WPI

Soccer Playing MuJoCo Agent

Reinforcement Learning – Fall 2019

Advisor: Prof Yanhua Li

Project by Group 2

1. Abhishek Shah [DS]
2. Ankur Gupta [CS]
3. Rajendra Kante [DS]
4. Sai Vineeth K V [RBE]

ABSTRACT

There has been significant work done in playing games such as Go or Atari where there is a clearly identifiable goal of an agent trying to win the game with a well-defined reward function. However, in the case of an agent learning to do actions such as walking and running, the agent will need to learn a much more" implicit" reward function. The behavior is complex and continuous, and the agent needs to acquire special skills. Additionally, in the case of the games, the state space can be discretized to simplify the learning process. However, in the case of agents containing multiple degrees of freedom, the action space is continuous and hence simple reward signals would not provide the desired learning behavior. This project will attempt to train an agent to move towards a target, a soccer ball in this case, and kick the ball towards the goal post.

INTRODUCTION

In this project, we aim to have an agent learn the physical laws and be able to manipulate them to play soccer in a virtual environment. The approach to solve this problem is to use reinforcement learning concepts, such as Policy Gradient variants, to explore and learn the dynamics of the MuJoCo emulator. There has been significant work in making an agent be able to walk, run and perform even more complicated locomotion tasks within a MuJoCo environment. We extend upon this work by introducing a reward-based goal (kicking a soccer ball into a goal post) into the environment and have the agent learn the game mechanics.

We have worked on a MuJoCo “quadruped” agent to learn to play soccer (i.e. Go to the ball ⟶ Move the ball into the goal post). The quadruped agent will try to learn a policy to exhibit the desired behavior achieved through a Reinforcement Learning method. Since the agent must explore a vast environment coupled with the fact that the action space is continuous, we have chosen Policy Gradient methods to train the agent to perform the task. Primarily, we worked on two algorithms: A2C (Advantage Actor Critic) & PPO (Proximal Policy Optimization).

After exploring our several available options, we decided to use MuJoCo with DeepMind’s dm_control package. This package offers a good middle ground between directly interfacing with the low-level MuJoCo interface and using a top-level wrapper (such as OpenAI’s gym, which doesn’t allow you to modify the environment). Because we were aiming to create a problem which is more complex than what OpenAI gym offered, we needed the ability to modify the environment objects and physics; however, we wanted to minimize our interactions with the low-level MuJoCo program as much as possible.

Initially, we started off with the idea of using a humanoid agent playing the sport. However, after seeing the sheer magnitude of the action space associated with that agent, we decided to simplify the problem by moving to a quadruped (think a four-legged spider) agent to play soccer instead.

Our environment included a simplified soccer game, where there is an agent, ball, and goal. Then, we had to experiment with different reinforcement learning methods to have the agent learn to play the game.

RELATED WORK

For the case of playing games, Deep Q-Learning (DQN) [1] can be utilized to solve the problem of high dimensional state spaces. However, when it comes to physical control tasks, DQN fails to handle the huge action spaces. Recently, Policy Gradient methods with deep function approximators such as TRPO (Trust Region Policy Optimization) [2] and DDPG (Deep Deterministic Policy Gradient) [3] have shown promising results in handling these continuous action spaces. Like the DDPG algorithm, TRPO also maintains two network instances for actors and critics each. The first actor - critic pair of networks belongs to the current policy that we want to refine. The second pair belongs to the policy that we last used to collect the samples.

Also, transfer learning is one of the methods existing in deep learning literature that has the ability to use pre-trained models to solve new complex tasks. Adopting transfer learning for reinforcement learning is a difficult task and hence a new method called Hierarchical learning [4] is used for adopting the transfer learning process where the single large task is divided into a series of sequential sub-tasks.

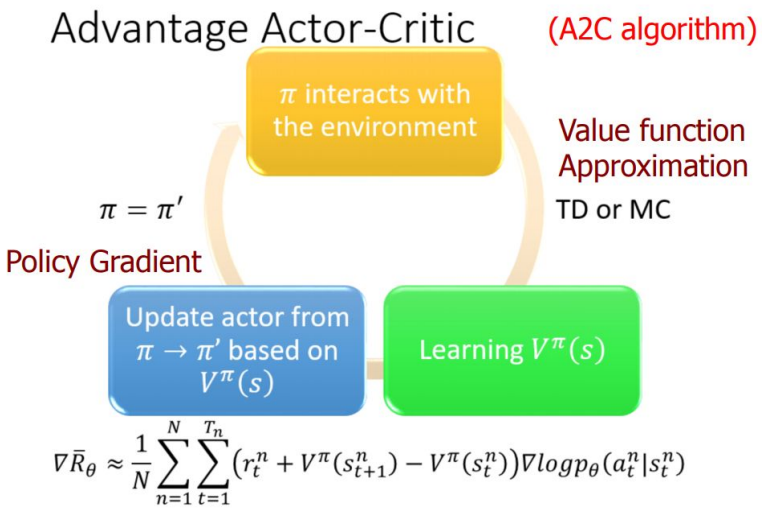
Another related area of work is in evolutionary methods, which are often straightforward to parallelize by distributing fitness evaluations over multiple machines or threads [5]. Such parallel evolutionary approaches have recently been applied to some visual reinforcement learning tasks. In one example, [6] evolved convolutional neural network controllers for the TORCS driving simulator by performing fitness evaluations on 8 CPU cores in parallel.

APPROACH

In this project, we explored two variants of Policy Gradient variants of which one is an Actor-Critic method (A2C) & other is a Policy Optimization method (PPO).

Advantage Actor Critic (A2C)

Policy Gradient methods particularly Monte-carlo methods (REINFORCE) are bound to result in high variance due to inefficient sampling due to lack of value-function. Actor-Critic methods are a new class of Policy gradient methods which have an actor (Policy Network) and a critic (Value network) which work in tandem to give less variant policies. A2C is a synchronous actor-critic method where the agent policy is updated through learnt state-value functions and the policy gradients are updated via an "Advantage" function (which captures the relative performance of an action at a particular state). This method is that it can more effectively use of GPUs, which perform best with large batch sizes.



Parameters considered during the training process include:

Parameter	**Value**
No. of processes	8
No. of forward steps in each process	128
Mini-batch size	4
No. of environment steps	10e6, 20e6

Proximal Policy Optimization (PPO)

PPO is an on-policy, first-order Policy Gradient method which tries to maintain low variance among the new policies it computes every time. It can be used in both discrete and continuous action spaces. It uses a trust region similar to TRPO but is less complex due to its first order nature. PPO has two variants: PPO-Clip & PPO-Penalty. In this project, the variant which we utilized is PPO-Clip with parallelization enabled.

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Parameters considered during training include:

Parameter	**Value**
Learning Rate of Adam Optimizer	0.0025
Discount Factor (Gamma)	0.99
PPO Clip parameter	0.1
Value Loss Coefficient	0.5
No. of processes (parallel)	8
No. of forward steps for each process	128
No. of batches for PPO	4
Entropy-coefficient	0.01
PPO Epochs	4
No. of Environment Steps	10e6, 20e6
Generalized Advantage Estimation Param	0.95

TOOLS & LIBRARIES USED

MuJoCo

MuJoCo stands for **Mu**lti-**Jo**int dynamics with **Co**ntact. It is a physics engine aimed to help researchers create accurate simulations by accounting for joint dynamics and contact forces, leaving the user to only bother about the RL algorithm. MuJoCo makes it possible to scale up computationally-intensive techniques such as optimal control, physically-consistent state estimation, system identification and automated mechanism design, and apply them to complex dynamical systems in contact-rich behaviors. It also has more traditional applications such as testing and validation of control schemes before deployment on physical robots, interactive scientific visualization, virtual environments, animation and gaming. MuJoCo is not open source and hence the first step to work with MuJoCo is to obtain License.

License

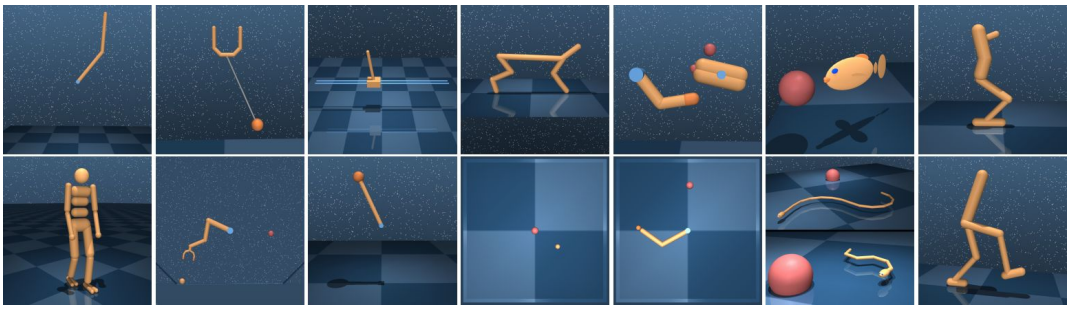
There are various levels of Licensing available given below:

1. **MuJoCo Trial License: 30 days** - This trial license is the most easiest and quickest to obtain and it is valid for 30 days. This license is limited to one per user and we may request up to 3 activation keys for 3 different computers during the 30 day period.

2. **MuJoCo Personal License: 1 year** - There are three types of personal license: Student, Non-commercial and Commercial. While it is free for Students, there's a fee of $500 for non-commercial and $2000 for commercial license.Students get a free license for 1 year and may use software only on one computer ID. Once the student requests the license, it takes approx 5 days to receive the activation key and once the activation key is used on a machine, it can't be used elsewhere. We all requested our individual student personal license for 1 year each.

   Though there are several other licensing options available such as institutional, academic class, academic lab etc, above 2 licenses works best for students who want to explore about MuJoCo software.

DeepMind Control Suite



DeepMind Control Suite is a set of continuous control tasks with a standardized structure and interpretable rewards, intended to serve as performance benchmarks for reinforcement learning agents. The tasks are written in Python and powered by the MuJoCo physics engine, making them easy to use and modify. The OpenAI Gym currently includes a set of continuous control domains that has become the de-facto benchmark in continuous RL. The Control Suite is also a set of tasks for benchmarking continuous RL algorithms, with a few notable differences focusing exclusively on continuous controls. Standardised action, observation and reward structures make benchmarking simple and learning curves easy to interpret. DeepMind offers various domains and agents to perform various tasks and actions. In our project, we extended the pre-defined task set to include a new "soccer" task for the “Quadruped” agent.

OpenAI-Baselines

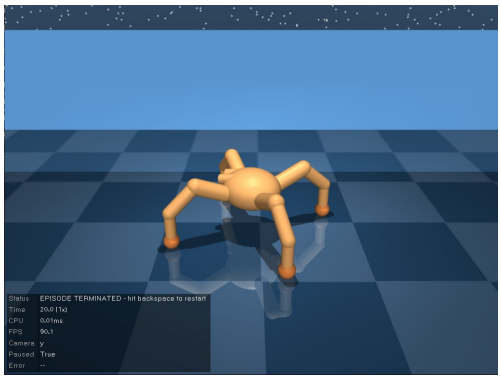
OpenAI Baselines is a set of high-quality implementations of reinforcement learning algorithms. These algorithms will make it easier for the research community to replicate, refine, and identify new ideas, and will create good baselines to build research on top of.

Computational / Hardware Resources

The Hardware Platform chosen to run the RL based task is a Linux (Ubuntu 16.04) system with NVIDIA capabilities. The training process on this system took as long as 12 hours to create our trained model.

ENVIRONMENTAL SETUP

Agent

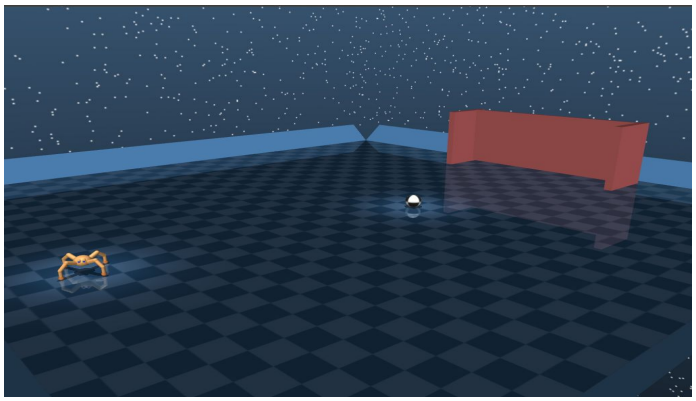


Our agent is a MuJoCo "Quadruped" who is roughly based on the 'ant' model. The agent has a 12-dimensional continuous action space. Main characteristics are:

- 4 DoFs per leg, 1 constraining tendon.
- 3 actuators per leg: 'yaw', 'lift', 'extend'.
- Filtered position actuators with timescale of 100ms.
- Sensors include an IMU, force/torque sensors, and rangefinders.

Environment

The environment is a smooth bounded field containing a target goal post and a soccer ball in which the agent is expected to train and perform the task. Both the ball and the agents are initialized at a random start point within the field. The ball doesn't have initial velocity to move on the field.

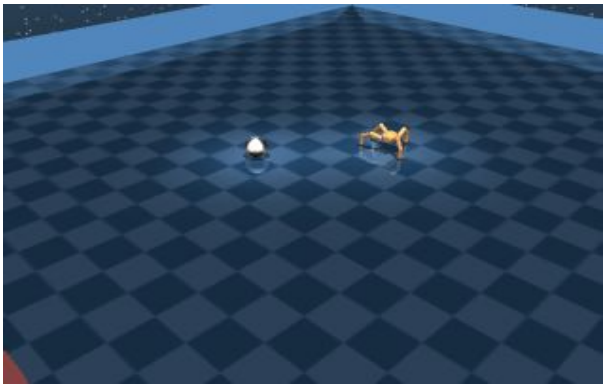


Reward

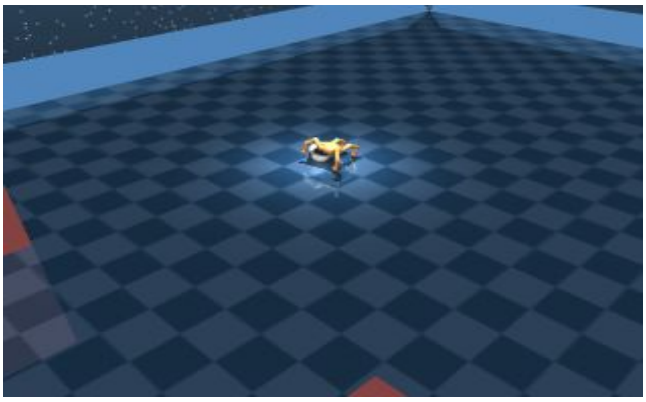
The overall reward the agent obtains, is based on the agent's relative position towards the ball **(Reach_reward)**, ball's relative position from the goal, and the agent's orientation **(Move_reward)**.

Overall reward: (**upright reward**) * (w1* **Reach_reward** + w2* **Move_reward**)

where w1 & w2 are weights assigned to each reward , *w1 = w2 = 0.5* chosen in our case.



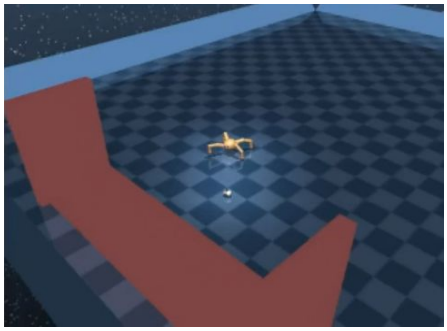
Agent trying to reach the ball



Agent trying to move the ball towards the goal

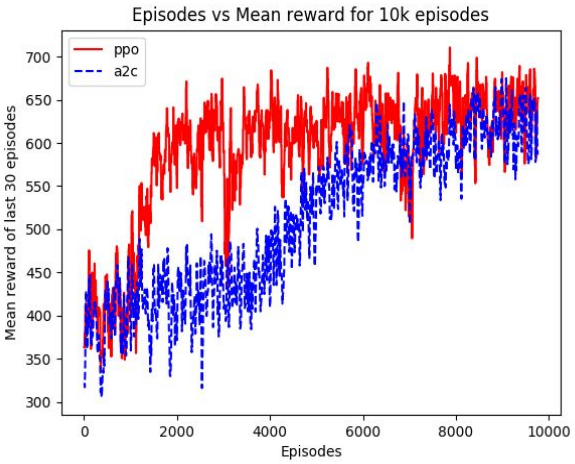
Visualized during some random model sample collected during the training phase

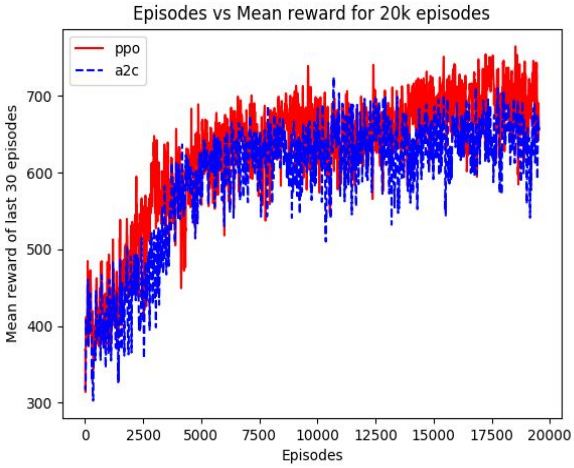
RESULTS

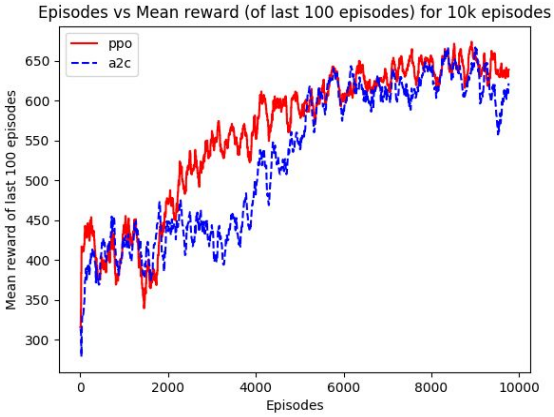


Agent trained with PPO trying to move the ball towards the goal post

Graphical plots of the learning curves were observed during the training process in A2C & PPO agents for 10K and 20K episodes spanning 1 Million & 2 Million timesteps approximately. From the plots, we observe that PPO learns marginally at a better rate than A2C. Also, the trained agent with PPO exhibits the desired task behaviour when compared to A2C.







With increasing deque size for mean reward calculation, the plot of episodes vs mean reward becomes less noisy and shows less variance but there is no significant improvement observed in the model performance.

CONCLUSION

The reward function formulated & considered for our required objective seems to produce very satisfactory results and induce the required task behaviour in the quadruped agent. PPO trained agent gives relatively better performance than its A2C counterpart in task fulfillment. PPO agent learnt the task marginally at a better rate. There is intelligence observed in both the cases, i.e. there is motion observed in the agent that maintains a trajectory towards the ball & then towards the goal. However, the A2C agent seems to fail multiple times in moving the ball towards the goal while PPO agent succeeds in achieving the objective. Below is our project GitHub link.

https://github.com/kvsaivineeth/Soccer-Playing-Quadruped-Agent

FUTURE WORK

The future work is expected to focus on Agent's robustness and adaptability to dynamic environment scenarios. The environment considered in this project is static with no external factors influencing the agent's behaviour. Also the agent is not currently in a state to recover from failures, like falling down, correct the trajectory course towards the goal post. Currently, any intermediate failure within an episode will degrade the agent performance in that episode. Also, future work is expected to consider a more complicated reward function taking into account the agent's dynamics as well as the current reward function seem to produce an unplanned locomotion in the agent in either cases.

REFERENCES

1. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Playing Atari With Deep Reinforcement Learning, 2013
2. J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," 2015
3. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa,D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", 2015
4. M.E.Taylor and P.Stone,"Transfer learning for reinforcement learning domains: A survey, "J. Mach. Learn.Res., vol. 10, pp. 1633–1685, Dec. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755839
5. Marco Tomassini, 1999 , "Parallel and Distributed Evolutionary Algorithms: A Review"
6. Jan Koutník, Jürgen Schmidhuber, Faustino Gomez, 2014, "Evolving Deep Unsupervised Convolutional Networks for Vision-Based Reinforcement Learning"
7. Deep Mind Control Suite Tech Report, 2017 : https://arxiv.org/abs/1801.00690
8. Deep Mind Control Suite, 2017 : https://github.com/deepmind/dm_control
9. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, "Proximal Policy Optimization Algorithms", https://arxiv.org/abs/1707.06347
10. Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, Jimmy Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation", https://arxiv.org/abs/1708.05144
11. Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning", https://arxiv.org/abs/1602.01783
12. Ilya Kostrikov, PyTorch Implementations of Reinforcement Learning Algorithms, 2018, https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail
13. Dhariwal, Prafulla and Hesse, Christopher and Klimov, Oleg and Nichol, Alex and Plappert, Matthias and Radford, Alec and Schulman, John and Sidor, Szymon and Wu, Yuhuai and Zhokhov, Peter, OpenAI Baselines, 2017, https://github.com/openai/baselines
14. OpenAI Baselines Blog: https://openai.com/blog/baselines-acktr-a2c/
15. "Mujoco." [Online]. Available: http://www.mujoco.org/book/