Run, skeleton, run: skeletal model in a physics-based simulation

Mikhail Pavlov[⋆] **Sergey Kolesnikov**[⋆] **Sergey M. Plis**[⋆]
[⋆]Reason8

Abstract

In this paper, we present our approach to solve a physics-based reinforcement learning challenge "Learning to Run" with objective to train physiologically-based human model to navigate a complex obstacle course as quickly as possible. The environment is computationally expensive, has a high-dimensional continuous action space and is stochastic. We benchmark state of the art policy-gradient methods and test several improvements, such as layer normalization, parameter noise, action and state reflecting, to stabilize training and improve its sample-efficiency. We found that the Deep Deterministic Policy Gradient method is the most efficient method for this environment and the improvements we have introduced help to stabilize training. Learned models are able to generalize to new physical scenarios, e.g. different obstacle courses.

1 Introduction

Reinforcement Learning (RL) (Sutton and Barto 1998) is a significant subfield of Machine Learning and Artificial Intelligence along the supervised and unsupervised subfields with numerous applications ranging from trading to robotics and medicine. It already achieved high levels of performance on Atari games (Mnih et al. 2015), board games (Silver et al. 2016) and 3D navigation tasks (Mnih et al. 2016; Jaderberg et al. 2016).

All of above tasks have one feature in common - there is always some well-defined reward function, for example, game score, which can be optimized to produce the required behaviour. Nevertheless, there are are many other tasks and environments, for which it is still unclear what is the "correct" reward function to optimize. And it is even a harder problem, when we talk about continuous control tasks, such as physics-based environments (Todorov, Erez, and Tassa 2012) and robotics (Gu et al. 2017).

Yet, recently a substantial interest is directed to research employing physics-based based environment. These environments are significantly more interesting, challenging and realistic than the well defined games; at the same time they are still simpler than real conditions with physical agents, while being cheap and more accessible. One of the interesting researches is the work of Schulman et al. where a simulated robot learned to run and get up off the ground (Schulman et al. 2015b). Another paper is by Heess et al. where the authors trained several simulated bodies on a diverse set of challenging terrains and obstacles, using a simple reward function based on forward progress (Heess et al. 2017).

To solve the problem of continuous control in simulation environments it has become generally accepted to adapt reward signal for specific environment. Still it can lead to unexpected results when the reward function is modified even slightly, and for more advanced behaviors the appropriate reward function is often non-obvious. To address this problem, the community came up with several environment-independent approaches such as unsupervised auxiliary tasks (Jaderberg et al. 2016) and unsupervised exploration rewards (Pathak et al. 2017). All these suggestions are trying to solve the main challenge of reinforcement learning: how an agent can learn for itself, directly from a limited reward signal, to achieve best performance.

Besides the difficult to define reward function, physically realistic environments usually have a lot of stochasticity, computationally very expensive, and have high-dimensional action spaces. To support learning in such settings it is necessary to have a reliable, scalable and sample-efficient reinforcement learning algorithm. In this paper we evaluate several existing approaches and then improve upon the best performing approach for a physical simulator setting. We present the approach that we have used to solve the "Learning to run" – NIPS 2017 competition challenge[1] with an objective to learn to control a physiologically-based human model and make it run as quickly as possible. The model that we present here has won the third place at the challenge: `https://www.crowdai.org/challenges/nips-2017-learning-to-run/leaderboards`.

This paper proceeds as follows: first we review the basics of reinforcement learning, then we describe environment used in challenge and models used in our experiment, after that we present results of our experiments and finally we discuss the results and conclude the work.

[1] `https://www.crowdai.org/challenges/nips-2017-learning-to-run`

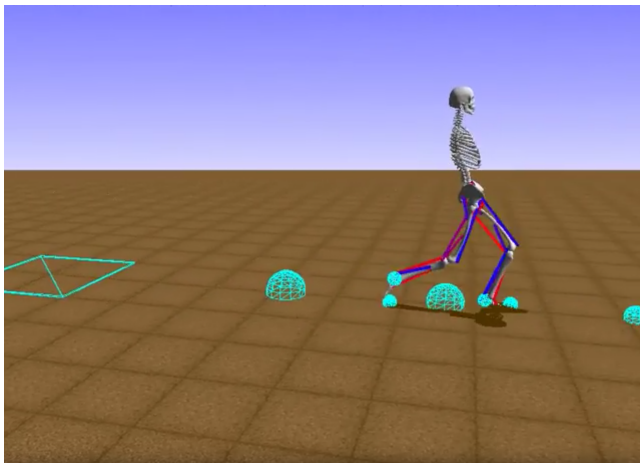


Figure 1: OpenSim screenshot that demonstrates the agent.

2 Background

We approach the problem in a basic RL setup of an agent interacting with an environment. The "Learning to run" environment is fully observable and thus can be modeled as a Markov Decision Process (MDP) (Bellman 1957). MDP is defined as a set of states ($\mathcal{S} : \{s_i\}$), a set of actions ($\mathcal{A} : \{a_i\}$), a distribution over initial states $p(s_0)$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, transition probabilities $p(s_{t+1}|s_t, a_t)$, time horizon T, and a discount factor $\gamma \in [0, 1)$. A policy parametrized by θ is denoted with π_θ. The policy can be either deterministic, or stochastic. The agent's goal is to maximize the expected discounted return $\eta(\pi_\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T} \gamma_t r(s_t, a_t)]$, where $\tau = (s_0, a_0, \ldots, s_T)$ denotes a trajectory with $s_0 \sim p(s_0)$, $a_t \sim \pi_\theta(a_t|s_t)$, and $s_t \sim p(s_t|s_{t-1}, a_{t-1})$.

3 Environment

The environment is a musculoskeletal model that includes body segments for each leg, a pelvis segment, and a single segment to represent the upper half of the body (trunk, head, arms). See Figure 1 for a clarifying screenshot. The segments are connected with joints (e.g., knee and hip) and the motion of these joints is controlled by the excitation of muscles. The muscles in the model have complex paths (e.g., muscles can cross more than one joint and there are redundant muscles). The muscle actuators themselves are also highly nonlinear.

The purpose is to navigate a complex obstacle course as quickly as possible. The agent operates in 2D world. The obstacles are balls randomly located along the agent's way. Simulation is done using OpenSim (Delp et al. 2007) library which relies on the Simbody (Sherman, Seth, and Delp 2011) physics engine. Environment is described in Table 1. More detailed description of environment can be found on competition github page.[2]

Due to a complex physics engine the environment is quite slow compared to standard locomotion environments (Todorov, Erez, and Tassa 2012; OpenAI Roboschool 2017). Some steps in environment could take seconds. Yet, the other environments can be as fast as three orders of magnitudes faster.[3] So it is crucial to train agent using the most sample-efficient method.

Table 1: Description of the OpenSim environment.

parameters	description
state (a_t)	$\mathbb{R}^{41}$, coordinates and velocities of various body parts and obstacle locations. All (x, y) coordinates are absolute. To improve generalization of our controller and use data more efficiently, we modified the original version of environment making all x coordinates relative to the x coordinate of pelvis.
action (a_t)	$\mathbb{R}^{18}$, muscles activations, 9 per leg, each in $[0, 1]$ range.
reward	$\mathbb{R}$, change in x coordinate of pelvis plus a small penalty for using ligament forces.
terminal state	agent falls (pelvis $x < 0.65$) or 1000 steps in environment
stochasticity	• random strength of the psoas muscles • random location and size of obstacles

4 Methods

In this section we briefly describe the models we have evaluated in the task of the "Learning to run" challenge. We also describe our improvements to the model best performing in the competition: Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015).

4.1 On-policy methods

On-policy RL methods can only update agent's behavior according to the current policy. We consider two popular on-policy algorithms, namely Trust Region Policy Optimization (TRPO) (Schulman et al. 2015a) and Proximal Policy Optimization (PPO) (Schulman et al. 2017) as the baseline algorithms for environment solving.

Trust Region Policy Optimization (TRPO) is one of the notable state-of-the-art RL algorithms, developed by Schulman et al., that has theoretical monotonic improvement guarantee. As a basis, TRPO (Schulman et al. 2015a) using REINFORCE (Williams 1992) algorithm, that estimates the gradient of expected return $\nabla_\theta \eta(\pi_\theta)$ via likelihood ratio:

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

[2] https://github.com/stanfordnmbl/osim-rl

[3] https://github.com/stanfordnmbl/osim-rl/issues/78

where N is the number of episodes, T is the number of steps per episode, $R_t^i = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}^i$ is the cumulative reward and b_t^t is a variance reducing baseline (Duan et al. 2016). After that, an ascent step is taken along the estimated gradient. TRPO improves upon REINFORCE by computing an ascent direction that ensures a small change in the policy distribution. As the baseline TRPO we have used the agent described in (Schulman et al. 2015a).

Proximal Policy Optimization (PPO) as TRPO tries to estimate an ascent direction of gradient of expected return that restricts the changes in policy to small values. We used clipped surrogate objective variant of proximal policy optimization (Schulman et al. 2017). This modification of PPO is trying to compute an update at each step that minimizes following cost function:

$$Ł\theta = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \quad (2)$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$ is a probability ratio (the new divided by the old policy), $\hat{A}_t = R_t - b_t$ is empirical return minus the baseline. This cost function is very easy to implement and allows multiple epochs of minibatch updates.

4.2 Off-policy methods

In contrast to on-policy algorithms, off-policy methods allow learning based on all data from arbitrary policies. It significantly increases sample-efficiency of such algorithms relative to on-policy based methods. Due to simulation speed litimations of the environment, we will only consider Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015).

Deep Deterministic Policy Gradient (DDPG) consists of actor and critic networks. Critic is trained using Bellman equation and off-policy data:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi_\theta(s_{t+1})), \quad (3)$$

where π_θ is the actor policy. The actor is trained to maximize the critic's estimated Q-values by back-propagating through critic and actor networks. As in original article we used replay buffer and the target network to stabilize training and more efficiently use samples from environment.

DDPG improvements Here we present our improvements of DDPG method. We used some standard reinforcement learning techniques: action repeat (we repeat each action 5 times), scaled reward and additional fail penalty reward. After several attempts, we choose a scale factor of 10 (i.e. multiply reward by ten) for our experiments. For exploration we used Ornstein-Uhlenbeck (OU) process (Uhlenbeck and Ornstein 1930) to generate temporally correlated noise for efficient exploration in physical environments. Our DDPG implementation was parallelized as follows: n processes collected samples with fixed weights all of which were processed by the learning process at the end of an episode, which updated their weights. Since DDPG is an off-policy method, the stale weights of the samples only improved the performance providing each sampling process with its own weights and thus improving exploration.

Parameter noise Another improvement is the recently proposed parameters noise (Plappert et al. 2017) that perturbs network weights encouraging state dependent exploration. We used parameter noise only for the actor network. Standard deviation σ for the Gaussian noise is chosen according to the original work (Plappert et al. 2017) so that measure d:

$$d(\pi, \widetilde{\pi}) = \sqrt{\left(\frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_s[(\pi(s)_i - \widetilde{\pi}(s)_i)^2]\right)}, \quad (4)$$

where $\widetilde{\pi}$ is the policy with noise, equals to σ in OU. For each training episode we switched between the action noise and the parameter noise choosing them with 0.7 and 0.3 probability respectively.

Layer norm Henderson et al. showed that layer normalization (Ba, Kiros, and Hinton 2016) stabilizes the learning process in a wide range of reward scaling. We have investigated this claim in our settings. Additionally, layer normalization allowed us to use same perturbation scale across all layers despite the use of parameters noise (Uhlenbeck and Ornstein 1930). We normalized the output of each layer except the last for critic and actor by standardizing the activations of each sample. We also give each neuron its own adaptive bias and gain. We applied layer normalization before the nonlinearity.

Actions and states reflection symmetry The model has bilateral body symmetry. State components and actions can be reflected to increase sample size by factor of 2. We sampled transitions from replay memory, reflected states and actions and used original states and actions as well as reflected as batch in training step. This procedure improves stability of learned policy. If we dont use this step our model learned suboptimal policies, when for example muscles for only one leg are active and other leg just follows first leg.

5 Results

It this section we presents our experiments and setup. For all experiments we used environment with 3 obstacles and random strengths of the psoas muscles. We tested models on setups running 8 and 20 threads. For comparing different PPO, TRPO and DDPG settings we used 20 threads per model configuration. We have compared various combinations of improvements of DDPG in two identical settings that only differed in the number of threads used per configuration: 8 and 20. The goal was to determine whether the model rankings are consistent when the number of threads changes. For n threads (where n is either 8 or 20) we used $n-2$ threads for sampling transitions, 1 thread for training, and 1 thread for testing. For all models we used identical architecture of actor and critic networks. All hyperparameters are listed in Table 2. Our code used for competition and described experiments can be found in a github repo.[4]

[4]Theano: `https://github.com/fgvbrt/nips_rl` and PyTorch: `https://github.com/Scitator/Run-Skeleton-Run`

Table 2: Hyperparameters used in the experiments.

parameters	Value
Actor network architecture	$[64, 64]$, elu activation
Critic network architecture	$[64, 32]$, tanh activation
Actor learning rate	linear decay from $1e{-3}$ to $5e{-5}$ in $10e6$ steps
Critic learning rate	linear decay from $2e{-3}$ to $5e{-5}$ in $10e6$ steps
Batch size	200
γ	0.9
replay buffer size	$5e6$
rewards scaling	10
parameter noise probability	0.3
OU exploration parameters	$\theta = 0.1$, $\mu = 0$, $\sigma = 0.2$, $\sigma_{min} = 0.05$, $dt = 1e{-2}$, n_{steps} annealing $\sigma_{\text{decay}} 1e6$ per thread

Experimental evaluation is based on the undiscounted return $\mathbb{E}_\tau[\sum_{t=0}^{T} r(s_t, a_t)]$.

5.1 Benchmarking different models

Comparison of our winning model with the baseline approaches is presented in Figure 2. Among all methods the DDPG significantly outperformed PPO and TRPO. The environment is time expensive and method should utilized experience as effectively as possible. DDPG due to experience replay (re)uses each sample from environment many times making it the most effective method for this environment.

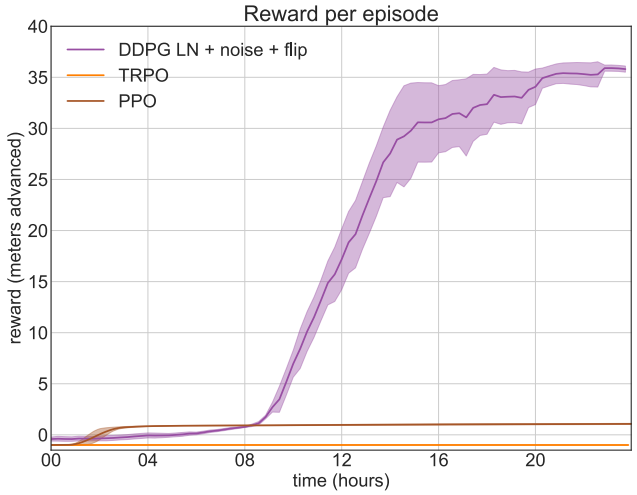


Figure 2: Comparing test reward of the baseline models and the best performing model that we have used in the "Learning to run" competition.

Table 3: Best achieved reward for each DDPG modification.

agent \ # threads	8	20
DDPG + noise + flip	0.39	23.58
DDPG + LN + flip	25.29	31.91
DDPG + LN + noise	25.57	30.90
DDPG + LN + noise + flip	**31.25**	**38.46**

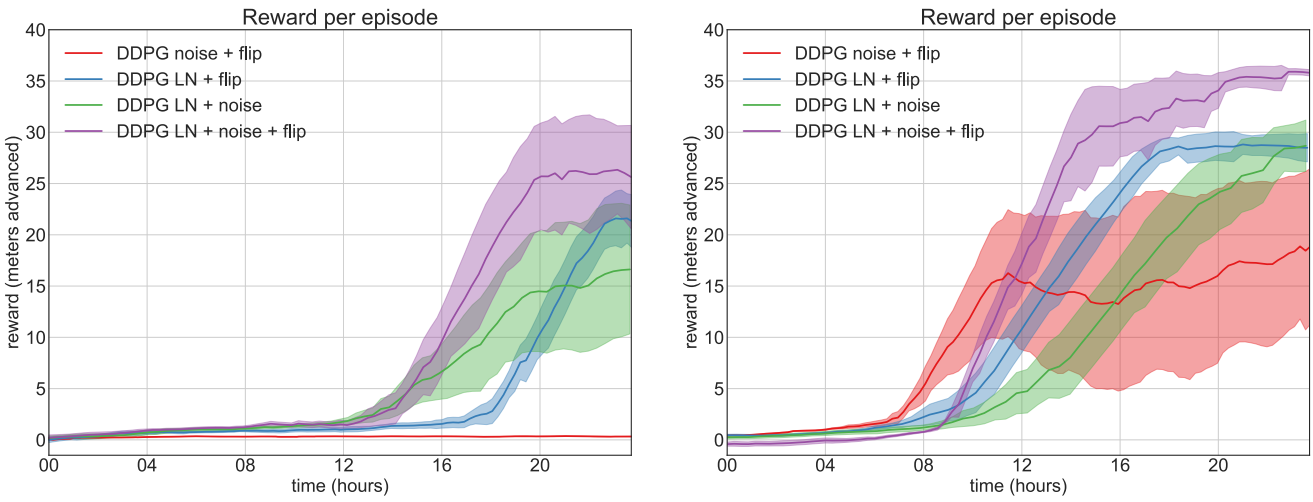
5.2 Testing improvements of DDPG

To evaluate each component we used an ablation study as it was done in the rainbow article (Hessel et al. 2017). In each ablation, we removed one component from the full combination. Results of experiments are presented in Figure 3a and Figure 3b for 8 and 20 threads respectively. The figures demonstrate that each modification leads to a statistically significant performance increase. The model containing all modifications scores the highest reward. Note, the substantially lower reward in the case, when parameter noise was employed without the layer norm. One of the reasons is our use of the same perturbation scale across all layers, which does not work that well without normalization. Also note, the behavior is quite stable across number of threads, as well as the model ranking. As expected, increasing the number of threads improves the result.

Maximal rewards achieved in the given time for 8 and 20 threads cases for each of the combinations of the modifications is summarized in Table 3. The main things to observe is a substantial improvement effect of the number of threads, and stability in the best and worst model rankings, although the models in the middle are ready to trade places.

6 Conclusions

Our results in OpenSim experiments indicate that in computationally expensive stochastic environments that have high-dimensional continuous action space the best performing method is off-policy DDPG. We have tested 3 modifications to DDPG and each turned out to be important for learning. Action states reflection doubles the size of the training data and improves stability of learning and encourages the agent to learn to use left and right muscles equally well. With this approach the agent truly learns to run. Examples of the learned policies with and without the reflection are present at this URL `https://tinyurl.com/ycvfq8cv`. Parameter and Layer noise additionally improves stability of learning due to introduction of state dependent exploration. In general, we believe that investigation of human-based agents in physically realistic environments is a promising direction for future research.



(a) 8 threads

(b) 20 threads

Figure 3: Comparing test reward for various modifications of the DDPG algorithm with 8 threads per configuration (Figure 3a) and 20 threads per configuration (Figure 3b). Although the number of threads significantly affects performance, the model ranking approximately stays the same.

References

[Ba, Kiros, and Hinton 2016] Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

[Bellman 1957] Bellman, R. 1957. A markovian decision process. *Journal of Mathematics and Mechanics* 679–684.

[Delp et al. 2007] Delp, S. L.; Anderson, F. C.; Arnold, A. S.; Loan, P.; Habib, A.; John, C. T.; Guendelman, E.; and Thelen, D. G. 2007. Opensim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering* 54(11):1940–1950.

[Duan et al. 2016] Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 1329–1338.

[Gu et al. 2017] Gu, S.; Holly, E.; Lillicrap, T.; and Levine, S. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 3389–3396. IEEE.

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

[Henderson et al. 2017] Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2017. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*.

[Hessel et al. 2017] Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2017. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*.

[Jaderberg et al. 2016] Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2016. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

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

[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

[Mnih et al. 2016] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

[OpenAI Roboschool 2017] OpenAI Roboschool. 2017. OpenAI Roboschool (`https://github.com/openai/roboschool`).

[Pathak et al. 2017] Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363*.

[Plappert et al. 2017] Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2017. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

[Schulman et al. 2015a] Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015a. Trust region policy opti-

mization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 1889–1897.

[Schulman et al. 2015b] Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015b. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

[Sherman, Seth, and Delp 2011] Sherman, M. A.; Seth, A.; and Delp, S. L. 2011. Simbody: multibody dynamics for biomedical research. *Procedia Iutam* 2:241–261.

[Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.

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

[Todorov, Erez, and Tassa 2012] Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 5026–5033. IEEE.

[Uhlenbeck and Ornstein 1930] Uhlenbeck, G. E., and Ornstein, L. S. 1930. On the theory of the brownian motion. *Physical review* 36(5):823.

[Williams 1992] Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.