Project Report for CMPT 888 Computer Animation

Comparison of Deep Reinforcement Learning Algorithms for Learning Control Policies for Physics-based Locomotion Tasks

Anmol Sharma
Medical Image Analysis Laboratory
School of Computing Science
Simon Fraser University
Burnaby, Canada
anmol_sharma@sfu.ca

Abstract*—Reinforcement learning (RL) have recently become the framework of choice to develop intelligent agents that can control various entities inside an environment through learning by trial and error. One such task where RL algorithms have been applied is to control simulated robots in a physics-based environment to achieve a set specific goal. These goals can be to run, walk, hop, grasp objects and so on depending on the environment. Recently due to a surge in deep learning (DL) research, RL algorithms have underwent a transformation where many algorithms were revived by utilizing DL methods in some of the internal workings of the RL algorithms. This class of algorithms were named as deep reinforcement learning (DRL). In this project we compare three different DRL algorithms with respect to their ability to control a simulated physics-based robot in an environment. More specifically, we compare Vanilla Policy Gradient (VPG) or REINFORCE, Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) algorithms in three different physics-based locomotion environments HalfCheetah-v2, Hopper-v2, and Walker2d-v2 defined in MuJoCo framework ordered by increasing difficulty. Through our experiments, we found the PPO exhibited better performance in all environments in terms of the total reward gained in an episode. VPG performed worse than PPO but better than TRPO due to its simplistic policy gradient approach, though the high variance in training process due to reliance on absolute episode rewards was evident from the experiments. We also observed the TRPO's convergence is inherently highly predictable, stable, and mostly monotonically increasing, leading to good convergence guarantees but with increased training time.

1. Introduction

Locomotion is key for the survival of species in the animal kingdom. Biologically, it can be defined as an act of moving from point A to B. Physically, it can be represented as a change in position of an entity over time. The motion is usually described in terms of physical quantities such as distance, velocity, acceleration, and time. In animation, it is one of the most fundamental motions to be simulated or modeled using keyframing.

Developing physics-based locomotion models for a physically-controlled character has been a long standing research problem in the animation and robotics community [1]. Inducing robust control and balance to the character have been challenging due to the under-actuated and high dimensional nature of the character [2]. Locomotion in animation (under the umbrella term character animation) has been handled using many approaches, namely - manual keyframe based designing, using recorded motion capture data, algorithmic physics-based methods [3], and more recently controller learning based methods which constitute some form of machine learning algorithm [4] [5] [6].

Manual keyframe based animation of gaits provides a lot of control over the character motion, but comes with a huge cost of manual labour intensive work. Physics-based simulation for character walking animation has shown promise due to its robustness against unexpected environmental stimuli. However their use for actual real-time character animation in practice has been limited mostly to objects populating the game world, such as furniture, vehicles, hair, clothing, fluids, and so on [7] [8]. Active simulation is not always employed for character control in commercial frameworks as the character movement (walking/locomotion) is commonly handled with kinematic techniques [3] [8]. Moreover, physics based controllers tend to provide lesser fine grained control over developing motion gaits.

Learning based approaches have also shown great promise. One of the approaches towards learning controllers for physics-based character control is by using Reinforcement Learning (RL) [3] [8]. RL is a class of machine learning methods that learn using the idea of trial-and-error. An "agent" learns to perform a specific task by seeing numerous "episodes" of trials, in the hope of finding a viable "trajectory" to navigate through that task. Classical RL has been applied to this problem with varying degree of success [9], mostly due to the fact that a generate-and-test approach

is often impractical because of the large number of trials and very-high dimensional systems [3]. More recently, inspired by the success of Deep Learning [10] techniques in a variety of different fields, modified versions of RL algorithms in the form of Deep Reinforcement Learning (DRL) [11] [12] [13] [14] [15] [5] [6] have been proposed. Although some of these recent works applied DRL methods to the problem of physics-based locomotion [12] [15] [5] [6] using different physics environments, a single resource which compares these algorithms on a standard test platform is still not present to the best of our knowledge.

In this project, we compare some of the recently proposed DRL methods, specifically the Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO) and Vanilla Policy Gradients (VPG) for the task of controlling an agent in a physics-based environment. For this, we utilize OpenAI Gym [16] toolkit and MuJoCo [17] as our physics simulation engine which provides multiple environments for different agents (biped humanoid, ant, walker2d, hopper and others). To provide a fair comparison amongst the algorithms in learning control policies in environments with varying levels of complexity, we test the methods on three physics-based environments, namely HalfCheetah-v2, Walker2d-v2, and Hopper-v2, ordered in increasing levels of difficulty.

The remainder of the paper is structured as follows: Section 2 discusses some of the related work. Section 3 provides a detailed background on the RL problem. Experimental setup is described in Section 4. Section 5 outlines the outcomes of this project and discusses some of the insights and comments on the performance of the algorithms. Section 6 concludes the paper.

2. Related Work

Computer animation using physics-based locomotion simulation provides a different approach compared to kinematic data-driven or keyframing approaches. One of the earliest work by Armstrong et al. [18] discussed physics-based simulation of virtual characters. This lead to a plethora of literature springing up to apply this idea. Since a thorough review of the literature is beyond the scope of this report, we provide a fairly recent review of the field. Yin et al. [2] proposed a simple physics-based controller for biped locomotion. The offline approach allowed interactive designing of gaits, which were robust against unexpected environmental stimuli. Coros et al. [19] proposed GENBICON, which supported multiple different gait styles, and was shown to generalize towards a variety of different tasks. In later work, Coros et al. [20] develop a control framework for several four-legged creatures, incorporating a flexible spine control model.

Learning based methods, specifically Reinforcement Learning (RL) were first applied by Grzeszczuk et al. [21] towards physics-based simulation for physically realistic animation generation. One of the early seminal works in RL was by Ronald. J. Williams in [22] called the Vanilla Policy Gradients (VPG)/REINFORCE family of algorithms, upon whose ideas many of the state-of-art algorithms today are based. Recently, after the Deep Learning [23] [10] revolution, RL algorithms enjoyed renewed interest, and several methods were proposed [12] [13] [15] [5] [6]. Lillicrap et al. [12] proposed a continuous version of the Deep Q-Learning algorithm called Deep Deterministic Policy Gradient (DDPG), and applied the method on the task of legged locomotion. Schulman et al. [13] build upon DDPG by proposing an iterative procedure for optimizing policies with guaranteed monotonic improvement, called the Trust Region Policy Optimization (TRPO) algorithm. In their later work, Schulman et al. [15] proposed changes to the original TRPO algorithm, by removing many hard-to-implement constraints in the form of a clipped surrogate function. The algorithm was called Proximal Policy optimization (PPO). Peng et al. [5] propose a two stage deep RL based dynamic locomotion controller where a low level controller is responsible for low-level coordination of the characters limbs for locomotion, and a high level controller is responsible for high-level task specific objectives such as navigation. Liu et al. [6] utilize the Deep Q-Learning framework by Mnih et al. [24] for building schedulers for highly dynamic behaviours.

3. Background

Reinforcement Learning (RL) refers to a family of machine learning algorithms which learn by trial and error. More specifically, an RL agent learns to perform actions in an environment through a series of delayed reward signals. An RL agent is typically composed of two components: an agent and an environment as illustrated in Figure 1.

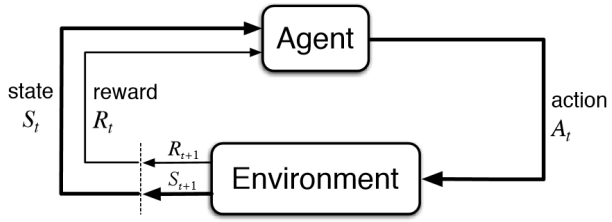

Figure 1. Reinforcement Learning: Agent and Environment

The loop starts by the environment sending a state S_t to the agent, which is used the agent to decide upon an action A_t to take. The action is fed back into the environment, which then outputs the new state S_{t+1} and a reward value R_{t+1} back to the agent. The loop continues until the environment sends a terminal state ending the episode.

Mathematically, the RL problem uses a number of common terms, which will enable understanding of the subsequent algorithms. Define $\mathcal{A}$ be the space of all possible actions a, $\mathcal{S}$ be the space of all possible states s. r_t be the immediate reward signal at time t from the environment after an action a_t is performed at state s_t. Let π be the policy which determines which action a_t to take at a particular state s_t. $V(s_t)$ be the value function which predicts the expected long-term return with discount γ given the agent is

in the state s_t. A function $Q(s, a)$, similar to to V except it takes an extra parameter a, outputs the long term expected reward with discount. Also, define $A(s, a) = Q(s, a) - V(s)$ called the advantage function which is commonly used in some algorithms. Intuitively it means how good an action is compared to the average action for a specific state, where the average action value is computed by V.

In this project, we compare three different Reinforcement Learning (RL) algorithms, namely Vanilla Policy Gradient (VPG)/REINFORCE, Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO). In the subsequent subsections we give a brief overview of these methods.

3.1. Vanilla Policy Gradient (VPG)/REINFORCE

Vanilla Policy Gradient (VPG) or REINFORCE is a gradient based method where the instead of parameterizing the functions V and Q, the policy function π is parameterized:

$$\pi_\theta(s, a) = \mathbb{P}[a|s, \theta] \tag{1}$$

The goal of the algorithm is to find the best set of parameters θ for the policy, by optimizing a cost function $J(\theta)$ using gradient ascent:

$$\nabla\theta = \alpha\nabla_\theta J(\theta) \tag{2}$$

where $\nabla_\theta J(\theta)$ is the policy gradient denoted by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log\pi_\theta(s, a)v_t] \tag{3}$$

α is the step-size parameter, the expected return v_t is sampled directly from the episode, which is simply the total episodic reward, and used as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$. Policy based methods demonstrate better convergence properties, while being effective in high-dimensional or continuous spaces. However, they typically converge to local optimum, and evaluation of policy is typically inefficient and exhibit high variance because the sampled rewards can be very different from one episode to another.

3.2. Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) is a policy gradient algorithm proposed by Schulman et al. [13], which builds upon the ideas of another algorithm DDPG and tries to address some of it's drawbacks, specifically the choice of step-size. TRPO tries to mitigate this issue using a method which guarantees that the parameter updates always leads to policy improvement, or more specifically, the expected discounted long-term reward η to be always increasing:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \tag{4}$$

Where $r(s_t)$ is a reward function, and γ is the discount parameter. For any new policy $\overline{\pi}$, $\eta(\overline{\pi})$ can be viewed as the expected return of policy $\overline{\pi}$ in terms of the advantage over π which is the old policy:

$$\eta(\overline{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \ldots\ \overline{\pi}}\left[\sum_{t=0}^{\inf}\gamma^t A_\pi(s_t, a_t)\right] \tag{5}$$

η can also be rewritten in the form of discounted visitation frequencies as follows:

$$\eta(\overline{\pi}) = \eta(\pi) + \sum_s \rho_{\overline{\pi}} \sum_a \overline{\pi}(a|s) A_\pi(s, a) \tag{6}$$

Where $\rho_{\overline{\pi}}$ is the discounted visitation frequencies, given as:

$$\rho_{\overline{\pi}} = P(s_o = s) + \gamma P(s_1 = s) + \gamma^2 P(s2 = s)\ldots. \tag{7}$$

Optimizing $\eta(\overline{\pi})$ is hard, hence an approximation is proposed:

$$L_\pi(\overline{\pi}) = \eta(\pi) + \sum_s \rho_{\overline{\pi}}(s) \sum_a \overline{\pi}(a|s) A_\pi(s, a) \tag{8}$$

and then the policy is updated using the equation:

$$\overline{\pi}(a|s) = (1 - \alpha)\pi(a|s) + \alpha\pi'(a|s) \tag{9}$$

The new policy is constrained using the following theorem:

$$\eta(\overline{\pi}) \geq L_\pi(\overline{\pi}) - CD_{KL}^{max}(\pi, \overline{\pi}) \tag{10}$$

$$\text{where } C = \frac{4\epsilon\gamma}{(1 - \gamma)^2} \tag{11}$$

C represents the penalty coefficient, whereas D_{KL}^{max} denotes the maximum KL divergence of the two parameters for each of the policy. The concept of KL divergence originated from information theory, in this case describing information loss or difference between the parameters π and $\overline{\pi}$. The equation above implies that the expected discounted long term reward η will be monotonically improving as the RHS is maximized. The theorem to prove this is beyond the scope of this report, and the readers are redirected to the original paper [13] for a rigorous proof. The final optimization problem that the algorithm solves is as follows:

$$maximize_\theta \sum_s \rho_{\theta_{old}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{old}}(s, a)$$
$$subject\ to\ \overline{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta \tag{12}$$

This objective function is also called a "surrogate" function as it contains a probability ratio between current policy π and next policy $\overline{\pi}$. TRPO successfully addresses the issue with DDPG, in a way that the performance increases monotonically, where the subset of region lying within the constraint is called trust region, hence the name Trust Region Policy Optimization.

3.3. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) was a follow-up work by Schulman et al. [15] that addressed many of the issues with TRPO. TRPO despite its guarantees of monotonic improvement and impressive empirical performance, proved to be hard to implement due to its KL divergence based constraints, which either need to be approximated by Fisher Information Matrix, or by Conjugate Gradient method, either of which adds complexity to the computations. Also, the hard constraints in TRPO leads to slower convergence times. PPO applies similar constraints as TRPO but in a computationally effectively, and easy way, by proposing a clipped surrogate objective function.

The proposed PPO objective is as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \quad (13)$$

where ϵ is a hyperparameter, and $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}(a_t|s_t)}}\hat{A}_t$. The second term modifies the surrogate objective by clipping the probability ratio r_t which removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$. Finally the minimum of clipped and unclipped objective function is selected.

4. Experiments

For our comparison of the previously detailed methods, we highlight the experimental methodology that we adopted below.

4.1. Environment

In this work, we use the physics-based environments defined in the software package MuJoCo [17], using abstract interface provided by the OpenAI Gym [16] open-source library. Since different physics-based simulated robotic task environments have different mechanics, and in turn, different complexities, we test the agents in three different environments, namely HalfCheetah-v2, Hopper-v2, and Walker2d-v2, ordered in increasing level of difficulty. These environments provide different physics mechanics, where the agent has to learn a varied set of actions (in terms of different number of joints). We use the standard XML files supplied by OpenAI[1] to create the environment without any modification to the reward function or the dynamics of the character. The environments are shown in the Figure 2.

4.2. Setup

The agents were implemented using an open-source reinforcement learning libary called TensorForce [25] which is based upon another open-source library called TensorFlow [26]. TensorForce allows building RL agents using a JSON-like definition which is highly modular, in the sense that any component can be plugged in and out of the agent and tested. This allows high flexibility while experimentation. The code for this work will be uploaded along with this paper. The hyperparameters for each of the agent were chosen to be as close to the original definitions as possible, except for VPG where the network for the actor network was changed to a deeper one than the one defined in the original paper. The exact values of the hyperparameters used are defined in the Tables 1, 2, 3. We train all the agents for a total of 15000 episodes due to limited computational capabilities. Due to a hard limit on the number of episodes to train, it cannot be guaranteed that each agent would be able to successfully learn how to control the robot in the environment. This limitation is demonstrated in the Walker2d-v2 environment where none of the agents could learn how to control the character even in the most rudimentary form. A thorough analysis of the convergence limits for each agent is beyond the scope of this project due to hardware and time limitations.

1. https://github.com/openai/gym/tree/master/gym/envs/mujoco/assets

TABLE 1. HYPERPARAMETER VALUES FOR VPG AGENT

Hyperparameter Name	**Value**
baseline_network	[32, 32]
baseline_optimizer	Adam
baseline_optimizer_lr	$1e^{-3}$
num_steps	5
optimizer	Adam
learning_rate (η)	$2e^{-2}$
discount (γ)	0.99
memory_capacity	5000
batch_size	20 episodes
update_frequency	20 episodes
network	[64, tanh, 64, tanh, linear]

TABLE 2. HYPERPARAMETER VALUES FOR TRPO AGENT

Hyperparameter Name	**Value**
baseline_network	None
baseline_optimizer	None
optimizer	Adam
learning_rate (η)	0.01
discount (γ)	0.99
memory_capacity	5000
batch_size	20 episodes
update_frequency	20 episodes
network	[64, tanh, 64, tanh, linear]

4.3. Evaluation Metric

For our comparison, we evaluate the algorithms using the total reward gained by the agent in the episodes by the

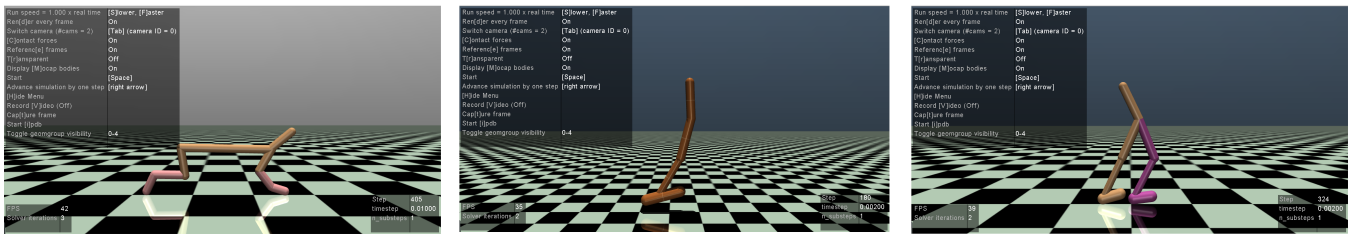

Figure 2. MuJoCo environments on which the algorithms were compared. a) HalfCheetah-v2, b) Hopper-v2, and c) Walker2d-v2.

TABLE 3. HYPERPARAMETER VALUES FOR PPO AGENT

Hyperparameter Name	**Value**
baseline_network	[32, 32]
baseline_optimizer	Adam
learning_rate (η)	0.001
num_steps	5
optimization_steps	50
step_optimizer	Adam
step_optimizer_learning_rate	0.001
discount (γ)	0.99
memory_capacity	5000
batch_size	10 episodes
update_frequency	10 episodes
network	[64, tanh, 64, tanh, linear]

end of the training process. The total reward for an episode is calculated as follows:

$$\mathcal{R} = \sum_{t=0}^{n} r_t \tag{14}$$

where n = number of time steps for the particular episode.

5. Results and Discussion

The results from our comparison are illustrated in Figure 3. We discuss the results for each algorithm with respect to every environment, in the subsequent subsections. For qualitative results, we also release a video of each agent performing in the environment[2].

5.1. HalfCheetah-v2

The HalfCheetah-v2 environment presents a cheetah robot with 6 joints to be controlled by an agent, with the ultimate goal to run as fast as it can. In this environment, all the algorithms were able to train to atleast partially achieve the task. However, the PPO agent performed the best in this environment with its fast pace and good running gait. PPO was followed closely by the VPG agent, which was able to make the robot run, but with a lower speed. TRPO on the other hand, could not learn to move forward during the limited training period of 15000 episodes, but could only learn how to walk/hop in place, trying to run forward. This is expected, since due to the hard constraints applied in TRPO which allows only monotonically increasing (in terms of policy performance) updates, it leads to slower convergence time. The performance of PPO can be explained due to its simplistic nature compared to TRPO, but still exhibiting similar theoretical constraints as TRPO, which lead it to learn and converge faster than all other algorithms. This is also clear from the convergence plot, where PPO and VPG converge to similar high reward values by the end of training, while TRPO although increasing monotonically, does not converge to a high enough reward. Although we note that given more time to train TRPO would have definitely converged, and might even have performed at par with PPO. The convergence behaviour of TRPO is stable, while PPO diverges and falls in terms of total reward per episode towards the end of training. VPG on the other hand is highly erratic due to its inherent limitation of the high variance and reliance on absolute episodic rewards rather than predicted long-term rewards.

5.2. Hopper-v2

Hopper-v2 environment in OpenAI Gym provides a one-legged robot whose aim is to hop as far as possible. This is an intermediate control task, where although the number of learnable joints are lower, there is complexity in the form of timing of actions, that play an integral part towards solving the environment. During our experiments, none of the agents was able to learn a highly robust gate to solve the environment and hop very far. Hence we compare the algorithms in terms of the number of steps they take before the robot falls on the ground. Using this metric, PPO agent performed the best, where the agent was able to take atleast 3 steps before the robot would fall over. However both TRPO and VPG were not able to learn to control the robot in this environment, where the TRPO controlled robot would fall instantaneously, followed by the VPG agent which would fall after struggling to balance on one step. This is also evident from the Figure 3 where the convergence plot shows that both VPG and TRPO get similar rewards for episodes towards the end of training, while PPO after falling down still regains a higher reward value during the end of training.

2. https://youtu.be/tlNHA7d1aLA

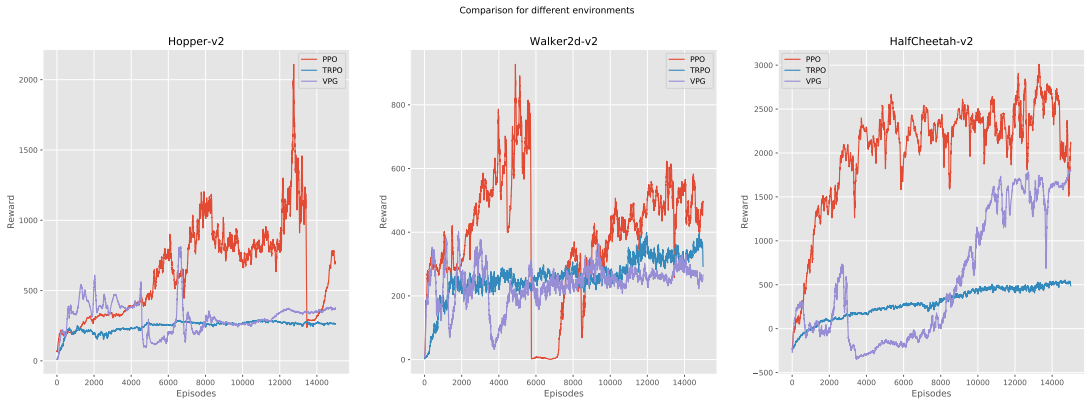

Figure 3. Convergence plots for each algorithm and environment.

The convergence graph for TRPO however is one of the most stable.

5.3. Walker-2d

Walker-2d environment presents a biped robot with the task of developing a walking gait suitable to walk over an extended period. The environment does not provide any obstacles to force the agent to develop a robust gait. This environment was the most complex simulation to be solved by any of the agent, due to its high-dimensional nature of joint control. Qualitatively, due to the limited nature of our hardware and training time, none of the agents were able to solve this environment within 15000 episodes. Needless to say the agents must be trained much longer to solve this environment, for about 50000+ episodes. However for the sake of comparison we provide the convergence plots of the algorithms for the limited training time of 15000 episodes. It can be seen that the PPO agent starts to converge to a high reward value by the end of training, before falling down. TRPO expectedly follows a monotonically increasing convergence path, though it does not come close to the highest rewards compared to TRPO and PPO. PPO's plot is very interesting, in the sense that first it increases, drops, and then sharply increases again. Qualititavely, it can be seen that PPO agent controlled biped was at a more advanced stage of learning to walk than other agents.

6. Conclusion

In this projecct we compare three different deep reinforcement learning (DRL) algorithms on three different physics-based tasks defined inside the MuJoCo environment. We implement the agents in TensorForce framework using similar hyperparameter values to provide a fair comparison. We also choose environments with varying levels of complexity in terms of control ability and overall goal to achieve. In our analysis we found that PPO agent converges much faster to a high episode reward value compared to TRPO and VPG. Out of all three, TRPO had the most ideal and predictable convergence plot due to its strong constraints on policy improvement, though the time required to successfully train a TRPO agent was not investigated in this project due to hardware constraints.

7. Acknowledgement

Thanks to Ben Cardoen and Harmanpreet Kaur for insightful discussions and comments on the project.

References

[1] S. Collins, A. Ruina, R. Tedrake, and M. Wisse, "Efficient bipedal robots based on passive-dynamic walkers," *Science*, vol. 307, no. 5712, pp. 1082–1085, 2005.

[2] K. Yin, K. Loken, and M. Van de Panne, "Simbicon: Simple biped locomotion control," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 105.

[3] T. Geijtenbeek and N. Pronost, "Interactive character animation using simulated physics: A state-of-the-art review," in *Computer Graphics Forum*, vol. 31, no. 8. Wiley Online Library, 2012, pp. 2492–2515.

[4] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 138, 2016.

[5] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[6] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Transactions on Graphics*, vol. 36, no. 3, 2017.

[7] I. Millington, *Game physics engine development: how to build a robust commercial-grade physics engine for your game*. CRC Press, 2010.

[8] B. Leonard-Cannon, "Physics-based character control with deep reinforcement learning," Ph.D. dissertation, McGill University Libraries, 2016.

[9] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2849–2854.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] 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.

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[14] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

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

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

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

[18] W. W. Armstrong and M. W. Green, "The dynamics of articulated rigid bodies for purposes of animation," *The visual computer*, vol. 1, no. 4, pp. 231–240, 1985.

[19] S. Coros, P. Beaudoin, and M. Van de Panne, "Generalized biped walking control," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, p. 130, 2010.

[20] S. Coros, A. Karpathy, B. Jones, L. Reveret, and M. Van De Panne, "Locomotion skills for simulated quadrupeds," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4. ACM, 2011, p. 59.

[21] R. Grzeszczuk, D. Terzopoulos, and G. Hinton, "Neuroanimator: Fast neural network emulation and control of physics-based models," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, pp. 9–20.

[22] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," in *Reinforcement Learning*. Springer, 1992, pp. 5–32.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[25] M. Schaarschmidt, A. Kuhnle, and K. Fricke, "Tensorforce: A tensorflow library for applied reinforcement learning," Web page, 2017. [Online]. Available: https://github.com/reinforceio/tensorforce

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.