PALADYN Journal of Behavioral Robotics

VERSITA

Review Article · DOI: 10.2478/s13230-010-0002-4 · JBR · 1(1) · 2010 · 14-24

# Exploring Parameter Space in Reinforcement Learning

**Thomas Rückstieß[1]*, Frank Sehnke[1], Tom Schaul[2], Daan Wierstra[2], Yi Sun[2], Jürgen Schmidhuber[2]**

*1 Technische Universität München Institut für Informatik VI, Boltzmannstr. 3, 85748 Garching, Germany,*

*2 Dalle Molle Institute for Artificial Intelligence (IDSIA) Galleria 2, 6928 Manno-Lugano, Switzerland*

**Received 20 February 2010**

**Accepted 16 March 2010**

**Abstract**

This paper discusses parameter-based exploration methods for reinforcement learning. Parameter-based methods perturb parameters of a general function approximator directly, rather than adding noise to the resulting actions. Parameter-based exploration unifies reinforcement learning and black-box optimization, and has several advantages over action perturbation. We review two recent parameter-exploring algorithms: Natural Evolution Strategies and Policy Gradients with Parameter-Based Exploration. Both outperform state-of-the-art algorithms in several complex high-dimensional tasks commonly found in robot control. Furthermore, we describe how a novel exploration method, State-Dependent Exploration, can modify existing algorithms to mimic exploration in parameter space.

**Keywords**

*reinforcement learning · optimization · exploration · policy gradients*



## 1. Introduction

Reinforcement learning (RL) is the method of choice for many complex real-world problems where engineers are unable to explicitly determine the desired policy of a controller. Unfortunately, as the indirect reinforcement signal provides less information to the learning algorithm than the teaching signal in supervised learning, learning requires a large number of trials.

Exploration is a critical component of RL, affecting both the number of trials required and the quality of the solution found. Novel solutions can be found only through effective exploration. Preferably, exploration should be broad enough not to miss good solutions, economical enough not to require too many trials and intelligent in the sense that the information gained through it is high. Clearly, those objectives are difficult to trade off. In practice, unfortunately, many RL practitioners do not focus on exploration, instead relying on small random perturbations of the actions of the current policy.

In this paper we review some alternative methods of exploration for Policy Gradient (PG) based RL that go beyond action-based exploration, directly perturbing policy parameters instead. We will look at two recent parameter-exploring algorithms: Natural Evolution Strategies (NES) [33] and Policy Gradients with Parameter-Based Exploration (PGPE) [22]. Both algorithms have been shown to outperform state-of-the-art PG methods in several complex high-dimensional tasks commonly found in robot control. We will review further a novel exploration technique, called State-Dependent Exploration (SDE), first introduced in [18]. SDE can modify existing PG algorithms to mimic exploration in parameter space and has also demonstrated to improve state-of-the-art PG methods.

We take a stand for parameter exploration (PE), the common factor of the above mentioned methods, which has the advantage of reducing the variance of exploration, while at the same time easing credit assignment. Furthermore, we establish how these methods relate to the field of Black-Box Optimization (BBO), and Evolution Strategies (ES) in particular. We highlight the topic of parameter-based exploration from different angles and give an overview of the superiority of this approach. We also give reasons for the increase in performance and insights in additional properties of this exploration approach.

The general outline of this paper is as follows: In Section 2 we introduce the general frameworks of RL and BBO and review the state-of-the-art. Section 3 then details parameter-based exploration, reviews two parameter-exploring algorithms and demonstrates how traditional algorithms can be brought to behave like parameter-exploring ones. Experimental results are described in Section 4 and the paper concludes in Section 5.

## 2. Background

In this section we introduce the RL framework in general and fix the terminology. In that context, we then formalize the concept of exploration. Finally we provide a brief overview of policy gradients and black-box optimization.

### 2.1. Reinforcement Learning

RL generally tries to optimize an agent's behavior in its environment. Unlike supervised learning, the agent is not told the correct behavior

*E-mail: ruecksti@in.tum.de

directly, but only informed about how well it did, usually in terms of a scalar value called *reward*.
Learning proceeds in a cycle of interactions with the environment. In time $t$, the agent observes a state $s_t$ from the environment, performs an action $a_t$ and receives a reward $r_t$. The environment then determines the next state $s_{t+1}$. We will use the term *history* for the concatenation of all encountered states, actions and rewards up to time step $t$ as $h_t = \{s_0, a_0, r_0, \ldots, s_{t-1}, a_{t-1}, r_{t-1}, s_t\}$.
Often, the environment fulfills the Markov assumption, i.e. the probability of the next state depends only on the last observed state and the current action: $p(s_{t+1}|h_t) = p(s_{t+1}|s_t, a_t)$. The environment is *episodic* if it admits at least one terminal, absorbing state.
The objective of RL is to maximize the 'long term' return $R_t$, which is the (discounted) sum of future rewards $R_t = \sum_{k=t}^{T} \gamma^k r_{t+1+k}$. In the episodic case, the objective is to maximize $R_1$, i.e., the discounted sum of rewards at the initial time step. The discounting factor $\gamma \in [0, 1]$ can put more or less emphasis on the most recent rewards. For infinite-horizon tasks that do not have a foreseeable end, $\gamma < 1$ prevents unbounded sums.
Actions are selected by a policy $\pi$ that maps a history $h_t$ to a probability of choosing an action: $\pi(h_t) = p(a_t|h_t)$. If the environment is Markovian, it is sufficient to consider Markovian policies that satisfy $p(a_t|h_t) = p(a_t|s_t)$.
We define $J(\theta)$ to be the performance of a policy $\pi$ with parameters $\theta$: $J(\theta) = E\{R_t\}$. For deterministic policies, we also write $\pi(s') = a$. In either case, the selected action is not necessarily the one executed, as it can be modified or substituted by a different action, depending on the exploration component. We describe exploration in detail in Section 2.2.
Three main categories can be found in RL: direct, value-based and model-based learning, as depicted in Figure 2.1.
Value-based RL tries to map each combination of state and action to a Q-value. A Q-value is the expected return, if we execute action $a$ in state $s$ at time $t$ and follow the policy $\pi$ greedily thereafter: $Q^\pi(s, a) = E\{R_t|s_t = s, a_t = a\}$. If the task is episodic, we can approximate the values by Monte-Carlo methods, generating several episodes and averaging over the received returns.
For infinite-horizon tasks, the definition of the Q-values can be transformed: $Q^\pi(s, a) = E\{R_t|s_t = s, a_t = a\} = E\{r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1})|s_t = s, a_t = a\}$. Using an exponential moving average and replacing $Q(s_{t+1}, a_{t+1})$ by $\max_a Q(s_{t+1}, a)$, we end up with the well-known Q-Learning algorithm [31]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (1)$$

A greedy policy then selects the action with the highest Q-value in a given state.

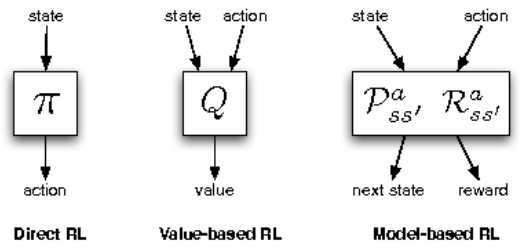


*Figure 1. Different Reinforcement Learning categories.*

In a continuous action space, action selection is much harder, mainly because calculating $\arg\max_a Q(s, a)$ is not trivial. Using a general function approximator (FA) to estimate the Q-values for state-action pairs, it is possible but expensive to follow the gradient $\frac{\partial Q(s,a)}{\partial a}$ towards an action that returns a higher Q-value. In actor-critic architectures [27], where the policy (the actor) is separated from the learning component (the critic), one can backpropagate the temporal difference error through the critic FA (usually implemented as neural networks) to the actor FA and train the actor to output actions that return higher Q-values [16, 30].
Direct reinforcement learning methods, in particular Policy Gradient methods [13, 14, 34], avoid the problem of finding $\arg\max_a Q(s, a)$ altogether, thus being popular for continuous action and state domains. Instead, states are mapped to actions directly by means of a parameterized function approximator, without utilizing Q-values. The parameters $\theta$ are changed by following the performance gradient $\nabla_\theta J(\theta)$. Different approaches exist to estimate this gradient [14].
Finally, model-based RL aims to estimate the transition probabilities $\mathcal{P}^a_{ss'} = p(s'|s, a)$ and the rewards $\mathcal{R}^a_{ss'} = E\{r|s, s', a\}$ going from state $s$ to $s'$ with action $a$. Having a model of the environment allows one to use direct or value-based techniques within the simulation of the environment, or even for dynamic programming solutions.

## 2.2. Exploration

The exploration/exploitation dilemma is one of the main problems that needs to be dealt with in Reinforcement Learning: Without exploration, the agent can only go for the best solution found so far, not learning about potentially better solutions. Too much exploration leads to mostly random behavior without exploiting the learned knowledge. A good exploration strategy carefully balances exploration and greedy policy execution.
Many exploration techniques have been developed for the case of discrete actions [28, 32], commonly divided into undirected and directed exploration. The most popular—albeit not the most effective—undirected exploration method is $\epsilon$-greedy exploration, where the probability of choosing a random action decreases over time. In practice, a random number $r$ is drawn from a uniform distribution $r \sim \mathcal{U}(0, 1)$, and action selection follows this rule:

$$\pi(s) = \begin{cases} \arg\max_a Q(s, a) & \text{if } r \geq \epsilon \\ \text{random action from } \mathcal{A}(s) & \text{if } r < \epsilon \end{cases}$$

where $\mathcal{A}(s)$ is the set of valid actions from state $s$ and $0 \leq \epsilon \leq 1$ is the trade-off parameter, which is reduced over time to slowly transition from exploration to exploitation. Other exploration methods for discrete actions are presented in [28]. For the remainder of the paper, we will ignore the well-established area of discrete action exploration and concentrate on continuous actions.
In the case of continuous actions, exploration is often neglected. If the policy is considered to be stochastic, a Gaussian distribution of actions is usually assumed, where the mean is often selected and interpreted as the greedy action:

$$a \sim \mathcal{N}(a_{\text{greedy}}, \sigma^2) = a_{\text{greedy}} + \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, \sigma^2) \quad (2)$$

During learning, exploration then occurs implicitly—almost as a side-effect— by sampling actions from the stochastic policy. While this is convenient, it conceals the fact that two different stochastic elements

are involved here: the exploration and the stochastic policy itself. This becomes most apparent if we let $\sigma$ adapt over time as well, following the gradient $\frac{\partial J(\theta)}{\partial \sigma}$, which is well-defined if the policy is differentiable. If the best policy is in fact a deterministic one, $\sigma$ will decrease quickly and therefore exploration comes to a halt as well. This clearly undesirable behavior can be circumvented by adapting the variance manually, e.g. by decreasing it slowly over time.
Another disadvantage of this implicit exploration is the independence of samples over time. In each time step, we draw a new $\epsilon$ and add it to the actions, leading to a very noisy trajectory through action space (see Figure 2 top). A robot controlled by such actions would exhibit a very shaky behavior, with a severe impact on the performance. Imagine an algorithm with this kind of exploration controlling the torques of a robot end-effector directly. Obviously, the trembling movement of the end-effector will worsen the performance in almost any object manipulation task. And we ignore the fact that such consecutive contradicting motor commands might even damage the robot or simply cannot be executed. Thus applying such methods requires the use of motion primitives [19] or other transformations. Despite these problems, many current algorithms [13, 17, 34] use this kind of Gaussian, action-perturbing exploration (cf. Equation 2).

## 2.3. Policy Gradients

In this paper, we will compare the parameter-exploring parameters to policy gradient algorithms that perturb the resulting action, in particular to REINFORCE [34] and eNAC [13]. Below, we will give a short overview of the derivation of policy gradient methods.
we start with the probability of observing history $h^\pi$ under policy $\pi$, which is given by the probability of starting with an initial observation $s_0$, multiplied by the probability of taking action $a_0$ under $h_0$, multiplied by the probability of receiving the next observation $s_1$, and so on. Thus, (3) gives the probability of encountering a certain history $h^\pi$.

$$p(h^\pi) = p(s_0)\prod_{t=0}^{T-1}\pi(a_t|h_t^\pi)\,p(s_{t+1}|h_t^\pi, a_t) \tag{3}$$

Inserting this into the definition of the performance measure $J(\theta)$, we can rewrite the equation by multiplying with $1 = {}^{p(h^\pi)}/_{p(h^\pi)}$ and using $\frac{1}{x}\nabla x = \nabla \log(x)$ to get

$$\nabla_\theta J(\pi) = \int \frac{p(h^\pi)}{p(h^\pi)}\nabla_\theta p(h^\pi)R(h^\pi)\,dh^\pi \tag{4}$$

$$= \int p(h^\pi)\nabla_\theta \log p(h^\pi)R(h^\pi)\,dh^\pi. \tag{5}$$

For now, let us consider the gradient $\nabla_\theta \log p(h^\pi)$. Substituting the probability $p(h^\pi)$ according to (3) gives

$$\nabla_\theta \log p(h^\pi) = \nabla_\theta \log\Big[p(s_0)\prod_{t=0}^{T-1}\pi(a_t|h_t^\pi)\,p(s_{t+1}|h_t^\pi, a_t)\Big]$$

$$= \nabla_\theta\Big[\log p(s_0) + \sum_{t=0}^{T-1}\log\pi(a_t|h_t^\pi + \Big.$$

$$\Big. + \sum_{t=0}^{T-1}\log p(s_{t+1}|h_t^\pi, a_t)\Big]. \tag{6}$$

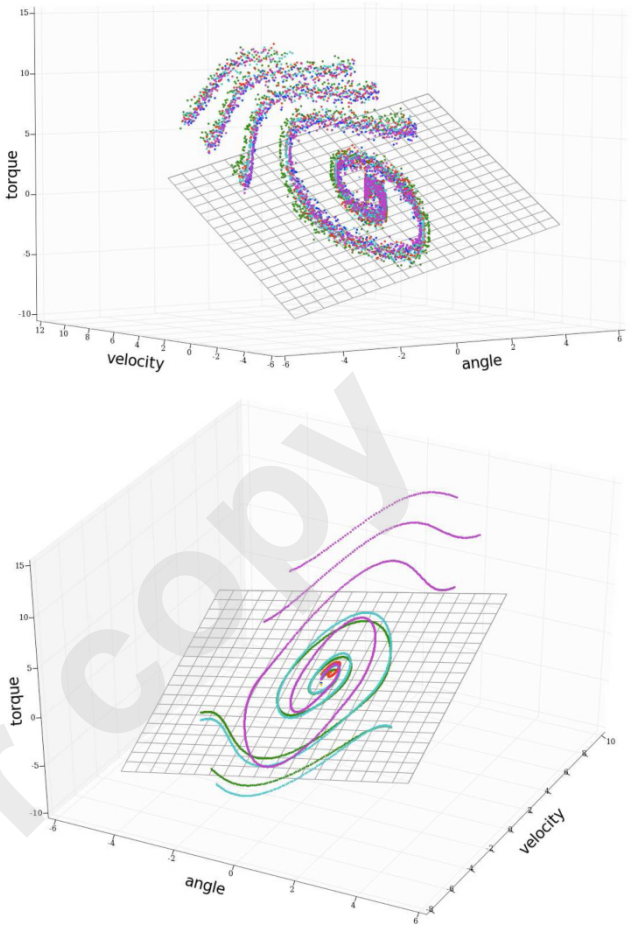


*Figure 2. Illustration of the main difference between action (left) and parameter (right) exploration. Several rollouts in state-action space of a task with state $x \in \mathbb{R}^2$ (velocity and angle axes) and action $a \in \mathbb{R}$ (torque axis) are plotted. While exploration based on action perturbation follows the same trajectory over and over again (with added noise), parameter exploration instead tries different* strategies *and can quickly find solutions that would take a long time to discover otherwise.*

On the right side of (6), only the policy $\pi$ is dependent on $\theta$, so the gradient can be simplified to

$$\nabla_\theta \log p(h^\pi) = \sum_{t=0}^{T-1}\nabla_\theta \log\pi(a_t|h_t^\pi). \tag{7}$$

We can now resubstitute this term into (5) and get

$$\nabla_\theta J(\pi) = \int p(h^\pi)\sum_{t=0}^{T-1}\nabla_\theta \log\pi(a_t|h_t^\pi)R(h^\pi)\,dh^\pi$$

$$= E\left\{\sum_{t=0}^{T-1}\nabla_\theta \log\pi(a_t|h_t^\pi)R(h^\pi)\right\}. \tag{8}$$

Unfortunately, the probability distribution $p(h^\pi)$ over the histories produced by $\pi$ is not known in general. Thus we need to approximate the expectation, e.g. by *Monte-Carlo sampling*. To this end, we collect $N$ samples through world interaction, where a single sample comprises a complete history $h^\pi$ (one episode or rollout) to which a return $R(h^\pi)$ can be assigned, and sum over all samples which basically yields Williams' [34] episodic REINFORCE gradient estimation:

$$\nabla_\theta J(\pi) \approx \frac{1}{N} \sum_{h^\pi} \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|h_t^\pi) R(h^\pi) \tag{9}$$

The derivation of eNAC is based on REINFORCE but taken further. It is an actor-critic method that uses the natural gradient to estimate the performance gradient. Its derivation can be found in [13].

### 2.4. Black-Box Optimization

The objective of optimization is to find parameters $x \in \mathcal{D}$ that maximize a given fitness function $f(x)$. Black-box optimization places little restrictions on $f$, in particular, it does not require $f$ to be differentiable, continuous or deterministic. Consequently, black-box optimization algorithms have access only to a number of *evaluations*, that is $(x, f(x))$-tuples, without the additional information (e.g., gradient, Hessian) typically assumed by other classes of optimization algorithms. Furthermore, as evaluations are generally considered costly, black-box optimization attempts to maximize $f$ with a minimal number of them. Such optimization algorithms allow domain experts to search for good or near-optimal solutions to numerous difficult real-world problems in areas ranging from medicine and finance to control and robotics.
The literature on real-valued blackbox optimization is too rich to be exhaustively surveyed. To our best knowledge, among various families of algorithms (e.g., hill-climbing, differential evolution [23], evolution strategies [15], immunological algorithms, particle swarm optimization [8] or estimation of distribution algorithms [10]), methods from the evolution strategy family seem to have an edge, particularly in cases where the fitness functions are high dimensional, non-separable or ill-shaped. The common approach in evolution strategies is to have a Gaussian mutation distribution maintained and updated through generations. The Gaussian mutation distribution is updated such that the probability of generating points with better fitness is high. Evolution strategies have been extended to account for correlations between the dimensions of the parameter space $\mathcal{D}$, leading to state of the art methods like covariance-matrix adaptation (CMA-ES) [6] and natural evolution strategies (see section 3.4).

## 3. Parameter-based Exploration

A significant problem with policy gradient algorithms such as REINFORCE [34] is that the high variance in the gradient estimation leads to slow convergence. Various approaches have been proposed to reduce this variance [1, 3, 14, 26]. 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 [12], since each state may depend on the entire sequence of previous samples. An alternative to the action-perturbing exploration described, as described in Section 2.2, is to manipulate the parameters $\theta$ of the policy directly. In this section we will start by showing how this parameter-based exploration can be realized (section 3.1) and provide a concrete algorithm for doing so (section 3.2). Realizing how this relates to black-box optimization, specifically evolution strategies (section 3.3), we then describe a related family of algorithms from that field (section 3.4). Finally we introduce a methodology for bridging the gap between action-based and parameter-based exploration (section 3.5).

### 3.1. Exploring in Parameter Space

Instead of manipulating the resulting actions, parameter exploration adds a small perturbation $\delta\theta$ directly to the parameter $\theta$ of the policy before each episode, and follow the resulting policy throughout the whole episode. *Finite differences* are a simple approach to estimate the gradient $\nabla_\theta J(\theta)$ towards better performance:

$$\nabla_\theta J(\theta) \approx \frac{J(\theta + \delta\theta) - J(\theta)}{\delta\theta} \tag{10}$$

In order to get a more accurate approximation, several parameter perturbations are usually collected (one for each episode) and the gradient is then estimated through linear regression. For this, we generate several rollouts by adding some exploratory noise to our policy parameters, resulting in an action $a = f(s; \theta + \delta\theta)$. From the rollouts, we generate the matrix $\Theta$ which has one row for each parameter perturbation $\delta\theta_i$. We also generate a column vector $J$ with the corresponding $J(\theta + \delta\theta)$ in each row:

$$\Theta_i = [\,\delta\theta_i \quad 1\,] \tag{11}$$
$$J_i = [\,J(\theta + \delta\theta_i)\,] \tag{12}$$

The ones in the right column of $\Theta$ are needed for the bias in the linear regression. The gradient can now be estimated with

$$\beta = (\Theta^T \Theta)^{-1} \Theta^T J \tag{13}$$

where the first $n$ elements of $\beta$ are the components of the gradient $\nabla_\theta J(\theta)$, one for each dimension of $\theta$.
Parameter-based exploration has several advantages. First, we no longer need to calculate the derivative of the policy with respect to its parameters, since we already know which choice of parameters has caused the changes in behavior. Therefore policies are no longer required to be differentiable, which in turn provides more flexibility in choosing a suitable policy for the task at hand.
Second, when exploring in parameter space, the resulting actions come from an instance of the same family of functions. This contrasts with action-perturbing exploration, which might result in actions that the underlying function could never have delivered itself. In the latter case, gradient descent could continue to change the parameters into a certain direction without improving the overall behavior. For example, in a neural network with sigmoid outputs, the explorative noise could push the action values above $+1.0$.
Third, parameter exploration avoids noisy trajectories that are due to adding i.i.d. noise in each timestep. This fact is illustrated in Figure 2. Each episode is executed entirely with the same parameters, which are only altered between episodes, resulting in much smoother action trajectories. Furthermore, this introduces much less variance in the rollouts, which facilitates credit assignment and generally leads to faster convergence [18, 22].
Lastly, the finite difference gradient information is much more accessible and simpler to calculate, as compared to the likelihood ratio gradient in [34] or [13].

### 3.2. Policy Gradients with Parameter-based Exploration

Following the idea above, in [22] it is proposed to use *policy gradients with parameter-based exploration*, where a distribution over the parameters of a controller is maintained and updated. Therefore PGPE explores purely in parameter space. 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 only on a single sample, the gradient estimates are significantly less noisy, even in stochastic environments.
In what follows, we briefly summarize [21], outlining the derivation that leads to PGPE. We give a short summary of the algorithm as far as it is needed for the rest of the paper.
Given the RL formulation from 2.1 we can associate a cumulative reward $R(h)$ 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 $\theta$ that maximize the agent's expected reward

$$J(\theta) = \int_H p(h|\theta)R(h)dh. \tag{14}$$

An obvious way to maximize $J(\theta)$ is to find $\nabla_\theta J$ and use it to carry out gradient ascent. Noting that $R(h)$ is independent of $\theta$, and using the standard identity $\nabla_x y(x) = y(x)\nabla_x \log y(x)$, we can write

$$\nabla_\theta J(\theta) = \int_H p(h|\theta)\nabla_\theta \log p(h|\theta)R(h)dh. \tag{15}$$

Assuming the environment is Markovian, and the states are conditionally independent of the parameters given the agent's choice of actions, we can write $p(h|\theta) = p(s_1)\prod_{t=1}^{T} p(s_{t+1}|s_t, a_t)p(a_t|s_t, \theta)$. Substituting this into Eq. (15) yields

$$\nabla_\theta J(\theta) = \int_H p(h|\theta)\sum_{t=1}^{T}\nabla_\theta p(a_t|s_t, \theta)R(h)dh. \tag{16}$$

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

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T}\nabla_\theta p(a_t^n|s_t^n, \theta)r(h^n), \tag{17}$$

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 $\theta$ 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 $\theta$, i.e.

$$p(a_t|s_t, \rho) = \int_\Theta p(\theta|\rho)\delta_{F_\theta(s_t),a_t}d\theta, \tag{18}$$

where $\rho$ are the parameters determining the distribution over $\theta$, $F_\theta(s_t)$ is the (deterministic) action chosen by the model with parameters $\theta$ in state $s_t$, and $\delta$ 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 $\rho$ is

$$J(\rho) = \int_\Theta\int_H p(h, \theta|\rho)R(h)dhd\theta. \tag{19}$$

Noting that $h$ is conditionally independent of $\rho$ given $\theta$, we have $p(h, \theta|\rho) = p(h|\theta)p(\theta|\rho)$ and therefore $\nabla_\rho \log p(h, \theta|\rho) = \nabla_\rho \log p(\theta|\rho)$, we have

$$\nabla_\rho J(\rho) = \int_\Theta\int_H p(h|\theta)p(\theta|\rho)\nabla_\rho \log p(\theta|\rho)R(h)dhd\theta, \tag{20}$$

where $p(h|\theta)$ is the probability distribution over the parameters $\theta$ and $\rho$ are the parameters determining the distribution over $\theta$. Clearly, integrating over the entire space of histories and parameters is unfeasible, and we therefore resort to sampling methods. This is done by first choosing $\theta$ from $p(\theta|\rho)$, then running the agent to generate $h$ from $p(h|\theta)$:

$$\nabla_\rho J(\rho) \approx \frac{1}{N}\sum_{n=1}^{N}\nabla_\rho \log p(\theta|\rho)r(h^n). \tag{21}$$

In the original formulation of PGPE, $\rho$ consisted of a set of means $\{\mu_i\}$ and standard deviations $\{\sigma_i\}$ that determine an independent normal distribution for each parameter $\theta_i$ in $\theta$ of the form $p(\theta_i|\rho_i) = \mathcal{N}(\theta_i|\mu_i, \sigma_i^2)$ Some rearrangement gives the following forms for the derivative of $\log p(\theta|\rho)$ with respect to $\mu_i$ and $\sigma_i$:

$$\begin{aligned}\nabla_{\mu_i} \log p(\theta|\rho) &= \frac{(\theta_i - \mu_i)}{\sigma_i^2}\\ \nabla_{\sigma_i} \log p(\theta|\rho) &= \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3},\end{aligned} \tag{22}$$

which can then be substituted into (21) to approximate the $\mu$ and $\sigma$ gradients that gives the PGPE update rules. Note the similarity to REINFORCE [34]. But in contrast to REINFORCE, $\theta$ defines the parameters of the model, not the probability of the actions.

### 3.3. Reinforcement Learning as Optimization

There is a double link between RL and optimization. On one hand, we may consider optimization to be a simple sub-problem of RL, with only a single state and a single timestep per episode, where the fitness corresponds to the reward (i.e. a bandit problem).
On the other hand, more interestingly, the return of a whole RL episode can be interpreted as a single fitness evaluation, where the parameters $x$ now map onto the policy parameters $\theta$. In this case, parameter-based exploration in RL is equivalent to black-box optimization. Moreover, when the exploration in parameter space is normally distributed, a directly link between RL and evolution strategies can be established.

### 3.4. Natural Evolution Strategies

The original PGPE can be seen as a stochastic optimization algorithm, where the parameters are adapted separately. However, in practice, parameters representing the policies are often strongly correlated, rendering per-parameter update less efficient.

The family of natural evolution strategies [24, 25, 33] offers a principled alternative by following the natural gradient of the expected fitness. NES maintains and iteratively updates a multivariate Gaussian mutation distribution. Parameters are updated by estimating a ***natural evolution gradient***, i.e., the natural gradient on the parameters of the mutation distribution, and following it towards better expected fitness. The evolution gradient obtained automatically takes into account the correlation between parameters, thus is particularly suitable here. A well-known advantage of natural gradient methods over 'vanilla' gradient ascent is isotropic convergence on ill-shaped fitness landscapes [2]. Although relying exclusively on function value evaluations, the resulting optimization behavior closely resembles second order optimization techniques. This avoids drawbacks of regular gradients which are prone to slow or even premature convergence.

The core idea of NES is its strategy adaptation mechanism: NES follows a sampled natural gradient of expected fitness in order to update (the parameters of) the search distribution. NES uses a Gaussian distribution with a fully adaptive covariance matrix, but it may in principle be used with a different family of search distributions. NES exhibits the typical characteristics of evolution strategies. It maintains a population of vector-valued candidate solutions, and samples new offspring and adapts its search distribution generation-wise. The essential concepts of NES are briefly revisited in the following.

We collect the parameters of the Gaussian, the mean $\mu$ and the covariance matrix $C$, in the variable $\theta = (\mu, C)$. However, to sample efficiently from this distribution we need a square root of the covariance matrix (a matrix $A$ fulfilling $AA^T = C$). Then $x = \mu + Az$ transforms a standard normal vector $z \sim \mathcal{N}(0, I)$ into a sample $x \sim \mathcal{N}(\mu, C)$. Let

$$p(x \mid \theta) = \frac{1}{(2\pi)^{\frac{d}{2}} \det(A)} \cdot \exp\left(-\frac{1}{2}\left\|A \cdot (x - \mu)\right\|^2\right)$$

denote the density of the normal search distribution $\mathcal{N}(\mu, C)$, and let $f(x)$ denote the fitness of sample $x$ (which is of dimension $d$). Then,

$$J(\theta) = \mathbb{E}[f(x) \mid \theta] = \int f(x)\, p(x \mid \theta)\, dx \tag{23}$$

is the expected fitness under the search distribution given by $\theta$. Again, the log-likelihood trick enables us to write

$$\nabla_\theta J(\theta) = \int \left[f(x)\, \nabla_\theta \log(p(x \mid \theta))\right] p(x \mid \theta)\, dx \ .$$

From this form we obtain the Monte Carlo estimate

$$\nabla_\theta J(\theta) \approx \nabla_\theta \widehat{J}(\theta) = \frac{1}{n} \sum_{i=1}^{n} f(x_i)\, \nabla_\theta \log(p(x \mid \theta))$$

of the expected fitness gradient. For the Gaussian search distribution $\mathcal{N}(\mu, C)$, the term $\nabla_\theta \log(p(x \mid \theta))$ can be computed efficiently, see e.g. [24].

Instead of using the stochastic gradient directly for updates, NES follows the ***natural*** gradient [2]. In a nutshell, the natural gradient amounts to $G = F^{-1} \nabla_\theta J(\theta)$, where $F$ denotes the Fisher information matrix of the parametric family of search distributions. Natural gradient ascent has well-known advantages over vanilla gradient ascent. Most prominently it results in isotropic convergence on ill-shaped fitness landscapes because the natural gradient is invariant under linear transformations of the search space.

Additional techniques were developed to enhance NES' performance and viability, including importance mixing to reduce the number of required samples [24, 25], and exponential parametrization of the search distribution to guarantee invariance while at the same time providing an elegant and efficient way of computing the natural gradient without the need of the explicit Fisher information matrix or its costly inverse (under review). NES' results are now comparable to the well-known CMA-ES [5, 9] algorithm, the de facto 'industry standard' for continuous black-box optimization.

### 3.5. State-Dependent Exploration

An alternative to parameter-based exploration, that addresses most of the shortcomings of action-based exploration is State-Dependent Exploration [18]. Its core benefit is that it is compatible with standard policy gradient methods like REINFORCE in a way that it can simply replace or augment the existing Gaussian exploration described in Section 2.2 and Equation 2. Actions are generated as follows, where $f$ is the parameterized function approximator:

$$a = f(s, \theta) + \hat{\epsilon}(s, \hat{\theta}), \quad \hat{\theta} \sim \mathcal{N}(0, \hat{\sigma}^2). \tag{24}$$

Instead of adding i.i.d. noise in each time step (cf. Equation 2), [18] introduces a pseudo-random function $\hat{\epsilon}(s)$, that takes the current state as input and itself is parameterized with parameters $\hat{\theta}$. These exploration parameters are in turn drawn from a Normal distribution with zero mean. The exploration parameters are varied between episodes (just like introduced in Section 3.1) and held constant during the rollout. Therefore, the exploration function $\hat{\epsilon}$ can still carry the necessary exploratory randomness through variation between episodes, but will always return the same value in the same state within an episode.

Effectively, by drawing $\hat{\theta}$, we actually create a ***policy delta***, similar to finite difference methods. In fact, if both $f(s; \Theta)$ with $\Theta = [\theta_{ji}]$ and $\hat{\epsilon}(x, \hat{\Theta})$ with $\hat{\Theta} = [\hat{\theta}_{ji}]$ are linear functions, we see that

$$\begin{aligned} a &= f(s; \Theta) + \hat{\epsilon}(s; \hat{\Theta}) \\ &= \Theta s + \hat{\Theta} s \\ &= (\Theta + \hat{\Theta}) s, \end{aligned} \tag{25}$$

which shows that direct parameter perturbation methods (cf. Equation (10)) are a special case of SDE and can be expressed in this more general framework.

In effect, state-dependent exploration can be seen as a converter from action-exploring to parameter-exploring methods. A method equipped with the SDE converter does not benefit from all the advantages mentioned in Section 3.1, e.g. actions are not chosen from the same family of functions, since the exploration value is still added to the greedy action. It does, however, cause smooth trajectories and thus mitigates the credit assignment problem (as illustrated in Figure 2).

For a linear exploration function $\hat{\epsilon}(s; \Theta) = \Theta s$ it is also possible to calculate the derivative of the log likelihood with respect to the variance.

Following the derivation in [18], we see that the action element $a_j$ is distributed as

$$a_j \sim \mathcal{N}(f_j(s,\Theta), \sum_i (s_i \hat{\sigma}_{ji})^2). \quad (26)$$

Therefore, differentiation of the policy with respect to the free parameters $\hat{\sigma}_{ji}$ yields:

$$\begin{aligned} \frac{\partial \log \pi(a|s)}{\partial \hat{\sigma}_{ji}} &= \sum_k \frac{\partial \log \pi_k(a_k|s)}{\partial \sigma_j} \frac{\partial \sigma_j}{\partial \hat{\sigma}_{ji}} \\ &= \frac{(a_j - \mu_j)^2 - \sigma_j^2}{\sigma_j^4} s_i^2 \hat{\sigma}_{ji}, \end{aligned} \quad (27)$$

which can directly be inserted into the gradient estimator of REINFORCE. For more complex exploration functions, calculating the exact derivative for the sigma adaptation might not be possible and heuristic or manual adaptation (e.g. with slowly decreasing $\hat{\sigma}$) is required.

# 4. Experimental Results

In this section we compare the reviewed parameter-exploring methods PGPE and NES to action-exploring policy gradient algorithms REINFORCE and eNAC on several simulated control scenarios. We also demonstrate how both policy gradient algorithms can perform better when equipped with the SDE converter to act more like parameter-exploring methods. The experiments are all executed in simulations, but their complexity is similar to today's real-life RL problems [11, 14].
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. All the experiments were conducted with empirically tuned meta-parameters. The benchmarks and algorithms are included in the open source Machine Learning library PyBrain [20].
Section 4.1 describes each of the experiments briefly, and section 4.2 lists the results which are discussed in section 4.3.

## 4.1. Benchmark Environments

### 4.1.1. Pole Balancing

The first scenario is the extended pole balancing benchmark as described in [17]. Pole balancing is a standard benchmark in reinforcement learning. In contrast to [17] however, we do not initialize the controller with a previously chosen stabilizing policy but rather start with random policies, which makes the task more difficult. 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.

### 4.1.2. Biped Robust Standing

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 [29] 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 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 is a Jordan network [7] with 41 inputs, 20 hidden units and 11 output units. 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 Figure 3) that would knock it over unless it counterbalanced. Figure 3 shows a typical scenario of the robust standing task.

### 4.1.3. Object Grasping

The task in this scenario was to grasp an object from a table. The simulation, based on the CCRL robot [4] 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 7 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 [7] with 35 inputs, 1 hidden units and 7 output units.
The system has only 7 DoF while having 8 joints, because the actual grasping is realized as a *reflex*. Has the hand reached the center of gravity of the object sufficiently close enough the hand closes automatically. Other simplifications are that the object is always at the same position at the very edge of the table.
By this simplifications the task is easy enough to be learned from scratch within 20,000 episodes. The needed controller could be constructed small enough to allow also methods that use a covariance matrix to learn the task. Figure 4 shows a typical solution of the grasping task.

### 4.1.4. Ball Catching

This series of experiments is based on a simulated robot hand with realistically modeled physics. We chose this experiment to show the predominance of policy gradients equipped with SDE, especially in a realistic robot task. We used the Open Dynamics Engine to model the hand, arm, body, and object. The arm has 3 degrees of freedom: shoulder, elbow, and wrist, where each joint is assumed to be a 1D hinge joint, which limits the arm movements to forward-backward and up-down. The hand itself consists of 4 fingers with 2 joints each, but for simplicity we only use a single actor to move all finger joints together, which gives the system the possibility to open and close the hand, but it cannot control individual fingers. These limitations to hand and arm movement reduce the overall complexity of the task while giving the system enough freedom to catch the ball. A 3D visualization of the robot attempting a catch is shown in Figure 5.
The reward function is defined as follows: upon release of the ball, in each time step the reward can either be $-3$ if the ball hits the ground (in which case the episode is considered a failure, because the system cannot recover from it) or else the negative distance between ball center and palm center, which can be any value between $-3$ (we capped the distance at 3 units) and $-0.5$ (the closest possible distance considering the palm heights and ball radius). The return for a whole episode is the mean over the episode: $R = \frac{1}{N}\sum_{n=1}^{N} r_t$. In practice, we found an overall episodic return of $-1$ or better to represent nearly optimal catching behavior, considering the time from ball release to impact on

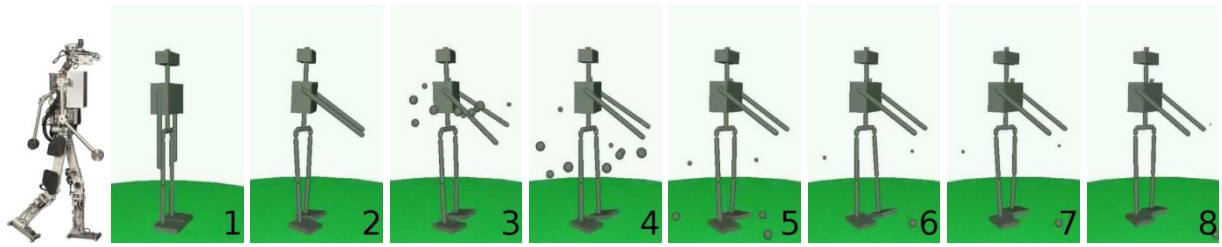


*Figure 3.* *The original Johnnie (left). From left to right, a typical solution which worked well in the robust standing task is shown: 1. Initial posture. 2. Stable posture. 3. Perturbation by heavy weights that are thrown randomly at the robot. 4. - 7. Backsteps right, left, right, left. 8. Stable posture regained.*

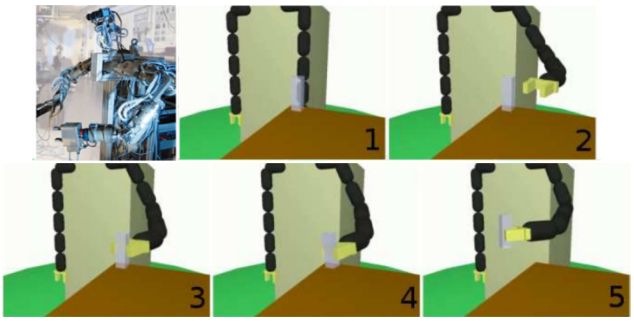


*Figure 4.* *The original CCRL robot (left). From left to right, a typical solution which worked well in the object grasping task is shown: 1. Initial posture. 2. Approach. 3. Enclose. 4. Take hold. 5. Lift.*

palm, which is penalized with the capped distance to the palm center.

## 4.2. Results

We present the results of our experiments below, ordered by benchmark. All plots show performance (e.g. average returns) over episodes. The solid lines are the mean over many repetitions of the same experiments, while the thick bars represent the variance. The thin vertical lines indicate best and worst performance of all repetitions. The Ball Catching experiment was repeated 100 times, the Object Grasping experiment was repeated 10 times. All other experiments were repeated 40 times.

### 4.2.1. Pole Balancing

For the first set of experiments, we compared PGPE to REINFORCE with varying action perturbation probabilities. Instead of changing the additive $\epsilon$ from Eqn. (2) in every time step, the probability of drawing a new $\epsilon$ was set to 0.125, 0.25, 0.5 and 1.0 respectively (the last one being the original REINFORCE again). Figure 6 shows the results. PGPE clearly outperformed all versions of REINFORCE, finding a better solution in shorter time. Original REINFORCE showed worst performance. The smaller the probability was to change the perturbation, the faster REINFORCE improved.

A second experiment on the pole balancing benchmark was conducted, comparing PGPE, NES and eNAC directly. The results are illustrated in Figure 7. NES converged fastest while PGPE came second. The action-perturbing eNAC got stuck in a plateau after approx. 1000 episodes but eventually recovered and found a equally good solution, although much slower.

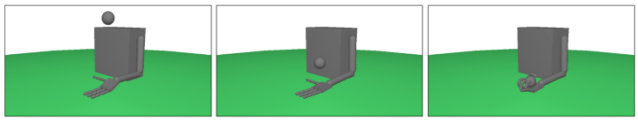

*Figure 5.* *Visualization of the simulated robot hand while catching a ball. The ball is released above the palm with added noise in x and y coordinates. When the fingers grasp the ball and do not release it throughout the episode, the best possible return (close to $-1.0$) is achieved.*

### 4.2.2. Biped Robust Standing

This complex, high-dimensional task was executed with REINFORCE and its parameter-exploring version PGPE, as shown in Figure 8. While PGPE is slower at first, it quickly overtakes REINFORCE and finds a robust posture in less than 1000 episodes, whereas REINFORCE needs twice as many episodes to reach that performance.

### 4.2.3. Object Grasping

The object grasping experiment compares the two parameter-exploring algorithms PGPE and NES. Object Grasping is a mid dimensional task. As can be seen in Figure 9 with a parameter dimension of 48 both algorithms perform nearly the same. Both algorithms manage to learn to grasp the object from scratch in reasonable time.

### 4.2.4. Ball Catching

Two experiments demonstrate the performance of eNAC and REINFORCE, both with and without SDE. The results are shown in Figure 10 and 11, respectively. REINFORCE enhanced with SDE clearly exceed its action-exploring counterpart, finding a superior solution with very low variance. REINFORCE without SDE on the other hand has a very high variance, which indicates that it finds good solutions sometimes while it gets stuck early on at other times. The same experiment was repeated with eNAC as policy gradient method, again with and without SDE. While the results are not as clear as in the case of REINFORCE, SDE still improved the performance significantly.

## 4.3. Discussion

### 4.3.1. PGPE

We previously asserted that the lower variance of PGPE's gradient estimates compared to action exploring methods is partly due to the fact that PGPE requires only one parameter sample per history, whereas

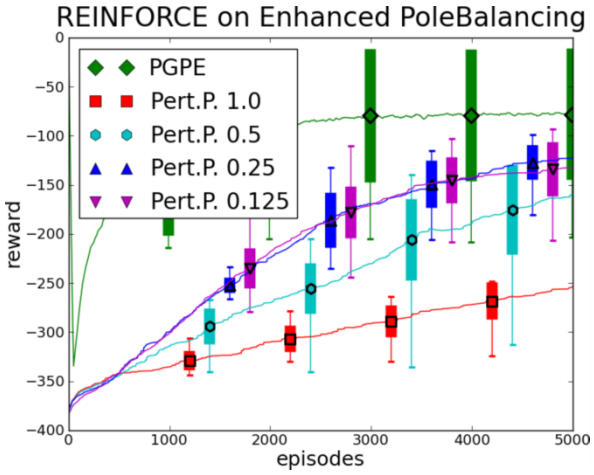


*Figure 6.* *REINFORCE on the pole balancing task, with various action perturbation probabilities (1, 0.5, 0.25, 0.125). PGPE is shown for reference.*

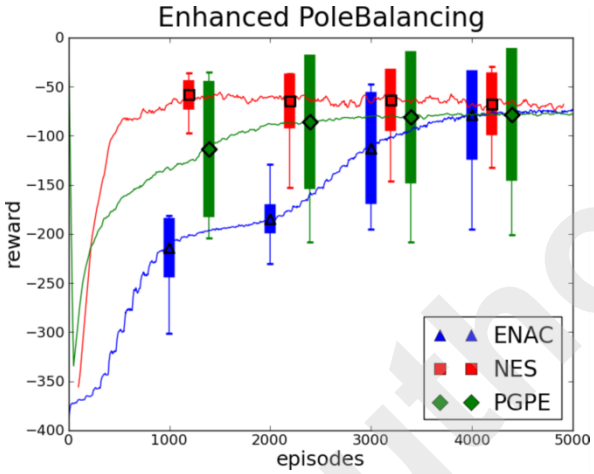


*Figure 7.* *PGPE and NES compared to eNAC on the pole balancing benchmark.*

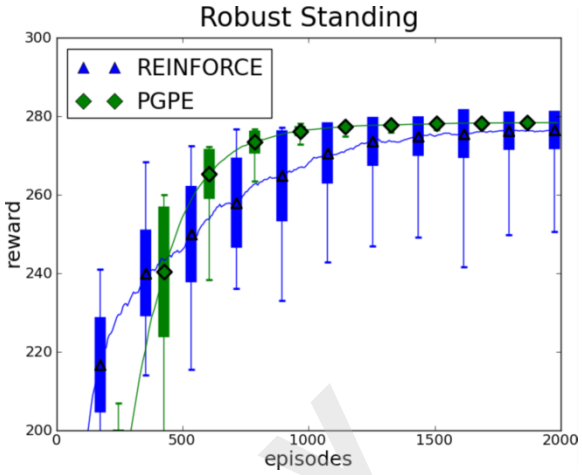


*Figure 8.* *PGPE compared to REINFORCE on the robust standing benchmark.*

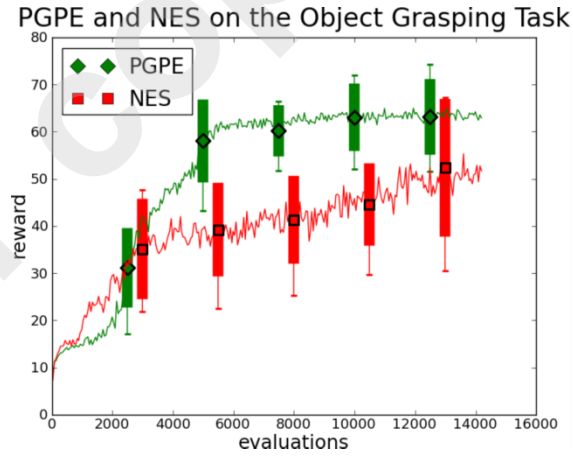


*Figure 9.* *PGPE and NES on the object grasping benchmark.*

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.

As can be seen in Figure 6 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 behavior. Because PGPE also uses only the log likelihood gradient for parameter update and does not adapt the full covariance matrix but only uses a single variance parameter per exploration dimension, the difference in performance is solely based on the different exploration strategies.

In Figure 8 PGPE is compared to its direct competitor and ancestor REINFORCE. The robust standing benchmark should clearly favor REINFORCE, because the policy has a lot more parameters than it has output dimensions (1111 parameters to 11 DoF) and is further quite noisy. Both properties are said to challenge parameter-exploring algorithms [14]. While these disadvantages are most likely responsible for the worse performance of PGPE in the beginning (less than 500 episodes), PGPE can still overtake REINFORCE and find a better solution is shorter time.

### 4.3.2. NES

NES has shown its superiority in the Pole Balancing experiment, and demonstrated its real strength, difficult low-dimensional tasks, which it derived from its origins in CMA-ES. Like eNAC it uses a natural gradient for the parameter updates of the policy. The difference in performance is therefore only due to the adaptation of the full covariance matrix and the exploration in parameter space.

eNAC's convergence temporarily slows down after 1000 episodes, where it reaches a plateau. By looking at the intermediate solution of

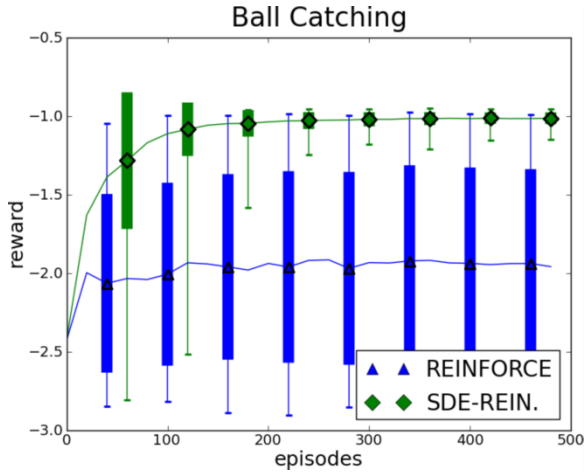


*Figure 10.* *REINFORCE compared to the SDE version of REINFORCE. While SDE managed to learn to catch the ball quickly in every single case, original REINFORCE occasionally found a good solution, but in most cases did not learn to catch the ball.*

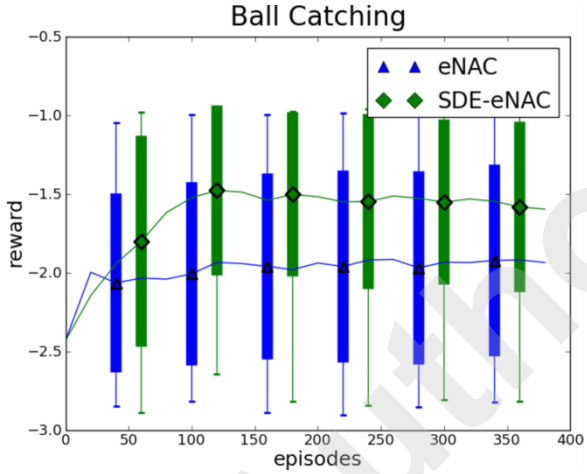


*Figure 11.* *eNAC compared to the SDE version. Both learning curves had relatively high variances. While original eNAC often didn't find a good solution, SDE found a catching behavior in almost every case, but many times lost it again due to continued exploration, hence the high variance.*

eNAC on plateau level, we discovered an interesting reason for this behavior. eNAC learns to balance the pole first, and only later proceeds to learn to drive to the middle of the area. NES and PGPE (and in fact all tested parameter-exploring techniques, including CMA-ES) learn both subtasks simultaneously. This is because action-perturbing methods try out small action variations and thus are more greedy. In effect, they learn small subtasks first. Parameter exploring methods on the other hand vary the strategy in each episode and are thus able to find the overall solution more directly. Admittedly, while the latter resulted in faster convergence in this task, it is possible that action perturbing methods can use the greedyness to their advantage, by learning step-by-step rather than tackling the big problem at once. This issue should be examined further in a future publication.

We also compared NES to PGPE on the Object Grasping task with a medium number of dimensions. As can be seen in Figure 9 with a parameter dimension of 48 both algorithms perform similar. This underlines that NES is a specialist on difficult low-dimensional problems while PGPE was constructed to cope with high-dimensional problems.

### 4.3.3. SDE

The experiment that supports our claim most is probably the Ball Catching task, where REINFORCE and eNAC are compared to their SDE-converted counterparts. The underlying algorithms are exactly the same, only the exploration method has been switched from a normally distributed action exploration to a state-dependent exploration. As mentioned in section 3.5, this does not make them fully parameter-exploring, but it does carry over some of its advantageous properties.

As can be seen in Figures 10 and 11, the proposed kind of exploration results in a completely different quality of the found solutions. The high difference of performance on this challenging task, where torques are directly affected by the output of the controller policy, strengthens our arguments for parameter exploration further.

## 5. Conclusion

We studied the neglected issue of exploration for continuous RL, especially Policy Gradient methods. Parameter-based exploration has many advantages over perturbing actions by Gaussian noise. Several of our novel RL algorithms profit from it, despite stemming from different subfields and pursuing different goals. All find reliable gradient estimates and outperform traditional policy gradient methods in many applications, converging faster and often also more reliably than standard PG methods with random exploration of action space.

SDE replaces the latter by state dependent exploration searching the parameter space of an exploration function approximator. It combines the advantages of policy gradients, in particular the advanced gradient estimation techniques found in eNAC, with the reduced variance of parameter-exploring methods. PGPE goes one step further and explores by perturbing the policy parameters directly. It lacks the methodology of existing PG methods, but works also for non-differentiable controllers, learning to execute smooth trajectories. NES finally combines desirable properties of CMA-ES with natural gradient estimates that replace the population/distribution-based convergence mechanism of CMA, while keeping the covariance matrix for more informed exploration.

We believe that parameter-based exploration should play a more important role not only for PG methods but for continuous RL in general, and continuous value-based RL in particular. This is subject of ongoing research.

## References

[1] D. Aberdeen. Policy-Gradient Algorithms for Partially Observable Markov Decision Processes. PhD thesis, Australian National University, 2003.

[2] S. Amari and S. C. Douglas. Why natural gradient? In Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98), volume 2, pages 1213–1216, 1998.

[3] J. Baxter and P. L. Bartlett. Reinforcement learning in POMDPs via direct gradient ascent. In Proc. 17th International Conf. on Ma-

chine Learning, pages 41–48. Morgan Kaufmann, San Francisco, CA, 2000.
[4] M. Buss and S. Hirche. Institute of Automatic Control Engineering, TU München, Germany, 2008. http://www.lsr.ei.tum.de/.
[5] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation, 9 (2):159–195, 2001.
[6] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation, 11(1):1–18, 2003.
[7] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. Proc. of the Eighth Annual Conference of the Cognitive Science Society, 8:531–546, 1986.
[8] J. Kennedy, R. C. Eberhart, et al. Particle swarm optimization. In Proceedings of IEEE international conference on neural networks, volume 4, pages 1942–1948. Piscataway, NJ: IEEE, 1995.
[9] S. Kern, S. D. Müller, N. Hansen, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms–a comparative review. Natural Computing, 3(1):77–112, 2004.
[10] P. Larranaga and J. A. Lozano. Estimation of distribution algorithms: A new tool for evolutionary computation. Kluwer Academic Pub, 2002.
[11] H. Müller, M. Lauer, R. Hafner, S. Lange, A. Merke, and M. Riedmiller. Making a robot learn to play soccer. Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI-2007), 2007.
[12] R. Munos and M. Littman. Policy gradient in continuous time. Journal of Machine Learning Research, 7:771–791, 2006.
[13] J. Peters and S. Schaal. Natural actor-critic. Neurocomputing, 71 (7-9):1180–1190, 2008.
[14] J. Peters and S. Schaal. Policy gradient methods for robotics. In Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006.
[15] I. Rechenberg. Evolution strategy. Computational Intelligence: Imitating Life, pages 147–159, 1994.
[16] M. Riedmiller. Neural fitted Q iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. Lecture notes in computer science, 3720:317, 2005.
[17] M. Riedmiller, J. Peters, and S. Schaal. Evaluation of policy gradient methods and variants on the cart-pole benchmark. In Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007.
[18] T. Rückstieß, M. Felder, and J. Schmidhuber. State-Dependent Exploration for policy gradient methods. In European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2008, Part II, LNAI 5212, pages 234–249, 2008.
[19] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In International symposium on robotics research. Citeseer, 2004.
[20] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. Journal of Machine Learning Research, 11:743–746, 2010.
[21] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Parameter-exploring policy gradients. Neural Networks, Special Issue, December 2009.
[22] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Policy gradients with parameter-based exploration for control. In Proceedings of the International Conference on Artificial Neural Networks ICANN, 2008.
[23] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11(4):341–359, 1997.
[24] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Stochastic Search using the Natural Gradient. In International Conference on Machine Learning (ICML), 2009.
[25] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Efficient Natural Evolution Strategies. In Genetic and Evolutionary Computation Conference (GECCO), 2009.
[26] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. NIPS-1999, pages 1057–1063, 2000.
[27] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.
[28] S. B. Thrun. The role of exploration in learning control. Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches, pages 527–559, 1992.
[29] H. Ulbrich. Institute of Applied Mechanics, TU München, Germany, 2008. http://www.amm.mw.tum.de/.
[30] H. van Hasselt and M. Wiering. Reinforcement learning in continuous action spaces. In Proc. 2007 IEEE Symp. Approx. Dynamic Programming and Reinforcement Learning, volume 272, page 279. Citeseer, 2007.
[31] C. J. C. H. Watkins and P. Dayan. Q-learning. Machine Learning, 8(3):279–292, 1992.
[32] M. Wiering and J. Schmidhuber. Efficient Model-Based Exploration. From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior, 1998.
[33] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural evolution strategies. In IEEE World Congress on Computational Intelligence (WCCI 2008), 2008.
[34] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8:229–256, 1992.