compartment. The first two frames are movable with respect to the arm as they may or may not be located inside the model depending upon its configuration (see Fig. 2).

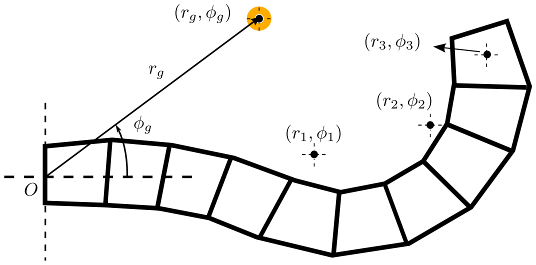


**Fig. 2.** Simulated octopus arm model with geometric description.

**Table 1**
State variables of octopus arm.

| | | |
|---|---|---|
| $s_0 = (r_g - 10)/3$ | $s_4 = \dot{r}_1/0.01$ | $s_8 = (r_3 - r_g)/3$ |
| $s_1 = \phi_g$ | $s_5 = \dot{\phi}_1/0.01$ | $s_9 = \phi_3 - \phi_g$ |
| $s_2 = (r_1 - r_g)/3$ | $s_6 = (r_2 - r_g)/3$ | $s_{10} = \text{sign}(r_3 - r_g)$ |
| $s_3 = \phi_1 - \phi_g$ | $s_7 = \phi_2 - \phi_g$ | $s_{11} = \text{sign}(\phi_3 - \phi_g)$ |

Based on the three frames and the goal frame $(r_g, \phi_g)$, the state space of our model consists of 12 state variables given in Table 1. Each state variable is normalized to roughly cover the interval $[-1, 1]$. Majority of the variables define the position of three frames with reference to the goal frame. Additionally, variables $s_4$ and $s_5$ express velocity of the center of mass, and discrete variables $s_{10}$ and $s_{11}$ describe whether the last compartment is 'in front of' or 'behind' the target. While these variables are redundant in defining arm configuration, they are quite useful inputs to the control policy.

*Action space.* The action space consists of a set of 6 actions each of which pre-defines the activation level in the arm's muscles, as used in Engel et al. (2005). The action space is shown in Fig. 3.

*Episodes and reward function.* The learning task is carried out in episodes. An episode terminates with success when the last compartment touches the goal (which is transparent for other compartments). If this goal is not reached within 500 steps, the episode terminates. A new target position is sampled in the beginning of every episode within the area described by a circular region:

$$r_g \in (7.5, 9.5)$$
$$\phi_g \in (-\pi/4, \pi/4).$$

The position of the arm is initialized such that it is set in its default position, an action is chosen randomly from the set {Action 1, Action 2}, and it is applied to the arm for a number of steps chosen randomly from the set $\{1, \dots, 100\}$.

The reward function is defined such that the controller is penalized with a negative score of $-1$ for all the learning steps in which the goal has not been reached. Moreover, the arm is rewarded for moving towards the goal in proportion to its speed. The reward function is given by:

$$r_t = \begin{cases} d_t - d_{t+1} - 1 & \text{if the target is not reached at } t+1 \\ d_t & \text{if the target is reached at } t+1. \end{cases} \qquad (19)$$

The above term $d_t$ denotes Euclidean distance between the tip of the arm and the goal position measured at instant $t$. The goal of the octopus arm is to maximize the reward function by reaching the goal position as quickly as possible. The total reward to be obtained within an episode is therefore equal to the difference between the initial distance of the last compartment to the goal and the number of steps in which the goal is reached.

### 6.1.2. Actor and critic structure

The actor and critic are based on feedforward neural networks, namely 2-layer perceptrons with sigmoidal neurons in their hidden layers and linear neurons in their output layers. The input of both the networks is the scaled state of the octopus arm. All neurons in the networks have a constant input (bias). The initial weights in the hidden layers are drawn randomly from the normal distribution $N(0, 1)$ and weights of the output layers are initially set to zero.

The actor is the combination of a neural network and a generator of discrete numbers. The network has $N_A$ neurons in its hidden layer and 6 neurons in the output layer corresponding to the size of the discrete action set. Given state $s$ and the observed neural network outputs

$$\mu_i(s, \theta), \quad i = 1 \dots 6,$$

the likelihood function for choosing a discrete action, Action $i$, is given by:

$$P(\text{Action } i|s) = \frac{\exp(\mu_i(s,\theta)/10)}{\sum_{j=1}^{6} \exp(\mu_j(s,\theta)/10)}. \qquad (20)$$

The exploration takes place as each action has a non-zero probability. This is ensured by the penalty function of the form

$$l(s, \theta) = \sum_{i=1}^{6} \max\{0, |\mu_i(s, \theta)| - 20\}^2.$$

As a result, the policy is not penalized for the network's outputs in the range $[-20, 20]$, which means that the best action is not more than $e^4 \approx 54$ times more probable than the worst one.

The critic is a 2-layer perceptron with $N_C$ neurons in its hidden layer and one neuron in the output layer.

### 6.1.3. Experiments and results

The experiments of the learning control problem of an octopus arm with the proposed algorithm are now presented. The parameter setting of the actor–critic reinforcement learning algorithm is as follows: $N_A = 50, N_C = 100, \gamma = 0.98, b = 2, \alpha = \gamma = 0.98$, and $\rho = \lambda = 0.8$. The experience database contained $M = 10^4$ events. The $\nu$ function took the form (18) with *min_capacity* $= 1000$ and *comp_steps* $= 100$. The fixed point algorithm parameters were: $\alpha_1 = 0.9, \alpha_2 = 0.5, \alpha_0 = 0.2$ for the actor and $\alpha_0 = 0.3$ for the critic.

Fig. 4 shows different stages of the octopus arm in reaching the desired goal. Figs. 5–9 present the result of experiments performed with the above parameter setting. The actor–critic with experience replay and various constant step-sizes is compared against those estimated by the fixed point method. Fig. 5 presents learning curves (average rewards vs. episode number), Fig. 6 presents efficiency (ratio of runs in which the goal was reached), Figs. 7 and 8 present average duration of episodes, and Fig. 9 presents the step-sizes estimated by the fixed point method.

What is not demonstrated in the figures is that the learning process is unstable for constant step-sizes greater or equal to 0.1 for both the actor and the critic. For $\beta^\theta = \beta^\upsilon = 10^{-2}$, the algorithm does not converge and produces poor performance. Smaller step-sizes lead to better performance but for $\beta^\theta \leq 10^{-4}$ or $\beta^\upsilon \leq 10^{-4}$ the learning process becomes very slow. The fixed point method of adjusting step-sizes enables at least as good performance of the learning algorithm, at any stage of its operation, as given by any constant step-size. The training is completed (no further improvement is observed) after 240 episodes which is equivalent to 80 min of Octopus time.

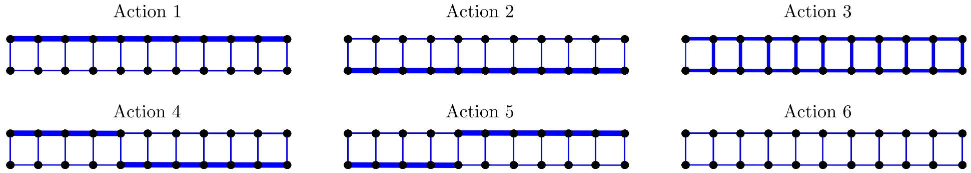


**Fig. 3.** Action set. The activated muscles are indicated by thick blue lines. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

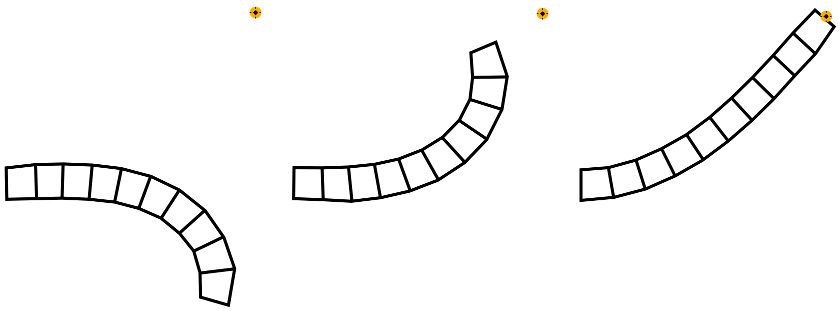


**Fig. 4.** Simulated goal reaching movement of octopus arm. *Left:* Arm position in the start of episode. *Middle:* Arm position trying to reach the goal. *Right:* Arm position when reached the goal. After learning, the arm reaches the goal in about 100 steps with maximum efficiency.

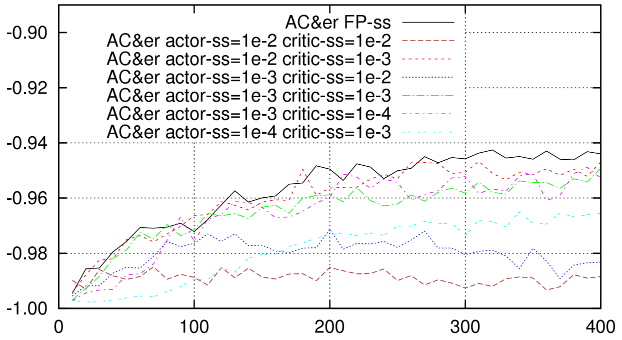


**Fig. 5.** *Octopus arm:* Average rewards vs. episode no. for actor–critic with experience replay and constant step-sizes against those estimated by the fixed point method. Each curve averages 10 runs.

Fig. 9 shows the evolution of step-sizes driven by the fixed point method. They are averaged over runs and episodes according to the formula

$$\bar{\beta} = \exp\left(\frac{1}{N}\sum_{i}^{N}\ln\beta_i\right), \tag{21}$$

where $N$ is the number of registered step-sizes. This way of averaging is better here than the normal one because it does not underweight very small step-sizes. It is seen that the actor step-size starts from a large initial value and then gradually decreases. The critic step-size initially decreases to a rather small value and then, after about 100 episodes, almost remains the same. This suggests that slowing down of the actor learning does not necessarily corresponds to slowing down of the critic learning.

### 6.2. Half-cheetah

Running in biological systems has largely inspired the development of legged robotics. Starting from the seminal spring-loaded hopping robots to the state-of-the-art quadrupeds and bipeds with compliant legs, a number of prototypes have been proposed to mimic versatility, speed and robustness in natural running. The dramatic progress is hindered by the lack of ability in robots to acquire these skills by online learning. In this section, we apply our proposed algorithm on a planar model of a large cat, called half-cheetah, to learn the complex skill of running.

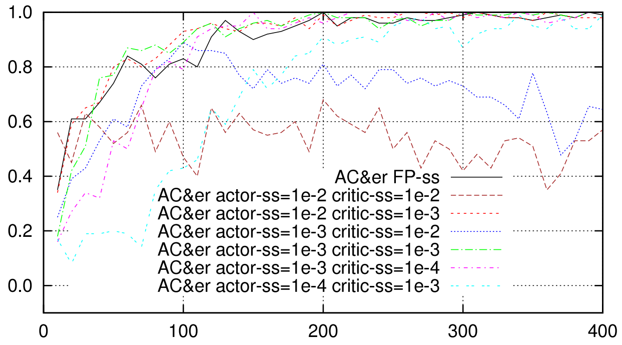


**Fig. 6.** *Octopus arm:* Efficiency vs. episode no. for actor–critic with experience replay, step-sizes estimated by the fixed point method, and constant step-sizes. Each curve averages 10 runs.

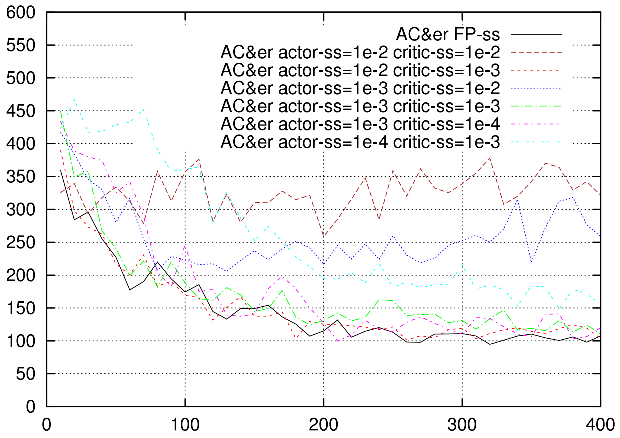


**Fig. 7.** *Octopus arm:* Duration in steps vs. episode no. for actor–critic with experience replay, step-sizes estimated by the fixed point method, and constant step-sizes. Each curve averages 10 runs.

#### 6.2.1. Control problem

Half-Cheetah is a 6 degrees of freedom planar robot introduced in Wawrzyński (2009). It is composed of nine links, eight joints and two paws (see Fig. 10). The angles of the fourth and the fifth joint are fixed while others are controllable. The torque applied at each

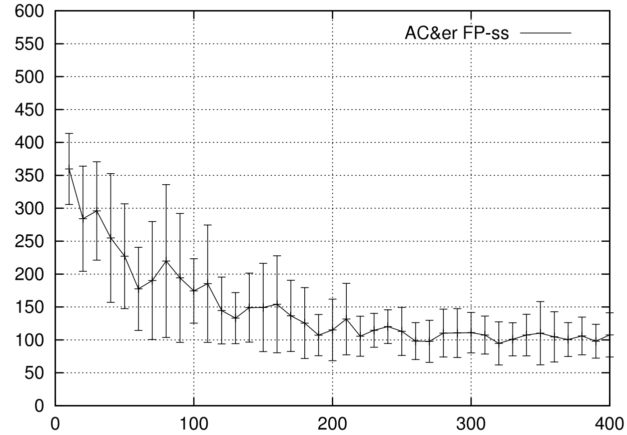


**Fig. 8.** *Octopus arm:* Duration in steps vs. episode no. for actor–critic with experience replay and step-sizes estimated by the fixed point method. The one-sigma limits are calculated to show run-to-run variability of trial averages .

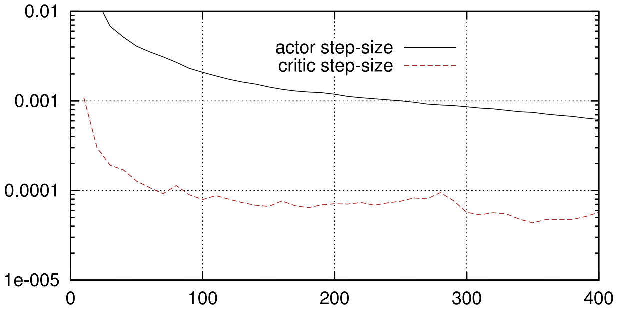


**Fig. 9.** *Octopus arm:* The fixed point method of step-size estimation vs. episode no. for actor–critic with experience replay. Each curve averages 10 runs.

joint acts as input to the model and the next position of the robot is obtained as output by integrating its dynamic equations of motion.

The control problem is to learn a reactive policy under the MDP framework to make half-cheetah run as fast as possible. State of half-cheetah is defined by 31 variables. The action space is continuous, contrary to that of octopus arm, with 6 dimensions each corresponding to one actuated joint independently. Learning is divided into episodes with an average duration of 250 steps, for 0.02 s duration of each step. The robot is mainly rewarded for its speed of moving forward. Other components are minor penalties for violating torque limits, joint limits, not moving the trunk in idle position and touching the ground with other body parts than paws. In this study, we use the simulator of half-cheetah, as reported in Wawrzyński (2009), with same experimental settings including its dynamics, state and action definition, and reward function.

### 6.2.2. Actor and critic structure

The actor, $\pi$, is composed of two parts: a neural network and a normal distribution. The input of the network is the state of half-cheetah. The network has a hidden layer with $N_A$ sigmoidal neurons and six linear output neurons corresponding to the dimensionality of the action space. The output, $\mu(s;\theta)$, becomes a mean value of the normal distribution with covariance matrix equal to $I\sigma^2$. The distribution generates actions. Because each component of the action is projected on the interval $[-30, 30]$, the penalty function keeps each network output in this interval for a given state. It is of the form

$$l(s,\theta) = \sum_{i=1}^{6} 10^{-3} \max\{0, |\mu_i(s,\theta)| - 30\}^2.$$

The critic is a neural network of the same structure as the actor network with $N_C$ neurons in its hidden layer. The initial weights in the hidden layers are drawn randomly from the normal distribution $N(0, 1)$ and weights of the output layers are initially set to zero.

### 6.2.3. Experiments and results

The experiments to make half-cheetah run are configured with following parameters: $N_A = 160, N_C = 160, \sigma = 5, \gamma = 0.99, b = 2, \alpha = \gamma = 0.99, \rho = \lambda = 0.9$. The experience database contained $M = 10^5$ events. The $\nu$ function took the form (18) with *min_capacity* $= 100$ and *comp_steps* $= 30$. The fixed point algorithm parameters were kept same as for octopus arm with: $\alpha_1 = 0.9, \alpha_2 = 0.5, \alpha_0 = 0.2$ for the actor and $\alpha_0 = 0.3$ for the critic.

Fig. 10 shows various stages of the learned running gait of half-cheetah. The cat robot starts from a standing position and first learns to move forward with the added noise in the control system. The awkward walk transforms into a trot gait which at the end of training becomes a smooth nimble run. The use of experience replay speeds up the learning process of running in proportion to the intensity of replaying computations.

In this study, we are interested in analyzing the effect of the proposed fixed-point method of step-size estimation on the learned policy. To this end, Figs. 11 and 12 compare the learning curve of the adaptive step-size estimation algorithm with the ones obtained by setting various constant step-sizes. Among the learning curves with constant step-sizes, the learning algorithm produces optimum results for $\beta^\theta = \beta^\upsilon = 10^{-5}$. For larger values, the policy evolves with a reasonable speed in the initial period but the ultimate performance is worse. Smaller step-sizes slow down the learning speed significantly. The fixed point method of estimating the step-sizes makes the learning algorithm perform better at every stage of the learning process than with any manually selected step-sizes. The algorithm required 3500 episodes (about 5 h of half-cheetah time) to learn to receive average reward of 6, which corresponds to a really nimble run.

Fig. 13 presents evolution of step-sizes driven by the fixed point method. They are averaged over runs and episodes according to formula (21). It is seen that the actor and critic step-sizes start descending from the same value and then gradually converge to about $5.10^{-6}$ and $10^{-6}$ respectively.

## 6.3. Discussion

The purpose of this experimental study is to evaluate the feasibility of autonomously determining the step-sizes for actor–critic class of reinforcement learning algorithms with experience replay. The results are promising: the augmented algorithm has been used to find control policies for two challenging problems similar to those encountered in robotics. The algorithm gave at least as good performance at any stage of the learning process as Actor–Critic with experience replay and constant step-sizes.

In all the experiments, the step-sizes of the fixed point method were initialized blindly, with a value of 1.0 (Algorithm 3, Line 8). As experiments with constant step-sizes showed, the value of 1.0 was very large and surely caused instability of the learning, which was identified by the algorithm (Line 19), and the step-sizes were halved (Line 20) until they ensured normal learning. Therefore, despite their large initial value, improper for the problems at hand, the step-sizes are always smaller than 1.0 in Figs. 9 and 13. Further, the step-sizes were autonomously adjusted with time (Algorithm 3, Line 25), which usually (but not always) meant their slow decrease.

The fixed point method of estimating step-sizes gave at least as good performance as the manually defined constant step-sizes used in the experiments. We argue that constant step-sizes generally do not ensure good performance as even their

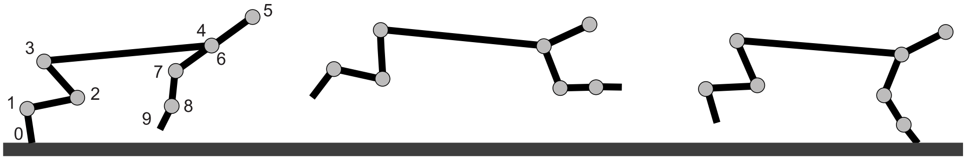


**Fig. 10.** Simulated run of Half-Cheetah. *Left:* Initial stance. *Middle:* Middle stance with feet in the air. *Right:* Landing stance. After learning, half-cheetah runs with a speed of about 6.5 m/s.

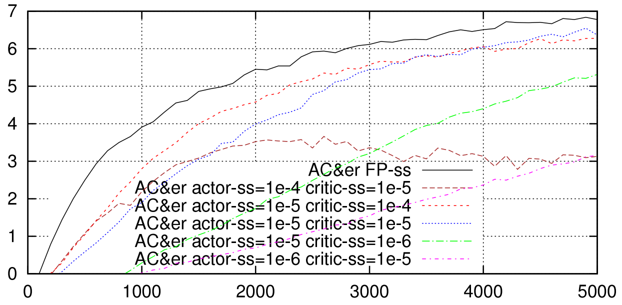


**Fig. 11.** *Half-Cheetah:* Average rewards vs. episode no. for actor–critic with experience replay and constant step-sizes against those estimated by the fixed point method. Each curve averages 10 runs .

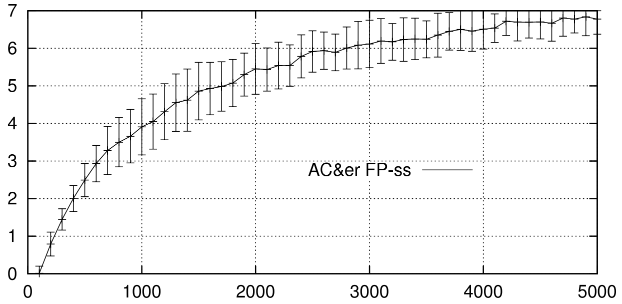


**Fig. 12.** *Half-Cheetah:* Average rewards vs. episode no. for actor–critic with experience replay and step-sizes estimated by the fixed point method. The one-sigma limits are calculated to show run-to-run variability of trial averages .

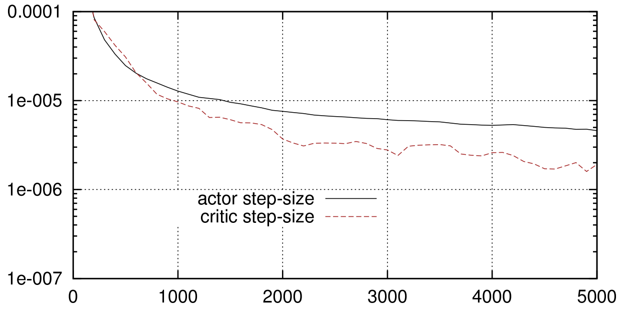


**Fig. 13.** *Half-Cheetah:* The fixed point method of step-size estimation vs. episode no. for actor–critic with experience replay. Each curve averages 10 runs.

handcrafted values yield slow learning in the early stages and fluctuations in the final stages. The fixed point method estimates large step-sizes initially and gradually decreases them during the course of learning. However, it has been observed in our supplementary experiments that the critic step-sizes that were twice larger than estimated by the fixed-point algorithm led to even better performance. Step-size estimation for the critic is an especially difficult problem as the critic learns mainly on the basis of values that it produces alone, thereby, going beyond the model of stochastic optimization for which the fixed-point algorithm along with most other methods of step-size estimation is designed. Still, the results presented here suggest the correctness of the principles that govern the fixed-point method.

Step-sizes are not the only input parameters of the learning algorithm. The rest of the parameters, however, either have the same values for both tasks or can be assigned by the experimenters based on their experience. The first category encompasses the parameters of the fixed-point method and parameter $b$ of the randomized-truncated estimators (Eqs. (8) and (9)). The second category includes the parameters whose values are in principle problem-dependent, but the learning is not very sensitive to them and not much experience of an experimenter is required to set their proper values. Those are: (1) the sizes of the neural networks, $N_A$ and $N_C$—the more difficult the control problem gets, the larger is the size of required networks, (2) the discount factor, $\gamma$, and the $\lambda$ parameter—the more farsighted the control policy is required to be, the larger are the values of $\gamma$ and $\lambda$, (3) the size, $M$, of the database with experience—it should be sufficiently large for this experience to be representative, (4) the intensity of experience replay, *comp_steps*, results from the available computer power.

In comparison to the parameters discussed above, the step-sizes have different nature: (1) the learning is quite sensitive to them, (2) their proper values differ orders of magnitude from one task to another, (3) for the same task they may differ substantially between stages of the learning. Their on-line estimation is therefore crucial for autonomy of learning.

## 7. Conclusions and future work

In this paper, an actor–critic reinforcement learning algorithm was introduced that combines experience replay (Wawrzyński, 2009) with the fixed point method of step-size estimation, improved upon its version presented in Wawrzyński (2010). Potentially problematic issues associated with this combination were addressed, namely overtraining, wrong fixed point initialization, and mutual adjustment of the actor and critic. The experimental study on octopus arm and half-cheetah revealed that the algorithm learned control policies in multidimensional continuous space within a short time. It is also autonomous in the sense that it achieved its goal essentially within a single run, without auxiliary experiments aiming at manual tuning of the problematic step-size parameters.

As the proposed approach has been designed for robotic applications, we plan its evaluation on a real humanoid walking robot. While the fixed point method of step-size estimation is justified by a simplified model and empirical performance, there is room for its further development and analysis of its convergence. This method has been applied here for optimization of the actor and critic step-sizes separately, while the learning of actor and of critic are coupled processes. The question whether the principles underlying the fixed point method can be formally extended to such coupling is another interesting subject of further research.

## References

Abbeel, P., & Ng, A. Y. (2005). Exploration and apprenticeship learning in reinforcement learning. In *Proc. of the 22nd ICML* (pp. 1–8). ACM.

Adam, S., Busoniu, L., & Babuska, R. (2012). Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, *42*(2), 201–212.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can learn difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*, 834–846.

Behera, L., Kumar, S., & Patnaik, A. (2006). On adaptive learning rate that guarantees convergence in feedforward networks. *IEEE Transactions on Neural Networks*, *17*(5), 1116–1125.

Bhatnagar, S., Sutton, R., Ghavamzadeh, M., & Lee, M. (2009). Natural actor–critic algorithms. *Automatica*, *45*, 2471–2482.

Chiaverini, S., Oriolo, G., & Walker, I. D. (2008). Kinematically redundant manipulators. In *Springer handbook of robotics* (pp. 245–268).

Cichosz, P. (1999). An analysis of experience replay in temporal difference learning. *Cybernetics and Systems*, *30*, 341–363.

Engel, Y., Szabó, P., & Volkinshtein, D. (2005). Learning to control an octopus arm with gaussian process temporal difference methods. In *NIPS*.

George, A. P., & Powell, W. B. (2006). Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, *65*(1), 167–198.

Hachiya, H., Peters, J., & Sugiyama, M. (2011). Reward-weighted regression with sample reuse for direct policy search in reinforcement learning. *Neural Computation*, *23*(11), 2798–2832.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, *1*(4), 295–308.

Kathirvalavakumar, T., & Subavathi, S. J. (2009). Neighborhood based modified backpropagation algorithm using adaptive learning parameters for training feedforward neural networks. *Neurocomputing*, *72*, 3915–3921.

Kimura, H., & Kobayashi, S. (1998). An analysis of actor/critic algorithm using eligibility traces: reinforcement learning with imperfect value functions. In *Proc. of the 15th ICML* (pp. 278–286).

Kober, J., & Peters, J. (2011). Policy search for motor primitives in robotics. *Machine Learning*, *84*(1-2), 171–203.

Konda, V., & Tsitsiklis, J. (2003). Actor-critic algorithms. *SIAM Journal on Control and Optimization*, *42*(4), 1143–1166.

Kormushev, P., Ugurlu, B., Calinon, S., Tsagarakis, N., & Caldwell, D.G. (2011). Bipedal walking energy minimization by reinforcement learning with evolving policy parameterization. In *Proc. IEEE/RSJ intl conf. on intelligent robots and systems*, IROS (pp. 318–324). San Francisco, USA.

Koutnik, J., Gomez, F., & Schmidhuber, J. (2010). Evolving neural networks in compressed weight space. In *Proceedings of the conference on genetic and evolutionary computation*, GECCO-10 (pp. 619–626).

Kushner, H. J., & Yin, G. (1997). *Stochastic approximation algorithms and applications*. Springer-Verlag.

McMahan, W., Chitrakaran, V.K., Csencsits, M.A., Dawson, D.M., Walker, I.D., & Jones, B.A. et al. (2006). Field trials and testing of the octarm continuum manipulator. In *ICRA* (pp. 2336–2341).

Noda, I. (2009). Recursive adaptation of stepsize parameter for non-stationary environments. In *Principles of practice in multi-agent systems* (pp. 525–533).

Octopus-sources (2006). http://www.cs.mcgill.ca/~dprecup/workshops/ICML06/Octopus/octopus-code-distribution.zip.

Peshkin, L., & Mukherjee, S. (2001). Bounds on sample size for policy evaluation in Markov environments. In *Proc. of the 14th annual conf. on computational learning theory, vol. 2111* (pp. 616–629). Berlin: Springer.

Peters, J., Vijayakumar, S., & Schaal, S. (2005). Natural actor–critic. In *Proc. of ECML* (pp. 280–291). Berlin Heidelberg: Springer-Verlag.

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo method*. John Wiley & Sons, Inc..

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). *Neurocomputing: foundations of research*. MIT Press, (pp. 696–699) (Chap. Learning representations by back-propagating errors).

Schraudolph, N.N., & Giannakopoulos, X. (2000). Online independent component analysis with local learning rate adaptation. In *Advances in NIPS, vol. 12* (pp. 789–795).

Schraudolph, N.N., Yu, J., & Aberdeen, D. (2006). Fast online policy gradient learning with SMD gain vector adaptation. In *Advances in NIPS, vol. 18* (pp. 1185–1192).

Silva, F.M., & Almeida, L.B. (1990). Acceleration techniques for the backpropagation algorithm. In *Neural networks EURASIP workshop*. Sesim.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the 7th ICML* (pp. 216–224). Morgan Kaufmann.

Sutton, R.R. (1992a). Gain adaptation beats least squares? In *YWALS* (pp. 161–166).

Sutton, R.S. (1992b). Adapting bias by gradient descent: an incremental version of delta-bar-delta. In *Proc. of the 10th NCAI* (pp. 171–176).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction*. MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in NIPS, vol. 12* (pp. 1057–1063). MIT Press.

Wawrzyński, P. (2009). Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, *22*, 1484–1497.

Wawrzyński, P. (2010). Fixed point method of step-size estimation for on-line neural network training. In *IJCNN* (pp. 2012–2017).

Wawrzyński, P., & Papis, B. (2011). Fixed point method for autonomous on-line neural network training. *Neurocomputing*, *74*, 2893–2905.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning* (pp. 229–256).

Woolley, B. G., & Stanley, K. O. (2010). Evolving a single scalable controller for an octopus arm with a variable number of segments. In *Proceedings of the 11th international conference on parallel problem solving from nature*, PPSN-2010. Springer.

Yekutieli, Y., Sagiv-Zohar, R., Aharonov, R., Engel, Y., Hochner, B., & Flash, T. (2005). Dynamic model of the octopus arm. i. biomechanics of the octopus reaching movement. *Journal of Neurophysiology*, *94*(2), 1443–1458.