Neural Networks 23 (2010) 551–559

Contents lists available at ScienceDirect

Neural Networks

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

2010 Special Issue

Parameter-exploring policy gradients

Frank Sehnke [a,*], Christian Osendorfer [a], Thomas Rückstieß [a], Alex Graves [a], Jan Peters [c], Jürgen Schmidhuber [a,b]

[a] Faculty of Computer Science, Technische Universität München, Boltzmannstr.3, 85748 Garching, Germany

[b] IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland

[c] Max-Planck Institute for Biological Cybernetics, Spemannstrae 38, 72076 Tübingen, Germany

ARTICLE INFO

Article history:
Received 6 February 2009
Received in revised form 30 November 2009
Accepted 9 December 2009

Keywords:
Policy gradients
Stochastic optimisation
Reinforcement learning
Robotics
Control

ABSTRACT

We present a model-free reinforcement learning method for partially observable Markov decision problems. Our method estimates a likelihood gradient by sampling directly in parameter space, which leads to lower variance gradient estimates than obtained by regular policy gradient methods. We show that for several complex control tasks, including robust standing with a humanoid robot, this method outperforms well-known algorithms from the fields of standard policy gradients, finite difference methods and population based heuristics. We also show that the improvement is largest when the parameter samples are drawn symmetrically. Lastly we analyse the importance of the individual components of our method by incrementally incorporating them into the other algorithms, and measuring the gain in performance after each step.

1. Introduction

Policy gradient methods, so called because they search in policy space instead of deriving the policy directly from a value function, are among the few feasible optimisation strategies for complex, high dimensional reinforcement learning problems with continuous states and actions (Benbrahim & Franklin, 1997; Peters & Schaal, 2006; Peters, Vijayakumar & Schaal, 2005; Schraudolph, Yu & Aberdeen, 2006). However a significant problem with policy gradient algorithms such as REINFORCE (Williams, 1992) is that the high variance in their gradient estimates leads to slow convergence. Various approaches have been proposed to reduce this variance (Aberdeen, 2003; Baxter & Bartlett, 2000; Peters & Schaal, 2006; Sutton, McAllester, Singh & Mansour, 2000).

However, none of these methods address the underlying cause of the high variance, which is that repeatedly sampling from a probabilistic policy has the effect of injecting noise into the gradient estimate at every time step. Furthermore, the variance increases linearly with the length of the history (Munos & Littman, 2006), since each state depends on the entire sequence of previous samples. As an alternative, we propose using *policy gradients with parameter-based exploration* (PGPE), where the policy is defined by a distribution over the parameters of a controller. The parameters are sampled from this distribution at the start of each sequence, and thereafter the controller is deterministic. Since the reward for each sequence depends on only a single sample, the gradient estimates are significantly less noisy, even in stochastic environments.

Among the advantages of PGPE is that it does not have the same credit assignment problem. By contrast, a standard policy gradient method must first determine the reward gradient with respect to the policy, then differentiate the parameters with respect to that reward gradient resulting in two drawbacks. Firstly, it assumes that the controller is always differentiable with respect to its parameters, while our approach does not. Secondly, it makes optimisation more difficult since very different parameter settings can determine very similar policies, and vice versa.

An important refinement of the basic PGPE algorithm is the use of symmetric parameter sample pairs, similar to those found in finite difference methods (Spall, 1998a). As we will see, symmetric sampling improves both convergence time and final performance.

The PGPE algorithm is derived in detail in Section 2. In Section 3 we test PGPE on five control experiments, and compare its performance with REINFORCE, evolution strategies (Schwefel, 1995), simultaneous perturbation stochastic adaptation (Spall, 1998a), and episodic natural actor critic (Peters & Schaal, 2008a, 2008b).

In Section 4 we analyse the relationship between PGPE and the above algorithms, by iteratively modifying each of them so that they become more like PGPE, and measuring the performance

* Corresponding author. Tel.: +49 0 89 289 17852; +49 0 179 2030653 (mobile).
E-mail addresses: sehnke@in.tum.de (F. Sehnke), osendorf@in.tum.de (C. Osendorfer), ruecksti@in.tum.de (T. Rückstieß), alex.graves@gmail.com (A. Graves), jan.peters@tuebingen.mpg.de (J. Peters), juergen@idsia.ch (J. Schmidhuber).

0893-6080/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2009.12.004

gain at each step. Conclusions and an outlook on future work are presented in Section 5.

2. Method

In this section we derive the PGPE algorithm from the general framework of episodic reinforcement learning in a Markovian environment. In Particular, we highlight the differences between PGPE and policy gradient methods such as REINFORCE. In Section 2.3 we introduce symmetric sampling and explain why it improves convergence speed.

2.1. Policy gradients with parameter-based exploration

Consider an agent whose action a_t is based on the state s_t at time t and results in the state s_{t+1} in the next step. As we are interested in continuous state and action spaces required for the control of most technical systems, we represent both a_t and s_t by real valued vectors. We assume that the environment is Markovian, i.e., that the conditional probability distribution over the next states s_{t+1} is entirely determined by the preceding state–action pair, $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. We also assume that a stochastic policy suffices, i.e., the distribution over actions only depends on the current state and the real valued vector θ of agent parameters: $a_t \sim p(a_t|s_t, \theta)$. Lastly, we assume that each state–action pair produces a scalar Markovian reward $r_t(a_t, s_t)$. We refer to a length T sequence of state–action pairs produced by an agent as a *history* $h = [s_{1:T}, a_{1:T}]$ (elsewhere in the literature such sequences are called *trajectories* or *roll-outs*).

Given the above formulation we can associate a cumulative reward r with each history h by summing over the rewards at each time step: $r(h) = \sum_{t=1}^{T} r_t$. In this setting, the goal of reinforcement learning is to find the parameters θ that maximize the agent's expected reward

$$J(\theta) = \int_H p(h|\theta) r(h) \mathrm{d}h. \tag{1}$$

An obvious way to maximize $J(\theta)$ is to estimate $\nabla_\theta J$ and use it to carry out gradient ascent optimisation. Noting that the reward for a particular history is independent of θ, we can use the standard identity $\nabla_x y(x) = y(x)\nabla_x \log x$ to obtain

$$\nabla_\theta J(\theta) = \int_H p(h|\theta) \nabla_\theta \log p(h|\theta) r(h) \mathrm{d}h. \tag{2}$$

Since the environment is Markovian, and the states are conditionally independent of the parameters given the agent's choice of actions, we can write $p(h|\theta) = p(s_1) \Pi_{t=1}^{T} p(s_{t+1}|s_t, a_t) p(a_t|s_t, \theta)$. Substituting this into Eq. (2) yields

$$\nabla_\theta J(\theta) = \int_H p(h|\theta) \sum_{t=1}^{T} \nabla_\theta p(a_t|s_t, \theta) r(h) \mathrm{d}h. \tag{3}$$

Clearly, integrating over the entire space of histories is unfeasible, and we therefore resort to sampling methods

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \nabla_\theta p(a_t^n|s_t^n, \theta) r(h^n) \tag{4}$$

where the histories h^i are chosen according to $p(h^i|\theta)$. The question then is how to model $p(a_t|s_t, \theta)$. In policy gradient methods such as REINFORCE, the parameters θ are used to determine a probabilistic *policy* $\pi_\theta(a_t|s_t) = p(a_t|s_t, \theta)$. A typical policy model would be a parametric function approximator whose outputs define the probabilities of taking different actions. In this case the histories can be sampled by choosing an action at each time step according to the policy distribution, and the final gradient is then calculated by differentiating the policy with respect to the parameters. However, sampling from the policy on every time step leads to a high variance in the sample over histories, and therefore to a noisy gradient estimate.

PGPE addresses the variance problem by replacing the probabilistic policy with a probability distribution over the parameters θ, i.e.

$$p(a_t|s_t, \rho) = \int_\Theta p(\theta|\rho) \delta_{F_\theta(s_t), a_t} \mathrm{d}\theta, \tag{5}$$

where ρ are the parameters determining the distribution over θ, $F_\theta(s_t)$ is the (deterministic) action chosen by the model with parameters θ in state s_t, and δ is the Dirac delta function. The advantage of this approach is that the actions are deterministic, and an entire history can therefore be generated from a single parameter sample. This reduction in samples-per-history is what reduces the variance in the gradient estimate. As an added benefit the parameter gradient is estimated by direct parameter perturbations, without having to backpropagate any derivatives, which allows the use of non-differentiable controllers.

The expected reward with a given ρ is

$$J(\rho) = \int_\Theta \int_H p(h, \theta|\rho) r(h) \mathrm{d}h \, \mathrm{d}\theta. \tag{6}$$

Differentiating this form of the expected return with respect to ρ and applying the log trick as before we obtain

$$\nabla_\rho J(\rho) = \int_\Theta \int_H p(h, \theta|\rho) \nabla_\rho \log p(h, \theta|\rho) r(h) \mathrm{d}h \, \mathrm{d}\theta. \tag{7}$$

Noting that h is conditionally independent of ρ given θ, we have $p(h, \theta|\rho) = p(h|\theta)p(\theta|\rho)$ and therefore $\nabla_\rho \log p(h, \theta|\rho) = \nabla_\rho \log p(\theta|\rho)$. Substituting this into Eq. (7) yields

$$\nabla_\rho J(\rho) = \int_\Theta \int_H p(h|\theta) p(\theta|\rho) \nabla_\rho \log p(\theta|\rho) r(h) \mathrm{d}h \, \mathrm{d}\theta. \tag{8}$$

Sampling methods can again be applied, this time by first choosing θ from $p(\theta|\rho)$, then running the agent to generate h from $p(h|\theta)$. This process yields the following gradient estimator:

$$\nabla_\rho J(\rho) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\rho \log p(\theta|\rho) r(h^n). \tag{9}$$

Assuming that ρ consists of a set of means $\{\mu_i\}$ and standard deviations $\{\sigma_i\}$ that determine an independent normal distribution for each parameter θ_i in θ,[1] some rearrangement gives the following forms for the derivative of $\log p(\theta|\rho)$ with respect to μ_i and σ_i

$$\begin{aligned} \nabla_{\mu_i} \log p(\theta|\rho) &= \frac{(\theta_i - \mu_i)}{\sigma_i^2}, \\ \nabla_{\sigma_i} \log p(\theta|\rho) &= \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3}, \end{aligned} \tag{10}$$

which can be substituted into Eq. (9) to approximate the μ and σ gradients.

2.2. Sampling with a baseline

Given enough samples, Eq. (9) will determine the reward gradient to arbitrary accuracy. However each sample requires rolling out an entire state–action history, which is expensive. Following Williams (1992), we obtain a cheaper gradient estimate by drawing a single sample θ and comparing its reward r to a baseline reward b given by a moving average over previous samples. Intuitively, if $r > b$ we adjust ρ so as to increase the

[1] More complex forms for the dependency of θ on ρ could be used, at the expense of higher computational cost.

Algorithm 1 The PGPE Algorithm without reward normalization: Left side shows the basic version, right side shows the version with symmetric sampling. $\boldsymbol{T}$ and $\boldsymbol{S}$ are $P \times N$ matrices with P the number of parameters and N the number of histories. The baseline is updated accordingly after each step. α is the learning rate or step size.

Initialize $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_{\text{init}}$	Initialize $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_{\text{init}}$
Initialize $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}_{\text{init}}$	Initialize $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}_{\text{init}}$
while TRUE **do**	**while** TRUE **do**
for $n = 1$ to N **do**	**for** $n = 1$ to N **do**
draw $\boldsymbol{\theta}^n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I}\boldsymbol{\sigma}^2)$	draw perturbation $\boldsymbol{\epsilon}^n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}\boldsymbol{\sigma}^2)$
	$\boldsymbol{\theta}^{+,n} = \boldsymbol{\mu} + \boldsymbol{\epsilon}^n$
	$\boldsymbol{\theta}^{-,n} = \boldsymbol{\mu} - \boldsymbol{\epsilon}^n$
evaluate $r^n = r(h(\boldsymbol{\theta}^n))$	evaluate $r^{+,n} = r(h(\boldsymbol{\theta}^{+,n}))$
	evaluate $r^{-,n} = r(h(\boldsymbol{\theta}^{-,n}))$
end for	**end for**
$\boldsymbol{T} = [\,t_{ij}\,]_{ij}$ with $t_{ij} := (\theta_i^j - \mu_i)$	$\boldsymbol{T} = [\,t_{ij}\,]_{ij}$ with $t_{ij} := \epsilon_i^j$
$\boldsymbol{S} = [\,s_{ij}\,]_{ij}$ with $s_{ij} := \frac{t_{ij}^2 - \sigma_i^2}{\sigma_i}$	$\boldsymbol{S} = [\,s_{ij}\,]_{ij}$ with $s_{ij} := \frac{(\epsilon_i^j)^2 - \sigma_i^2}{\sigma_i}$
$\boldsymbol{r} = [(r^1 - b), \ldots, (r^N - b)]^{\mathrm{T}}$	$\boldsymbol{r}_T = [(r^{+,1} - r^{-,1}), \ldots, (r^{+,N} - r^{-,N})]^{\mathrm{T}}$
	$\boldsymbol{r}_S = [\frac{(r^{+,1} + r^{-,1}}{2} - b), \ldots, (\frac{(r^{+,N} + r^{-,N}}{2} - b)]^{\mathrm{T}}$
update $\boldsymbol{\mu} = \boldsymbol{\mu} + \alpha \boldsymbol{T}\boldsymbol{r}$	update $\boldsymbol{\mu} = \boldsymbol{\mu} + \alpha \boldsymbol{T}\boldsymbol{r}_{\boldsymbol{T}}$
update $\boldsymbol{\sigma} = \boldsymbol{\sigma} + \alpha \boldsymbol{S}\boldsymbol{r}$	update $\boldsymbol{\sigma} = \boldsymbol{\sigma} + \alpha \boldsymbol{S}\boldsymbol{r}_{\boldsymbol{S}}$
update baseline b accordingly	update baseline b accordingly
end while	**end while**

probability of θ, and $r < b$ we do the opposite. If, as in Williams (1992), we use a step size $\alpha_i = \alpha\sigma_i^2$ in the direction of positive gradient (where α is a constant) we get the following parameter update equations:

$$\begin{aligned} \Delta\mu_i &= \alpha(r - b)(\theta_i - \mu_i), \\ \Delta\sigma_i &= \alpha(r - b)\frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i}. \end{aligned} \tag{11}$$

2.3. Symmetric sampling

While sampling with a baseline is efficient and reasonably accurate for most scenarios, it has several drawbacks. In particular, if the reward distribution is strongly skewed then the comparison between the sample reward and the baseline reward is misleading. A more robust gradient approximation can be found by measuring the difference in reward between two symmetric samples on either side of the current mean. That is, we pick a perturbation ϵ from the distribution $\mathcal{N}(0, \sigma)$, then create symmetric parameter samples $\theta^+ = \mu + \epsilon$ and $\theta^- = \mu - \epsilon$. Defining r^+ as the reward given by θ^+ and r^- as the reward given by θ^-, we can insert the two samples into Eq. (9) and make use of Eq. (10) to obtain

$$\nabla_{\mu_i} J(\rho) \approx \frac{\epsilon_i(r^+ - r^-)}{2\sigma_i^2}, \tag{12}$$

which resembles the *central difference* approximation used in finite difference methods. Using the same step sizes as before gives the following update equation for the μ terms

$$\Delta\mu_i = \frac{\alpha\epsilon_i(r^+ - r^-)}{2}. \tag{13}$$

The updates for the standard deviations are more involved. As θ^+ and θ^- are by construction equally probable under a given σ, the difference between them cannot be used to estimate the σ gradient. Instead we take the mean $\frac{r^+ + r^-}{2}$ of the two rewards and compare it to the baseline reward b. This approach yields

$$\Delta\sigma_i = \alpha\left(\frac{r^+ + r^-}{2} - b\right)\left(\frac{\epsilon_i^2 - \sigma_i^2}{\sigma_i}\right). \tag{14}$$

Compared to the method in Section 2.2, symmetric sampling removes the problem of misleading baselines, and therefore improves the μ gradient estimates. It also improves the σ gradient estimates, since both samples are equally probable under the current distribution, and therefore reinforce each other as predictors of the benefits of altering σ. Even though symmetric sampling requires twice as many histories per update, our experiments show that it gives a considerable improvement in convergence quality and time (see Figs. 1, 2, 5 and 8).

As a final refinement, we make the step size independent from the (possibly unknown) scale of the rewards by introducing a normalization term. Let m be the maximum reward the agent can receive, if this is known, or the maximum reward received so far if it is not. We normalize the μ updates by dividing them by the difference between m and the mean reward of the symmetric samples. We normalize the σ updates by dividing by the difference between m and the baseline b. This insight gives

$$\begin{aligned} \Delta\mu_i &= \frac{\alpha\epsilon_i(r^+ - r^-)}{2m - r^+ - r^-}, \\ \Delta\sigma_i &= \frac{\alpha}{m - b}\left(\frac{r^+ + r^-}{2} - b\right)\left(\frac{\epsilon^2 - \sigma_i^2}{\sigma_i}\right). \end{aligned} \tag{15}$$

Pseudocode for PGPE with and without symmetric sampling is provided by Algorithm 1. Note that the reward normalization terms have been omitted for brevity.

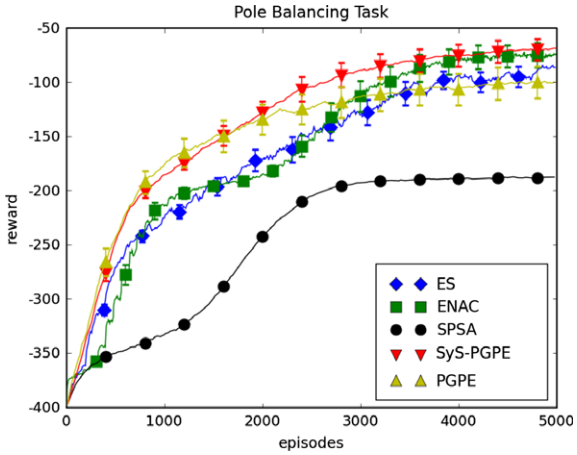

Fig. 1. PGPE with and without SyS compared to ES, SPSA and eNAC on the pole balancing benchmark. All plots show the mean and half standard deviation of 40 runs.

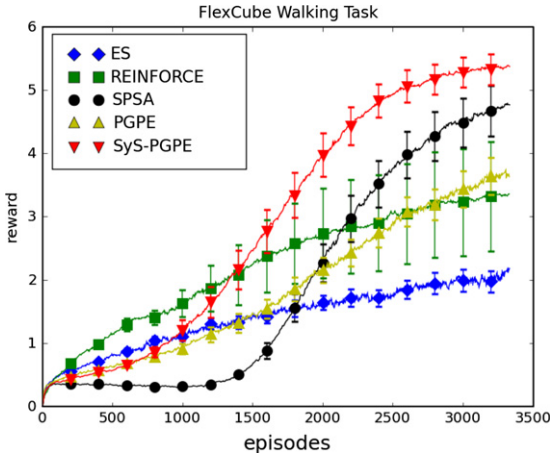

Fig. 2. PGPE with and without SyS compared to ES, SPSA and REINFORCE on the FlexCube walking task. All plots show the mean and half standard deviation of 40 runs.

3. Experiments

In this section we compare PGPE with REINFORCE, simultaneous perturbation stochastic adaptation (SPSA), episodic natural actor critic (eNAC) and evolution strategies (ES) on three simulated control scenarios. These scenarios allow us to model problems of similar complexity to today's real-life RL problems (Müller, Lauer, Hafner, Lange, Merke & Riedmiller, 2007; Peters & Schaal, 2006). For most experiments we also compare the performance of PGPE with and without symmetric sampling (SyS).

For all experiments we plot the agent's reward against the number of training *episodes*. An episode is a sequence of T interactions of the agent with the environment, where T is fixed for each experiment, during which the agent makes one attempt to complete the task. For all methods, the agent and the environment are reset at the beginning of every episode.

For eNAC and REINFORCE we employed an improved algorithm that perturbs the actions at randomly sampled time steps instead of perturbing at every time step.

For all the ES experiments we used a local mutation operator. We did not examine correlated mutation and covariance matrix adaptation-ES because both mutation operators add $n(n-1)$ strategy parameters to the genome; given the more than 1000 parameters for the largest controller, this approach would lead to a prohibitive memory and computation requirement. In addition, the local mutation operator is more similar to the perturbations in PGPE, making it easier to compare the algorithms.

All plots show the average results of 40 independent runs. All the experiments were conducted with hand-optimized metaparameters, including the perturbation probabilities for eNAC and REINFORCE. For PGPE we used the metaparameters $\alpha_\mu = 0.2$, $\alpha_\sigma = 0.1$ and $\sigma_{init} = 2.0$ in all tasks.

3.1. Pole balancing

The first scenario is the extended pole balancing benchmark as described in Riedmiller, Peters and Schaal (2007). In contrast to Riedmiller et al. (2007) however, we do not initialize the controller with a previously chosen stabilizing policy but rather start with random policies. In this task the agent's goal is to maximize the length of time a movable cart can balance a pole upright in the centre of a track. The agent's inputs are the angle and angular velocity of the pole and the position and velocity of the cart. The agent is represented by a linear controller with four inputs and one output unit. The simulation is updated 50 times a second. The initial position of the cart and angle of the pole are chosen randomly.

Fig. 1 shows the performance of the various methods on the pole balancing task. All algorithms quickly learned to balance the pole, and all eventually learned to do so in the centre of the track. PGPE with SyS was both the fastest to learn and the most effective algorithm on this benchmark.

3.2. FlexCube walking task

The second scenario is a mass-particle system with 8 particles. The particles are modelled as point masses on the vertices of a cube, with every particle connected to every other by a spring (see Fig. 3). The agent can set the desired lengths of the 12 edge springs to be anywhere between 0.5 and 1.5 times the original spring lengths. Included in the physics simulation are gravity, collision with the floor, a simple friction model for particles colliding with the floor and the spring forces. We refer to this scenario as the *FlexCube* framework. Though relatively simple, FlexCube can be used to perform sophisticated tasks with continuous state and action spaces. In this case the task is to make the cube "walk"—that is, to maximize the distance of its centre of gravity from the starting point. Its inputs are the 12 current edge spring lengths, the 12 previous desired edge spring lengths (fed back from its own output at the last time step) and the 8 floor contact sensors in the vertices. The policy of the agent is represented by a Jordan network (Jordan, 1986) with 32 inputs, 10 hidden units and 12 output units. Fig. 2 shows the results on the walking task. All the algorithms learn to move the FlexCube. PGPE substantially outperforms the other methods, both in learning speed and final reward. Here SyS has a big impact on both as well. Fig. 3 shows a typical scenario of the walking task. For better understanding please refer to the video on Sehnke (2009).

3.3. Biped robot standing task

The task in this scenario was to keep a simulated biped robot standing while perturbed by external forces. The simulation, based on the biped robot Johnnie (Ulbrich, 2008) was implemented using the Open Dynamics Engine. The lengths and masses of the body parts, the location of the connection points, and the range of allowed angles and torques in the joints were matched with those of the original robot. Due to the difficulty of accurately simulating the robot's feet, the friction between them and the ground was approximated by a Coulomb friction model. The framework has 11

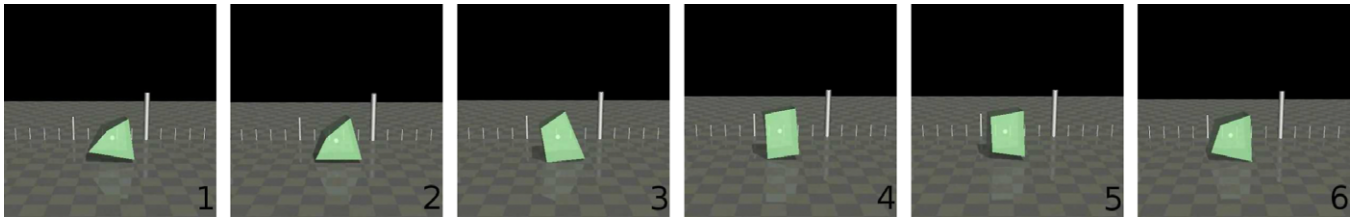

Fig. 3. From left to right, a typical solution which worked well in the walking task is shown: 1. Stretching forward. 2. Off the ground. 3. Landing on front vertices. 4. Retracting back vertices 5. Bouncing off front vertices, landing on back vertices. 6. Stretching forward (cycle closed).

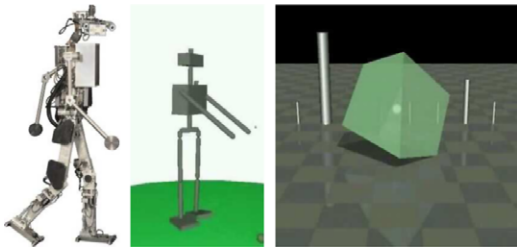

Fig. 4. The real Johnnie robot (left), its simulation (centre) and the FlexCube (right).

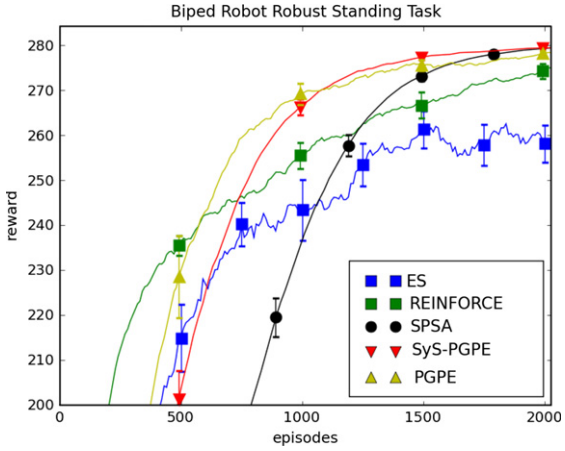

Fig. 5. PGPE with and without SyS compared to ES, SPSA and REINFORCE on the robust standing benchmark. All plots show the mean and half standard deviation of 40 runs.

degrees of freedom and a 41 dimensional observation vector (11 angles, 11 angular velocities, 11 forces, 2 pressure sensors in feet, 3 degrees of orientation and 3 degrees of acceleration in the head).

The controller was a Jordan network (Jordan, 1986) with 41 inputs, 20 hidden units and 11 output units (Fig. 4).

The aim of the task is to maximize the height of the robot's head, up to the limit of standing completely upright. The robot is continually perturbed by random forces (depictured by the particles in Fig. 6) that would knock it over unless it counterbalanced.

As can be seen from the results in Fig. 5, the task was relatively easy, and all the methods were able to quickly achieve a high reward. REINFORCE learned especially quickly, and outperformed PGPE in the early stages of learning. However PGPE overtook it after about 500 training episodes. Fig. 6 shows a typical scenario of the robust standing task. For more details please refer to the video on Sehnke (2009).

3.4. Ship steering task

In this task an ocean-going ship with substantial inertia in both forward motion and rotation (plus noise resembling the impact of the waves) is simulated. The task in this scenario was to keep the ship on course while keeping maximal speed. While staying on a predefined course with an error less than 5 degrees, the reward is equal to the ship's speed. Otherwise the agent receives a reward of 0. The framework has 2 degrees of freedom and a three dimensional observation vector (velocity, angular velocity, error angle). The agent is represented by a linear controller with 3 inputs and 2 output units. The simulation is updated every 4 s. The initial angle of the ship is chosen randomly (Fig. 7).

Table 1
Overview of algorithms used in the ship steering task. The results are plotted in Fig. 8 with respective marker shapes and colors.

#	Markers	SyS	Reward normalization	# Samples
1	Red circle	Yes	Yes	2
2	Green triangle	No	Yes	1
3	Blue diamond	No	Yes	2
4	Yellow square	Yes	No	2

In this experiment we only compare different versions of our algorithm with each other. Table 1 gives an overview of the different properties used. As can be seen from the results in Fig. 8, the task could be solved easily with PGPE. PGPE with SyS (version 1) is faster in convergence speed than its non-symmetric counterpart (version 2). However, both versions 1 and 2 reach the same optimal control strategy.

The improvement cannot be explained by the fact that SyS uses 2 samples rather than one, as can be seen when comparing it to the third version: PGPE with random sampling and 2 samples batch size. This algorithm performs even worse than the two others. Hence the improvement does in fact come from the symmetry of the two samples. Version 4 of the algorithm assumes that the maximum reward r_{max} is known in advance. Instead of the reward normalization introduced in Eq. (15) the reward is then simply divided by r_{max}. Our experiments show, that even if knowledge of r_{max} is available, it is still beneficial to use the adaptive reward normalization instead of the real maximum, since it accelerates convergence during the early learning phase. In Wierstra, Schaul, Peters and Schmidhuber (2008) other beneficial reward normalization techniques are discussed, especially for bigger batch sizes.

3.5. Grasping task

The task in this scenario was to grasp an object from different positions on a table. The simulation, based on the CCRL robot (Buss & Hirche, 2008) (Fig. 9) was implemented using the Open Dynamics Engine. The lengths and masses of the body parts and the location of the connection points were matched with those of the original robot. Friction was approximated by a Coulomb friction model. The framework has 8 degrees of freedom and a 35 dimensional observation vector (8 angles, 8 angular velocities, 8 forces, 2 pressure sensors in hand, 3 degrees of orientation and 3 values of position in hand, 3 values of position of object). The controller was a Jordan network (Jordan, 1986) with 35 inputs, 10 hidden units and 8 output units. The task was learned in 4 phases, with progressively more difficult initial positions for the object. In the first phase, the

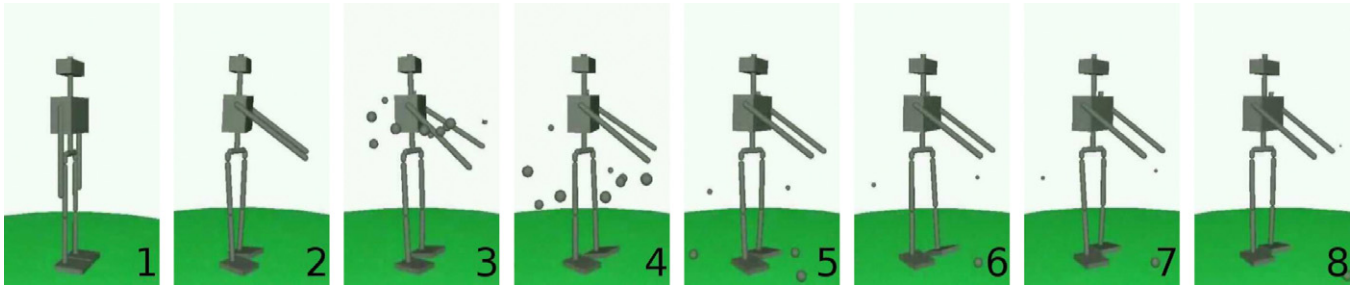

Fig. 6. From left to right, a typical solution which worked well in the robust standing task is shown: 1. Initial posture. 2. Stable posture. 3. Perturbation by heavy weights that are thrown randomly at the robot. 4–7. Backsteps right, left, right, left. 8. Stable posture regained.

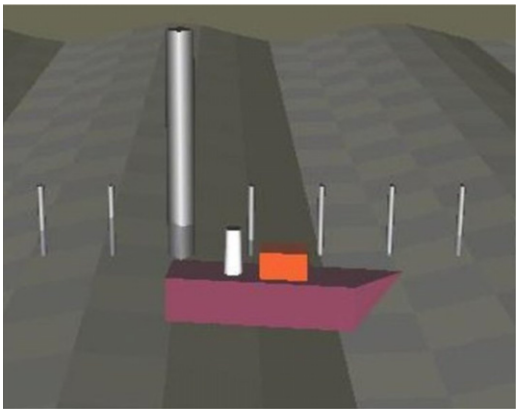

Fig. 7. The ship steering simulation. The color of the "cargo container" corresponds to the agent's reward. The color changes continuously from green for maximal reward to red for minimal reward. The scale in the background shows the distance the ship has travelled. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

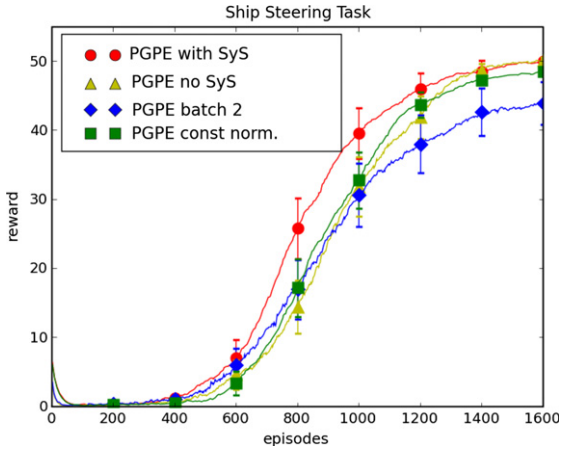

Fig. 8. PGPE with SyS, without SyS, with two samples batch size and with classical reward normalization. All plots show the mean and half standard deviation of 40 runs.

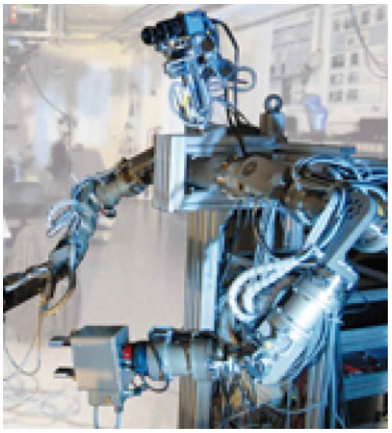

Fig. 9. The CCRL robot (courtesy: Institute of Automatic Control Engineering (Buss & Hirche, 2008)).

object was always in the same place on the edge of the table. In the second phase it was still in a fixed position, but away from the edge. In the third phase it was normally distributed around the centre of the reachable region (standard deviation of 10 cm). In the last phase it was placed with equal probability anywhere in the reachable area. Every phase required 10,000 episodes and used the final controller of the preceding phase. Fig. 10 shows a typical solution of the grasping task. For more detailed views of the solution please see the video on Sehnke (2009).

3.6. Discussion

One general observation from our experiments was that the longer the episodes the more PGPE outperformed policy gradient methods. This effect is a result of the variance increase in REINFORCE gradient estimates with the number of actions. As most interesting real-world problems require much longer episodes than in our experiments, this improvement can have a strong impact. For example, in biped walking (Benbrahim & Franklin, 1997), object manipulation (Peters & Schaal, 2006) and other robot control tasks (Müller et al., 2007) update rates of hundreds of Hertz and task lengths of several seconds are common. Another observation was that symmetric sampling has a stronger impact on tasks with more complex, multimodal reward functions, such as in the FlexCube walking task.

4. Relationship to other algorithms

In this section we attempt to evaluate the differences between PGPE, SPSA, ES and REINFORCE. Fig. 11 shows an overview of the relationship of PGPE to the other compared learning methods. Starting with each of the other algorithms, we incrementally alter them so that their behaviour (and performance) becomes closer to that of PGPE. In the case of SPSA we end up with an algorithm identical to PGPE; for the other methods, the transformed algorithm is similar but still inferior to PGPE.

4.1. From SPSA to PGPE

Three changes are required to transform SPSA into PGPE. First the uniform sampling of perturbations is replaced by Gaussian sampling. Second, correspondingly the finite differences gradient is replaced by the likelihood gradient. Third, the variances of the perturbations are turned into free parameters and trained with the

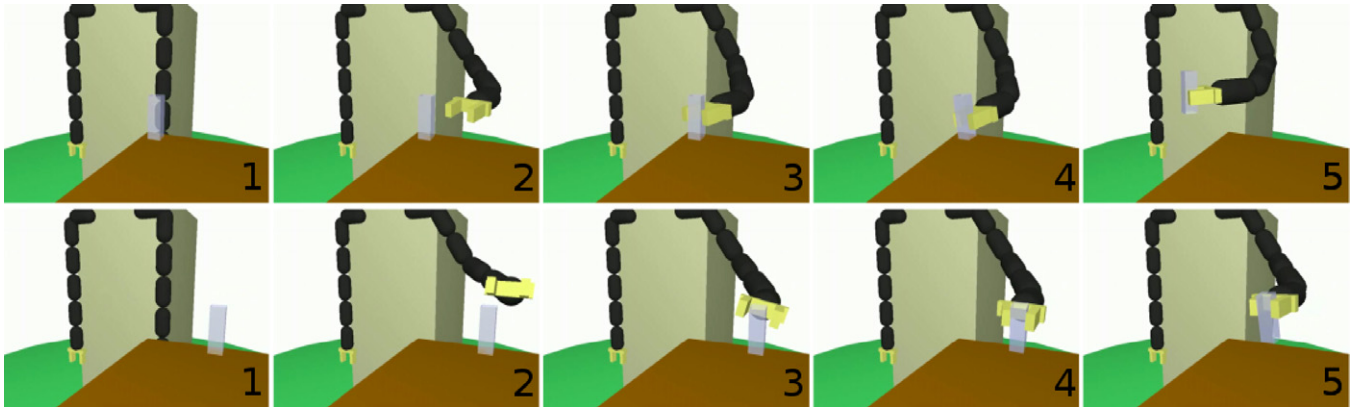

Fig. 10. From left to right, a typical solution which worked well in the grasping task is shown for 2 different positions of the object with the same controller: 1. Initial posture. 2. Approach. 3. Enclose. 4. Take hold. 5. Lift.

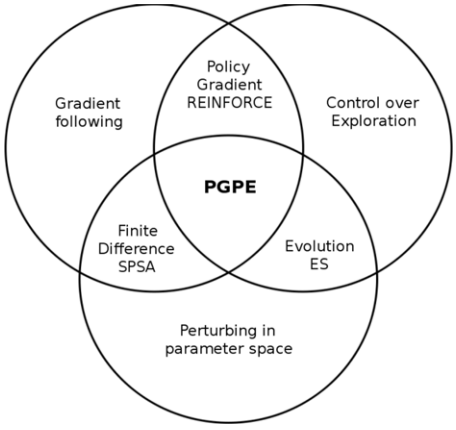

Fig. 11. Relationship of PGPE to other stochastic optimisation methods.

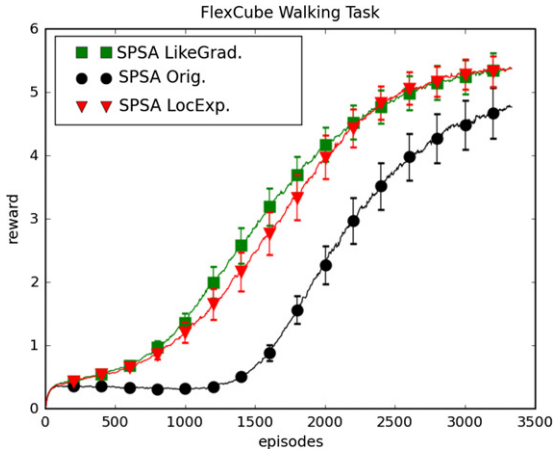

Fig. 12. Three variants of SPSA on the FlexCube walking task: the original algorithm (SPSA Original), the algorithm with normally distributed sampling and likelihood gradient (SPSA LikeGrad.), and with adaptive variance (SPSA LocExp.). All plots show the mean and half standard deviation of 40 runs.

rest of the model. Initially the Gaussian sampling is carried out with fixed variance, just as the range of uniform sampling is fixed in SPSA.

Fig. 12 shows the performance of the three variants of SPSA on the walking task. Note that the final variant is identical to PGPE (triangle). For this task the main improvement comes from the switch to Gaussian sampling and the likelihood gradient (circles). Adding adaptive variances actually gives slightly slower learning at first, although the two converge later on.

The original parameter update rule for SPSA is

$$\theta_i(t+1) = \theta_i(t) - \alpha \frac{y_+ - y_-}{2\epsilon} \tag{16}$$

with $y_+ = r(\theta + \Delta\theta)$ and $y_- = r(\theta - \Delta\theta)$, where $r(\theta)$ is the evaluation function and $\Delta\theta$ is drawn from a Bernoulli distribution scaled by the time-dependent step size $\epsilon(t)$, i.e. $\Delta\theta_i(t) = \epsilon(t)\,\mathrm{rand}(-1, 1)$. In addition, a set of metaparameters is used to help SPSA converge. The step size ϵ decays according to $\epsilon(t) = \frac{\epsilon(0)}{t^\gamma}$ with $0 < \gamma < 1$. Similarly, the step size α decreases over time with $\alpha = a/(t + A)^E$ for some fixed a, A and E (Spall, 1998a). The choice of initial parameters $\epsilon(0)$, γ, a, A and E is critical to the performance of SPSA. In Spall (1998b) some guidance is provided for the selection of these coefficients (note that the nomenclature differs from the one used here).

To switch from uniform to Gaussian sampling we simply modify the perturbation function to $\Delta\theta_i(t) = \mathcal{N}(0, \epsilon(t))$. We then follow the derivation of the likelihood gradient outlined in Section 2, to obtain the same parameter update rule as used for PGPE (Eq. (11)). The correspondence with PGPE becomes exact when we calculate the gradient of the expected reward with respect to the sampling variance, giving the standard deviation update rule of Eq. (11).

As well as improved performance, the above modifications greatly reduce the number of hand-tuned metaparameters in the algorithm, leaving only the following: a step size α_μ for updating the parameters, a step size α_σ for updating the standard deviations of the perturbations, and an initial standard deviation σ_{init}. We found that the parameters $\alpha_\mu = 0.2$, $\alpha_\sigma = 0.1$ and $\sigma_{init} = 2.0$ worked very well for a wide variety of tasks.

4.2. From ES to PGPE

We now examine the effect of two modifications that bring ES closer to PGPE. First we switch from standard ES to derandomized ES (Hansen & Ostermeier, 2001), which somewhat resembles the gradient based variance updates found in PGPE. Then we change from population based search to following a likelihood gradient. The results are plotted in Fig. 13. As can be seen, both modifications bring significant improvements, although neither can match PGPE. While ES performs well initially, it is slow to converge to good optima. This is partly because, as well as having stochastic mutations, ES has stochastic updates to the standard deviations of the mutations, and the coupling of these terms slows down convergence. Derandomized ES addresses that problem by using instead a deterministic standard deviation update rule, based on the change in parameters between the parent and child. Population based search has advantages in the early phase of search, when broad, relatively undirected exploration is desirable.

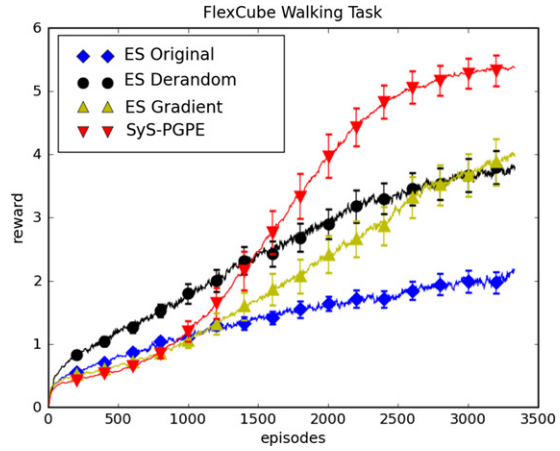

Fig. 13. Three variants of ES on the FlexCube walking task: the original algorithm (ES Original), derandomized ES (ES Derandom) and gradient-following (ES Gradient). PGPE is shown for reference. All plots show the mean and half standard deviation of 40 runs.

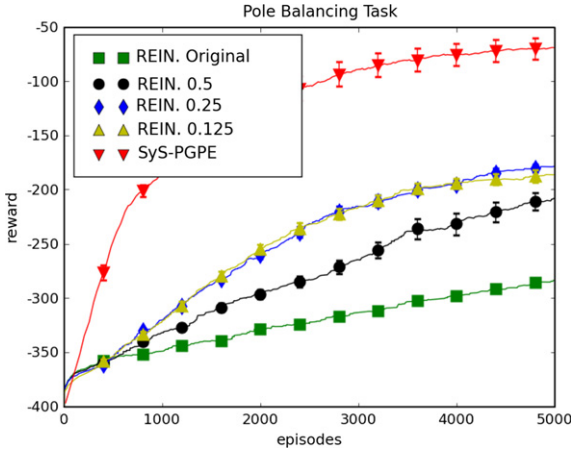

Fig. 14. REINFORCE on the pole balancing task, with various action perturbation probabilities (1, 0.5, 0.25, 0.125). PGPE is shown for reference. All plots show the mean and half standard deviation of 40 runs.

This is particularly true for the multimodal fitness spaces typical of realistic control tasks. However in later phases convergence is usually more efficient with gradient based methods. Applying the likelihood gradient, the ES parameter update rule becomes

$$\Delta\theta_i = \alpha \sum_{m=1}^{M} (r_m - b)(y_{m,i} - \theta_i), \tag{17}$$

where M is the number of samples and $y_{m,i}$ is parameter i of sample m.

4.3. From REINFORCE to PGPE

We previously asserted that the lower variance of PGPE's gradient estimates is partly due to the fact that PGPE requires only one parameter sample-per-history, whereas REINFORCE requires samples every time step. This suggests that reducing the frequency of REINFORCE perturbations should improve its gradient estimates, thereby bringing it closer to PGPE.

Fig. 14 shows the performance of episodic REINFORCE with a perturbation probability of 1, 0.5, 0.25, and 0.125 per time step. In general, performance improved with decreasing perturbation probability. However the difference between 0.25 and 0.125 is negligible. This is because reducing the number of perturbations constrains the range of exploration at the same time as it reduces the variance of the gradient, leading to a saturation point beyond which performance does not increase.

Note that the above trade off does not exist for PGPE, because a single perturbation of the parameters can lead to a large change in the overall behaviour.

A related approach is taken in Rückstieß, Felder and Schmidhuber (2008) where the exploratory noise in each time step is replaced by a state-dependent exploratory function (SDE) with additional parameters. Rather than generating random noise in each time step, the parameters of this exploration function are drawn at the beginning of each episode and kept constant thereafter, which leads to smooth trajectories and reduced variance. SDE allows to use any gradient estimation technique like Natural Actor Critic or the classical REINFORCE algorithm by Williams, while still ensuring smooth trajectories, a property that PGPE naturally has.

5. Conclusion and future work

We have introduced PGPE, a novel algorithm for episodic reinforcement learning based on a gradient based search through model parameter space. We have derived the PGPE equations from the basic principle of reward maximization, and explained why they lead to lower variance gradient estimates than those obtained by policy gradient methods. We compared PGPE to a range of stochastic optimisation algorithms on three control tasks, and found that it gave superior performance in every case. We have also shown that PGPE is substantially improved by symmetric parameter sampling. Lastly, we provided a detailed analysis of the relationship between PGPE and the other algorithms.

A possible objection to PGPE is that the parameter space is generally higher dimensional than the action space, and therefore has higher sampling complexity. However, standard policy gradient methods in fact train the same number of parameters—in PGPE they are just trained explicitly instead of implicitly. Additionally, recent results (Riedmiller et al., 2007) indicate that this drawback was overestimated in the past. In this paper we present experiments where PGPE successfully trains a controller with more than 1000 parameters. Another issue is that PGPE, at least in its present form, is episodic, because the parameter sampling is carried out once per history. This contrasts with policy gradient methods, which can be applied to infinite horizon settings as long as frequent rewards can be computed.

One direction for future work would be to establish whether Williams' local convergence proofs for REINFORCE can be generalised to PGPE. Another would be to combine PGPE with recent improvements in policy gradient methods, such as natural gradients and baseline approximation (Peters et al., 2005). We will also compare PGPE to the related SDE method (Rückstieß et al., 2008) to investigate performance differences, especially for problem sets with a large number of parameters. Lastly, we would like to see if symmetric sampling is beneficial to other stochastic optimisation methods such as evolution strategies.

Acknowledgement

This work was partly funded within the Excellence Cluster *Cognition for Technical Systems* (CoTeSys) by the German Research Foundation (DFG).

References

Aberdeen, D. (2003). Policy-gradient algorithms for partially observable markov decision processes. Ph.D. thesis, Australian National University.

Baxter, J., & Bartlett, P. L. (2000). Reinforcement learning in POMDPs via direct gradient ascent. In *Proc. 17th international conf. on machine learning* (pp. 41–48). San Francisco, CA: Morgan Kaufmann.

Benbrahim, H., & Franklin, J. (1997). Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, *22*, 283–302.
Buss, M., & Hirche, S. (2008). Institute of Automatic Control Engineering, TU München, Germany. http://www.lsr.ei.tum.de/.
Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, *9*(2), 159–195.
Jordan, M. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proc. of the eighth annual conference of the cognitive science society* (pp. 531–546) *Vol. 8*.
Müller, H., Lauer, M., Hafner, R., Lange, S., Merke, A., & Riedmiller, M. (2007). Making a robot learn to play soccer. In *Proceedings of the 30th annual German conference on artificial intelligence (KI-2007)*.
Munos, R., & Littman, M. (2006). Policy gradient in continuous time. *Journal of Machine Learning Research*, *7*, 771–791.
Peters, J., & Schaal, S. (2006). Policy gradient methods for robotics. In *IROS-2006* (pp. 2219–2225).
Peters, J., & Schaal, S. (2008a). Natural actor–critic. *Neurocomputing*, *71*, 1180–1190.
Peters, J., & Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 682–697.
Peters, J., Vijayakumar, S., & Schaal, S. (2005). Natural actor–critic. In *ECML-2005* (pp. 280–291).
Riedmiller, M., Peters, J., & Schaal, S. (2007). Evaluation of policy gradient methods and variants on the cart-pole benchmark. In *ADPRL-2007*.
Rückstieß, T., Felder, M., & Schmidhuber, J. (2008). State-dependent exploration for policy gradient methods. In W.D., et al., (Eds.), *LNAI*: *Vol. 5212. European conference on machine learning and principles and practice of knowledge discovery in databases 2008, Part II* (pp. 234–249).
Schraudolph, N., Yu, J., & Aberdeen, D. (2006). Fast online policy gradient learning with smd gain vector adaptation. In Y. Weiss, B. Schölkopf, & J. Platt (Eds.), *Advances in neural information processing systems*: *Vol. 18*. Cambridge, MA: MIT Press.
Schwefel, H. (1995). *Evolution and optimum seeking*. New York: Wiley.
Sehnke, F. (2009). PGPE – Policy Gradients with Parameter-based exploration – demonstration video: Learning in robot simulatons. http://www.pybrain.org/videos/jnn10/.
Spall, J. (1998a). An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest*, *19*(4), 482–492.
Spall, J. (1998b). Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems*, *34*(3), 817–823.
Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *NIPS-1999* (pp. 1057–1063).
Ulbrich, H. (2008). Institute of Applied Mechanics, TU München, Germany. http://www.amm.mw.tum.de/.
Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008). Fitness expectation maximization. In *Proceedings of the 10th international conference on parallel problem solving from nature*.
Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.