State-Dependent Exploration for Policy Gradient Methods

Thomas Rückstieß, Martin Felder, and Jürgen Schmidhuber

Technische Universität München, 85748 Garching, Germany
{ruecksti,felder,schmidhu}@in.tum.de

Abstract. Policy Gradient methods are model-free reinforcement learning algorithms which in recent years have been successfully applied to many real-world problems. Typically, Likelihood Ratio (LR) methods are used to estimate the gradient, but they suffer from high variance due to random exploration at every time step of each training episode. Our solution to this problem is to introduce a state-dependent exploration function (SDE) which during an episode returns the same action for any given state. This results in less variance per episode and faster convergence. SDE also finds solutions overlooked by other methods, and even improves upon state-of-the-art gradient estimators such as Natural Actor-Critic. We systematically derive SDE and apply it to several illustrative toy problems and a challenging robotics simulation task, where SDE greatly outperforms random exploration.

1 Introduction

Reinforcement Learning (RL) is a powerful concept for dealing with semi-supervised control tasks. There is no teacher to tell the agent the correct action for a given situation, but it does receive feedback (the *reinforcement signal*) about how well it is doing. While exploring the space of possible actions, the reinforcement signal can be used to adapt the parameters governing the agent's behavior. Classical RL algorithms [1,2] are designed for problems with a limited, discrete number of states. For these scenarios, sophisticated exploration strategies can be found in the literature [3,4].

In contrast, Policy Gradient (PG) methods as pioneered by Williams [5] can deal with continuous states and actions, as they appear in many real-life settings. They can handle function approximation, avoid sudden discontinuities in the action policy during learning, and were shown to converge at least locally [6]. Successful applications are found e.g. in robotics [7], financial data prediction [8] or network routing [9].

However, a major problem in RL remains that feedback is rarely available at every time step. Imagine a robot trying to exit a labyrinth within a set time, with a default policy of driving straight. Feedback is given at the end of an *episode*, based on whether it was successful or not. PG methods most commonly use a random exploration strategy [5,7], where the deterministic action ("if wall ahead, go straight") at each time step is perturbed by Gaussian noise. This way,

the robot may wiggle free from time to time, but it is very hard to improve the policy based on this success, due to the high variance in the gradient estimation. Obviously, a lot of research has gone into devising smarter, more robust ways of estimating the gradient, as detailed in the excellent survey by Peters [7].

Our novel approach is much simpler and targets the exploration strategy instead: In the example, the robot would use a deterministic function providing an exploration offset consistent throughout the episode, but still depending on the state. This might easily change the policy into something like "if wall ahead, veer a little left", which is much more likely to lead out of the labyrinth, and thus can be identified easily as a policy improvement. Hence, our method, which we call *state-dependent exploration* (SDE), causes considerable variance reduction and therefore faster convergence. Because it only affects exploration and does not depend on a particular gradient estimation technique, SDE can be enhanced with any episodic likelihood ratio (LR) method, like REINFORCE [5], GPOMDP [10], or ENAC [11], to reduce the variance even further.

Our exploration strategy is in a sense related to Finite Difference (FD) methods like SPSA [12] as both create policy deltas (or strategy variations) rather than perturbing single actions. However, direct parameter perturbation has to be handled with care, since small changes in the policy can easily lead to unforseen and unstable behavior and a fair amount of system knowledge is therefore necessary. Furthermore, FD are very sensitive to noise and hence not suited for many real-world tasks. SDE does not suffer from these drawbacks—it embeds the power of FD exploration into the stable LR framework.

The remainder of this paper is structured as follows: Section 2 introduces the policy gradient framework together with a thorough derivation of the equations and applications to function approximation. Our novel exploration strategy SDE will be explained in detail in section 3. Experiments and their results are described in section 4. The paper concludes with a short discussion in section 5.

2 Policy Gradient Framework

An advantage of policy gradient methods is that they don't require the environment to be Markovian, i.e. each controller action may depend on the whole history encountered. So we will introduce our policy gradient framework for general non-Markovian environments but later assume a Markov Decission Process (MDP) for ease of argument.

2.1 General Assumptions

A policy $\pi(u|h,\theta)$ is the probability of taking action u when encountering history h under the policy parameters θ. Since we use parameterized policies throughout this paper, we usually ommit θ and just write $\pi(u|h)$. We will use h^π for the history of all the observations x, actions u, and rewards r encountered when following policy π. The history at time $t=0$ is defined as the sequence $h_0^\pi = \{x_0\}$, consisting only of the start state x_0. The history at time t consists of

all the observations, actions and rewards encountered so far and is defined as $h_t^\pi = \{x_0, u_0, r_0, x_1, \dots, u_{t-1}, r_{t-1}, x_t\}$.

The return for the controller whose interaction with the environment produces history h^π is written as $R(h^\pi)$, which is defined as $R(h^\pi) = a_\Sigma \sum_{t=0}^{T} a_D\, r_t$ with $a_\Sigma = (1-\gamma)$, $a_D = \gamma^t$ for discounted (possibly continuous) tasks and $a_\Sigma = 1/T$, $a_D = 1$ for undiscounted (and thus necessarily episodic) tasks. In this paper, we deal with episodic learning and therefore will use the latter definition. The expectation operator is written as $E\{\cdot\}$.

The overall performance measure of policy π, independent from any history h, is denoted $J(\pi)$. It is defined as $J(\pi) = E\{R(h^\pi)\} = \int p(h^\pi)R(h^\pi)\, dh^\pi$. Instead of $J(\pi)$ for policy π parameterized with θ, we will also write $J(\theta)$.

To optimize policy π, we want to move the parameters θ along the gradient of J to an optimum with a certain learning rate α:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi). \tag{1}$$

The gradient $\nabla_\theta J(\pi)$ is

$$\nabla_\theta J(\pi) = \nabla_\theta \int_{h^\pi} p(h^\pi)R(h^\pi)\, dh^\pi = \int_{h^\pi} \nabla_\theta p(h^\pi)R(h^\pi)\, dh^\pi. \tag{2}$$

2.2 Likelihood Ratio Methods

Rather than perturbing the policy directly, as it is the case with FD methods [12,7], LR methods [5] perturb the resulting action instead, leading to a stochastic policy (which we assume to be differentiable with respect to its parameters θ), such as

$$u = f(h, \theta) + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2) \tag{3}$$

where f is the controller and ϵ the exploration noise. Unlike FD methods, the new policy that leads to this behavior is not known and consequently the difference quotient

$$\frac{\partial J(\theta)}{\partial \theta_i} \approx \frac{J(\theta + \delta\theta) - J(\theta)}{\delta \theta_i} \tag{4}$$

can not be calculated. Thus, LR methods use a different approach in estimating $\nabla_\theta J(\theta)$.

Following the derivation of e.g. Wierstra et al. [13], we start with the probability of observing history h^π under policy π, which is given by the probability of starting with an initial observation x_0, multiplied by the probability of taking action u_0 under h_0, multiplied by the probability of receiving the next observation x_1, and so on. Thus, (5) gives the probability of encountering a certain history h^π.

$$p(h^\pi) = p(x_0) \prod_{t=0}^{T-1} \pi(u_t | h_t^\pi)\, p(x_{t+1} | h_t^\pi, u_t) \tag{5}$$

Inserting this into (2), 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

$$\nabla_\theta J(\pi) = \int \frac{p(h^\pi)}{p(h^\pi)} \nabla_\theta p(h^\pi) R(h^\pi)\, dh^\pi \tag{6}$$

$$= \int p(h^\pi) \nabla_\theta \log p(h^\pi) R(h^\pi)\, dh^\pi. \tag{7}$$

For now, let us consider the gradient $\nabla_\theta \log p(h^\pi)$. Substituting the probability $p(h^\pi)$ according to (5) gives

$$\begin{aligned} \nabla_\theta \log p(h^\pi) &= \nabla_\theta \log \Big[p(x_0) \prod_{t=0}^{T-1} \pi(u_t|h_t^\pi)\, p(x_{t+1}|h_t^\pi, u_t) \Big] \\ &= \nabla_\theta \Big[\log p(x_0) + \sum_{t=0}^{T-1} \log \pi(u_t|h_t^\pi) + \sum_{t=0}^{T-1} \log p(x_{t+1}|h_t^\pi, u_t) \Big]. \end{aligned} \tag{8}$$

On the right side of (8), only the policy π is dependent on θ, so the gradient can be simplified to

$$\nabla_\theta \log p(h^\pi) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi(u_t|h_t^\pi). \tag{9}$$

We can now resubstitute this term into (7) and get

$$\begin{aligned} \nabla_\theta J(\pi) &= \int p(h^\pi) \sum_{t=0}^{T-1} \nabla_\theta \log \pi(u_t|h_t^\pi) R(h^\pi)\, dh^\pi \\ &= E\left\{ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(u_t|h_t^\pi) R(h^\pi) \right\}. \end{aligned} \tag{10}$$

Unfortunately, the probability distribution $p(h^\pi)$ over the histories produced by π 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^π (one episode or rollout) to which a return $R(h^\pi)$ can be assigned, and sum over all samples which basically yields Williams' [5] episodic REINFORCE gradient estimation:

$$\nabla_\theta J(\pi) \approx \frac{1}{N} \sum_{h^\pi} \sum_{t=0}^{T-1} \nabla_\theta \log \pi(u_t|h_t^\pi) R(h^\pi) \tag{11}$$

At this point there are several approaches to improve gradient estimates, as mentioned in the introduction. Neither these nor ideas like baselines [5], the PEGASUS trick [14] or other variance reduction techniques [15] are treated here. They are complementary to our approach, and their combination with SDE will be covered by a future paper.

2.3 Application to Function Approximation

Here we describe how the results above, in particular (11), can be applied to general parametric function approximation. Because we are dealing with multi-dimensional states $\boldsymbol{x}$ and multi-dimensional actions $\boldsymbol{u}$, we will now use bold font for (column) vectors in our notation for clarification.

To avoid the issue of a growing history length and to simplify the equations, we will assume the world to be Markovian for the remainder of the paper, i.e. the current action only depends on the last state encountered, so that $\pi(u_t|h_t^\pi) = \pi(u_t|x_t)$. But due to its general derivation, the idea of SDE is still applicable to non-Markovian environments.

The most general case would include a multi-variate normal distribution function with a covariance matrix $\boldsymbol{\Sigma}$, but this would square the number of parameters and required samples. Also, differentiating this distribution requires calculation of $\boldsymbol{\Sigma}^{-1}$, which is time-consuming. We will instead use a simplification here and add independent uni-variate normal noise to each element of the output vector seperately. This corresponds to a covariance matrix $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_n)$.[1] The action $\boldsymbol{u}$ can thus be computed as

$$\boldsymbol{u} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{e} = \begin{bmatrix} f_1(\boldsymbol{x}, \boldsymbol{\theta}) \\ \vdots \\ f_n(\boldsymbol{x}, \boldsymbol{\theta}) \end{bmatrix} + \begin{bmatrix} e_1 \\ \vdots \\ e_n \end{bmatrix} \tag{12}$$

with $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots]$ being the parameter vector and f_j the jth controller output element. The exploration values e_j are each drawn from a normal distribution $e_j \sim \mathcal{N}(0, \sigma_j^2)$. The policy $\boldsymbol{\pi}(\boldsymbol{u}|\boldsymbol{x})$ is the probability of executing action $\boldsymbol{u}$ when in state $\boldsymbol{x}$. Because of the independence of the elements, it can be decomposed into $\boldsymbol{\pi}(\boldsymbol{u}|\boldsymbol{x}) = \prod_{k \in \mathbb{O}} \pi_k(u_k|\boldsymbol{x})$ with $\mathbb{O}$ as the set of indices over all outputs, and therefore $\log \boldsymbol{\pi}(\boldsymbol{u}|\boldsymbol{x}) = \sum_{k \in \mathbb{O}} \log \pi_k(u_k|\boldsymbol{x})$. The element-wise policy $\pi_k(u_k|\boldsymbol{x})$ is the probability of receiving value u_k as kth element of action vector $\boldsymbol{u}$ when encountering state $\boldsymbol{x}$ and is given by

$$\pi_k(u_k|\boldsymbol{x}) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(u_k - \mu_k)^2}{2\sigma_k^2}\right), \tag{13}$$

where we substituted $\mu_k := f_k(\boldsymbol{x}, \boldsymbol{\theta})$. We differentiate with respect to the parameters θ_j and σ_j:

$$\begin{aligned} \frac{\partial \log \boldsymbol{\pi}(\boldsymbol{u}|\boldsymbol{x})}{\partial \theta_j} &= \sum_{k \in \mathbb{O}} \frac{\partial \log \pi_k(u_k|\boldsymbol{x})}{\partial \mu_k} \frac{\partial \mu_k}{\partial \theta_j} \\ &= \sum_{k \in \mathbb{O}} \frac{(u_k - \mu_k)}{\sigma_k^2} \frac{\partial \mu_k}{\partial \theta_j} \end{aligned} \tag{14}$$

[1] A further simplification would use $\boldsymbol{\Sigma} = \sigma \boldsymbol{I}$ with $\boldsymbol{I}$ being the unity matrix. This is advisable if the optimal solution for all parameters is expected to lay in similar value ranges.

$$\begin{aligned} \frac{\partial \log \boldsymbol{\pi}(\boldsymbol{u}|\boldsymbol{x})}{\partial \sigma_j} &= \sum_{k \in \mathbb{O}} \frac{\partial \log \pi_k(u_k|\boldsymbol{x})}{\partial \sigma_j} \\ &= \frac{(u_j - \mu_j)^2 - \sigma_j^2}{\sigma_j^3} \end{aligned} \tag{15}$$

For the linear case, where $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\Theta x}$ with the parameter matrix $\boldsymbol{\Theta} = [\theta_{ji}]$ mapping states to actions, (14) becomes

$$\frac{\partial \log \boldsymbol{\pi}(\boldsymbol{u}|\boldsymbol{x})}{\partial \theta_{ji}} = \frac{(u_j - \sum_i \theta_{ji} x_i)}{\sigma_j^2} x_i \tag{16}$$

An issue with nonlinear function approximation (NLFA) is a parameter dimensionality typically much higher than their output dimensionality, constituting a huge search space for FD methods. However, in combination with LR methods, they are interesting because LR methods only perturb the resulting outputs and not the parameters directly. Assuming the NLFA is differentiable with respect to its parameters, one can easily calculate the log likelihood values for each single parameter.

The factor $\frac{\partial \mu_k}{\partial \theta_j}$ in (14) describes the differentiation through the function approximator. It is convenient to use existing implementations, where instead of an error, the log likelihood derivative with respect to the mean, i.e. the first factor of the sum in (14), can be injected. The usual backward pass through the NLFA—known from supervised learning settings—then results in the log likelihood derivatives for each parameter [5].

3 State-Dependent Exploration

As indicated in the introduction, adding noise to the action u of a stochastic policy (3) at each step enables random exploration, but also aggravates the credit assignment problem: The overall reward for an episode (also called *return*) cannot be properly assigned to individual actions because information about which actions (if any) had a positive effect on the return value is not accessible.[2]

Our alternative approach adds a *state-dependent offset* to the action at each timestep, which 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. We define a function $\hat{\boldsymbol{\epsilon}}(\boldsymbol{x}; \hat{\boldsymbol{\theta}})$ on the states, which will act as a pseudo-random function that takes the state $\boldsymbol{x}$ as input. Randomness originates from parameters $\hat{\boldsymbol{\theta}}$ being drawn from a normal distribution $\hat{\theta}_j \sim \mathcal{N}(0, \hat{\sigma}_j^2)$. As discussed in section 2.3, simplifications to reduce the number of variance parameters can be applied. The action is then calculated by

$$\boldsymbol{u} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}) + \hat{\boldsymbol{\epsilon}}(\boldsymbol{x}; \hat{\boldsymbol{\theta}}), \quad \hat{\theta}_j \sim \mathcal{N}(0, \hat{\sigma}_j^2). \tag{17}$$

[2] GPOMDP [10], also known as the Policy Gradient Theorem [6], does consider single step rewards. However, it still introduces a significant amount of variance to a rollout with traditional random exploration.

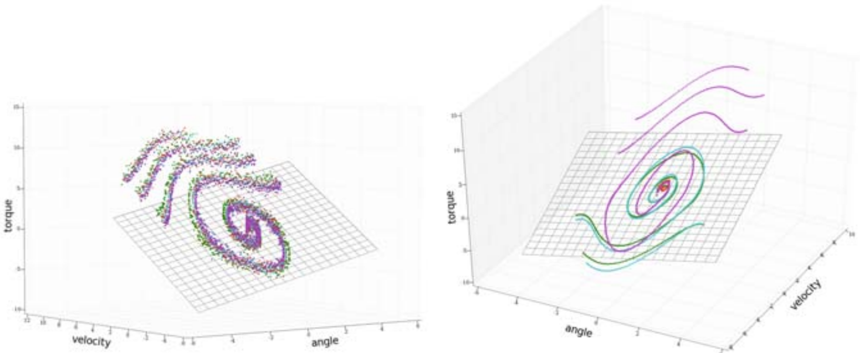


Fig. 1. Illustration of the main difference between random (left) and state-dependent (right) exploration. Several rollouts in state-action space of a task with state $\boldsymbol{x} \in \mathbb{R}^2$ (x- and y-axis) and action $u \in \mathbb{R}$ (z-axis) are plotted. While random exploration follows the same trajectory over and over again (with added noise), SDE instead tries different *strategies* and can quickly find solutions that would take a long time to discover with random exploration.

If the parameters $\hat{\boldsymbol{\theta}}$ are drawn at each timestep, we have an LR algorithm as in (3) and (12), although with a different exploration variance. However, if we keep $\hat{\boldsymbol{\theta}}$ constant for a full episode, then our action will have the same exploration added whenever we encounter the same state (Figure 1). Depending on the choice of $\hat{\boldsymbol{\epsilon}}(\boldsymbol{x})$, the randomness can further be "continuous", resulting in similar offsets for similar states. Effectively, by drawing $\hat{\boldsymbol{\theta}}$, we actually create a *policy delta*, similar to FD methods. In fact, if both $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\Theta})$ with $\boldsymbol{\Theta} = [\theta_{ji}]$ and $\hat{\boldsymbol{\epsilon}}(\boldsymbol{x},\hat{\boldsymbol{\Theta}})$ with $\hat{\boldsymbol{\Theta}} = [\hat{\theta}_{ji}]$ are linear functions, we see that

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\Theta}) + \hat{\boldsymbol{\epsilon}}(\boldsymbol{x};\hat{\boldsymbol{\Theta}}) \\ &= \boldsymbol{\Theta}\boldsymbol{x} + \hat{\boldsymbol{\Theta}}\boldsymbol{x} \\ &= (\boldsymbol{\Theta} + \hat{\boldsymbol{\Theta}})\boldsymbol{x}, \end{aligned} \tag{18}$$

which shows that direct parameter perturbation methods (cf. (4)) are a special case of SDE and can be expressed in this more general reinforcement framework.

3.1 Updates of Exploration Variances

For a linear exploration function $\hat{\boldsymbol{\epsilon}}(\boldsymbol{x};\hat{\boldsymbol{\Theta}}) = \hat{\boldsymbol{\Theta}}\boldsymbol{x}$ it is possible to calculate the derivative of the log likelihood with respect to the variance. We will derive the adaptation for general $\hat{\sigma}_{ji}$, any parameter reduction techniques from 2.3 can be applied accordingly.

First, we need the distribution of the action vector elements u_j:

$$u_j = f_j(\boldsymbol{x},\boldsymbol{\Theta}) + \hat{\boldsymbol{\Theta}}_j\boldsymbol{x} = f_j(\boldsymbol{x},\boldsymbol{\Theta}) + \sum_i \hat{\theta}_{ji}x_i \tag{19}$$

with $f_j(\boldsymbol{x},\boldsymbol{\Theta})$ as the jth element of the return vector of the deterministic controller f and $\hat{\theta}_{ji} \sim \mathcal{N}(0,\hat{\sigma}_{ji}^2)$. We now use two well-known properties of normal distributions: First, if X and Y are two independent random variables with $X \sim \mathcal{N}(\mu_a,\sigma_a^2)$ and $Y \sim \mathcal{N}(\mu_b,\sigma_b^2)$ then $U = X + Y$ is normally distributed with $U \sim \mathcal{N}(\mu_a + \mu_b, \sigma_a^2 + \sigma_b^2)$. Second, if $X \sim \mathcal{N}(\mu,\sigma^2)$ and $a,b \in \mathbb{R}$, then $aX + b \sim \mathcal{N}(a\mu + b, (a\sigma)^2)$.

Applied to (19), we see that $\hat{\theta}_{ji}x_i \sim \mathcal{N}(0,(x_i\hat{\sigma}_{ji})^2)$, that the sum is distributed as $\sum_i \hat{\theta}_{ji}x_i \sim \mathcal{N}(0,\sum_i(x_i\hat{\sigma}_{ji})^2)$, and that the action element u_j is therefore distributed as

$$u_j \sim \mathcal{N}(f_j(\boldsymbol{x},\boldsymbol{\Theta}), \sum_i (x_i\hat{\sigma}_{ji})^2), \tag{20}$$

where we will substitute $\mu_j := f_j(\boldsymbol{x},\boldsymbol{\Theta})$ and $\sigma_j^2 := \sum_i(x_i\hat{\sigma}_{ji})^2$ to obtain expression (13) for the policy components again. Differentiation of the policy with respect to the free parameters $\hat{\sigma}_{ji}$ yields:

$$\begin{aligned} \frac{\partial \log \boldsymbol{\pi}(\boldsymbol{u}|\boldsymbol{x})}{\partial \hat{\sigma}_{ji}} &= \sum_k \frac{\partial \log \pi_k(u_k|\boldsymbol{x})}{\partial \sigma_j} \frac{\partial \sigma_j}{\partial \hat{\sigma}_{ji}} \\ &= \frac{(u_j - \mu_j)^2 - \sigma_j^2}{\sigma_j^4} \, x_i^2 \hat{\sigma}_{ji} \end{aligned} \tag{21}$$

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.

3.2 Stochastic Policies

The original policy gradient setup as presented in e.g. [5] conveniently unifies the two stochastic features of the algorithm: the stochastic exploration and the stochasticity of the policy itself. Both were represented by the Gaussian noise added on top of the controller. While elegant on the one hand, it also conceals the fact that there are two different stochastic processes. With SDE, randomness has been taken out of the controller completely and is represented by the seperate exploration function. So if learning is switched off, the controller only returns deterministic actions. But in many scenarios the best policy is necessarily of stochastic nature.

It is possible and straight-forward to implement SDE with stochastic policies, by combining both random and state-dependent exploration in one controller, as in

$$\boldsymbol{u} = f(\boldsymbol{x};\boldsymbol{\theta}) + \boldsymbol{\epsilon} + \hat{\boldsymbol{\epsilon}}(\boldsymbol{x};\hat{\boldsymbol{\theta}}), \tag{22}$$

where $\epsilon_j \sim N(0,\sigma_j)$ and $\hat{\theta}_j \sim N(0,\hat{\sigma}_j)$. Since the respective noises are simply added together, none of them affects the derivative of the log-likelihood of the other and σ and $\hat{\sigma}$ can be updated independently. In this case, the trajectories through state-action space would look like a noisy version of Figure 1, righthand side.

3.3 Negative Variances

For practical applications, we also have to deal with the issue of negative variances. Obviously, we must prevent σ from falling below zero, which can happen since the right side of (15) can become negative. We therefore designed the following smooth, continuous function and its first derivative:

$$\operatorname{expln}(\sigma) = \begin{cases} \exp(\sigma) & \sigma \leq 0 \\ \ln(\sigma+1)+1 & \text{else} \end{cases} \tag{23}$$

$$\operatorname{expln}'(\sigma) = \begin{cases} \exp(\sigma) & \sigma \leq 0 \\ \frac{1}{\sigma+1} & \text{else} \end{cases} \tag{24}$$

Substitution of $\sigma^* := \operatorname{expln}(\sigma)$ will keep the variance above zero (exponential part) and also prevent it from growing too fast (logarithmic part). For this, the derivatives in (15) and (21) have to be multiplied by $\operatorname{expln}'(\sigma)$. In the experiments in section 4, this factor is included.

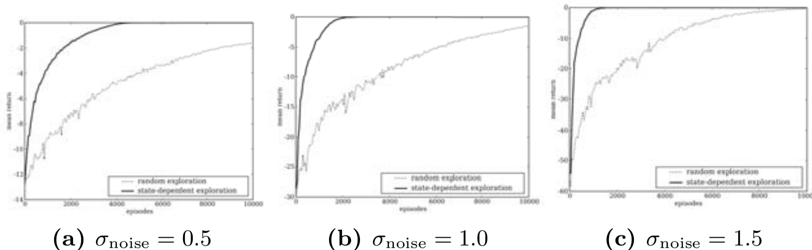
4 Experiments

Two different sets of experiments are conducted to investigate both the theoretical properties and the practical application of SDE. The first looks at plain function minimization and analyses the properties of SDE compared to REX. The second demonstrates SDE's usefulness for real-world problems with a simulated robot hand trying to catch a ball.

4.1 Function Minimization

The following sections compare SDE and random exploration (REX) with regard to sensitivity to noise, episode length, and parameter dimensionality. We chose a very basic setup where the task was to minimize $g(x) = x^2$. This is sufficient for first convergence evaluations since policy gradients are known to only converge locally. The agent's state x lies on the abscissa, its action is multiplied with a step-size factor s and the result is interpreted as a step along the abscissa in either direction. To make the task more challenging, we always added random noise to the agent's action. Each experiment was repeated 30 times, averaging the results. Our experiments were all episodic, with the return R for each episode being the average reward as stated in section 2.1. The reward per timestep was defined as $r_t = -g(x_t)$, thus a controller reducing the costs (negative reward) minimizes $g(x)$.

For a clean comparison of SDE and REX, we used the SDE algorithm in both cases, and emulated REX by drawing new exploration function parameters $\hat{\boldsymbol{\theta}}$ after each step (see Section 3). Unless stated otherwise, all experiments were conducted with a REINFORCE gradient estimator with optimal baseline [7] and the following parameters: learning rate $\alpha = 0.1$, step-size factor $s = 0.1$, initial parameter $\theta_0 = -2.0$, episode length EL $= 15$ and starting exploration noise $\hat{\sigma} = e^{-2} \approx 0.135$.



(a) $\sigma_{\text{noise}} = 0.5$ **(b)** $\sigma_{\text{noise}} = 1.0$ **(c)** $\sigma_{\text{noise}} = 1.5$

Fig. 2. Convergence for different levels of noise, averaged over 30 runs per curve. The upper solid curve shows SDE, the lower dotted curve REX.

Noise Level. First, we investigated how both SDE and REX deal with noise in the setting. We added normally distributed noise with variance σ^2_{noise} to each new state after the agent's action was executed: $x_{t+1} = x_t + su_t + \mathcal{N}(0, \sigma^2_{\text{noise}})$, where s is the step-size factor and u_t is the action at time t. The results of experiments with three different noise levels are given in Figure 2 and the right part of Table 1.

The results show that SDE is much more robust to noise, since its advantage over REX grows with the noise level. This is a direct effect of the credit assignment problem, which is more severe as the randomness of actions increases.

An interesting side-effect can also be found when comparing the convergence times for different noise levels. Both methods, REX and SDE, ran at better convergence rates with higher noise. The reason for this behavior can be shown best for a one-dimensional linear controller. In the absence of (environmental) noise, we then have:

$$x_t = x_{t-1} + su_{t-1}$$
$$u_t = \theta x_t + \epsilon_{\text{explore}}$$

Table 1. Noise and episode length (EL) sensitivity of REX and SDE. σ_{noise} is the standard deviation of the environmental noise. The steps designate the number of episodes until convergence, which was defined as $R_t > R_{\text{lim}}$ (a value that all controllers reached). The quotient REX/SDE is given as a speed-up factor.

			# steps						# steps		
σ_{noise}	EL	R_{lim}	REX	SDE	speed-up	σ_{noise}	EL	R_{lim}	REX	SDE	speed-up
	5		9450	3350	2.82	0.5			9950	2000	4.98
	15		9150	1850	4.95	1.0	15	-1.6	9850	1400	7.04
0.5	30	-1.8	9700	1050	9.24	1.5			7700	900	8.56
	45		8800	650	13.54						
	60		8050	500	16.10						

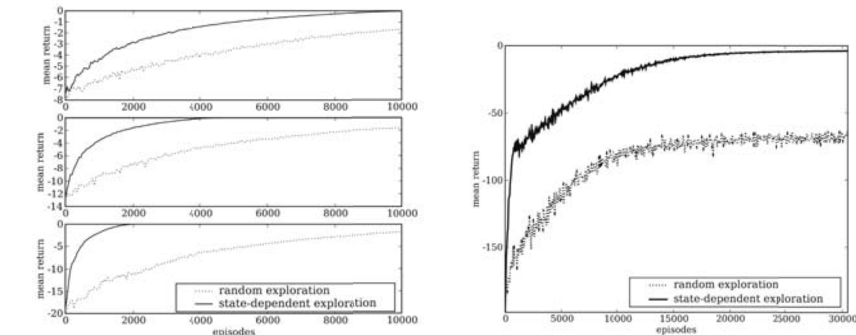


Fig. 3. Left: Results for different episode lengths, from top to bottom: 5, 15, 30. Right: Nonlinear controller with 18 free parameters on a 2-dimensional task. With REX, the agent became stuck in a local optimum, while SDE found the same optimum about 15 times faster and then converged to a better solution.

Adding noise to the state update results in

$$\begin{aligned} x'_t &= x_t + \epsilon_{\text{noise}} = x_{t-1} + s u_{t-1} + \epsilon_{\text{noise}} \\ u'_t &= \theta(x_{t-1} + s u_{t-1} + \epsilon_{\text{noise}}) + \epsilon_{\text{explore}} \\ &= \theta(x_{t-1} + s u_{t-1}) + \theta\epsilon_{\text{noise}} + \epsilon_{\text{explore}} \\ &= \theta x_t + \epsilon'_{\text{explore}} \end{aligned}$$

with $\epsilon'_{\text{explore}} = \theta\epsilon_{\text{noise}} + \epsilon_{\text{explore}}$. In our example, increasing the environmental noise was equivalent to increasing the exploratory noise of the agent, which obviously accelerated convergence.

Episode Length. In this series of experiments, we varied only the episode length and otherwise used the default settings with $\sigma_{\text{noise}} = 0.5$ for all runs. The results are shown in Figure 3 on the left side and Table 1, left part. Convergence speed with REX only improved marginally with longer episodes. The increased variance introduced by longer episodes almost completely outweighed the higher number of samples for a better gradient estimate. Since SDE does not introduce more noise with longer episodes during a single rollout, it could profit from longer episodes enormously. The speed-up factor rose almost proportionally with the episode length.

Parameter Dimensionality. Here, we increased the dimensionality of the problem in two ways: Instead of minimizing a scalar function, we minimized $\boldsymbol{g}(x, y) = [x^2, y^2]^T$. Further, we used a nonlinear function approximator, namely a multilayer perceptron with 3 hidden units with sigmoid activation and a bias unit connected to hidden and output layer. We chose a single parameter $\hat{\sigma}$ for exploration variance adaptation. Including $\hat{\sigma}$ the system consisted of 18 adjustable parameters, which made it a highly challenging task for policy gradient methods.

The exploration variance was initially set to $\hat{\sigma} = -2$ which corresponds to an effective variance of ~ 0.135. The parameters were initialized with $\theta_i \in [-1, 0[$ because positive actions quickly lead to high negative rewards and destabilized learning. For smooth convergence, the learning rate $\alpha = 0.01$ needed to be smaller than in the one-dimensional task.

As the righthand side of Figure 3 shows, the agent with REX became stuck after $15,000$ episodes at a local optimum around $R = -70$ from which it could not recover. SDE on the other hand found the same local optimum after a mere $1,000$ episodes, and subsequently was able to converge to a much better solution.

4.2 Catching a Ball

This series of experiments is based on a simulated robot hand with realistically modelled physics. We chose this experiment to show the predominance of SDE over random exploration, especially in a realistic robot task. We used the Open Dynamics Engine[3] 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 Fig. 4. First, we used REINFORCE gradient estimation with optimal baseline and a learning rate of $\alpha = 0.0001$. We then repeated the experiment with Episodic Natural Actor-Critic (ENAC), to see if SDE can be used for different gradient estimation techniques as well.

Experiment Setup. The information given to the system are the three coordinates of the ball position, so the robot "sees" where the ball is. It has four degrees of freedom to act, and in each timestep it can add a positive or negative torque to the joints. The controller therefore has 3 inputs and 4 outputs. We map inputs directly to outputs, but squash the outgoing signal with a tanh-function to ensure output between -1 and 1.

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

[3] The Open Dynamics Engine (ODE) is an open source physics engine, see `http://www.ode.org/` for more details.

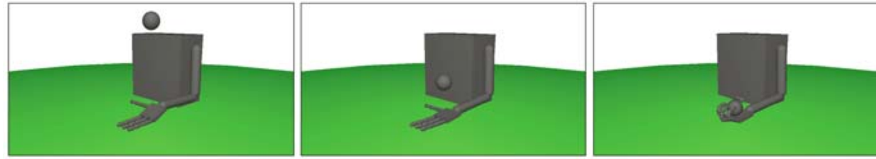


Fig. 4. Visualization of the simulated robot hand while catching a ball. The ball is released 5 units above the palm, where the palm dimensions are 1 x 0.1 x 1 units. When the fingers grasp the ball and do not release it throughout the episode, the best possible return (close to −1.0) is achieved.

catching behavior, considering the time from ball release to impact on palm, which is penalized with the capped distance to the palm center.

One attempt at catching the ball was considered to be one episode, which lasted for 500 timesteps. One simulation step corresponded to 0.01 seconds, giving the system a simulated time of 5 seconds to catch and hold the ball.

For the policy updates, we first executed 20 episodes with exploration and stored the complete history of states, actions, and rewards in an episode queue. After executing one learning step with the stored episodes, the first episode was discarded and one new rollout was executed and added to the front of the queue, followed by another learning step. With this "online" procedure, a policy update could be executed after each single step, resulting in smoother policy changes. However, we did not evaluate each single policy but ran every twentieth a few times without exploration. This yields a return estimate for the deterministic policy. Training was stopped after 500 policy updates.

Results. We will first describe the results with REINFORCE. The whole experiment was repeated 100 times. The left side of Figure 5 shows the learning curves over 500 episodes. Please note that the curves are not perfectly smooth because we only evaluated every twentieth policy. As can be seen, SDE finds a near-perfect solution in almost every case, resulting in a very low variance. The mean of the REX experiments indicate a semi-optimal solution, but in fact some of the runs found a good solution while others failed, which explains the high variance throughout the learning process.

The best controller found by SDE yielded a return of −0.95, REX reached −0.97. While these values do not differ much, the chances of producing a good controller are much higher with SDE. The right plot in Figure 5 shows the percentage of runs where a solution was found that was better than a certain value. Out of 100 runs, REX only found a mere 7 policies that qualified as "good catches", where SDE found 68. Almost all SDE runs, 98%, produced rewards $R \geq -1.1$, corresponding to behavior that would be considered a "catch" (closing the hand and holding the ball), although not all policies were as precise and quick as the "good catches". A typical behavior that returns $R \simeq -1.5$ can be described as one that keeps the ball on the fingers throughout the episode but hasn't learned to close the hand. $R \simeq -2.0$ corresponds to a behavior where

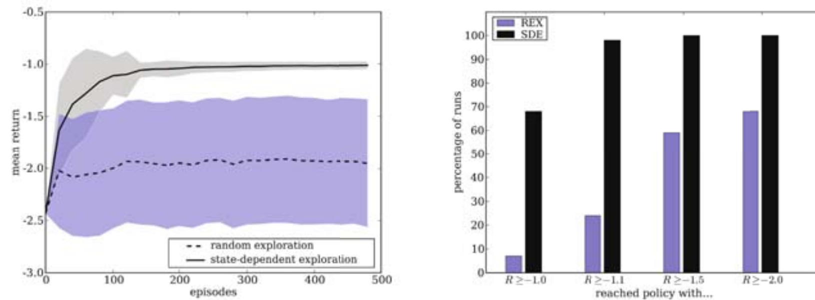


Fig. 5. Results after 100 runs with REINFORCE. Left: The solid and dashed curves show the mean over all runs, the filled envelopes represent the standard deviation. While SDE (solid line) managed to learn to catch the ball quickly in every single case, REX occasionally found a good solution but in most cases did not learn to catch the ball. Right: Cumulative number of runs (out of 100) that achieved a certain level. $R \geq -1$ means "good catch", $R \geq -1.1$ corresponds to all "catches" (closing the hand and holding the ball). $R \geq -1.5$ describes all policies managing to keep the ball on the hand throughout the episode. $R \geq -2$ results from policies that at least slowed down ball contact to the ground. The remaining policies dropped the ball right away.

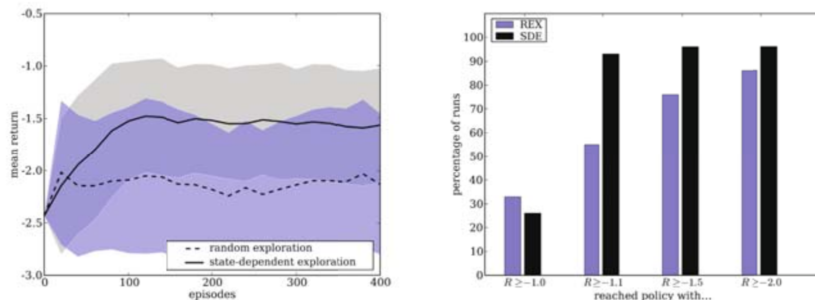


Fig. 6. Results after 100 runs with ENAC. Both learning curves had relatively high variances. While REX often didn't find a good solution, SDE found a catching behavior in almost every case, but many times lost it again due to continued exploration. REX also found slightly more "good catches" but fell far behind SDE considering both "good" and "average" catches.

the hand is held open and the ball falls onto the palm, rolls over the fingers and is then dropped to the ground. Some of the REX trials weren't even able to reach the −2.0 mark. A typical worst-case behavior is pulling back the hand and letting the ball drop to the ground immediately.

To investigate if SDE can be used with different gradient estimation techniques, we ran the same experiments with ENAC [11] instead of REINFORCE. We used a learning rate of 0.01 here, which lead to similar convergence speed. The results are presented in Figure 6. The difference compared to the results

with REINFORCE is, that both algorithms, REX and SDE had a relatively high variance. While REX still had problems to converge to stable catches (yet showed a 26% improvement over the REINFORCE version of REX for "good catches"), SDE in most cases (93%) found a "catching" solution but often lost the policy again due to continued exploration, which explains its high variance. Perhaps this could have been prevented by using tricks like reducing the learning rate over time or including a momentum term in the gradient descent. These advancements, however, are beyond the scope of this paper. SDE also had trouble reaching near-optimal solutions with $R \geq -1.0$ and even fell a little behind REX. But when considering policies with $R \geq -1.1$, SDE outperformed REX by over 38%. Overall the experiments show that SDE can in fact improve more advanced gradient estimation techniques like ENAC.

5 Conclusion

We introduced state-dependent exploration as an alternative to random exploration for policy gradient methods. By creating strategy variations similar to those of finite differences but without their disadvantages, SDE inserts considerably less variance into each rollout or episode. We demonstrated how various factors influence the convergence of both exploration strategies. SDE is much more robust to environmental noise and exhibits advantages especially during longer episodes. In problems involving many tunable parameters it not only converges considerably faster than REX, but can also overcome local minima where the other method gets stuck. In a robotics simulation task, SDE could clearly outperform REX and delivered a stable, near-optimal result in almost every trial. SDE also improves upon recent gradient estimation techniques such as ENAC. Furthermore, SDE is simple and elegant, and easy to integrate into existing policy gradient implementations. All of this recommends SDE as a valuable addition to the existing collection of policy gradient methods. Our toy experiment serves to illustrate basic properties of SDE, while the physics-based ball catching simulation gives a first hint of SDE's performance in real-world applications. Ongoing work is focusing on realistic robot domains.

Acknowledgements

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

References

1. Watkins, C., Dayan, P.: Q-learning. Machine Learning 8(3), 279–292 (1992)
2. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
3. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. Journal of AI research 4, 237–285 (1996)

4. Wiering, M.A.: Explorations in Efficient Reinforcement Learning. PhD thesis, University of Amsterdam / IDSIA (February 1999)
5. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8, 229–256 (1992)
6. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems (2000)
7. Peters, J., Schaal, S.: Policy gradient methods for robotics. In: Proc. 2006 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (2006)
8. Moody, J., Saffell, M.: Learning to trade via direct reinforcement. IEEE Transactions on Neural Networks 12(4), 875–889 (2001)
9. Peshkin, L., Savova, V.: Reinforcement learning for adaptive routing. In: Proc. 2002 Intl. Joint Conf. on Neural Networks (IJCNN 2002) (2002)
10. Baxter, J., Bartlett, P.: Reinforcement learning in POMDP's via direct gradient ascent. In: Proc. 17th Intl. Conf. on Machine Learning, pp. 41–48 (2000)
11. Peters, J., Vijayakumar, S., Schaal, S.: Natural actor-critic. In: Proceedings of the Sixteenth European Conference on Machine Learning (2005)
12. Spall, J.C.: Implementation of the simultaneous perturbation algorithm for stochastic optimization. IEEE Transactions on Aerospace and Electronic Systems 34(3), 817–823 (1998)
13. Wierstra, D., Foerster, A., Peters, J., Schmidhuber, J.: Solving deep memory POMDPs with recurrent policy gradients. In: Proc. Intl. Conference of Artificial Neural Networks (ICANN) (2007)
14. Ng, A., Jordan, M.: PEGASUS: A policy search method for large MDPs and POMDPs. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, pp. 406–415 (2000)
15. Aberdeen, D.: Policy-gradient Algorithms for Partially Observable Markov Decision Processes. Australian National University (2003)