Persistence Length Based Exploration in Reinforcement Learning

Riashat Islam*
Reasoning and Learning Lab
School of Computer Science
McGill University
riashat.islam@mail.mcgill.ca

Abstract

Scalable exploration methods in high dimensional continuous control tasks is a key challenge in reinforcement learning. Very little work has been done in developing optimally guaranteed exploration technique in high dimensional continuous action space deep RL scenarios. In this paper, we propose a trajectory based exploration method which explores trajectories as opposed to exploring state action pairs individually. Our method is based on the intuition of choosing the next exploratory action based on the trajectory so far, where the explored trajectories aims to explore the state space, using the notion of locally self avoiding random walks, often used in physics to describe the behaviour of polymer chains. We demonstrate our exploration method on a wide variety of continuous control tasks, using the OpenAI Gym's MuJoCo continuous control environments.

1 Introduction

Reinforcement Learning (RL) studies how the behaviour of an agent can be optimized such that it can act in an initially unknown environment by maximizing its cumulative reward Sutton and Barto ((1998)). The goal of the agent is to learn from its own experience by exploring the unknown environment, until it has sufficiently explored, and then to exploit the already learnt information about the environment. A long standing goal of reinforcement learning is to balance the trade-off between exploration and exploitation. During the exploration phase, the agent should actively seek out novel states and actions that might maximize its rewards, and during exploitation phase, the agent should maximize the short term rewards using its current knowledge and information about the environment.

Even though there are exploration techniques with theoretical guarantees in RL, the problem of exploration becomes difficult in environments with high dimensional state and action spaces. The theoretical guarantees for exploration are often in discrete state and action spaces, and often heuristic exploration strategies such as ϵ-greedy, random or Boltzmann exploration are used Mnih et al. ((2013)). However in cases where discretization of state and action space is difficult and not scalable, the problem of exploration becomes even harder. For example, in continuous action spaces often a Gaussian noise is added on the controls, such as in the policy gradient methods such as the deep deterministic policy gradient (DDPG) Lillicrap et al. ((2015)). There has not been much research effort for addressing exploration in continuous control where disretization of state and action space can scale exponentially in its dimensionality, such as in the Walker 2D task which has a 6 dimensional action space and the Humanoid task that has a 17 dimensional action space Duan et al. ((2016)), other than the recently proposed VIME explorarion method Houthooft et al. ((2016)).

*Work in progress in collaboration with Maziar Gomrokchi, Susan Amin and Doina Precup

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

In this paper, we propose a novel exploration strategy particularly suited for exploration in continuous control tasks, which is scalable and can be applied to action space of any dimensionality, uses the off-policy exploration samples to learn the action-value function and does not depend on the Markovian state of the environment. We propose a trajectory based exploration technique rather than state based exploration where the task for the agent during the exploration phase is to build up a trajectory of actions (without considering Markovian state) such as to explore the entire state space effectively. Our proposed algorithm relies on the local information about the visited state action pairs, and is derived from theoretical intuitions from the notion of polymer chains in physics. The action trajectory build up during exploration phase also ensures local self avoidance w.r.t to the action space.

Our proposed algorithm is based on the following intuitions. During the agent exploration phase, the choice of the next exploratory action should dependent not just on the Markovian state of the environment, but also on the explored trajectory of the state space so far. The trajectories should aim at filling up the existing environment and state space as quickly as possible. Our algorithm is based on the mechanism of locally self avoiding random walks, often used in physics to describe the behaviour of polymer chains, with the goal to explore effectively and efficiently in continuous control domains. The proposed algorithm is evaluated on a range of tasks as in the continous control MuJoCo environment.

2 Methodology

In section 2.1, we establish the notion of persistence length in the exploration phase, and its relations to polymer chains from physics. In section 2.2, we describe our algorithm applied on top of policy gradient methods, such as the deep deterministic policy gradient for continuous actions Lillicrap et al. ((2015)). We will describe how our method can be practically implemented, for the 2D action space, and higher order action spaces.

2.1 Persistence Length

Ideal chains are chains with no interaction between monomers that are far apart on the chain contour. The forumlation of a chain by a sequence of actions can be described in terms of bond vectors. We can represent the formulation of a chain as N bond vectors $\{\mu_i\}_{i=1,...N}$ each of length b_0 such that $\mu_i = \phi(s_i) - \phi(s_{i-1})$. A polymer chain can be described as a sequence of $N+1$ position vectors as well, $\{\phi(s_0), ...\phi(s_N)\}$ or N bond vectors, and the end to end vector of the chain U connects the first monomer to the last one. In a freely- rotating chain, the bond angle θ is invariant between two consecutive bond vectors while the azimuthal angle ϕ is drawn uniformly at random from the range $\big[0, 2\pi\big]$. The persistence length L_p is the contour length on the chain after which the chain forgets its initial orientation. A chain can also be thought of as a random walk with a constant length or step size. The persistence length guarantees local self avoidance on the chain.

We call our persistence length based algorithm for exploration as PolyRL algorithm. The PolyRL provides uniform coverage over the space of state action pairs. To develop some intuitions about the notion of building action chain trajectories for exploration in RL, we describe the following. Our PolyRL algorithm can be used in a phase of pure exploration in high dimensional continuous action spaces, and is controlled by a parameter, the persistence length of the walk. In the experimental section, we show that this algorithm can have significant advantages over simple random walks. Each exploratory action during pure exploration phase is based on the entire trajectory rather than just the current state of the environment. This is achieved implicitly by the properties of the walk, and each move is still chosen locally (based only on the current state and the previous move). Yet this locally controlled behavior gives rise to interesting global properties, for the trajectory, such as environment coverage.

The PolyRL algorithm emphasizes on trajectory-based exploration rather than the state-based exploration as well as a technique that relies on local information about visited state-action pairs. During the exploration phase, we can employ a range of persistence lengths and explore a range of low dimensional manifolds and then explore the selected manifold efficiently. The intention of our proposed algorithm is to guarantee the local self avoidance within a trajectory induced by the exploration policy. In the next section, we describe how our algorithm can be applied on top of the DDPG algorithm, to aid exploration in continuous control.

2.2 Implementation

Our complete algorithm is described in algorithm 1 for the 2D action space. Higher dimensional action space of the algorithm can be similarly extended (not discussed extensibly here).

Algorithm 1: PolyRL Algorithm (2D Action Space) on top of DDPG

Randomly initialize critic network Q(s,a | θ^Q) and actor network $\mu(s|\theta^\mu$ with weights θ^Q and θ^μ;
Initialise target network Q' and policy network μ';
Initialise two replay buffers B^e and B^d;
for *episode=1, 2, ... M* **do**
    PolyRL pure exploration phase $\rightarrow$ **for** *expl epoch until* $e = E$ **do**
        if $e == 0$ **then**
            Sample $\mathbf{A_0}$ and S_0 w.r.t ρ;
        else if $e == 1$ **then**
            Initialize $\mathbf{H_1}$ s.t. $||\mathbf{H_1}|| = b_o$;
            $\mathbf{A_1} \leftarrow \mathbf{A_o} + \mathbf{H_1}$;
        else
            Draw a sample θ from $\mathcal{N}(\mu, \sigma)$;
            $\theta_t \leftarrow$ toss a coin and choose between θ and $-\theta$;
            $\mathbf{A_e} \leftarrow \mathbf{A_{e-1}}+$ apply $\prod_2^{\theta_e}$ on $\mathbf{H_{e-1}}$;
        if A_e *is not valid* **then**
            Terminate the episode;
        else
            Apply step function on action $\mathbf{A_e}$ and observe S_{e+1} and R_{e+1};
            if S_{e+1} *is valid* **then**
                Continue;
            else
                End the episode and re-start the chain;
        Sample a random minibatch of transitions from buffer B^e;
        Update the Q critic network using off-policy exploration samples;
    Return trajectory of states and actions;
    Return end of trajectory state and action;
    Return updated Q critic network from PolyRL exploration phase;

    Deep Deterministic Policy Gradient (DDPG);
    for *t=1, 2,...T* **do**
        Select action a_t according to current policy $\mu(s_t|\theta^{mu})$;
        Executve action a_t and observe reward r_{t+1} and next state s_{t+1};
        Store transition to replay buffer B^d;
        Sample random minibatch from replay buffer B^d;
        Update the critic network by minimizing the loss;
        Update the actor policy network using sampled policy gradient ;
        Update the target networks;

Our algorithm can be described as follows. At the beginning of each episode, we perform pure exploration and explore trajectories of the unknown environment, based on building up a chain of actions using the notion of persistence length. We perform exploratory persistence based moves, and using the off-policy exploratory samples obtained from a persistence length based behavior policy, we update the Q network (in our case critic network in the actor-critic policy gradient method). The Q network is in other words trained using the exploratory samples. At the end of the exploration phase, we then follow the DDPG algorithm, but now the starting state of DDPG is the same as the last state visited during the exploratory phase. The actions following this are now taken based on the on-policy actor network and the critic and actor network are now trained based on the batch obtained

from the exploitation phase. Note that as done in DDPG, here we can also add a Gaussian noise on the controls, such that the critic network is again trained with a different off-policy behaviour policy, though now the behaviour policy is different then what we considered during the pure exploration phase.

3 Experiments

(a) (b) (c) (d) (e) (f) (g)

Figure 1: OpenAI Gym MuJoCo domains (Brockman et al., 2016; Duan et al., 2016):
(a) Ant, (b) HalfCheetah, (c) Hopper, (d) Humanoid, (e) Reacher, (f) Swimmer, (g) Walker.

We evaluated the PolyRL exploration algorithm on continuous control environments from the OpenAI Gym benchmark tasks which are developed using the MuJoCo physics simulator Brockman et al. ((2016)); Todorov et al. ((2012)). The PolyRL algorithm is built on top of the off-policy actor-critic DDPG algorithm, using the OpenAI rllab. We consider the following MuJoCo locomotion tasks for evaluating PolyRL and compare it with the current exploration method, VIME, on these tasks. The tasks for the experimental setup are as follows: Swimmer ($\mathcal{S} \subseteq \mathbb{R}^{33}, \mathcal{A} \subseteq \mathbb{R}^2$), Reacher ($\mathcal{S} \subseteq \mathbb{R}^{11}, \mathcal{A} \subseteq \mathbb{R}^2$), Hopper($\mathcal{S} \subseteq \mathbb{R}^{11}, \mathcal{A} \subseteq \mathbb{R}^3$), HalfCheetah ($\mathcal{S} \subseteq \mathbb{R}^{20}, \mathcal{A} \subseteq \mathbb{R}^6$), Walker ($\mathcal{S} \subseteq \mathbb{R}^{17}, \mathcal{A} \subseteq \mathbb{R}^6$), Humanoid ($\mathcal{S} \subseteq \mathbb{R}^{376}, \mathcal{A} \subseteq \mathbb{R}^{17}$).

The performance of the algorithms are measured through the average return over fixed number of steps (typically 1000) during evaluation phase. For each of the environments below, we built PolyRL on top of DDPG. We evaluated our algorithm for 15000 training epochs, with a fixed length of 1000 steps per epoch. Our algorithm was implemented using two replay buffers, one for the PolyRL exploration phase to train the Q network with off-policy exploration samples, and the other for the critic network in DDPG. Note that the same Q network is updated at the end of exploration phase in PolyRL, and then the pre-trained Q network is used as the critic in DDPG. Both replay buffers had a maximum size of 10^6 and trajectory sample tuples were added to the replay buffer after every 1000 steps. The networks were trained when the buffer size exceeded $10,000$, using batch samples of size 32. We used the ADAM optimizer for the Q network and the policy network update method, with the same learning rate for the actor critic with a starting value of 10^{-3}.The discount factor γ for the experiment was chosen to be 0.99.

During the PolyRL exploration phase, the self avoiding trajectory chain of actions were built with $20,000$ exploratory steps per every episode. Without fine-tuning for the hyperparameters, we evaluated the PolyRL with a persistence length parameter $L_p = 0.08$ and a step size $b_0 = 0.0004$. During the exploration phase, samples were added to the replay buffer after every 1000 steps. If an invalid state was reached during exploration, the exploration phase was restarted again with a new randomly sampled action, until the total length of the exploratory steps was equal to $20,000$.

The table below summarizes some of the current benchmark results on the MuJoCo environments we are considering for our work, taken from Duan et al. ((2016)); Gu et al. ((2016))

Few Benchmark Results (Max Return)			
Task	**Action Dim**	**TRPO**	**DDPG**
Swimmer	2D	110	150
Reacher	2D	-6.7	-6.6
Hopper	3D	2486	2604
HalfCheetah	6D	4734	7490
Walker	6D	3567	3626
Humanoid	17D	918	552

In the next section, we show the performance of our PolyRL algorithm with DDPG on the Hopper and the Swimmer environments as initial experiments, without taking averaged results (over multiple GPU runs) and without fine-tuning for the hyperparameters.

3.1 Hopper Environment

The hopper task is a planar monopod robot with 4 rigid links, corresponding to the torso, upper leg, lower leg, and foot, along with 3 actuated joints. It is known that for the Hopper task, more exploration is needed than the Swimmer task, since a stable hopping gait has to be learnt without falling. Otherwise, the robot may get stuck in a local optimum moving forward. The 20-dim observation includes joint angles, joint velocities, the coordinates of center of mass, and constraint forces. The reward is given by $r(s,a) = v_x - 0.005\dot{||}a||_2^2 + 1$, where v_x is the forward velocity.

We first evaluated our algorithm on the Hopper environment. Note that our experiment have been run for only 1000 episodes so far, and we reach a maximal reward of approximately 1000 as well, whereas the current benchmark with using DDPG algorithm (with Gaussian noise added to the controls for exploration) reaches a maximal reward of 2604. Our hope is that, with fine-tuned hyperparameters for the PolyRL exploration phase, and running our algorithm over long runs of episodes in GPU would make our approach outperform naive DDPG on Hopper. Our aim is to show that the PolyRL exploration method can enable faster and more data-efficient learning in off-policy actor-critic methods for continuous control.

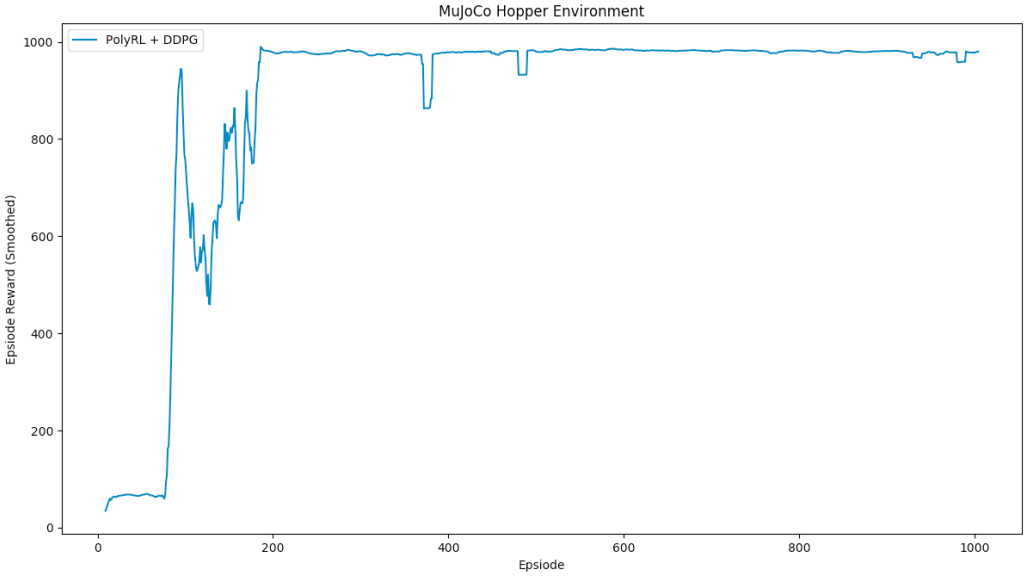

Figure 2: PolyRL + DDPG algorithm on the highly difficult MuJoCo Hopper Environment, with 3 dimensional action space

3.2 Swimmer Environment

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 again given by $r(s,a) = v_x - 0.005\dot{||}a||_2^2$, where v_x is again the forward velocity.

The swimmer environment is a highly difficult hierarchical task whose reward signal is naturally sparse. In this task, a two-link robot needs to reach "apples" while avoiding "bombs" that are

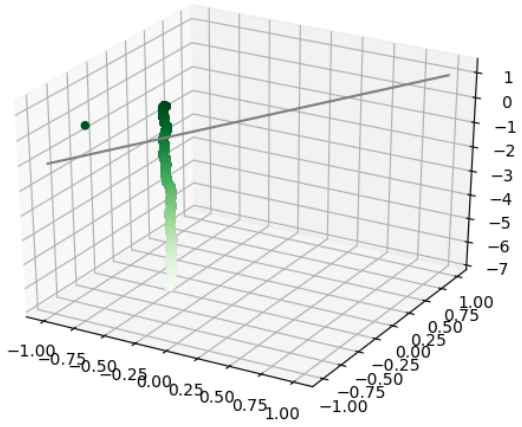

Figure 3: Explored trajectory of actions in PolyRL (Episode 1)

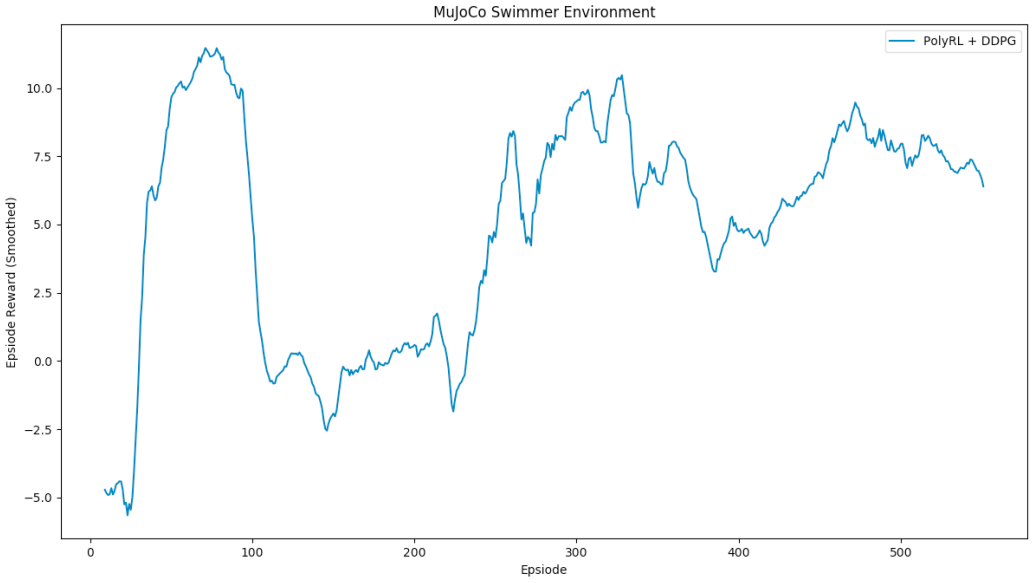

Figure 4: PolyRL + DDPG algorithm on the highly difficult MuJoCo Swimmer Environment, with 2 dimensional action space

perceived through a laser scanner. However, before it can make any forward progress, it has to learn complex locomotion primitives in the absence of any reward. Except for VIME, it is known that none of the RL methods on this task were able to make progress on this with naive exploration. Our results of PolyRL on top of DDPG on the Swimmer environment task is given below.

Figures show the exploratory action heatmap of our algorithm in the 2D action space on the swimmer environment. It shows that during the PolyRL pure exploration phase, the agent tries out different exploratory actions based on building a polymer chain of actions.

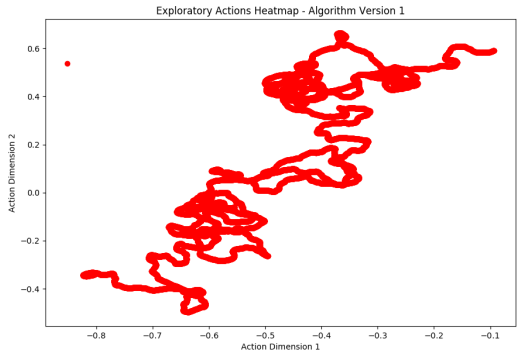

Figure 5: Explored trajectory of actions in PolyRL (Episode 1)

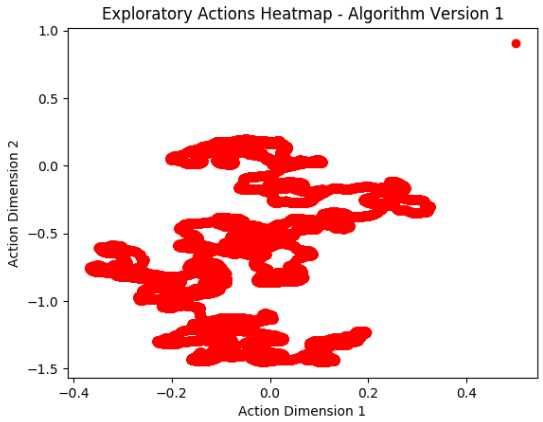

Figure 6: Explored trajectory of actions in PolyRL (Episode 2)

4 Conclusion and Future Work

In this work, we proposed a new method for exploration in reinforcement learning in the continuous action space. Our exploration method leverages the notion of locally self avoiding random walks, and it can be applied as a pure exploration phase to explore trajectories instead of state action pairs. The algorithm is controlled by a parameter, called the persistence length of the walk. As initial steps towards this project, we applied the PolyRL algorithm on the off-policy actor-critic DDPG framework, and evaluated our algorithm on several MuJoCo locomotion tasks. We initially show results on the Hopper 3D and the Swimmer 2D action space tasks.

This project is still in its initial stages. As the next step for this work, we will evaluate our algorithm on several higher dimensional action spaces, utilizing several experiment runs (typically of the order of 50 and taking averaged results) on the GPU. As a baseline for comparison, we will compare our work on the VIME exploration method, which was applied on the on-policy TRPO algorithm Schulman et al. ((2015)). However, compared to VIME + TRPO, our method is applied on the off-policy case, leading to PolyRL + DDPG. An interesting direction of future work will be to consider the PolyRL exploration algorithm on on-policy policy gradient methods such as TRPO. Since this exploration method is based on the intuition of exploring trajectories by building up a polymer chain of actions, this method can act as a benchmark exploration method to be applied for continuous control

Acknowledgements

We thank Maziar Gomrokchi and Susan Amin for offering this project as part of the COMP767 Reinforcement Learning course at McGill University. We thank Doina Precup for helpful advice, discussions and suggestions on this work, and Pierre-Luc Bacon for his insights. We also thank Shane Gu for sharing and answering questions about rllab code and using the OpenAI Gym MuJoCo simulator.

References

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1329–1338, 2016. URL `http://jmlr.org/proceedings/papers/v48/duan16.html`.

S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *CoRR*, abs/1611.02247, 2016. URL `http://arxiv.org/abs/1611.02247`.

R. Houthooft, X. Chen, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel. VIME: variational information maximizing exploration. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1109–1117, 2016. URL `http://papers.nips.cc/paper/6591-vime-variational-information-maximizing-exploration`.

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. *CoRR*, abs/1509.02971, 2015. URL `http://arxiv.org/abs/1509.02971`.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1889–1897, 2015. URL `http://jmlr.org/proceedings/papers/v37/schulman15.html`.

R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9(5):1054–1054, 1998. doi: 10.1109/TNN.1998.712192. URL `http://dx.doi.org/10.1109/TNN.1998.712192`.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. URL `http://dx.doi.org/10.1109/IROS.2012.6386109`.