Multimodal Parameter-exploring Policy Gradients

Frank Sehnke
Technische Universität München
Germany

Alex Graves
Technische Universität München
Germany

Christian Osendorfer
Technische Universität München
Germany

Jürgen Schmidhuber
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Lugano
Universitá della Svizzera italiana, Lugano
Switzerland

***Abstract*— Policy Gradients with Parameter-based Exploration (PGPE) is a novel model-free reinforcement learning method that alleviates the problem of high-variance gradient estimates encountered in normal policy gradient methods. It has been shown to drastically speed up convergence for several large-scale reinforcement learning tasks. However the independent normal distributions used by PGPE to search through parameter space are inadequate for some problems with multimodal reward surfaces. This paper extends the basic PGPE algorithm to use multimodal mixture distributions for each parameter, while remaining efficient. Experimental results on the Rastrigin function and the inverted pendulum benchmark demonstrate the advantages of this modification, with faster convergence to better optima.**

I. Introduction

Policy gradient methods, so called because they follow the gradient in policy space instead of deriving the policy directly from a value function, are among the few feasible optimization strategies for complex, high dimensional reinforcement learning problems with continuous states and actions [3], [4], [5]. However a significant problem with algorithms such as REINFORCE [11] is that the high variance in their gradient estimates leads to slow convergence. Various approaches have been proposed to reduce this variance [2], [1], [4], [10], but none of them address the underlying cause, which is that repeatedly sampling from a probabilistic policy injecting noise into the gradient estimate at every time-step.

Policy Gradients with Parameter-based Exploration (PGPE; [9]) tackles the problem directly by transferring the exploration from action space to parameter space. In this paradigm an implicit policy is defined by a distribution over the parameters of a deterministic controller. The policy is then sampled by picking a single point from the distribution at the start of each complete state-action trajectory. As well as being much less noisy than sampling at every time-step (see [9], this approach has the advantage (conferred by working directly in parameter space) of being applicable to non-differentiable controllers. PGPE has been shown to outperform evolution strategies (ES; [7]), natural actor critic, SPSA and REINFORCE for several complex control tasks [9], [8], including robust standing with a humanoid robot and different grasping tasks (See [6], [9]).

Clearly the success of PGPE depends on how well the parameter distribution converges on regions of the parameter space corresponding to good policies. In particular it relies on an effective tradeoff between exploring and exploiting the reward surface. In the original formulation this was achieved by using a separate Gaussian for each parameter, whose mean and variance were adapted during training to home in on areas of high reward. However for multimodal reward surfaces with widely separated optima the adaptation tends to be confused by the high rewards arriving from opposite directions. The obvious solution is to move from a unimodal single Gaussian to a multimodal mixture of Gaussians for each parameter. As we will see, this modification, which we refer to as Multimodal Policy Gradients with Parameter-based Exploration (MultiPGPE), can lead to substantial gains in performance, while leaving the underlying equations and computational efficiency of PGPE largely unchanged.

The MultiPGPE algorithm is derived in detail in Section II. In Section III we test MultiPGPE on three benchmark tasks that clearly show its advantages over both PGPE and Evolution Strategies. Conclusions and an outlook on future work are presented in Section IV.

II. Method

In this section we derive the MultiPGPE algorithm from the general framework of episodic reinforcement learning in a Markovian environment. In doing so we highlight the differences between PGPE, MultiPGPE and standard policy gradient methods such as REINFORCE.

A. Policy Gradients with Parameter-Based Exploration

Consider an agent interacting with an environment. Denote the state of the environment at time t as s_t and the action at time t as a_t. Because we are interested in continuous state and action spaces (usually required for control tasks), we represent both a_t and s_t with real valued vectors. Each state-action pair gives the agent a scalar reward $r_t(a_t, s_t)$. The agent's actions depend stochastically on the current state

and some real valued parameter vector θ: $a_t \sim p(a_t|s_t, \theta)$. We assume that the environment is Markovian, i.e. that the probability distribution over the possible next states is conditionally independent of all previous state-action pairs given the current one: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. We refer to a length T sequence h of state-action pairs produced by an agent as a *history*: $h = [s_{1:T}, a_{1:T}]$.

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

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

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 θ, and using the standard identity $\nabla_x y(x) = y(x) \nabla_x \log y(x)$, we can write

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

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

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

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

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

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

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

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

where ρ are the parameters determining the distribution over θ, $F_\theta(s_t)$ is the (deterministic) action chosen by the model with parameters θ in state s_t, and δ is the Dirac delta function. The advantage of this approach is that the actions are deterministic, and an entire history can therefore be generated from a single parameter sample. This reduction in samples-per-history is what reduces the variance in the gradient estimate (see [9]). As an added benefit the parameter gradient is estimated by direct parameter perturbations, without having to backpropagate any derivatives, which allows the use of non-differentiable controllers.

The expected reward with a given ρ is

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

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

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

where $p(h|\theta)$ is the probability distribution over the parameters θ and ρ are the parameters determining the distribution over θ. 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 θ from $p(\theta|\rho)$, then running the agent to generate h from $p(h|\theta)$:

$$\nabla_\rho J(\rho) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\rho \log p(\theta|\rho) r(h^n). \tag{8}$$

B. Unimodal Parameter Distributions

In the original formulation of PGPE, ρ consisted of a set of means $\{\mu_i\}$ and standard deviations $\{\sigma_i\}$ that determine an independent normal distribution for each parameter θ_i in θ 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 μ_i and σ_i:

$$\nabla_{\mu_i} \log p(\theta|\rho) = \frac{(\theta_i - \mu_i)}{\sigma_i^2},$$
$$\nabla_{\sigma_i} \log p(\theta|\rho) = \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3}, \tag{9}$$

which can then be substituted into (8) to approximate the μ and σ gradients that gives the PGPE update rules. Note the similarity to REINFORCE [11]. But in contrast to REINFORCE, θ defines the parameters of the model, not the probability of the actions.

C. Multimodal Parameter Distributions

In MultiPGPE ρ consists of a set of mixing coefficients $\{\pi_i^k\}$, means $\{\mu_i^k\}$ and standard deviations $\{\sigma_i^k\}$ defining

an independent mixture of Gaussians for each parameter θ_i:

$$p(\theta_i|\rho_i) = \sum_{k=1}^{K} \pi_i^k \mathcal{N}(\theta_i|\mu_i^k, (\sigma_i^k)^2), \tag{10}$$

where

$$\sum_{k=1}^{K} \pi_i^k = 1,\ \pi_i^k \in [0,1]\ \forall i,k.$$

The derivatives of $\log p(\theta|\rho)$ are now as follows:

$$\nabla_{\pi_i^k} \log p(\theta|\rho) = l_i^k,\ \nabla_{\mu_i^k} \log p(\theta|\rho) = l_i^k \frac{(\theta_i - \mu_i^k)}{(\sigma_i^k)^2},$$
$$\nabla_{\sigma_i^k} \log p(\theta|\rho) = l_i^k \frac{(\theta_i - \mu_i^k)^2 - (\sigma_i^k)^2}{(\sigma_i^k)^3}, \tag{11}$$

where $l_i^k \stackrel{\text{def}}{=} \frac{\mathcal{N}(\theta_i|\mu_i^k,(\sigma_i^k)^2)}{p(\theta_i|\rho_i)}$. These can again be substituted into Eq. (8) to approximate $\nabla_\rho J(\rho)$.

We use a fix number of 10 Gaussians per mixture. The μ and σ of each Gaussian is chosen in a way that the mixture forms a uniform distribution over the search interval.

D. Simplified MultiPGPE

Sampling from a mixture distribution is equivalent to first choosing a component according to the probability distribution defined by the mixing coefficients, then sampling from that component. This suggests the following simplification to the MultiPGPE gradient calculations: first pick k with probability π_i^k, then set $l_i^k = 1$ and $l_i^{k'} = 0\ \forall k' \neq k$. Substituting into Eq. (11) we see that $\nabla_{\pi_i^k} \log p(\theta|\rho) = 1$ and the other gradients reduce to their unimodal form in Eq. (9).

As well removing the computational effort of calculating the l_i^k terms, this simplification has other advantage. First, it tends to push the mixing coefficients towards one or zero, which prevents multiple components with similar means and variances "shadowing" each other, and speeds up the decision between overlapping components. Second, as we will see in Section II-F, it allows us to improve convergence by picking symmetric parameter samples from either side of the mean of the chosen component.

E. Sampling with a baseline

Given enough samples, Eq. (8) will determine the reward gradient to arbitrary accuracy. However each sample requires rolling out an entire state-action history, which is expensive. Following [11], we obtain a cheaper gradient estimate by drawing a single sample θ and comparing its reward r to a baseline reward b given by a moving average over previous samples. Intuitively, if $r > b$ we adjust ρ so as to increase the probability of θ, and $r < b$ we do the opposite. If, as in [11], we use a step size $\alpha_i = \alpha\sigma_i^2$ in the direction of positive gradient (where α is a constant) we get the following parameter update equations:

$$\Delta\pi_i^k = \alpha_\pi (r-b) l_i^k, \quad \Delta\mu_i^k = \alpha_\mu (r-b) l_i^k (\theta_i - \mu_i^k),$$
$$\Delta\sigma_i^k = \alpha_\sigma (r-b) l_i^k \frac{(\theta_i - \mu_i^k)^2 - (\sigma_i^k)^2}{\sigma_i^k}. \tag{12}$$

F. Symmetric sampling

While sampling with a baseline is efficient and reasonably accurate for most scenarios, it has several drawbacks. In particular, if the reward distribution is strongly skewed then the comparison between the sample reward and the baseline reward is misleading. A more robust gradient approximation can be found by measuring the difference in reward between two symmetric samples on either side of the current mean. That is, we pick a perturbation ϵ from the distribution $\mathcal{N}(0,\sigma)$, then create symmetric parameter samples $\theta^+ = \mu + \epsilon$ and $\theta^- = \mu - \epsilon$. Note that this is only possible for unimodal PGPE or the simplified version of MultiPGPE described in Section II-D, since the full mixture distribution used for MultiPGPE is not symmetrical. Defining r^+ as the reward given by θ^+ and r^- as the reward given by θ^-, we can insert the two samples into Eq. (8) and make use of Eq. (11) with $l_i^k = 1$ to obtain

$$\nabla_{\mu_i^k} J(\rho) \approx \frac{\epsilon_i (r^+ - r^-)}{2(\sigma_i^k)^2}, \tag{13}$$

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

$$\Delta\mu_i^k = \frac{\alpha_\mu \epsilon_i (r^+ - r^-)}{2}. \tag{14}$$

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

$$\Delta\sigma_i^k = \alpha_\sigma \left(\frac{r^+ + r^-}{2} - b\right) \left(\frac{\epsilon_i^2 - (\sigma_i^k)^2}{\sigma_i^k}\right),$$
$$\Delta\pi_i^k = \alpha_\pi \left(\frac{r^+ + r^-}{2} - b\right). \tag{15}$$

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

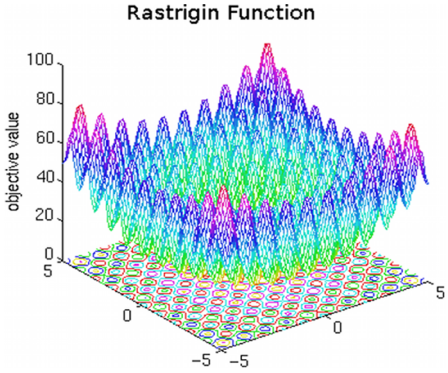

Figure 1. The Rastrigin function for $\vec{x} \in [-5, 12, 5.12]$. Note the global maxima near the corners and the single global minimum in the center.

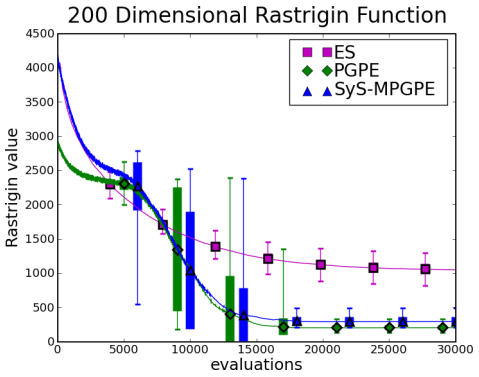

Figure 3. SyS-MPGPE, PGPE and ES minimizing the 200 dimensional Rastrigin function.

G. Reward Normalization

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

$$\Delta\pi_i^k = \frac{\alpha_\pi}{m-b}\left(\frac{r^+ + r^-}{2} - b\right), \quad \Delta\mu_i^k = \frac{\alpha_\mu \epsilon_i^k (r^+ - r^-)}{2m - r^+ - r^-},$$
$$\Delta\sigma_i^k = \frac{\alpha_\sigma}{m-b}\left(\frac{r^+ + r^-}{2} - b\right)\left(\frac{(\epsilon_i^k)^2 - (\sigma_i^k)^2}{\sigma_i^k}\right), \quad (16)$$

where $\epsilon_i^k \stackrel{\text{def}}{=} \theta_i - \mu_i^k$. For non-simplified MultiPGPE the reward normalization is applied to the baseline sampling of Sec. II-E:

$$\Delta\pi_i^k = \alpha_\pi l_i^k \frac{r-b}{m-b}, \quad \Delta\mu_i^k = \alpha_\mu l_i^k \frac{r-b}{m-b}(\theta_i - \mu_i^k)$$
$$\Delta\sigma_i^k = \alpha_\sigma l_i^k \frac{r-b}{m-b}\frac{(\theta_i - \mu_i^k)^2 - (\sigma_i^k)^2}{\sigma_i^k}. \quad (17)$$

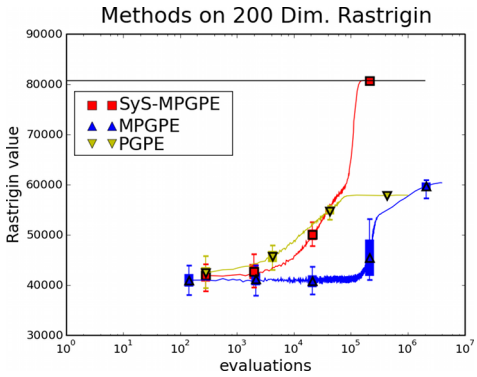

Figure 2. SyS-MPGPE, MPGPE and PGPE maximizing the 200 dimensional Rastrigin function.

III. Experiments

This section compares MultiPGPE with PGPE and ES on two multimodal benchmark functions. We used 10 Gaussians per mixture for MultiPGPE and its variants in all experiments. We will use the following abbreviations throughout to distinguish the algorithm variants:

- MPGPE: standard MultiPGPE (Sec. II-C) with baseline sampling (Sec. II-E)
- SyS-MPGPE: simplified MultiPGPE with symmetric sampling (Sec. II-F)

A. Rastrigin Function

The n-dimensional Rastrigin function, illustrated in Fig 1, is a well-known optimization benchmark with one global minimum, 2^n global maxima, and a exponential growing number of local optima. It is defined as follows:

$$Ras(\vec{x}) = 10n + 10 \cdot \sum_{i=1}^{n} \cos(2\pi x_i) - \sum_{i=1}^{n} x_i^2. \quad (18)$$

As is standard practice for the Rastrigin benchmark, our experiments restrict all the x_i values to the interval

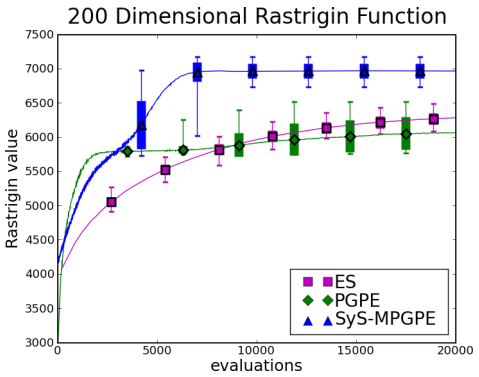

Figure 4. SyS-MPGPE, PGPE and ES maximizing the 200 dimensional Rastrigin function.

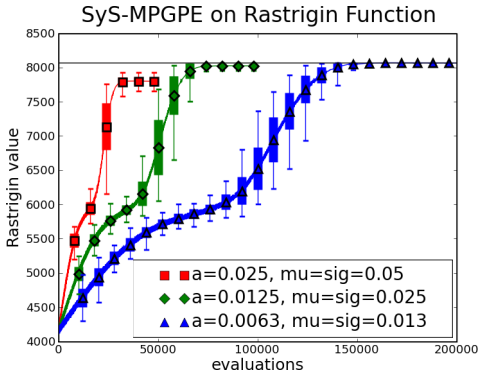

Figure 5. SyS-MPGPE maximizing the 200 dimensional Rastrigin function with different meta parameters.

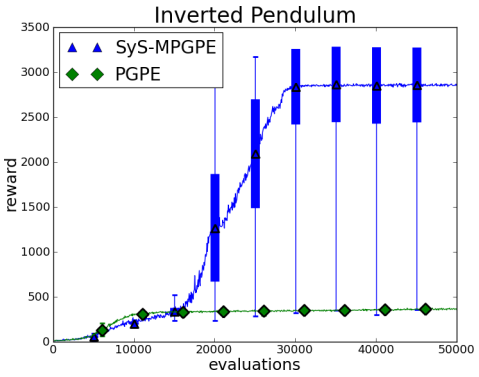

Figure 7. SyS-MPGPE and PGPE on the Inverted Pendulum. All plots show the mean and standard deviation of 10 runs.

$[-5.12, 5.12]$. This ensures that the global maxima lie close to, but not at, the extremal values of the domain.

All experiments were conducted on the 200 dimensional Rastrigin function if not noted otherwise. Also all experiments depict the mean, standard deviation and min/max of 100 independent runs.

Our first experiments compared the ability of PGPE, MPGPE and Sys-MPGPE to maximize the 200 dimensional Rastrigin function. For this task the meta parameters were $\alpha_\mu = \alpha_\sigma = 0.0125$ and $\alpha_\pi = 0.00625$ for SyS-MPGPE and PGPE and $\alpha_\mu = \alpha_\sigma = 0.025$ and $\alpha_\pi = 0.0125$ for MPGPE. The results, presented in figure 2, show that PGPE converges to a suboptimal solution. Its failure stems from its use of unimodal parameter distributions, which are confused by the 2^n equal but widely separated global optima. This results in an overextension of the parameter variances during the search process. MPGPE also fails to find a global optimum in the timeframe we set for computational reasons. In particular it is very slow at the initial stages of adaptation. The switch to SyS-MPGPE brings a dramatic speedup, reaching a global optimum in about $1.5 \cdot 10^5$ evaluations.

Our second set of experiments compared SyS-MPGPE,

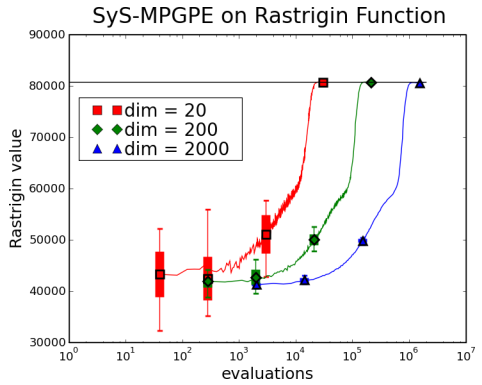

Figure 6. SyS-MPGPE maximizing the n dimensional Rastrigin function for different values of n.

PGPE and ES at both maximizing and minimizing the 200 dimensional Rastrigin function with the standard meta parameters. The PGPE and SyS-MPGPE meta parameters were therefore $\alpha_\mu = \alpha_\sigma = 0.2$ and $\alpha_\pi = 0.1$. For ES we hand-optimized the meta parameters to give the best performance possible within the time-frame set by the PGPE experiments; in particular we chose the best population size found in a thorough search in the range from (6,36) to (600, 3600). As can be seen from Figure 4, the advantage of SyS-MPGPE over PGPE in the maximization task vanishes in the minimization task.

At first glance this seems surprising, since one would expect the multimodal surface of the Rastrigin function to favor MPGPE for both kinds of optimization. However, while the Rastrigin function has 2^n global maxima, it has only one global minimum. Therefore PGPE no longer needs to overextend the variance in its parameter distributions to successfully optimize; all it needs is enough variance to escape the local minima — just like MPGPE. Although ES slightly outperforms PGPE at maximization, SyS-MPGPE is clearly better than ES at both tasks. Figure 5 shows how the performance of SyS-MPGPE varies with different meta parameters. While its ability to find an optimum appears robust to parameter choice, the time taken to converge can vary substantially. Figure 6 shows how the number of evaluations required for convergence grows with the dimension of the problem for SyS-MPGPE. The meta parameters were halved for every factor of 10 by which the dimensionality was increased (from $\alpha_\mu = \alpha_\sigma = 0.025$ and $\alpha_\pi = 0.0125$ for 20 dimensions to $\alpha_\mu = \alpha_\sigma = 0.00625$ and $\alpha_\pi = 0.003125$ for 2000 dimensions). Note that training time grows sublinearly with the number of dimensions (increasing the number of dimensions by a factor of 10 increases the number of required evaluations by a factor about 6.5). Given that the number of local optima in the Rastrigin function grows exponentially with the number of dimensions, this bodes well for the scalability of SyS-MPGPE to high dimensional parameter spaces.

B. Inverted Pendulum

The task in the Inverted Pendulum benchmark is to swing a pendulum to an upright position, and balance it there, using a motor attached to its axis. The motor is not strong enough to move the pendulum up directly, so that the only way to solve the task is to swing back and forth several times to gain momentum. The agent receives a reward only while the pendulum is almost completely vertical (meaning that the absolute value of the joint angle is less than 0.14 Radians). The input variables to the agent are a bias signal, the angle of the pendulum and the angular velocity of the pendulum. The starting search interval was $[-20, 20]$ for all parameters in our experiments, but they were allowed to exceed these limits during learning. We used the ODE physics engine to simulate the motion of the pendulum. Note that we used a much harder version of this benchmark. In the standard task the pendulum needs only one extra swing to gather enough momentum to finally swing up. Here the motor is under-powered such that at least 3 extra swings are needed to gain enough momentum. The main difficulty of the task is the gradient of most of the reward space leads to suboptimal solutions, as illustrated in Fig. 8. These solutions correspond to parameters that will spin the pendulum around the axis, thereby collecting a reward only during the brief period when the pendulum passes through the upright position. We compared SyS-MPGPE and PGPE on this benchmark with the meta parameters $\alpha_\mu = 0.05, \alpha_\sigma = \alpha_\pi = 0.025$. As can be seen from figure 7, PGPE converges to a much lower reward (the spinning behavior) than SyS-MPGPE. In fact, SyS-MPGPE only failed once in ten trials to find the global optimum. As with the Rastrigin function, PGPE is confused by the multimodal reward surface and the sharp global optimum. SyS-MPGPE, on the other hand, uses its multimodal distributions to track both regions of the reward space simultaneous and eventually choose the better one.

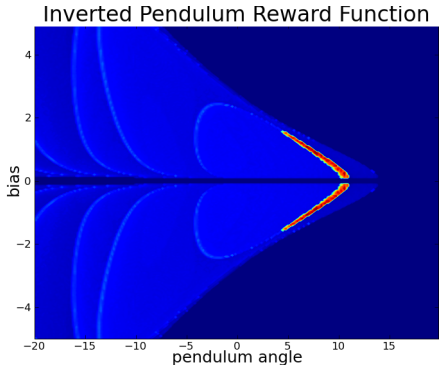

Figure 8. A 2D slice of the reward space for the inverted pendulum task. The plot shows how the reward varies with the pendulum angle and the bias weight while the angular velocity is fixed at 25. Note the sharp global optimum (which corresponds to balancing the pendulum upright) and the smooth gradients leading towards the multiple local optima (which correspond to spinning the pendulum round and round).

IV. Conclusion and Future Work

We derived the MultiPGPE algorithm from the general framework of episodic reinforcement learning. In the process we showed how MultiPGPE can be simplified and how standard techniques for PGPE, like symmetric sampling and reward normalization, transfer straightforwardly to MultiPGPE. Our experiments demonstrate that for certain types of task MultiPGPE clearly outperforms PGPE, which has already proved superior to standard evolutionary and policy-gradient methods for several reinforcement learning problems. MultiPGPE introduces a new meta parameter, the step size α_π for adjusting the mixing coefficients π_i^k. This new meta parameter seems to be more dependent on the actual problem than the other meta parameters already present in PGPE. One direction for future work is to show how this step size can be estimated for certain problem domains. Another direction is to assess how much impact this enhancement of PGPE has on real-world applications, now that the theoretical benefits have been demonstrated.

References

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

[2] Baxter, J., Bartlett, P.L.: Reinforcement learning in POMDPs via direct gradient ascent. In: Proc. 17th International Conf. on Machine Learning. pp. 41–48 (2000)

[3] Benbrahim, H., Franklin, J.: Biped dynamic walking using reinforcement learning. Robotics and Autonomous Systems Journal (1997)

[4] Peters, J., Schaal, S.: Policy gradient methods for robotics. In: IROS-2006. pp. 2219 – 2225. Beijing, China (2006)

[5] Peters, J., Vijayakumar, S., Schaal, S.: Natural actor-critic. In: ECML-2005. pp. 280–291 (2005)

[6] Rückstieß, T., Sehnke, F., Schaul, T., Wierstra, D., Sun, Y., Schmidhuber, J.: Exploring parameter space in reinforcement learning. Paladyn 1(1), 14–24 (2010)

[7] Schwefel, H.: Evolution and optimum seeking. Wiley New York (1995)

[8] Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Policy gradients with parameter-based exploration for control. In: Proc. of the International Conference on Artificial Neural Networks. vol. I, pp. 387–396 (2008)

[9] Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Parameter-exploring policy gradients. Neural Networks 23(4), 551–559 (2010)

[10] Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. NIPS-1999 pp. 1057–1063 (2000)

[11] Williams, R.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8, 229–256 (1992)