Benchmarking Deep Reinforcement Learning for Continuous Control

Yan Duan[†] ROCKYDUAN@EECS.BERKELEY.EDU
Xi Chen[†] C.XI@EECS.BERKELEY.EDU
Rein Houthooft[†‡] REIN.HOUTHOOFT@UGENT.BE
John Schulman[†§] JOSCHU@EECS.BERKELEY.EDU
Pieter Abbeel[†] PABBEEL@CS.BERKELEY.EDU
[†] University of California, Berkeley, Department of Electrical Engineering and Computer Sciences
[‡] Ghent University - iMinds, Department of Information Technology
[§] OpenAI

Abstract

Recently, researchers have made significant progress combining the advances in deep learning for learning feature representations with reinforcement learning. Some notable examples include training agents to play Atari games based on raw pixel data and to acquire advanced manipulation skills using raw sensory inputs. However, it has been difficult to quantify progress in the domain of continuous control due to the lack of a commonly adopted benchmark. In this work, we present a benchmark suite of continuous control tasks, including classic tasks like cart-pole swing-up, tasks with very high state and action dimensionality such as 3D humanoid locomotion, tasks with partial observations, and tasks with hierarchical structure. We report novel findings based on the systematic evaluation of a range of implemented reinforcement learning algorithms. Both the benchmark and reference implementations are released at https://github.com/rllab/rllab in order to facilitate experimental reproducibility and to encourage adoption by other researchers.

1. Introduction

Reinforcement learning addresses the problem of how agents should learn to take actions to maximize cumulative reward through interactions with the environment. The traditional approach for reinforcement learning algorithms requires carefully chosen feature representations, which are usually hand-engineered. Recently, significant progress has been made by combining advances in deep learning for learning feature representations (Krizhevsky et al., 2012; Hinton et al., 2012) with reinforcement learning, tracing back to much earlier work of Tesauro (1995) and Bertsekas & Tsitsiklis (1995). Notable examples are training agents to play Atari games based on raw pixels (Guo et al., 2014; Mnih et al., 2015; Schulman et al., 2015a) and to acquire advanced manipulation skills using raw sensory inputs (Levine et al., 2015; Lillicrap et al., 2015; Watter et al., 2015). Impressive results have also been obtained in training deep neural network policies for 3D locomotion and manipulation tasks (Schulman et al., 2015a;b; Heess et al., 2015b).

Along with this recent progress, the Arcade Learning Environment (ALE) (Bellemare et al., 2013) has become a popular benchmark for evaluating algorithms designed for tasks with high-dimensional state inputs and discrete actions. However, these algorithms do not always generalize straightforwardly to tasks with continuous actions, leading to a gap in our understanding. For instance, algorithms based on Q-learning quickly become infeasible when naive discretization of the action space is performed, due to the curse of dimensionality (Bellman, 1957; Lillicrap et al., 2015). In the continuous control domain, where actions are continuous and often high-dimensional, we argue that the existing control benchmarks fail to provide a comprehensive set of challenging problems (see Section 7 for a review of existing benchmarks). Benchmarks have played a significant role in other areas such as computer vision and speech recognition. Examples include MNIST (LeCun et al., 1998), Caltech101 (Fei-Fei et al., 2006), CIFAR (Krizhevsky & Hinton, 2009), ImageNet (Deng et al., 2009), PASCAL VOC (Everingham et al., 2010), BSDS500 (Martin et al., 2001), SWITCHBOARD (Godfrey et al., 1992), TIMIT (Garofolo et al., 1993), Aurora (Hirsch & Pearce, 2000), and VoiceSearch (Yu et al., 2007). The lack

Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).
Also available at https://arxiv.org/abs/1604.06778

of a standardized and challenging testbed for reinforcement learning and continuous control makes it difficult to quantify scientific progress. Systematic evaluation and comparison will not only further our understanding of the strengths of existing algorithms, but also reveal their limitations and suggest directions for future research.

We attempt to address this problem and present a benchmark consisting of 31 continuous control tasks. These tasks range from simple tasks, such as cart-pole balancing, to challenging tasks such as high-DOF locomotion, tasks with partial observations, and hierarchically structured tasks. Furthermore, a range of reinforcement learning algorithms are implemented on which we report novel findings based on a systematic evaluation of their effectiveness in training deep neural network policies. The benchmark and reference implementations are available at https://github.com/rllab/rllab, allowing for the development, implementation, and evaluation of new algorithms and tasks.

2. Preliminaries

In this section, we define the notation used in subsequent sections.

The implemented tasks conform to the standard interface of a finite-horizon discounted Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma, T)$, where $\mathcal{S}$ is a (possibly infinite) set of states, $\mathcal{A}$ is a set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the initial state distribution, $\gamma \in (0, 1]$ is the discount factor, and T is the horizon.

For partially observable tasks, which conform to the interface of a partially observable Markov decision process (POMDP), two more components are required, namely Ω, a set of observations, and $\mathcal{O} : \mathcal{S} \times \Omega \to \mathbb{R}_{\geq 0}$, the observation probability distribution.

Most of our implemented algorithms optimize a stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$. Let $\eta(\pi)$ denote its expected discounted reward: $\eta(\pi) = \mathbb{E}_\tau \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right]$, where $\tau = (s_0, a_0, \ldots)$ denotes the whole trajectory, $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi(a_t | s_t)$, and $s_{t+1} \sim P(s_{t+1} | s_t, a_t)$.

For deterministic policies, we use the notation $\mu_\theta : \mathcal{S} \to \mathcal{A}$ to denote the policy instead. The objective for it has the same form as above, except that now we have $a_t = \mu(s_t)$.

3. Tasks

The tasks in the presented benchmark can be divided into four categories: basic tasks, locomotion tasks, partially observable tasks, and hierarchical tasks. We briefly describe them in this section. More detailed specifications are given in the supplementary materials and in the source code.

We choose to implement all tasks using physics simulators rather than symbolic equations, since the former approach is less error-prone and permits easy modification of each task. Tasks with simple dynamics are implemented using Box2D (Catto, 2011), an open-source, freely available 2D physics simulator. Tasks with more complicated dynamics, such as locomotion, are implemented using MuJoCo (Todorov et al., 2012), a 3D physics simulator with better modeling of contacts.

3.1. Basic Tasks

We implement five basic tasks that have been widely analyzed in reinforcement learning and control literature: Cart-Pole Balancing (Stephenson, 1908; Donaldson, 1960; Widrow, 1964; Michie & Chambers, 1968), Cart-Pole Swing Up (Kimura & Kobayashi, 1999; Doya, 2000), Mountain Car (Moore, 1990), Acrobot Swing Up (DeJong & Spong, 1994; Murray & Hauser, 1991; Doya, 2000), and Double Inverted Pendulum Balancing (Furuta et al., 1978). These relatively low-dimensional tasks provide quick evaluations and comparisons of RL algorithms.

3.2. Locomotion Tasks

In this category, we implement six locomotion tasks of varying dynamics and difficulty: Swimmer (Purcell, 1977; Coulom, 2002; Levine & Koltun, 2013; Schulman et al., 2015a), Hopper (Murthy & Raibert, 1984; Erez et al., 2011; Levine & Koltun, 2013; Schulman et al., 2015a), Walker (Raibert & Hodgins, 1991; Erez et al., 2011; Levine & Koltun, 2013; Schulman et al., 2015a), Half-Cheetah (Wawrzyński, 2007; Heess et al., 2015b), Ant (Schulman et al., 2015b), Simple Humanoid (Tassa et al., 2012; Schulman et al., 2015b), and Full Humanoid (Tassa et al., 2012). The goal for all the tasks is to move forward as quickly as possible. These tasks are more challenging than the basic tasks due to high degrees of freedom. In addition, a great amount of exploration is needed to learn to move forward without getting stuck at local optima. Since we penalize for excessive controls as well as falling over, during the initial stage of learning, when the robot is not yet able to move forward for a sufficient distance without falling, apparent local optima exist including staying at the origin or diving forward slowly.

3.3. Partially Observable Tasks

In real-life situations, agents are often not endowed with perfect state information. This can be due to sensor noise, sensor occlusions, or even sensor limitations that result in partial observations. To evaluate algorithms in more realistic settings, we implement three variations of partially ob-

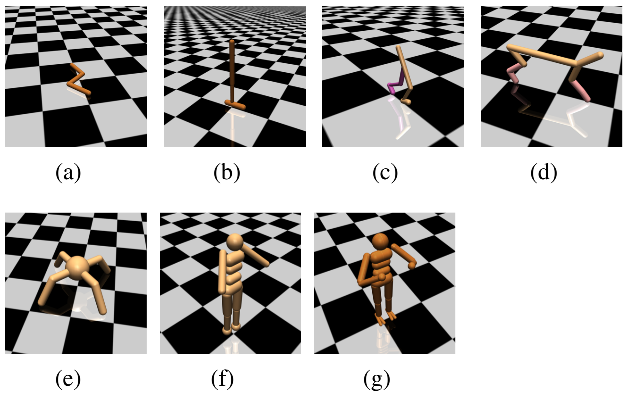

Figure 1. Illustration of locomotion tasks: (a) Swimmer; (b) Hopper; (c) Walker; (d) Half-Cheetah; (e) Ant; (f) Simple Humanoid; and (g) Full Humanoid.

servable tasks for each of the five basic tasks described in Section 3.1, leading to a total of 15 additional tasks. These variations are described below.

Limited Sensors: For this variation, we restrict the observations to only provide positional information (including joint angles), excluding velocities. An agent now has to learn to infer velocity information in order to recover the full state. Similar tasks have been explored in Gomez & Miikkulainen (1998); Schäfer & Udluft (2005); Heess et al. (2015a); Wierstra et al. (2007).

Noisy Observations and Delayed Actions: In this case, sensor noise is simulated through the addition of Gaussian noise to the observations. We also introduce a time delay between taking an action and the action being in effect, accounting for physical latencies (Hester & Stone, 2013). Agents now need to learn to integrate both past observations and past actions to infer the current state. Similar tasks have been proposed in Bakker (2001).

System Identification: For this category, the underlying physical model parameters are varied across different episodes (Szita et al., 2003). The agents must learn to generalize across different models, as well as to infer the model parameters from its observation and action history.

3.4. Hierarchical Tasks

Many real-world tasks exhibit hierarchical structure, where higher level decisions can reuse lower level skills (Parr & Russell, 1998; Sutton et al., 1999; Dietterich, 2000). For instance, robots can reuse locomotion skills when exploring the environment. We propose several tasks where both low-level motor controls and high-level decisions are needed. These two components each operates on a different time scale and calls for a natural hierarchy in order to efficiently learn the task.

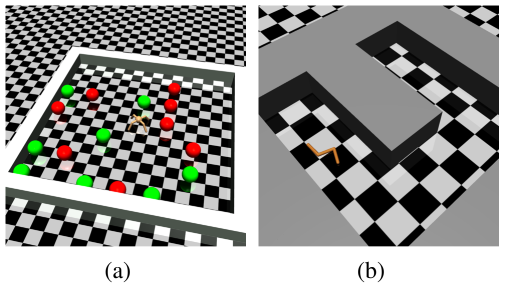

Figure 2. Illustration of hierarchical tasks: (a) Locomotion + Food Collection; and (b) Locomotion + Maze.

Locomotion + Food Collection: For this task, the agent needs to learn to control either the swimmer or the ant robot to collect food and avoid bombs in a finite region. The agent receives range sensor readings about nearby food and bomb units. It is given a positive reward when it reaches a food unit, or a negative reward when it reaches a bomb.

Locomotion + Maze: For this task, the agent needs to learn to control either the swimmer or the ant robot to reach a goal position in a fixed maze. The agent receives range sensor readings about nearby obstacles as well as its goal (when visible). A positive reward is given only when the robot reaches the goal region.

4. Algorithms

In this section, we briefly summarize the algorithms implemented in our benchmark, and note any modifications made to apply them to general parametrized policies. We implement a range of gradient-based policy search methods, as well as two gradient-free methods for comparison with the gradient-based approaches.

4.1. Batch Algorithms

Most of the implemented algorithms are batch algorithms. At each iteration, N trajectories $\{\tau_i\}_{i=1}^N$ are generated, where $\tau_i = \{(s_t^i, a_t^i, r_t^i)\}_{t=0}^T$ contains data collected along the ith trajectory. For on-policy gradient-based methods, all the trajectories are sampled under the current policy. For gradient-free methods, they are sampled under perturbed versions of the current policy.

REINFORCE (Williams, 1992): This algorithm estimates the gradient of expected return $\nabla_\theta \eta(\pi_\theta)$ using the likelihood ratio trick:

$$\widehat{\nabla_\theta \eta(\pi_\theta)} = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=0}^{T} \nabla_\theta \log \pi(a_t^i | s_t^i; \theta)(R_t^i - b_t^i),$$

where $R_t^i = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}^i$ and b_t^i is a baseline that only depends on the state s_t^i to reduce variance. Hereafter, an ascent step is taken in the direction of the estimated gradient. This process continues until θ_k converges.

Truncated Natural Policy Gradient (TNPG) (Kakade, 2002; Peters et al., 2003; Bagnell & Schneider, 2003; Schulman et al., 2015a): Natural Policy Gradient improves upon REINFORCE by computing an ascent direction that approximately ensures a small change in the policy distribution. This direction is derived to be $I(\theta)^{-1}\nabla_\theta\eta(\pi_\theta)$, where $I(\theta)$ is the Fisher information matrix (FIM). We use the step size suggested by Peters & Schaal (2008): $\alpha = \sqrt{\delta_{\text{KL}} \left(\nabla_\theta\eta(\pi_\theta)^T I(\theta)^{-1}\nabla_\theta\eta(\pi_\theta)\right)^{-1}}$. Finally, we replace $\nabla_\theta\eta(\pi_\theta)$ and $I(\theta)$ by their empirical estimates.

For neural network policies with tens of thousands of parameters or more, generic Natural Policy Gradient incurs prohibitive computation cost by forming and inverting the empirical FIM. Instead, we study Truncated Natural Policy Gradient (TNPG) in this paper, which computes the natural gradient direction without explicitly forming the matrix inverse, using a conjugate gradient algorithm that only requires computing $I(\theta)v$ for arbitrary vector v. TNPG makes it practical to apply natural gradient in policy search setting with high-dimensional parameters, and we refer the reader to Schulman et al. (2015a) for more details.

Reward-Weighted Regression (RWR) (Peters & Schaal, 2007; Kober & Peters, 2009): This algorithm formulates the policy optimization as an Expectation-Maximization problem to avoid the need to manually choose learning rate, and the method is guaranteed to converge to a locally optimal solution. At each iteration, this algorithm optimizes a lower bound of the log-expected return: $\theta = \arg\max_{\theta'} \mathcal{L}(\theta')$, where

$$\mathcal{L}(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=0}^{T}\log\pi(a_t^i|s_t^i;\theta)\rho(R_t^i - b_t^i)$$

Here, $\rho : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a function that transforms raw returns to nonnegative values. Following Deisenroth et al. (2013), we choose ρ to be $\rho(R) = R - R_{\text{min}}$, where R_{min} is the minimum return among all trajectories collected in the current iteration.

Relative Entropy Policy Search (REPS) (Peters et al., 2010): This algorithm limits the loss of information per iteration and aims to ensure a smooth learning progress (Deisenroth et al., 2013). At each iteration, we collect all trajectories into a dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^M$, where M is the total number of samples. Then, we first solve for the dual parameters $[\eta^*, \nu^*] = \arg\min_{\eta',\nu'} g(\eta', \nu')$ s.t. $\eta > 0$, where

$$g(\eta, \nu) = \eta\delta_{\text{KL}} + \eta\log\left(\frac{1}{M}\sum_{i=1}^{M} e^{\delta_i(\nu)/\eta}\right).$$

Here $\delta_{\text{KL}} > 0$ controls the step size of the policy, and $\delta_i(\nu) = r_i + \nu^T(\phi(s_i') - \phi(s_i))$ is the sample Bellman error. We then solve for the new policy parameters:

$$\theta_{k+1} = \arg\max_\theta \frac{1}{M}\sum_{i=1}^{M} e^{\delta_i(\nu^*)/\eta^*}\log\pi(a_i|s_i;\theta).$$

Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a): This algorithm allows more precise control on the expected policy improvement than TNPG through the introduction of a surrogate loss. At each iteration, we solve the following constrained optimization problem (replacing expectations with samples):

$$\begin{array}{ll} \text{maximize}_\theta & \mathbb{E}_{s\sim\rho_{\theta_k}, a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A_{\theta_k}(s,a)\right] \\ \text{s.t.} & E_{s\sim\rho_{\theta_k}}[D_{\text{KL}}(\pi_{\theta_k}(\cdot|s)\|\pi_\theta(\cdot|s))] \leq \delta_{\text{KL}} \end{array}$$

where $\rho_\theta = \rho_{\pi_\theta}$ is the discounted state-visitation frequencies induced by π_θ, $A_{\theta_k}(s,a)$, known as the advantage function, is estimated by the empirical return minus the baseline, and δ_{KL} is a step size parameter which controls how much the policy is allowed to change per iteration. We follow the procedure described in the original paper for solving the optimization, which results in the same descent direction as TNPG with an extra line search in the objective and KL constraint.

Cross Entropy Method (CEM) (Rubinstein, 1999; Szita & Lőrincz, 2006): Unlike previously mentioned methods, which perform exploration through stochastic actions, CEM performs exploration directly in the policy parameter space. At each iteration, we produce N perturbations of the policy parameter: $\theta_i \sim \mathcal{N}(\mu_k, \Sigma_k)$, and perform a rollout for each sampled parameter. Then, we compute the new mean and diagonal covariance using the parameters that correspond to the top q-quantile returns.

Covariance Matrix Adaption Evolution Strategy (CMA-ES) (Hansen & Ostermeier, 2001): Similar to CEM, CMA-ES is a gradient-free evolutionary approach for optimizing nonconvex objective functions. In our case, this objective function equals the average sampled return. In contrast to CEM, CMA-ES estimates the covariance matrix of a multivariate normal distribution through incremental adaption along evolution paths, which contain information about the correlation between consecutive updates.

4.2. Online Algorithms

Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015): Compared to batch algorithms, the DDPG algorithm continuously improves the policy as it explores the environment. It applies gradient descent to the policy with minibatch data sampled from a replay pool, where the gradient is computed via

$$\widehat{\nabla_\theta \eta(\mu_\theta)} = \sum_{i=1}^{B} \nabla_a Q_\phi(s_i, a)\big|_{a=\mu_\theta(s_i)} \nabla_\theta \mu_\theta(s_i)$$

where B is the batch size. The critic Q is trained via gradient descent on the ℓ^2 loss of the Bellman error $L = \frac{1}{B}\sum_{i=1}^{B}(y_i - Q_\phi(s_i, a_i))^2$, where $y_i = r_i + \gamma Q'_{\phi'}(s'_i, \mu'_{\theta'}(s'_i))$. To improve stability of the algorithm, we use target networks for both the critic and the policy when forming the regression target y_i. We refer the reader to Lillicrap et al. (2015) for a more detailed description of the algorithm.

4.3. Recurrent Variants

We implement direct applications of the aforementioned batch-based algorithms to recurrent policies. The only modification required is to replace $\pi(a_t^i|s_t^i)$ by $\pi(a_t^i|o_{1:t}^i, a_{1:t-1}^i)$, where $o_{1:t}^i$ and $a_{1:t-1}$ are the histories of past and current observations and past actions. Recurrent versions of reinforcement learning algorithms have been studied in many existing works, such as Bakker (2001), Schäfer & Udluft (2005), Wierstra et al. (2007), and Heess et al. (2015a).

5. Experiment Setup

In this section, we elaborate on the experimental setup used to generate the results.

Performance Metrics: For each report unit (a particular algorithm running on a particular task), we define its performance as $\frac{1}{\sum_{i=1}^{I} N_i}\sum_{i=1}^{I}\sum_{n=1}^{N_i} R_{in}$, where I is the number of training iterations, N_i is the number of trajectories collected in the ith iteration, and R_{in} is the undiscounted return for the nth trajectory of the ith iteration,

Hyperparameter Tuning: For the DDPG algorithm, we used the hyperparametes reported in Lillicrap et al. (2015). For the other algorithms, we follow the approach in (Mnih et al., 2015), and we select two tasks in each category, on which a grid search of hyperparameters is performed. Each choice of hyperparameters is executed under five random seeds. The criterion for the best hyperparameters is defined as $\mathrm{mean(returns)} - \mathrm{std(returns)}$. This metric selects against large fluctuations of performance due to overly large step sizes.

For the other tasks, we try both of the best hyperparameters found in the same category, and report the better performance of the two. This gives us insights into both the maximum possible performance when extensive hyperparameter tuning is performed, and the robustness of the best hyperparameters across different tasks.

Policy Representation: For basic, locomotion, and hierarchical tasks and for batch algorithms, we use a feed-forward neural network policy with 3 hidden layers, consisting of 100, 50, and 25 hidden units with tanh nonlinearity at the first two hidden layers, which map each state to the mean of a Gaussian distribution. The log-standard deviation is parameterized by a global vector independent of the state, as done in Schulman et al. (2015a). For all partially observable tasks, we use a recurrent neural network with a single hidden layer consisting of 32 LSTM hidden units (Hochreiter & Schmidhuber, 1997).

For the DDPG algorithm which trains a deterministic policy, we follow Lillicrap et al. (2015). For both the policy and the Q function, we use the same architecture of a feed-forward neural network with 2 hidden layers, consisting of 400 and 300 hidden units with relu activations.

Baseline: For all gradient-based algorithms except REPS, we can subtract a baseline from the empirical return to reduce variance of the optimization. We use a linear function as the baseline with a time-varying feature vector.

6. Results and Discussion

The main evaluation results are presented in Table 1. The tasks on which the grid search is performed are marked with (*). In each entry, the pair of numbers shows the mean and standard deviation of the normalized cumulative return using the best possible hyperparameters.

REINFORCE: Despite its simplicity, REINFORCE is an effective algorithm in optimizing deep neural network policies in most basic and locomotion tasks. Even for high-DOF tasks like Ant, REINFORCE can achieve competitive results. However we observe that REINFORCE sometimes suffers from premature convergence to local optima as noted by Peters & Schaal (2008), which explains the performance gaps between REINFORCE and TNPG on tasks such as Walker (Figure 3(a)). By visualizing the final policies, we can see that REINFORCE results in policies that tend to jump forward and fall over to maximize short-term return instead of acquiring a stable walking gait to maximize long-term return. In Figure 3(b), we can observe that even with a small learning rate, steps taken by REINFORCE can sometimes result in large changes to policy distribution, which may explain the fast convergence to local optima.

TNPG and TRPO: Both TNPG and TRPO outperform other batch algorithms by a large margin on most tasks, confirming that constraining the change in the policy distribution results in more stable learning (Peters & Schaal, 2008).

Compared to TNPG, TRPO offers better control over each

Table 1. Performance of the implemented algorithms in terms of average return over all training iterations for five different random seeds (same across all algorithms). The results of the best-performing algorithm on each task, as well as all algorithms that have performances that are not statistically significantly different (Welch's t-test with $p < 0.05$), are highlighted in boldface.[a] In the tasks column, the partially observable variants of the tasks are annotated as follows: LS stands for limited sensors, NO for noisy observations and delayed actions, and SI for system identifications. The notation N/A denotes that an algorithm has failed on the task at hand, e.g., CMA-ES leading to out-of-memory errors in the Full Humanoid task.

Task	Random	REINFORCE	TNPG	RWR	REPS	TRPO	CEM	CMA-ES	DDPG
Cart-Pole Balancing	77.1 ± 0.0	4693.7 ± 14.0	**3986.4 ± 748.9**	**4861.5 ± 12.3**	565.6 ± 137.6	**4869.8 ± 37.6**	4815.4 ± 4.8	2440.4 ± 568.3	4634.4 ± 87.8
Inverted Pendulum*	−153.4 ± 0.2	13.4 ± 18.0	**209.7 ± 55.5**	84.7 ± 13.8	−113.3 ± 4.6	**247.2 ± 76.1**	38.2 ± 25.7	−40.1 ± 5.7	40.0 ± 244.6
Mountain Car	−415.4 ± 0.0	−67.1 ± 1.0	**-66.5 ± 4.5**	−79.4 ± 1.1	−275.6 ± 166.3	**-61.7 ± 0.9**	−66.0 ± 2.4	−85.0 ± 7.7	−288.4 ± 170.3
Acrobot	−1904.5 ± 1.0	−508.1 ± 91.0	−395.8 ± 121.2	−352.7 ± 35.9	−1001.5 ± 10.8	−326.0 ± 24.4	−436.8 ± 14.7	−785.6 ± 13.1	**-223.6 ± 5.8**
Double Inverted Pendulum*	149.7 ± 0.1	4116.5 ± 65.2	**4455.4 ± 37.6**	3614.8 ± 368.1	446.7 ± 114.8	**4412.4 ± 50.4**	2566.2 ± 178.9	1576.1 ± 51.3	2863.4 ± 154.0
Swimmer*	−1.7 ± 0.1	92.3 ± 0.1	**96.0 ± 0.2**	60.7 ± 5.5	3.8 ± 3.3	**96.0 ± 0.2**	68.8 ± 2.4	64.9 ± 1.4	85.8 ± 1.8
Hopper	8.4 ± 0.0	714.0 ± 29.3	**1155.1 ± 57.9**	553.2 ± 71.0	86.7 ± 17.6	**1183.3 ± 150.0**	63.1 ± 7.8	20.3 ± 14.3	267.1 ± 43.5
2D Walker	−1.7 ± 0.0	506.5 ± 78.8	**1382.6 ± 108.2**	136.0 ± 15.9	−37.0 ± 38.1	**1353.8 ± 85.0**	84.5 ± 19.2	77.1 ± 24.3	318.4 ± 181.6
Half-Cheetah	−90.8 ± 0.3	1183.1 ± 69.2	**1729.5 ± 184.6**	376.1 ± 28.2	34.5 ± 38.0	**1914.0 ± 120.1**	330.4 ± 274.8	441.3 ± 107.6	**2148.6 ± 702.7**
Ant*	13.4 ± 0.7	548.3 ± 55.5	**706.0 ± 127.7**	37.6 ± 3.1	39.0 ± 9.8	**730.2 ± 61.3**	49.2 ± 5.9	17.8 ± 15.5	326.2 ± 20.8
Simple Humanoid	41.5 ± 0.2	128.1 ± 34.0	**255.0 ± 24.5**	93.3 ± 17.4	28.3 ± 4.7	**269.7 ± 40.3**	60.6 ± 12.9	28.7 ± 3.9	99.4 ± 28.1
Full Humanoid	13.2 ± 0.1	262.2 ± 10.5	**288.4 ± 25.2**	46.7 ± 5.6	41.7 ± 6.1	**287.0 ± 23.4**	36.9 ± 2.9	N/A ± N/A	119.0 ± 31.2
Cart-Pole Balancing (LS)*	77.1 ± 0.0	420.9 ± 265.5	**945.1 ± 27.8**	68.9 ± 1.5	898.1 ± 22.1	**960.2 ± 46.0**	227.0 ± 223.0	68.0 ± 1.6	
Inverted Pendulum (LS)	−122.1 ± 0.1	−13.4 ± 3.2	**0.7 ± 6.1**	−107.4 ± 0.2	−87.2 ± 8.0	**4.5 ± 4.1**	−81.2 ± 33.2	−62.4 ± 3.4	
Mountain Car (LS)	−83.0 ± 0.0	−81.2 ± 0.6	**-65.7** ± 9.0	−81.7 ± 0.1	−82.6 ± 0.4	**-64.2 ± 9.5**	**-68.9 ± 1.3**	**-73.2 ± 0.6**	
Acrobot (LS)*	−393.2 ± 0.0	−128.9 ± 11.6	**-84.6 ± 2.9**	−235.9 ± 5.3	−379.5 ± 1.4	**-83.3 ± 9.9**	−149.5 ± 15.3	−159.9 ± 7.5	
Cart-Pole Balancing (NO)*	101.4 ± 0.1	616.0 ± 210.8	**916.3 ± 23.0**	93.8 ± 1.2	99.6 ± 7.2	606.2 ± 122.2	181.4 ± 32.1	104.4 ± 16.0	
Inverted Pendulum (NO)	−122.2 ± 0.1	6.5 ± 1.1	**11.5 ± 0.5**	−110.0 ± 1.4	−119.3 ± 4.2	**10.4 ± 2.2**	−55.6 ± 16.7	−80.3 ± 2.8	
Mountain Car (NO)	−83.0 ± 0.0	−74.7 ± 7.8	**-64.5 ± 8.6**	−81.7 ± 0.1	−82.9 ± 0.1	**-60.2 ± 2.0**	−67.4 ± 1.4	−73.5 ± 0.5	
Acrobot (NO)*	−393.5 ± 0.0	**-186.7 ± 31.3**	**-164.5 ± 13.4**	−233.1 ± 0.4	−258.5 ± 14.0	**-149.6 ± 8.6**	−213.4 ± 6.3	−236.6 ± 6.2	
Cart-Pole Balancing (SI)*	76.3 ± 0.1	431.7 ± 274.1	**980.5 ± 7.3**	69.0 ± 2.8	702.4 ± 196.4	**980.3 ± 5.1**	746.6 ± 93.2	71.6 ± 2.9	
Inverted Pendulum (SI)	−121.8 ± 0.2	−5.3 ± 5.6	**14.8 ± 1.7**	−108.7 ± 4.7	−92.8 ± 23.9	**14.1 ± 0.9**	−51.8 ± 10.6	−63.1 ± 4.8	
Mountain Car (SI)	−82.7 ± 0.0	−63.9 ± 0.2	**-61.8 ± 0.4**	−81.4 ± 0.1	−80.7 ± 2.3	**-61.6 ± 0.4**	−63.9 ± 1.0	−66.9 ± 0.6	
Acrobot (SI)*	−387.8 ± 1.0	**-169.1 ± 32.3**	**-156.6 ± 38.9**	−233.2 ± 2.6	−216.1 ± 7.7	**-170.9 ± 40.3**	−250.2 ± 13.7	−245.0 ± 5.5	
Swimmer + Gathering	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Gathering	−5.8 ± 5.0	−0.1 ± 0.1	−0.4 ± 0.1	−5.5 ± 0.5	−6.7 ± 0.7	−0.4 ± 0.0	−4.7 ± 0.7	N/A ± N/A	−0.3 ± 0.3
Swimmer + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	N/A ± N/A	0.0 ± 0.0

[a] Except for the hierarchical tasks

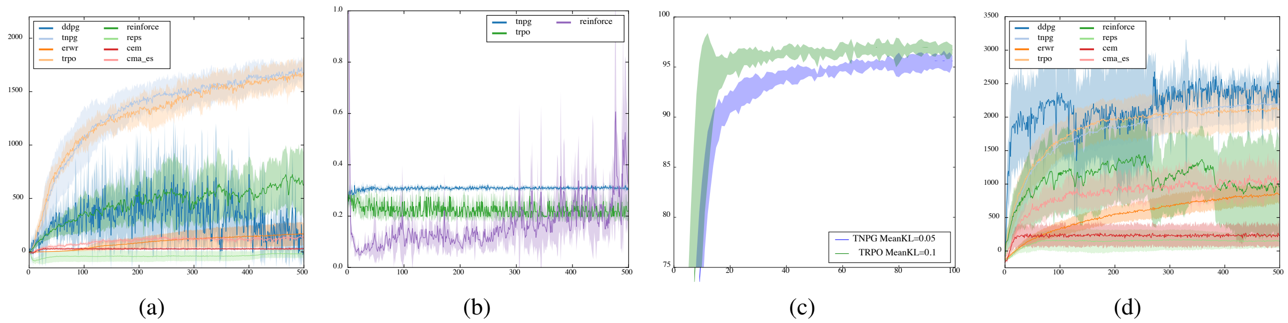

Figure 3. Performance as a function of the number of iterations; the shaded area depicts the mean $\pm$ the standard deviation over five different random seeds: (a) Performance comparison of all algorithms in terms of the average reward on the Walker task; (b) Comparison between REINFORCE, TNPG, and TRPO in terms of the mean KL-divergence on the Walker task; (c) Performance comparison on TNPG and TRPO on the Swimmer task; (d) Performance comparison of all algorithms in terms of the average reward on the Half-Cheetah task.

policy update by performing a line search in the natural gradient direction to ensure an improvement in the surrogate loss function. We observe that hyperparameter grid search tends to select conservative step sizes (δ_{KL}) for TNPG, which alleviates the issue of performance collapse caused by a large update to the policy. By contrast, TRPO can robustly enforce constraints with larger a δ_{KL} value and hence speeds up learning in some cases. For instance, grid search on the Swimmer task reveals that the best step size for TNPG is $\delta_{\text{KL}} = 0.05$, whereas TRPO's best step-size is larger: $\delta_{\text{KL}} = 0.1$. As shown in Figure 3(c), this larger step size enables slightly faster learning.

RWR: RWR is the only gradient-based algorithm we implemented that does not require any hyperparameter tuning. It can solve some basic tasks to a satisfactory degree, but fails to solve more challenging tasks such as locomotion. We observe empirically that RWR shows fast initial improvement followed by significant slow-down, as shown in Figure 3(d).

REPS: Our main observation is that REPS is especially prone to early convergence to local optima in case of continuous states and actions. Its final outcome is greatly affected by the performance of the initial policy, an observation that is consistent with the original work of Peters et al. (2010). This leads to a bad performance on average, although under particular initial settings the algorithm can perform on par with others. Moreover, the tasks presented here do not assume the existence of a stationary distribution, which is assumed in Peters et al. (2010). In particular, for many of our tasks, transient behavior is of much greater interest than steady-state behavior, which agrees with previous observation by van Hoof et al. (2015),

Gradient-free methods: Surprisingly, even when training deep neural network policies with thousands of parameters, CEM achieves very good performance on certain basic tasks such as Cart-Pole Balancing and Mountain Car, suggesting that the dimension of the searching parameter is not always the limiting factor of the method. However, the performance degrades quickly as the system dynamics becomes more complicated. We also observe that CEM outperforms CMA-ES, which is remarkable as CMA-ES estimates the full covariance matrix. For higher-dimensional policy parameterizations, the computational complexity and memory requirement for CMA-ES become noticeable. On tasks with high-dimensional observations, such as the Full Humanoid, the CMA-ES algorithm runs out of memory and fails to yield any results, denoted as N/A in Table 1.

DDPG: Compared to batch algorithms, we found that DDPG was able to converge significantly faster on certain tasks like Half-Cheetah due to its greater sample efficiency. However, it was less stable than batch algorithms, and the performance of the policy can degrade significantly during training. We also found it to be more susceptible to scaling of the reward. In our experiment for DDPG, we rescaled the reward of all tasks by a factor of 0.1, which seems to improve the stability.

Partially Observable Tasks: We experimentally verify that recurrent policies can find better solutions than feed-forward policies in Partially Observable Tasks but recurrent policies are also more difficult to train. As shown in Table 1, derivative-free algorithms like CEM and CMA-ES work considerably worse with recurrent policies. Also we note that the performance gap between REINFORCE and TNPG widens when they are applied to optimize recurrent policies, which can be explained by the fact that a small change in parameter space can result in a bigger change in policy distribution with recurrent policies than with feed-forward policies.

Hierarchical Tasks: We observe that all of our implemented algorithms achieve poor performance on the hierarchical tasks, even with extensive hyperparameter search and 500 iterations of training. It is an interesting direction to develop algorithms that can automatically discover and exploit the hierarchical structure in these tasks.

7. Related Work

In this section, we review existing benchmarks of continuous control tasks. The earliest efforts of evaluating reinforcement learning algorithms started in the form of individual control problems described in symbolic form. Some widely adopted tasks include the inverted pendulum (Stephenson, 1908; Donaldson, 1960; Widrow, 1964), mountain car (Moore, 1990), and Acrobot (DeJong & Spong, 1994). These problems are frequently incorporated into more comprehensive benchmarks.

Some reinforcement learning benchmarks contain low-dimensional continuous control tasks, such as the ones introduced above, including RLLib (Abeyruwan, 2013), MMLF (Metzen & Edgington, 2011), RL-Toolbox (Neumann, 2006), JRLF (Kochenderfer, 2006), Beliefbox (Dimitrakakis et al., 2007), Policy Gradient Toolbox (Peters, 2002), and ApproxRL (Busoniu, 2010). A series of RL competitions has also been held in recent years (Dutech et al., 2005; Dimitrakakis et al., 2014), again with relatively low-dimensional actions. In contrast, our benchmark contains a wider range of tasks with high-dimensional continuous state and action spaces.

Previously, other benchmarks have been proposed for high-dimensional control tasks. Tdlearn (Dann et al., 2014) includes a 20-link pole balancing task, DotRL (Papis & Wawrzyński, 2013) includes a variable-DOF octopus arm and a 6-DOF planar cheetah model, PyBrain (Schaul et al., 2010) includes a 16-DOF humanoid robot with standing and jumping tasks, RoboCup Keepaway (Stone et al., 2005) is a multi-agent game which can have a flexible dimension of actions by varying the number of agents, and SkyAI (Yamaguchi & Ogasawara, 2010) includes a 17-DOF humanoid robot with crawling and turning tasks. Other libraries such as CL-Square (Riedmiller et al., 2012) and RLPark (Degris et al., 2013) provide interfaces to actual hardware, e.g., Bioloid and iRobot Create. In contrast to these aforementioned testbeds, our benchmark makes use of simulated environments to reduce computation time and to encourage experimental reproducibility. Furthermore, it provides a much larger collection of tasks of varying difficulty.

8. Conclusion

In this work, a benchmark of continuous control problems for reinforcement learning is presented, covering a wide variety of challenging tasks. We implemented several reinforcement learning algorithms, and presented them in the context of general policy parameterizations. Results show that among the implemented algorithms, TNPG, TRPO, and DDPG are effective methods for training deep neural network policies. Still, the poor performance on the proposed hierarchical tasks calls for new algorithms to be developed. Implementing and evaluating existing and newly proposed algorithms will be our continued effort. By providing an open-source release of the benchmark, we encourage other researchers to evaluate their algorithms on the proposed tasks.

Acknowledgements

We thank Emo Todorov and Yuval Tassa for providing the MuJoCo simulator, and Sergey Levine, Aviv Tamar, Chelsea Finn, and the anonymous ICML reviewers for insightful comments. We also thank Shixiang Gu and Timothy Lillicrap for helping us diagnose the DDPG implementation. This work was supported in part by DARPA, the Berkeley Vision and Learning Center (BVLC), the Berkeley Artificial Intelligence Research (BAIR) laboratory, and Berkeley Deep Drive (BDD). Rein Houthooft is supported by a Ph.D. Fellowship of the Research Foundation - Flanders (FWO).

References

Abeyruwan, S. RLLib: Lightweight standard and on/off policy reinforcement learning library (C++). http://web.cs.miami.edu/home/saminda/rilib.html, 2013.

Bagnell, J. A. and Schneider, J. Covariant policy search. pp. 1019–1024. IJCAI, 2003.

Bakker, B. Reinforcement learning with long short-term memory. In *NIPS*, pp. 1475–1482, 2001.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The Arcade Learning Environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, 2013.

Bellman, R. *Dynamic Programming*. Princeton University Press, 1957.

Bertsekas, Dimitri P and Tsitsiklis, John N. Neuro-dynamic programming: an overview. In *CDC*, pp. 560–564, 1995.

Busoniu, L. ApproxRL: A Matlab toolbox for approximate RL and DP. http://busoniu.net/files/repository/readme-approxrl.html, 2010.

Catto, E. Box2D: A 2D physics engine for games, 2011.

Coulom, Rémi. *Reinforcement learning using neural networks, with applications to motor control*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2002.

Dann, C., Neumann, G., and Peters, J. Policy evaluation with temporal differences: A survey and comparison. *J. Mach. Learn. Res.*, 15(1):809–883, 2014.

Degris, T., Béchu, J., White, A., Modayil, J., Pilarski, P. M., and Denk, C. RLPark. http://rlpark.github.io, 2013.

Deisenroth, M. P., Neumann, G., and Peters, J. A survey on policy search for robotics, foundations and trends in robotics. *Found. Trends Robotics*, 2(1-2):1–142, 2013.

DeJong, G. and Spong, M. W. Swinging up the Acrobot: An example of intelligent control. In *ACC*, pp. 2158–2162, 1994.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255, 2009.

Dietterich, T. G. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res*, 13: 227–303, 2000.

Dimitrakakis, C., Tziortziotis, N., and Tossou, A. Beliefbox: A framework for statistical methods in sequential decision making. http://code.google.com/p/beliefbox/, 2007.

Dimitrakakis, Christos, Li, Guangliang, and Tziortziotis, Nikoalos. The reinforcement learning competition 2014. *AI Magazine*, 35(3):61–65, 2014.

Donaldson, P. E. K. Error decorrelation: a technique for matching a class of functions. In *Proc. 3th Intl. Conf. Medical Electronics*, pp. 173–178, 1960.

Doya, K. Reinforcement learning in continuous time and space. *Neural Comput.*, 12(1):219–245, 2000.

Dutech, Alain, Edmunds, Timothy, Kok, Jelle, Lagoudakis, Michail, Littman, Michael, Riedmiller, Martin, Russell, Bryan, Scherrer, Bruno, Sutton, Richard, Timmer, Stephan, et al. Reinforcement learning benchmarks and bake-offs ii. *Advances in Neural Information Processing Systems (NIPS)*, 17, 2005.

Erez, Tom, Tassa, Yuval, and Todorov, Emanuel. Infinite horizon model predictive control for nonlinear periodic tasks. *Manuscript under review*, 4, 2011.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The pascal visual object classes (VOC) challenge. *Int. J. Comput. Vision*, 88(2):303–338, 2010.

Fei-Fei, L., Fergus, R., and Perona, P. One-shot learning of object categories. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(4):594–611, 2006.

Furuta, K., Okutani, T., and Sone, H. Computer control of a double inverted pendulum. *Comput. Electr. Eng.*, 5(1):67–84, 1978.

Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., and Pallett, D. S. DARPA TIMIT acoustic-phonetic continuous speech corpus CD-ROM. NIST speech disc 1-1.1. *NASA STI/Recon Technical Report N*, 93, 1993.

Godfrey, J. J., Holliman, E. C., and McDaniel, J. SWITCHBOARD: Telephone speech corpus for research and development. In *ICASSP*, pp. 517–520, 1992.

Gomez, F. and Miikkulainen, R. 2-d pole balancing with recurrent evolutionary networks. In *ICANN*, pp. 425–430. 1998.

Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. Deep learning for real-time Atari game play using offline monte-carlo tree search planning. In *NIPS*, pp. 3338–3346. 2014.

Hansen, N. and Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, 2001.

Heess, N., Hunt, J., Lillicrap, T., and Silver, D. Memory-based control with recurrent neural networks. *arXiv:1512.04455*, 2015a.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, T. Learning continuous control policies by stochastic value gradients. In *NIPS*, pp. 2926–2934. 2015b.

Hester, T. and Stone, P. The open-source TEXPLORE code release for reinforcement learning on robots. In *RoboCup 2013: Robot World Cup XVII*, pp. 536–543. 2013.

Hinton, G., Deng, L., Yu, D., Mohamed, A.-R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Dahl, T. S. G., and Kingsbury, B. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Process. Mag*, 29(6):82–97, 2012.

Hirsch, H.-G. and Pearce, D. The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In *ASR2000-Automatic Speech Recognition: Challenges for the new Millenium ISCA Tutorial and Research Workshop (ITRW)*, 2000.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

Kakade, S. M. A natural policy gradient. In *NIPS*, pp. 1531–1538. 2002.

Kimura, H. and Kobayashi, S. Stochastic real-valued reinforcement learning to solve a nonlinear control problem. In *IEEE SMC*, pp. 510–515, 1999.

Kober, J. and Peters, J. Policy search for motor primitives in robotics. In *NIPS*, pp. 849–856, 2009.

Kochenderfer, M. JRLF: Java reinforcement learning framework. http://mykel.kochenderfer.com/jrlf, 2006.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. ImageNet classification with deep convolutional neural networks. In *NIPS*, pp. 1097–1105. 2012.

LeCun, Y., Cortes, C., and Burges, C. The MNIST database of handwritten digits, 1998.

Levine, S. and Koltun, V. Guided policy search. In *ICML*, pp. 1–9, 2013.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *arXiv:1504.00702*, 2015.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. arXiv:1509.02971, 2015.

Martin, D., C. Fowlkes, D. Tal, and Malik, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, pp. 416–423, 2001.

Metzen, J. M. and Edgington, M. Maja machine learning framework. http://mloss.org/software/view/220/, 2011.

Michie, D. and Chambers, R. A. BOXES: An experiment in adaptive control. *Machine Intelligence*, 2:137–152, 1968.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Moore, A. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990.

Murray, R. M. and Hauser, J. A case study in approximate linearization: The Acrobot example. Technical report, UC Berkeley, EECS Department, 1991.

Murthy, S. S. and Raibert, M. H. 3D balance in legged locomotion: modeling and simulation for the one-legged case. *ACM SIGGRAPH Computer Graphics*, 18(1):27–27, 1984.

Neumann, G. A reinforcement learning toolbox and RL benchmarks for the control of dynamical systems. *Dynamical principles for neuroscience and intelligent biomimetic devices*, pp. 113, 2006.

Papis, B. and Wawrzyński, P. dotrl: A platform for rapid reinforcement learning methods development and validation. In *FedCSIS*, pp. pages 129–136., 2013.

Parr, Ronald and Russell, Stuart. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pp. 1043–1049, 1998.

Peters, J. Policy Gradient Toolbox. http://www.ausy.tu-darmstadt.de/Research/PolicyGradientToolbox, 2002.

Peters, J. and Schaal, S. Reinforcement learning by reward-weighted regression for operational space control. In *ICML*, pp. 745–750, 2007.

Peters, J. and Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

Peters, J., Vijaykumar, S., and Schaal, S. Policy gradient methods for robot control. Technical report, 2003.

Peters, J., Mülling, K., and Altün, Y. Relative entropy policy search. In *AAAI*, pp. 1607–1612, 2010.

Purcell, E. M. Life at low Reynolds number. *Am. J. Phys*, 45(1): 3–11, 1977.

Raibert, M. H. and Hodgins, J. K. Animation of dynamic legged locomotion. In *ACM SIGGRAPH Computer Graphics*, volume 25, pp. 349–358, 1991.

Riedmiller, M., Blum, M., and Lampe, T. CLS2: Closed loop simulation system. http://ml.informatik.uni-freiburg.de/research/clsquare, 2012.

Rubinstein, R. The cross-entropy method for combinatorial and continuous optimization. *Methodol. Comput. Appl. Probab.*, 1 (2):127–190, 1999.

Schäfer, A. M. and Udluft, S. Solving partially observable reinforcement learning problems with recurrent neural networks. In *ECML Workshops*, pp. 71–81, 2005.

Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., Rückstieß, T., and Schmidhuber, J. PyBrain. *J. Mach. Learn. Res.*, 11:743–746, 2010.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. Trust region policy optimization. In *ICML*, pp. 1889–1897, 2015a.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. arXiv:1506.02438, 2015b.

Stephenson, A. On induced stability. *Philos. Mag.*, 15(86):233–236, 1908.

Stone, Peter, Kuhlmann, Gregory, Taylor, Matthew E, and Liu, Yaxin. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*, pp. 93–105. Springer, 2005.

Sutton, Richard S, Precup, Doina, and Singh, Satinder. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

Szita, I. and Lőrincz, A. Learning Tetris using the noisy cross-entropy method. *Neural Comput.*, 18(12):2936–2941, 2006.

Szita, I., Takács, B., and Lörincz, A. ε-MDPs: Learning in varying environments. *J. Mach. Learn. Res.*, 3:145–174, 2003.

Tassa, Yuval, Erez, Tom, and Todorov, Emanuel. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 4906–4913. IEEE, 2012.

Tesauro, G. Temporal difference learning and TD-Gammon. *Commun. ACM*, 38(3):58–68, 1995.

Todorov, E., Erez, T., and Tassa, Y. MuJoCo: A physics engine for model-based control. In *IROS*, pp. 5026–5033, 2012.

van Hoof, H., Peters, J., and Neumann, G. Learning of non-parametric control policies with high-dimensional state features. In *AISTATS*, pp. 995–1003, 2015.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, pp. 2728–2736, 2015.

Wawrzyński, P. Learning to control a 6-degree-of-freedom walking robot. In *IEEE EUROCON*, pp. 698–705, 2007.

Widrow, B. Pattern recognition and adaptive control. *IEEE Trans. Ind. Appl.*, 83(74):269–277, 1964.

Wierstra, D., Foerster, A., Peters, J., and Schmidhuber, J. Solving deep memory POMDPs with recurrent policy gradients. In *ICANN*, pp. 697–706. 2007.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8: 229–256, 1992.

Yamaguchi, A. and Ogasawara, T. SkyAI: Highly modularized reinforcement learning library. In *IEEE-RAS Humanoids*, pp. 118–123, 2010.

Yu, D., Ju, Y.-C., Wang, Y.-Y., Zweig, G., and Acero, A. Automated directory assistance system - from theory to practice. In *Interspeech*, pp. 2709–2712, 2007.

Supplementary Material

1. Task Specifications

Below we provide some specifications for the task observations, actions, and rewards. Please refer to the benchmark source code (https://github.com/rllab/rllab) for complete specification of physics parameters.

1.1. Basic Tasks

Cart-Pole Balancing: In this task, an inverted pendulum is mounted on a pivot point on a cart. The cart itself is restricted to linear movement, achieved by applying horizontal forces. Due to the system's inherent instability, continuous cart movement is needed to keep the pendulum upright. The observation consists of the cart position x, pole angle θ, the cart velocity $\dot{x}$, and the pole velocity $\dot{\theta}$. The 1D action consists of the horizontal force applied to the cart body. The reward function is given by $r(s,a) := 10 - (1-\cos(\theta)) - 10^{-5}\|a\|_2^2$. The episode terminates when $|x| > 2.4$ or $|\theta| > 0.2$.

Cart-Pole Swing Up: This is a more complicated version of the previous task, in which the system should not only be able to balance the pole, but first succeed in swinging it up into an upright position. This task extends the working range of the inverted pendulum to 360°. This is a nonlinear extension of the previous task. It has the same observation and action as in balancing. The reward function is given by $r(s,a) := \cos(\theta)$. The episode terminates when $|x| > 3$, with a penalty of -100.

Mountain Car: In this task, a car has to escape a valley by repetitive application of tangential forces. Because the maximal tangential force is limited, the car has to alternately drive up along the two slopes of the valley in order to build up enough inertia to overcome gravity. This brings a challenge of exploration, since before first reaching the goal among all trials, a locally optimal solution exists, which is to drive to the point closest to the target and stay there for the rest of the episode. The observation is given by the horizontal position x and the horizontal velocity $\dot{x}$ of the car. The reward is given by $r(s,a) := -1 + \text{height}$, with height the car's vertical offset. The episode terminates when the car reaches a target height of 0.6. Hence the goal is to reach the target as soon as possible.

Acrobot Swing Up: In this task, an under-actuated, two-link robot has to swing itself into an upright position. It consists of two joints of which the first one has a fixed position and only the second one can exert torque. The goal is to swing the robot into an upright position and stabilize around that position. The controller not only has to swing the pendulum in order to build up inertia, similar to the Mountain Car task, but also has to decelerate it in order to prevent it from tipping over. The observation includes the two joint angles, θ_1 and θ_2, and their velocities, $\dot{\theta}_1$ and $\dot{\theta}_2$. The action is the torque applied at the second joint. The reward is defined as $r(s,a) := -\|\text{tip}(s) - \text{tip}_{\text{target}}\|_2$, where $\text{tip}(s)$ computes the Cartesian position of the tip of the robot given the joint angles. No termination condition is applied.

Double Inverted Pendulum Balancing: This task extends the Cart-Pole Balancing task by replacing the single-link pole by a two-link rigid structure. As in the former task, the goal is to stabilize the two-link pole near the upright position. This task is more difficult than single-pole balancing, since the system is even more unstable and requires the controller to actively maintain balance. The observation includes the cart position x, joint angles (θ_1 and θ_2), and joint velocities ($\dot{\theta}_1$ and $\dot{\theta}_2$). We encode each joint angle as its sine and cosine values. The action is the same as in cart-pole tasks. The reward is given by $r(s,a) = 10 - 0.01x_{\text{tip}}^2 - (y_{\text{tip}} - 2)^2 - 10^{-3}\cdot\dot{\theta}_1^2 - 5\cdot 10^{-3}\cdot\dot{\theta}_2^2$, where $x_{\text{tip}}, y_{\text{tip}}$ are the coordinates of the tip of the pole. No termination condition is applied. The episode is terminated when $y_{\text{tip}} \leq 1$.

1.2. Locomotion Tasks

Swimmer: The swimmer is a planar robot with 3 links and 2 actuated joints. Fluid is simulated through viscosity forces, which apply drag on each link, allowing the swimmer to move forward. This task is the simplest of all locomotion tasks, since there are no irrecoverable states in which the swimmer can get stuck, unlike other robots which may fall down or flip over. This places less burden on exploration. The 13-dim observation includes the joint angles, joint velocities, as well as the coordinates of the center of mass. The reward is given by $r(s,a) = v_x - 0.005\|a\|_2^2$, where v_x is the forward velocity. No termination condition is applied.

Hopper: The hopper is a planar monopod robot with 4 rigid links, corresponding to the torso, upper leg, lower leg, and foot, along with 3 actuated joints. More exploration is needed than the swimmer task, since a stable hopping gait has to be learned without falling. Otherwise, it may get stuck in a local optimum of diving forward. The 20-dim observation includes joint angles, joint velocities, the coordinates of center of mass, and constraint forces. The reward is given by $r(s,a) := v_x - 0.005 \cdot \|a\|_2^2 + 1$, where the last term is a bonus for being "alive." The episode is terminated when $z_{body} < 0.7$ where z_{body} is the z-coordinate of the body, or when $|\theta_y| < 0.2$, where θ_y is the forward pitch of the body.

Walker: The walker is a planar biped robot consisting of 7 links, corresponding to two legs and a torso, along with 6 actuated joints. This task is more challenging than hopper, since it has more degrees of freedom, and is also prone to falling. The 21-dim observation includes joint angles, joint velocities, and the coordinates of center of mass. The reward is given by $r(s,a) := v_x - 0.005 \cdot \|a\|_2^2$. The episode is terminated when $z_{body} < 0.8$, $z_{body} > 2.0$, or when $|\theta_y| > 1.0$.

Half-Cheetah: The half-cheetah is a planar biped robot with 9 rigid links, including two legs and a torso, along with 6 actuated joints. The 20-dim observation includes joint angles, joint velocities, and the coordinates of the center of mass. The reward is given by $r(s,a) = v_x - 0.05 \cdot \|a\|_2^2$. No termination condition is applied.

Ant: The ant is a quadruped with 13 rigid links, including four legs and a torso, along with 8 actuated joints. This task is more challenging than the previous tasks due to the higher degrees of freedom. The 125-dim observation includes joint angles, joint velocities, coordinates of the center of mass, a (usually sparse) vector of contact forces, as well as the rotation matrix for the body. The reward is given by $r(s,a) = v_x - 0.005 \cdot \|a\|_2^2 - C_{\text{contact}} + 0.05$, where C_{contact} penalizes contacts to the ground, and is given by $5 \cdot 10^{-4} \cdot \|F_{\text{contact}}\|_2^2$, where F_{contact} is the contact force vector clipped to values between -1 and 1. The episode is terminated when $z_{body} < 0.2$ or when $z_{body} > 1.0$.

Simple Humanoid: This is a simplified humanoid model with 13 rigid links, including the head, body, arms, and legs, along with 10 actuated joints. The increased difficulty comes from the increased degrees of freedom as well as the need to maintain balance. The 102-dim observation includes the joint angles, joint velocities, vector of contact forces, and the coordinates of the center of mass. The reward is given by $r(s,a) = v_x - 5 \cdot 10^{-4}\|a\|_2^2 - C_{\text{contact}} - C_{\text{deviation}} + 0.2$, where $C_{\text{contact}} = 5 \cdot 10^{-6} \cdot \|F_{\text{contact}}\|$, and $C_{\text{deviation}} = 5 \cdot 10^{-3} \cdot (v_y^2 + v_z^2)$ to penalize deviation from the forward direction. The episode is terminated when $z_{body} < 0.8$ or when $z_{body} > 2.0$.

Full Humanoid: This is a humanoid model with 19 rigid links and 28 actuated joints. It has more degrees of freedom below the knees and elbows, which makes the system higher-dimensional and harder for learning. The 142-dim observation includes the joint angles, joint velocities, vector of contact forces, and the coordinates of the center of mass. The reward and termination condition is the same as in the Simple Humanoid model.

1.3. Partially Observable Tasks

Limited Sensors: The full description is included in the main text.

Noisy Observations and Delayed Actions: For all tasks, we use a Gaussan noise with $\sigma = 0.1$. The time delay is as follows: Cart-Pole Balancing 0.15 sec, Cart-Pole Swing Up 0.15 sec, Mountain Car 0.15 sec, Acrobot Swing Up 0.06 sec, and Double Inverted Pendulum Balancing 0.06 sec. This corresponds to 3 discretization frames for each task.

System Identifications: For Cart-Pole Balancing and Cart-Pole Swing Up, the pole length is varied uniformly between, 50% and 150%. For Mountain Car, the width of the valley varies uniformly between 75% and 125%. For Acrobot Swing Up, each of the pole length varies uniformly between 50% and 150%. For Double Inverted Pendulum Balancing, each of the pole length varies uniformly between 83% and 167%. Please refer to the benchmark source code for reference values.

1.4. Hierarchical Tasks

Locomotion + Food Collection: During each episode, 8 food units and 8 bombs are placed in the environment. Collecting a food unit gives $+1$ reward, and collecting a bomb gives -1 reward. Hence the best cumulative reward for a given episode is 8.

Locomotion + Maze: During each episode, a $+1$ reward is given when the robot reaches the goal. Otherwise, the robot receives a zero reward throughout the episode.

2. Experiment Parameters

For all batch gradient-based algorithms, we use the same time-varying feature encoding for the linear baseline:

$$\phi_{s,t} = \text{concat}(s, s \odot s, 0.01t, (0.01t)^2, (0.01t)^3, 1)$$

where s is the state vector and $\odot$ represents element-wise product.

Table 2 shows the experiment parameters for all four categories. We will then detail the hyperparameter search range for the selected tasks and report best hyperparameters, shown in Tables 3, 4, 5, 6, 7, and 8.

Table 2. Experiment Setup

	Basic & Locomotion	Partially Observable	Hierarchical
Sim. steps per Iter.	50,000	50,000	50,000
Discount(λ)	0.99	0.99	0.99
Horizon	500	100	500
Num. Iter.	500	300	500

Table 3. Learning Rate α for REINFORCE

	Search Range	Best
Cart-Pole Swing Up	$[1 \times 10^{-4}, 1 \times 10^{-1}]$	5×10^{-3}
Double Inverted Pendulum	$[1 \times 10^{-4}, 1 \times 10^{-1}]$	5×10^{-3}
Swimmer	$[1 \times 10^{-4}, 1 \times 10^{-1}]$	1×10^{-2}
Ant	$[1 \times 10^{-4}, 1 \times 10^{-1}]$	5×10^{-3}

Table 4. Step Size δ_{KL} for TNPG

	Search Range	Best
Cart-Pole Swing Up	$[1 \times 10^{-3}, 5 \times 10^{0}]$	5×10^{-2}
Double Inverted Pendulum	$[1 \times 10^{-3}, 5 \times 10^{0}]$	3×10^{-2}
Swimmer	$[1 \times 10^{-3}, 5 \times 10^{0}]$	1×10^{-1}
Ant	$[1 \times 10^{-3}, 5 \times 10^{0}]$	3×10^{-1}

Table 5. Step Size δ_{KL} for TRPO

	Search Range	Best
Cart-Pole Swing Up	$[1 \times 10^{-3}, 5 \times 10^{0}]$	5×10^{-2}
Double Inverted Pendulum	$[1 \times 10^{-3}, 5 \times 10^{0}]$	1×10^{-3}
Swimmer	$[1 \times 10^{-3}, 5 \times 10^{0}]$	5×10^{-2}
Ant	$[1 \times 10^{-3}, 5 \times 10^{0}]$	8×10^{-2}

Table 6. Step Size δ_{KL} for REPS

	Search Range	Best
Cart-Pole Swing Up	$[1 \times 10^{-3}, 5 \times 10^{0}]$	1×10^{-2}
Double Inverted Pendulum	$[1 \times 10^{-3}, 5 \times 10^{0}]$	8×10^{-1}
Swimmer	$[1 \times 10^{-3}, 5 \times 10^{0}]$	3×10^{-1}
Ant	$[1 \times 10^{-3}, 5 \times 10^{0}]$	8×10^{-1}

Table 7. Initial Extra Noise for CEM

	Search Range	Best
Cart-Pole Swing Up	$[1 \times 10^{-3}, 1]$	1×10^{-2}
Double Inverted Pendulum	$[1 \times 10^{-3}, 1]$	1×10^{-1}
Swimmer	$[1 \times 10^{-3}, 1]$	1×10^{-1}
Ant	$[1 \times 10^{-3}, 1]$	1×10^{-1}

Table 8. Initial Standard Deviation for CMA-ES

	Search Range	Best
Cart-Pole Swing Up	$[1 \times 10^{-3}, 1 \times 10^{3}]$	1×10^{3}
Double Inverted Pendulum	$[1 \times 10^{-3}, 1 \times 10^{3}]$	3×10^{-1}
Swimmer	$[1 \times 10^{-3}, 1 \times 10^{3}]$	1×10^{-1}
Ant	$[1 \times 10^{-3}, 1 \times 10^{3}]$	1×10^{-1}