Learning to Walk via Deep Reinforcement Learning

Tuomas Haarnoja*,[1,2], Sehoon Ha*,[1], Aurick Zhou[2], Jie Tan[1], George Tucker[1] and Sergey Levine[1,2]
[1]Google Brain [2]Berkeley Artificial Intelligence Research, University of California, Berkeley
Email: {tuomash,sehoonha,jietan,gjt}@google.com,{azhou42,svlevine}@berkeley.edu
*The first two authors contributed equally.

***Abstract*—Deep reinforcement learning (deep RL) holds the promise of automating the acquisition of complex controllers that can map sensory inputs directly to low-level actions. In the domain of robotic locomotion, deep RL could enable learning locomotion skills with minimal engineering and without an explicit model of the robot dynamics. Unfortunately, applying deep RL to real-world robotic tasks is exceptionally difficult, primarily due to poor sample complexity and sensitivity to hyperparameters. While hyperparameters can be easily tuned in simulated domains, tuning may be prohibitively expensive on physical systems, such as legged robots, that can be damaged through extensive trial-and-error learning. In this paper, we propose a sample-efficient deep RL algorithm based on maximum entropy RL that requires minimal per-task tuning and only a modest number of trials to learn neural network policies. We apply this method to learning walking gaits on a real-world Minitaur robot. Our method can acquire a stable gait from scratch directly in the real world in about two hours, without relying on any model or simulation, and the resulting policy is robust to moderate variations in the environment. We further show that our algorithm achieves state-of-the-art performance on simulated benchmarks with a single set of hyperparameters. Videos of training and the learned policy can be found on the project website[3].**

I. Introduction

Designing locomotion controllers for legged robots is a long-standing research challenge. Current state-of-the-art methods typically employ a pipelined approach, consisting of components such as state estimation, contact scheduling, trajectory optimization, foot placement planning, model-predictive control, and operational space control [1, 5, 13, 23]. Designing these components requires expertise and often an accurate dynamics model of the robot that can be difficult to acquire. In contrast, end-to-end deep reinforcement learning (deep RL) does not assume any prior knowledge of the gait or the robot's dynamics, and can in principle be applied to robotic systems without explicit system identification or manual engineering. If successfully applied, deep RL can automate the controller design, completely removing the need for system identification, and resulting in gaits that are directly optimized for a particular robot and environment. However, applying deep RL to learning gaits in the real world is challenging, since current algorithms often require a large number of samples—on the order of tens of thousands of trials [44]. Moreover, such algorithms are often highly sensitive to hyperparameter settings and require considerable tuning [21], further increasing the overall sample complexity. For this reason, many prior methods have studied learning of locomotion gaits in simulation [4, 20, 34, 50], requiring accurate system identification and modeling.

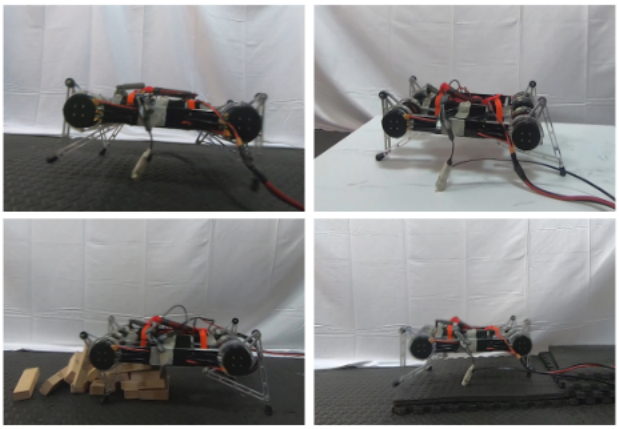


Fig. 1: Illustration of a walking gait learned in the real world. The policy is trained only on a flat terrain, but the learned gait is robust and can handle obstacles that were not seen during training.

In this paper, we aim to address these challenges by developing a deep RL algorithm that is both sample efficient and robust to the choice of hyperparameters, thus allowing us to learn locomotion gaits directly in the real world, without prior modeling. In particular, we extend the framework of maximum entropy RL. Methods of this type, such as soft actor-critic [17] and soft Q-learning [15], can achieve state-of-the-art sample efficiency [17] and have been successfully deployed in real-world manipulation tasks [16, 31], where they exhibit a high degree of robustness due to entropy maximization [16]. However, maximum entropy RL algorithms are sensitive to the choice of the temperature parameter, which determines the trade-off between exploration (maximizing the entropy) and exploitation (maximizing the reward). In practice, this temperature is considered as a hyperparameter that must be tuned manually for each task.

We propose an extension to the soft actor-critic algorithm [17] that removes the need for manually tuning of the temperature parameter. Our method employs gradient-based optimization of the temperature towards the targeted expected entropy over the visited states. In contrast to standard RL, our method controls only the expected entropy over the states, while the per-state entropy can still vary—a desirable property that allows the policy to automatically reduce entropy for states where acting deterministically is preferred, while still acting stochastically in other states. Consequently, our approach

[3]https://sites.google.com/view/minitaur-locomotion/

virtually eliminates the need for per-task hyperparameter tuning, making it practical for us to apply this algorithm to learn quadrupedal locomotion gaits directly on a real-world robotic system.

The principal contribution of our paper is an end-to-end RL framework for legged locomotion on physical robots, which includes a data efficient learning algorithm based on maximum entropy RL and an asynchronous learning system. We demonstrate the framework by training a Minitaur robot [26] (Figure 1) to walk. While we train the robot on flat terrain, the learned policy can generalize to unseen terrains and is moderately robust to perturbations. The training requires about 400 rollouts, equating to about two hours of real-world time. In addition to the robot experiments, we evaluate our algorithm on simulated benchmark tasks and show that it can achieve state-of-the-art performance and, unlike prior works based on maximum entropy RL, can use exactly the same hyperparameters for all tasks.

II. Related Work

Current state-of-the-art locomotion controllers typically adopt a pipelined control scheme. For example, the MIT Cheetah [5] uses a state machine over contact conditions, generates simple reference trajectories, performs model predictive control [9] to plan for desired contact forces, and then uses Jacobian transpose control to realize them. The ANYmal robot [23] plans footholds based on the inverted pendulum model [37], applies CMA-ES [19] to optimize a parameterized controller [12, 13], and solves a hierarchical operational space control problem [22] to produce joint torques, contact forces, and body motion. While these methods can provide effective gaits, they require considerable prior knowledge of the locomotion task and, more importantly, of the robot's dynamics. In contrast, our method aims to control the robot without prior knowledge of either the gait or the dynamics. We do not assume access to any trajectory design, foothold planner, or a dynamics model of the robot, since all learning is done entirely through real-world interaction. The only requirement is knowledge of the dimension and bounds of the state and action space, which in our implementation correspond to joint angles, IMU readings, and desired motor positions. While in practice, access to additional prior knowledge could be used to accelerate learning (see, e.g., [25]), end-to-end methods that make minimal prior assumptions are broadly applicable, and developing such techniques will make acquisition of gaits for diverse robots in diverse conditions scalable.

Deep RL has been used extensively to learn locomotion policies in simulation [4, 20, 34, 50] and even transfer them to real-world robots [24, 44], but this inevitably incurs a loss of performance due to discrepancies in the simulation, and requires accurate system identification. Using such algorithms directly in the real world has proven challenging. Real-world applications typically make use of simple and inherently stable robots [14] or low-dimensional gait parameterizations [8, 29, 36], or both [45]. In contrast, we show that we can acquire locomotion skills directly in the real world using neural-net policies.

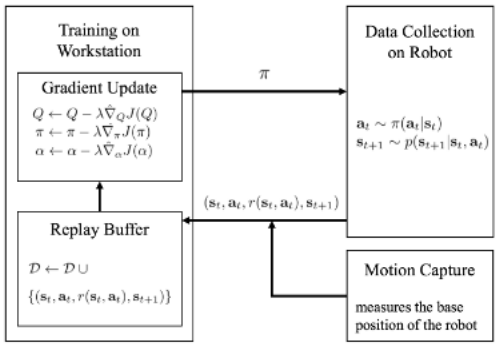


Fig. 2: Overview of our learning system. The learning system runs the training and data collection asynchronously across multiple machines.

Our algorithm is based on maximum entropy RL, which maximizes the weighted sum of the the expected return and the policy's expected entropy. This framework has been used in many contexts, from inverse RL [51] to optimal control [38, 48, 49]. One advantage of maximum entropy RL is that it produces relatively robust policies, since injection of structured noise during training causes the policy to explore the state space more broadly and improves the robustness of the policy [16]. However, the weight on the entropy term ("temperature") is typically chosen heuristically [15, 17, 32, 33, 40]. In our observation, this parameter is very sensitive and manual tuning can make real-world application of the maximum entropy framework difficult. Instead, we propose to constrain the expected entropy of the policy and adjust the temperature automatically to satisfy the constraint. Our formulation is an instance of constrained MDP, which has been studied recently in [3, 6, 47]. These works consider constraints that depend on the policy only via the sampling distribution, whereas in our case the constraint depends the policy explicitly. Our approach is also closely related to KL-divergence constraints that limit the policy change between iterations [2, 35, 39] but is applied directly to the current policy's entropy. We find that this simple modification drastically reduces the effort of parameter tuning on both simulated benchmarks and our robotic locomotion task.

III. Asynchronous Learning System

In this section, we will first describe our asynchronous robotic RL system, which we will use to evaluate real-world RL for robotic locomotion. The system, shown in Figure 2, consists of three components: a data collection job that collects robot experience, a motion capture job that computes the reward signal based on robot's position measured by a motion capture system, and a training job that updates the neural networks. These subsystems run asynchronously on different machines. When the learning system starts, the subsystems are synchronized to a common clock and use timestamps to sync the future data streams.

The data collection job runs on an on-board computer and executes the latest policy π produced by the training job. For each control step t, it collects observations $\mathbf{s}_t$, performs

neural network policy inference, and executes an action $\mathbf{a}_t$. The entire observed trajectory, or rollout, is recorded into tuples $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})_{t=0,\dots,N-1}$ and sent to the training job. The motion capture system measures the position of the robot and provides the reward signal $r(\mathbf{s}_t, \mathbf{a}_t)$. It periodically pulls data from the robot and the motion capture system, evaluates the reward function, and appends it to a replay buffer. The training subsystem runs on a workstation. At each iteration of training, the training job randomly samples a batch of data from this buffer and uses stochastic gradient descent to update the value network, the policy network, and the temperature parameter, as we will discuss in Section V. Once training is started, minimal human intervention is needed, except for the need to reset the robot if it falls or runs out of free space.

The asynchronous design allows us to pause or restart any subsystem without affecting the other subsystems. In practice, we found this particularly useful because we often encounter hardware and communication errors, in which case we can safely restart any of the subsystems without impacting the entire learning process. In addition, our system can be easily scaled to multiple robots by simply increasing the number of data collection jobs. In the following sections, we describe our proposed reinforcement learning method in detail.

IV. Reinforcement Learning Preliminaries

Reinforcement learning aims to learn a policy that maximizes the expected sum of rewards [43]. We consider Markov decision processes where the state space $\mathcal{S}$ and action space $\mathcal{A}$ are continuous. An agent starts at an initial state $\mathbf{s}_0 \sim p(\mathbf{s}_0)$, samples an action $\mathbf{a}_t$ from a policy $\pi(\cdot|\mathbf{s}_t) \in \Pi$, receives a bounded reward $r(\mathbf{s}_t, \mathbf{a}_t)$, and transitions to a new state $\mathbf{s}_{t+1}$ according to the dynamics $p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$. This generates a trajectory of states and actions $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots)$. We denote the trajectory distribution induced by π by $\rho_\pi(\tau) = p(\mathbf{s}_0)\prod_t \pi_t(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, and we overload the notation and use $\rho_\pi(\mathbf{s}_t, \mathbf{a}_t)$ and $\rho_\pi(\mathbf{s}_t)$ to denote the corresponding state-action and state marginals, respectively.

Maximum entropy RL optimizes both the expected return and the entropy of the policy. For finite-horizon MDPs, the corresponding objective can be expressed as

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{\tau\sim\rho_\pi}\left[r(\mathbf{s}_t, \mathbf{a}_t) - \alpha_t \log \pi_t(\mathbf{a}_t|\mathbf{s}_t)\right], \qquad (1)$$

which incentivizes the policy to explore more widely improving its robustness against perturbations [16]. The temperature parameter α determines the relative importance of the entropy term against the reward, and thus controls the stochasticity of the optimal policy. The maximum entropy objective differs from the standard maximum expected reward objective used in conventional reinforcement learning, though the conventional objective can be recovered in the limit as $\alpha \to 0$. In the finite horizon case, the policy is time dependent, and we write π and α to denote the set of all policies $(\pi_0, \pi_1, ..., \pi_T)$ or temperatures $(\alpha_0, \alpha_1, ..., \alpha_T)$. We can extend the objective to infinite horizon problems by introducing a discount factor γ to ensure that the sum of expected rewards and entropies is finite [15], in which case we overload the notation and denote a stationary policy and temperature as π and α.

One of the central challenges with the objective in (1) is that the trade-off between maximizing the return, or exploitation, versus the entropy, or exploration, is directly affected by the scale of the reward function[1]. Unlike in conventional RL, where the optimal policy is independent of scaling of the reward function, in maximum entropy RL the scaling factor has to be tuned per environment, and a sub-optimal scale can drastically degrade the performance [17].

V. Automating Entropy Adjustment for Maximum Entropy RL

Learning robotic tasks in the real world requires an algorithm that is sample efficient, robust, and insensitive to the choice of the hyperparameters. Maximum entropy RL is both sample efficient and robust, making it a good candidate for real-world robot learning [16]. However, one of the major challenges of maximum entropy RL is its sensitivity to the temperature parameter, which typically needs to be tuned for each task separately. In this section, we propose an algorithm that enables automated temperature adjustment at training time, substantially reducing the effort of hyperparameter tuning and making deep RL a viable solution for real-world robotic problems.

A. Entropy Constrained Objective

The magnitude of the reward differs not only across tasks, but it also depends on the policy, which improves over time during training. Since the optimal entropy depends on this magnitude, choosing the ideal temperature is particularly difficult: the entropy can vary unpredictably both across tasks and during training as the policy becomes better. Instead of requiring the user to set the temperature manually, we can automate this process by formulating a modified RL objective, where the entropy is treated as a constraint. Simply forcing the entropy to a fixed value is a poor solution, since the policy should be free to explore more in regions where the optimal action is uncertain, but remain more deterministic in states with a clear distinction between good and bad actions. Therefore, we constrain the *expected* entropy of the policy, while the entropy at different states can still vary. We show that the Lagrangian relaxation of this problem leads to the maximum entropy objective with respect to the policy, where the dual variable takes the role of the temperature.

In particular, our aim is to find a stochastic policy with maximal expected return that satisfies a minimum expected entropy constraint. Formally, we want to solve the constrained optimization problem

$$\max_{\pi\in\Pi} \mathbb{E}_{\tau\sim\rho_\pi}\left[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right]$$
$$\text{s.t. } \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[-\log\left(\pi_t(\mathbf{a}_t|\mathbf{s}_t)\right)\right] \geq \mathcal{H}, \ \forall t, \qquad (2)$$

[1]Reward scale is the reciprocal of temperature. We will use these two terms interchangeably throughout this paper.

where $\mathcal{H}$ is the desired minimum expected entropy. Note that, for fully observed MDPs, the policy that optimizes the expected return is deterministic, so we expect this constraint to usually be tight and do not need to impose an upper bound on the entropy.

We start by writing out the Lagrangian relaxation of (2), as typical in the prior works [3, 6, 46]:

$$\mathcal{L}(\pi, \alpha) = \mathbb{E}_{\tau\sim\rho_\pi}\left[\sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t) + \alpha_t \left(-\log\left(\pi_t(\mathbf{a}_t|\mathbf{s}_t)\right) - \mathcal{H}\right)\right]. \tag{3}$$

We optimize this objective using the dual gradient method. Note that for a fixed dual variable, the Lagrangian is exactly equal to the maximum entropy objective in (1) minus an additive constant ($\alpha_t\mathcal{H}$) per time step, and can thus be optimized with any off-the-shelf maximum entropy RL algorithm. Specifically, we resort to approximate dynamic programming, which turn out to correspond to the soft actor-critic algorithm [17]. We first define the soft Q-function and use it to bootstrap the algorithm. The optimal soft Q-function is defined as

$$Q_t^*(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}_{\mathbf{s}_{t+1}\sim\rho_\pi}\left[V_{t+1}^*(\mathbf{s}_{t+1})\right], \tag{4}$$

where

$$V_t^*(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t\sim\pi_t^*}\left[Q_t^*(\mathbf{s}_t, \mathbf{a}_t) - \alpha_t \log\left(\pi_t^*(\mathbf{a}_t|\mathbf{s}_t)\right)\right], \tag{5}$$

and π_t^* denotes the optimal policy for time t. We have omitted the dependency of the soft Q-function on the dual variable of the future time steps for brevity. We also abuse the notation slightly, and write Q_t^* to denote $Q_t^{\pi^*}$, which are equal only if Π is a set of all policies (and not for example the set of Gaussian policies). We initialize the iteration by setting $Q_T^*(\mathbf{s}_T, \mathbf{a}_T) = r(\mathbf{s}_T, \mathbf{a}_T)$. Assuming we have evaluated Q_t^* for some t, we can substitute it to the Lagrangian. We can now solve for the optimal policy at time t for all $\mathbf{s}_t \in \mathcal{S}$ by noting that the optimal policy at time t is independent of the policy at the previous time steps:

$$\begin{aligned}\pi_t^*(\,\cdot\,|\mathbf{s}_t) &\in \arg\max_{\pi_t\in\Pi} \mathbb{E}_{\mathbf{a}_t\sim\pi_t}\left[Q_t^*(\mathbf{s}_t, \mathbf{a}_t) - \alpha_t \log \pi_t(\mathbf{a}_t|\mathbf{s}_t)\right] \\ &= \arg\min_{\pi_t\in\Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi_t(\,\cdot\,|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(\frac{1}{\alpha_t} Q_t^*(\mathbf{s}_t, \,\cdot\,)\right)}{Z_t(\mathbf{s}_t)}\right).\end{aligned} \tag{6}$$

The partition function $Z_t(\mathbf{s}_t) = \int_{\mathcal{A}} \exp\left(\frac{1}{\alpha_t} Q_t^*(\mathbf{s}_t, \mathbf{a}_t)\right) d\mathbf{a}_t$ does not depend on π_t^*, so we can ignore it for optimizing π_t^*. This is exactly the soft policy improvement step introduced by [17], with an additional temperature parameter α_t. In contrast to [17], which shows that this update leads to an improvement in the infinite horizon case, we derived it starting from the finite horizon objective. By traversing backwards in time, we can optimize the Lagrangian with respect to the policy.

After solving for the policy for a fixed dual variable, we improve the dual in order to satisfy the entropy constraint. We can optimize the temperature by moving it in the direction of the negative gradient of (3):

$$\alpha_t \leftarrow \alpha_t + \lambda_\alpha \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\rho_{\pi^*}}\left[\log\left(\pi_t^*(\mathbf{a}_t|\mathbf{s}_t) + \mathcal{H}\right)\right], \tag{7}$$

where λ_α is the learning rate[2]. The equations (4), (6), and (7) constitute the core of our algorithm. However, solving these equations exactly is not practical for continuous state and actions, and in practice, we cannot compute the expectations, but instead have access to unbiased samples. Therefore, for a practical algorithm, we need to resort to function approximators and stochastic gradient descent as well as other standard tricks to stabilize training, as discussed in the next section.

B. Practical Algorithm

In practice, we parameterize a Gaussian policy with parameters ϕ, and learn them using stochastic gradient descent for the discounted, infinite horizon problem. We additionally use two parameterized Q-functions, with parameters θ_1 and θ_2, as suggested in [17]. We learn the Q-function parameters as a regression problem by minimizing the following loss $J_Q(\theta_i)$:

$$\mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\sim\mathcal{D}}\left[\left(Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t) - \left(r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V_{\theta_1,\theta_2}(\mathbf{s}_{t+1})\right)\right)^2\right] \tag{8}$$

using minibatches from a replay buffer $\mathcal{D}$. The value function $V_{\theta_1,\theta_2}(\mathbf{s}_t)$ is implicitly defined through the Q-functions and the policy as $\mathbb{E}_{\mathbf{a}_t\sim\pi_\phi}\left[\min_{i\in\{1,2\}} Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)\right]$. We learn a Gaussian policy by minimizing

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}, \mathbf{a}_t\sim\pi_\phi}\left[\alpha \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - \min_{i\in\{1,2\}} Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t)\right], \tag{9}$$

using the reparameterization trick [28]. This procedure is the same as the standard soft actor-critic algorithm [17], but with an explicit, dynamic temperature α.

To learn α, we need to minimize the dual objective, which can be done by approximating dual gradient descent. Instead of optimizing with respect to the primal variables to convergence, we use a truncated version that performs incomplete optimization and alternates between taking a single gradient step on each objective. While convergence to the global optimum is not guaranteed, we found this approach to work well in practice. Thus, we compute gradients for α with the following objective:

$$J(\alpha) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}, \mathbf{a}_t\sim\pi_\phi}\left[-\alpha \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - \alpha\mathcal{H}\right]. \tag{10}$$

The proposed algorithm alternates between a data collection phase and an optimization phase. In the optimization phase, the algorithm optimizes all objectives in (8) – (10) jointly. We also incorporate delayed target Q-function networks as is standard in prior work. Algorithm 1 summarizes the full algorithm, where $\hat{\nabla}$ denotes stochastic gradients.

[2]We also need to make sure α_t remains non-negative. In practice, we thus parameterize $\alpha_t = \exp(\beta_t)$ and optimize β_t instead.

Algorithm 1: Soft Actor-Critic with Automatic Entropy Adjustment

Initialize function approximators parameters θ_1 θ_2, ϕ, and a global temperature coefficient α.
for *each iteration* **do**
  for *each environment step* **do**
    $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$.
  end
  for *each gradient step* **do**
    $\theta_i \leftarrow \theta_i - \lambda\hat{\nabla}_\theta J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda\hat{\nabla}_\phi J_\pi(\phi)$
    $\alpha \leftarrow \alpha - \lambda\hat{\nabla}_\alpha J(\alpha)$
  end
  $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$
end

VI. Evaluation on Simulation Environments

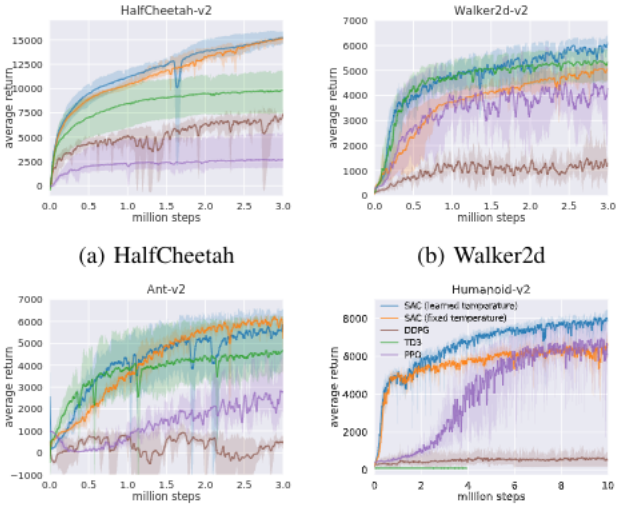
Before evaluating on real-world locomotion, we conduct a comprehensive evaluation in simulation to validate our algorithm. Our goal is to answer following four questions:

1) Does our method achieve the state-of-the-art data efficiency?
2) How sensitive is our method to the hyperparameter?
3) Is our method effectively regulating the entropy and dynamically adjusting the temperature during learning?
4) Can the learned policy generalize to unseen situations?

A. Evaluation on OpenAI Benchmark Environments

We first evaluate our algorithm on four standard benchmark environments for continuous locomotion tasks in OpenAI Gym benchmark suite [7]. We compare our method to soft actor-critic (SAC) [17] with a fixed temperature parameter that is tuned for each environment. We also compare to deep deterministic policy gradient (DDPG) [30], proximal policy optimization (PPO) [41], and twin delayed deep deterministic policy gradient algorithm (TD3) [11]. All of the algorithms use the same network architecture: all of the function approximators (policy and Q-functions for SAC) are parameterized with a two-layer neural network with 256 hidden units on each layer, and we use ADAM [27] with the same learning rate of 0.0003 to train all the networks and temperature parameter α. For standard SAC, we tune the reward scale per environment using grid search. Poorly chosen reward scales can degrade performance drastically (see Figure 4a). For our method, we simply set the target entropy to be -1 per action dimension (i.e., HalfCheetah has target entropy -6, while Humanoid uses -17).

1) Comparative Evaluation: Figure 3 shows a comparison of the algorithms. The solid line denotes the average performance over five random seeds, and the shaded region corresponds to the best and worst performing seeds. The results indicate that our method (blue) achieves practically identical or better performance compared to standard SAC (orange), which is tuned per environment for all environments. Overall, our method performs better or comparably to the other baselines, standard SAC, DDPG, TD3, and PPO.



(a) HalfCheetah (b) Walker2d

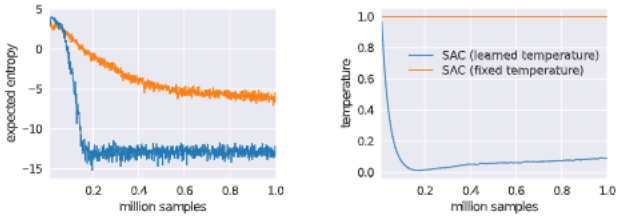
(c) Ant (d) Humanoid

Fig. 3: (a) – (d) Standard benchmark training results. Our method (blue) achieves similar or better performance compared to other algorithms. Note that all other algorithms except ours went through dense hyperparameter tuning to achieve the above learning curves.

(a) Standard SAC over reward scale (b) Our method over target entropy

Fig. 4: Average normalized performance over the last 100k samples on a range of environments. (a) Performance of standard SAC as a function of reward scale, and (b) Performance of our method as a function of target entropy. Our method is substantially less sensitive to the choice of the hyperparameter.

2) Sensitivity Analysis: We compare the sensitivity to the hyperparameter between our method (target entropy) and the standard SAC (reward scale). Both maximum entropy RL algorithms [17] and standard RL algorithms [21] can be very sensitive to the scale of the reward function. In the case of maximum entropy RL, this scale directly affects the trade-off between reward maximization and entropy maximization [17]. We first validate the sensitivity of standard SAC by running experiments on the HalfCheetah, Walker, Ant, and the simulated Minitaur robot (See Section VI-B for more details). Figure 4a shows the returns for a range of reward scale values that are normalized to the maximum reward of the given task. All benchmark environments achieve good performance for about the same range of values, between 1 to 10. On the other hand, the simulated Minitaur requires roughly two orders of magnitude larger reward scale to work properly. This result



(a) Entropy (b) Temperature

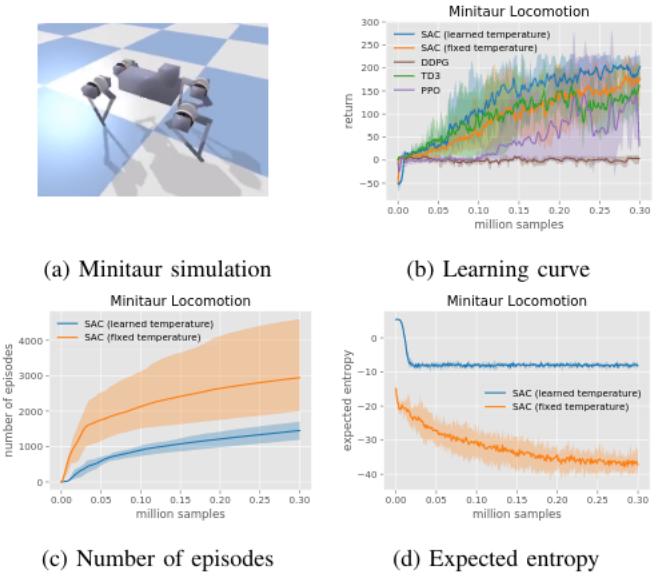
Fig. 5: Comparison of our method and standard SAC in terms of entropy and temperature on HalfCheetah. The target entropy for learning the temperature of SAC is -13 in this case.

indicates that, while standard benchmarks offer high variability in terms of task dimensionality, they are homogeneous in terms of other characteristics, and testing only on the benchmarks might not generalize well to seemingly similar tasks designed for different purposes. This suggests that the good performance of our method, with the same hyperparameters, on both the benchmark tasks and the Minitaur task accurately reflects its generality and robustness. Figure 4b compares the sensitivity of our method to the target entropy on the same tasks. In this case, the range of good target entropy values is essentially the same for all environments, making hyperparameter tuning substantially less laborious. It is also worth noting that this large range indicates that our algorithm is relatively insensitive to the choice of this hyperparameter.

3) Validation of Entropy Control: Next, we compared how the entropy and temperature evolve during training. Figure 5a compares the entropy (estimated as an expected negative log probability over a minibatch) on HalfCheetah for SAC with fixed temperature (orange) and our method (blue), which uses a target entropy of -13. The figure clearly indicates that our algorithm is able to match the target entropy in a relatively small number of steps. On the other hand, regular SAC has a fixed temperature parameter and thus the entropy slowly decreases as the Q-function increases. Figure 5b compares the temperature parameter of the two methods. Our method (blue) actively adjusts the temperature, particularly in the beginning of training when the Q-values are small and the entropy term dominates in the objective. The temperature is quickly pulled down so as to make the entropy to match the target. For other simulated environments, we observed similar entropy and temperature curves throughout the learning.

B. Evaluation on Simulated Minitaur Environment

Next, we evaluate our method on a simulated Minitaur locomotion task (Figure 6). Simulation allows us to quantify perturbation robustness, measure states that are not accessible on the robot, and more importantly, gather more data to evaluate our algorithm. To prevent bias of our conclusion, we have also conducted a careful system identification, following Tan et al. [44], such that our simulated robot moderately represents the real system. However, we emphasize that we do not transfer any simulated policy to the real world —all real-world experiments



(a) Minitaur simulation (b) Learning curve

(c) Number of episodes (d) Expected entropy

Fig. 6: (a) Illustration of the Minitaur environment. (b) Learning curves. For our method (blue), we used exactly the same hyperparameters as we used for the benchmarks whereas for the baseline (orange), we needed to tune the reward scale. (c) Number of episodes during training. (d) Expected entropy during training.

use only real-world training, without access to any simulator.

Figure 6b compares the learning curve of our method to the state-of-the-art deep reinforcement learning algorithms. Our method is the most data efficient. Note that in order to obtain the result of SAC (fixed temperature) in the plot, we had to sweep though a set of candidate temperatures and choose the best one. This mandatory hyperparameter tuning is equivalent to collecting an order of magnitude more samples, which is not shown in Figure 6b. While the *number of steps* is a common measure of data efficiency in the learning community, the *number of episodes* can be another important indicator for robotics because the number of episodes determines the number of experiment reset, which typically is time-consuming and require human intervention. Figure 6c indicates that our method takes fewer numbers of episodes for training a good policy. In the experiments, our algorithm effectively escapes a local minimum of "diving forward," which is a common cause of falling and early episode termination, by maintaining the policy entropy at higher values (Figure 6d).

The final learned policy in simulation qualitatively resembles the gait learned directly on the robots. We tested its robustness by applying lateral perturbations to its base for 0.5 seconds with various magnitudes. Even though no perturbation is injected during training, the simulated robot can withstand up to 220N lateral pushes and subsequently recover to normal walking. This is significantly larger than the maximum 130N of the best PPO-trained policy that is picked out of 1000 learning trials. We suspect that this robustness emerges automatically from the SAC method due to entropy maximization at training time.

VII. Learning in the Real World

In this section, we describe the real-world learning experiments on the Minitaur robot. We aim to answer the following

questions:

1) Can our method efficiently train a policy on hardware without hyperparameter tuning?
2) Can the learned policy generalize to unseen situations?

A. Experiment Setup

Quadrupedal locomotion presents substantial challenges for real-world reinforcement learning. The robot is underactuated, and must therefore delicately balance contact forces on the legs to make forward progress. A suboptimal policy can cause it to lose balance and fall, which will quickly damage the hardware, making sample-efficient learning essential. In this section, we test our learning method and system on a quadrupedal robot in the real world settings. We use the Minitaur robot, a small-scale quadruped with eight direct-drive actuators [26]. Each leg is controlled by two actuators that allow it to move in the sagittal plane. The Minitaur is equipped with motor encoders that measure the motor angles and an IMU that measures the orientation and angular velocity of Minitaur's base.

In our MDP formulation, the observation includes eight motor angles, roll and pitch angles, and their angular velocities. We choose to exclude the yaw measurement because it drifts quickly. The action space includes the swing angle and the extension of each leg, which are then mapped to desired motor positions and tracked with a PD controller [44]. For safety, we choose low PD gains $k_p = 0.3$ and $k_d = 0.003$ to ensure compliant motion. We find that the latencies in the hardware and the partial observation make the system non-Markovian, which significantly degrades the learning performance. We therefore augment an observation space to include a history of the last five observations and actions which results in a 112 dimensional observation space. The reward is defined as:

$$r(\mathbf{s}_t, \mathbf{a}_t) = w_1(x_t - x_{t-1}) - w_2|\ddot{\mathbf{a}}_t| - w_3|\phi| - w_4 \sum_{i\in\{1,2\}} \max(\bar{q} - q_i, 0).$$

The function encourages longer walking distance $(x_t - x_{t-1})$, which is measured using the motion capture system, and penalizes large joint accelerations ($\ddot{\mathbf{a}}_t$), computed via finite differences using the last three actions. We also find it necessary to penalize a large roll angle of the base (ϕ) and the joint angles when the front legs (q_1, q_2) are folded under the robot, which are the common failure cases. The weights are set to $1.0, 0.05, 0.5, 1.0$ respectively and the maximum angle threshold q is set to -0.3 radians.

We parameterize the policy and the value functions with fully connected feed-forward neural networks with two hidden-layers and 256 neurons per layer, which are randomly initialized. For preventing too jerky motions at the early stage, we smoothed out actions for the first 50 episodes.

B. Results

Our method successfully learns to walk from 160k control steps, or approximately 400 rollouts. Each rollout has the maximum length of 500 steps (equivalent to 10 seconds) and can terminate early if the robot falls. The whole training process takes about two hours. Figure 7 shows the learning curve, The performance is slightly less than the simulation, potentially due to fewer collected samples. Please refer to the supplemental video to see the learning process, the final policy, and more evaluations on different terrains.

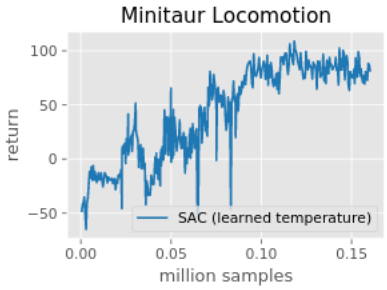
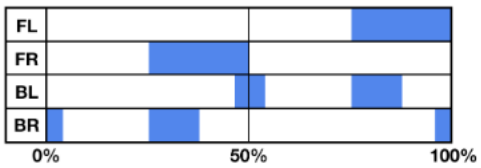
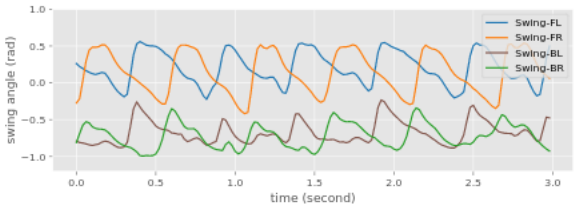


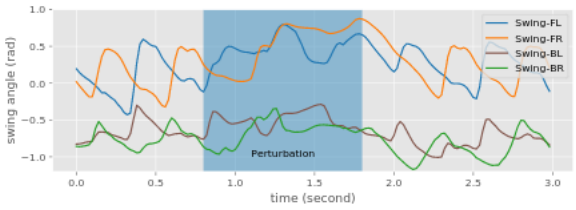
Fig. 7: Learning curve of SAC with learned temperature on the Minitaur robot.



(a) Footfall pattern of learned gait



(b) Swing angles of learned gait



(c) Swing angles of learned gait with perturbations

Fig. 8: Illustration of the learned policy. (a) Footfall pattern of a single cycle of the learned gait. Swing phases are drawn as blue bars. (b) Swing angles of four legs, which are periodic and synchronized. (c) Swing angles with an external perturbation. The learned policy successfully recovers to the nominal gait.

The trained robot is able to walk forward at a speed of 0.32m/s (∼0.8 body length per second). The learned gait swings the front legs once per cycle, while pushing against the ground multiple times with the rear legs (Figure 8a and b). Note that the learned gait is periodic and synchronized, though no explicit trajectory generator, symmetry constraint, or periodicity constraint is encoded into the system. Comparing to the default controller (trotting gait) provided by the manufacturer that

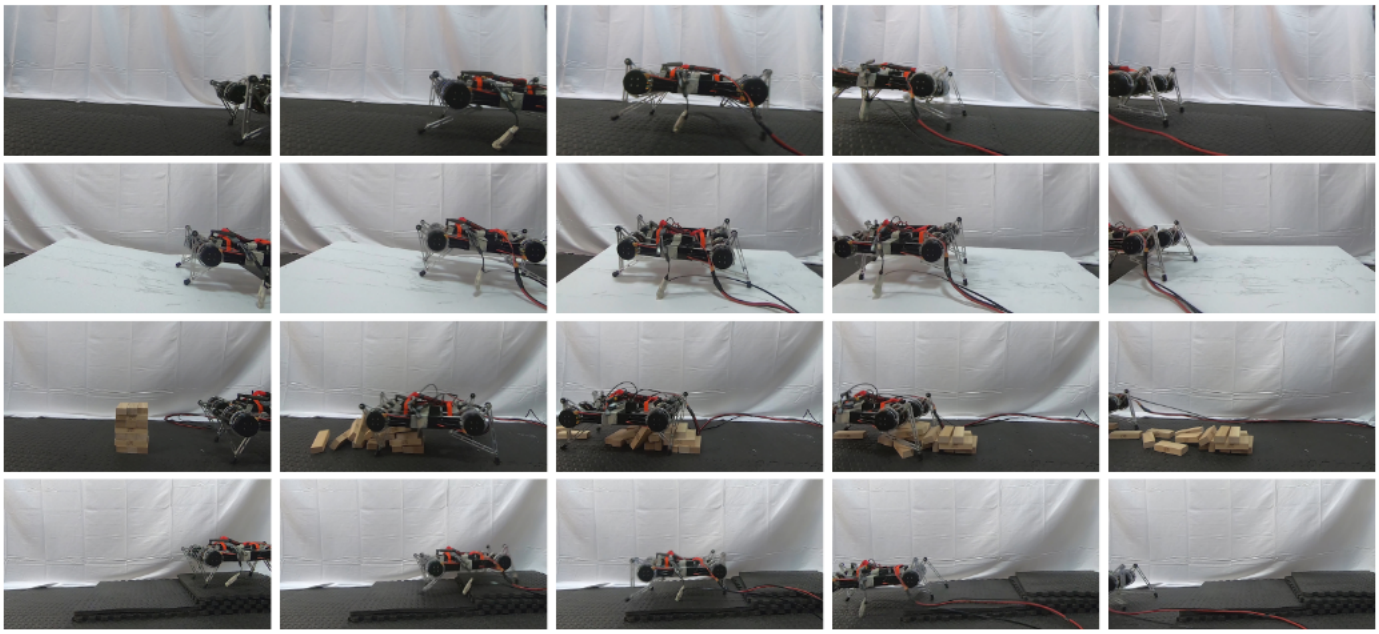


Fig. 9: We trained the Minitaur robot to walk on flat terrain (first row) in about two hours. At test time, we introduced obstacles, including a slope, wooden blocks, and steps, which were not present at training time, and the learned policy was able to generalize to the unseen situations without difficulty (other rows).

walks at a similar speed, the learned gait has similar frequencies ($\sim$2Hz) and swing amplitudes ($\sim$0.7 Rad), but has substantially different joint angle trajectories and foot placement. The learned gait has a much wider stance and a lower standing height. We evaluated the robustness of the trained policy against external perturbations by pushing the base of the robot backward (Figure 8c) for approximately one second, or side for around half second. Although the policy has never been trained with such perturbations, it successfully recovered and returned to a periodic gait for all 10 repeated tests.

In the real world, the utility of a locomotion policy hinges critically on its ability to generalize to different terrains and obstacles. Although we trained our policy only on flat terrain (Figure 9, first row), we tested it on varied terrains and obstacles (other rows). Because the SAC method learns robust policies due to entropy maximization at training time, the policy can readily generalize to these perturbations without any additional tweaking. The robot is able to walk up and down a slope (second row), ram through an obstacle made of wooden blocks (third row), and step down stairs (fourth row) without difficulty, despite not being trained in these settings. We repeated these tests for 10 times, and the robot succeeds on all cases.

VIII. Conclusion

We presented a complete end-to-end learning system for locomotion with legged robots. The core algorithm, which is based on a dual formulation of an entropy-constrained reinforcement learning objective, can automatically adjust the temperature hyperparameter during training, resulting in a sample-efficient and stable algorithm with respect to hyperparameter settings. It enables end-to-end learning of quadrupedal locomotion controllers from scratch on a real-world robot. In our experiments, a walking gait emerged automatically in two hours of training without the need of prior knowledge about the locomotion tasks or the robot's dynamic model. A further discussion of this method and results on other platforms can be found in an extended technical report [18]. Compared to sim-to-real approaches [44] that require careful system identification, learning directly on hardware can be more practical for systems where acquiring an accurate model is hard and expensive, such as for walking on diverse terrains or manipulation of deformable objects.

To the best of our knowledge, our experiment is the first example of an application of deep reinforcement learning to quadrupedal locomotion directly on a robot in the real world without any pretraining in the simulation, and it is the first step towards a new paradigm of fully autonomous real-world training of robot policies. Two of the most critical remaining challenges of our current system are the heavy dependency on manual resets between episodes and the lack of a safety layer that would enable learning on bigger robots, such as ANYmal [24] or HyQ[42]. In the future work, we plan to address these issues by developing a framework for learning policies that are safety aware and can be trained to automatically reset themselves [10, 24].

Acknowledgments

The authors gratefully thank Ken Caluwaerts, Tingnan Zhang, Julian Ibarz, Vincent Vanhoucke, and the anonymous reviewers for valuable discussion and suggestions.

References

[1] Taylor A., Patrick C., Kevin G., Alan F., and Jonathan W. H. Fast online trajectory optimization for the bipedal robot cassie. In *Robotics: Science and Systems (RSS)*, 2018.

[2] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

[3] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *International Conference on Machine Learning (ICML)*, 2017.

[4] G. Berseth, C. Xie, P. Cernek, and M. Van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. *International Conference on Learning Representations (ICLR)*, 2018.

[5] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim. MIT cheetah 3: Design and control of a robust, dynamic quadruped robot. In *International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[6] S. Bohez, A. Abdolmaleki, M. Neunert, J. Buchli, N. Heess, and R. Hadsell. Value constrained model-free continuous control. *arXiv preprint arXiv:1902.04623*, 2019.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[8] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2): 5–23, 2016.

[9] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[10] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[11] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, 2018.

[12] C. Gehring, S. Coros, M. Hutter, M. Bloesch, P. Fankhauser, M. A. Hoepflinger, and R. Siegwart. Towards automatic discovery of agile gaits for quadrupedal robots. In *International Conference on Robotics and Automation (ICRA)*, 2014.

[13] C. Gehring, M. Coros, S. Hutler, C. D. Bellicoso, H. Heijnen, R. Diethelm, M. Bloesch, P. Fankhauser, J. Hwangbo, M. Hoepflinger, et al. Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot. *IEEE Robotics & Automation Magazine*, 23(1):34–43, 2016.

[14] S. Ha, J. Kim, and K. Yamane. Automated deep reinforcement learning environment for hardware of a modular legged robot. In *International Conference on Ubiquitous Robots (UR)*. IEEE, 2018.

[15] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*, 2017.

[16] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine. Composable deep reinforcement learning for robotic manipulation. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018.

[18] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[19] N. Hansen. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[20] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[21] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Conference on Artificial Intelligence (AAAI)*. AAAI, 2018.

[22] M. Hutter, H. Sommer, C. Gehring, M. Hoepflinger, M. Bloesch, and R. Siegwart. Quadrupedal locomotion using hierarchical operational space control. *The International Journal of Robotics Research*, 33(8):1047–1062, 2014.

[23] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 38–44. IEEE, 2016.

[24] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019. doi: 10.1126/scirobotics.aau5872.

[25] A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani, and V. Vanhoucke. Policies modulating trajectory generators. In *Conference on Robot Learning (CoRL)*. PMLR.

[26] Gavin D Kenneally, Avik De, and Daniel E Koditschek. Design principles for a family of direct-drive legged robots. *IEEE Robotics and Automation Letters*, 1(2): 900–907, 2016.

[27] D. Kingma and J. Ba. Adam: A method for stochastic

optimization. In *International Conference for Learning Presentations (ICLR)*, 2015.

[28] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[29] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2004.

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

[31] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra. Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on Robot Learning (CoRL)*, 2018.

[32] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[33] Brendan O'Donoghue, Rémi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Pgq: Combining policy gradient and q. *arXiv preprint arXiv:1611.01626*, 2016.

[34] X. B. Peng, G. Berseth, and M. Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35 (4):81, 2016.

[35] J. Peters and S. Schaal. Policy gradient methods for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2006.

[36] A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. G. Atkeson. Bayesian optimization using domain knowledge on the ATRIAS biped. *arXiv preprint arXiv:1709.06047*, 2017.

[37] M. H. Raibert. *Legged robots that balance*. MIT press, 1986.

[38] K. Rawlik, M. Toussaint, and S. Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Robotics: Science and Systems (RSS)*, 2012.

[39] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[40] J. Schulman, P. Abbeel, and X. Chen. Equivalence between policy gradients and soft Q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[42] C. Semini, N. G Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell. Design of hyq–a hydraulically and electrically actuated quadruped robot. *Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 225(6):831–849, 2011.

[43] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[44] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems (RSS)*, 2018.

[45] R. Tedrake, T. W. Zhang, and H. S. Seung. Learning to walk in 20 minutes. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, volume 95585, pages 1939–1412. Yale University New Haven (CT), 2005.

[46] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 3, page 6, 2017.

[47] C. Tessler, D. J. Mankowitz, and S. Mannor. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*, 2018.

[48] E. Todorov. General duality between optimal control and estimation. In *Conference on Decision and Control (CDC)*, pages 4286–4292. IEEE, 2008.

[49] M. Toussaint. Robot trajectory optimization using approximate inference. In *International Conference on Machine Learning (ICML)*, pages 1049–1056. ACM, 2009.

[50] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne. Feedback control for cassie with deep reinforcement learning. *arXiv preprint arXiv:1803.05580*, 2018.

[51] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1433–1438, 2008.