Hierarchical Reinforcement Learning for Humanoids

Arpit Kapoor
RA1511003010744

Abhishek Warrier
RA1511003010532

Under the guidance of

Dr T. Sujithra

Department of Computer Science and Engineering
SRM Institute of Science and Technology

13th April 2019

Agenda

Abstract

Humanoid Robots have many degrees of freedom with continuous state and action spaces and thus controlling humanoid robots is a difficult problem that requires extensive domain knowledge. The extension of Deep Reinforcement Learning to continuous spaces has enabled such complex behaviour to be learned directly by specifying the appropriate reward function and recent research has even enabled learning from sparse rewards. But these behaviours are usually limited to simpler tasks such as walking. In this dissertation, the paradigm of Hierarchical Reinforcement Learning is explored where separately trained high level policies are combined with low level policies to learn complex tasks.

Introduction
Reinforcement Learning

Reinforcement learning is one of the paradigms of machine learning, alongside supervised learning and unsupervised learning. It is much more focused on goal directed learning from interaction than are other approaches to machine learning. Reinforcement learning is learning what to do i.e. how to map situations to actions so as to maximize a numerical reward signal. It was originally considered as an extension of optimal control theory.

But in recent years, there have been many breakthroughs in novel architectures and computational power which has led to the advent of Deep Learning. The combination of deep learning with traditional reinforcement learning techniques is referred to as Deep Reinforcement Learning.

Introduction
Reinforcement Learning

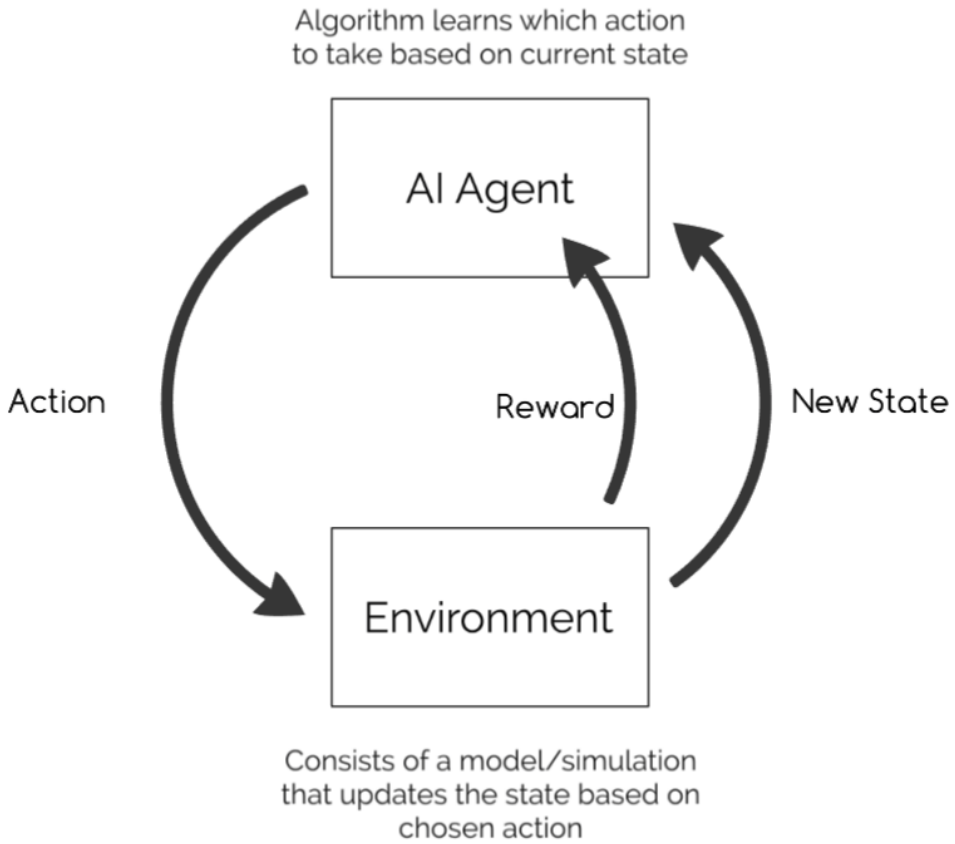


Figure 1: The Reinforcement Learning Paradigm

Introduction
Hierarchical RL

Recent techniques in RL have made training agents in continuous spaces with sparse rewards a viable option. But RL agents still face difficulty in learning complex tasks, especially with long time horizons(duration).

One possible approach to tackle this is Hierarchical Reinforcement Learning (HRL) in which rather than learning a single policy, an agent learns an entire hierarchy of policies. More specifically a complex task is divided into a hierarchy of sub tasks and a corresponding policy is learned for each sub task. Learning each of these sub-policies is much more simpler and feasible rather than learning to solve the problem as a whole.

Literature Survey

Deep Reinforcement Learning Techniques

SNo.	Objective	Main Contribution	Methodology	Performance Measure	Remarks
[1]	To adapt ideas from Deep Q-Learning to introduce an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces.	The authors introduce an algorithm called DDPG (Deep Deterministic Policy Gradient) which can be used to achieve continuous control with deep reinforcement learning.	DDPG is an Actor-Critic approach based on Deterministic Policy Gradients (DPG).	Success Rate on Gym + MuJoCo environments.	This paper is quite pivotal as it allowed state of the art techniques to be extended to continuous action-spaces which makes it quite useful for application in Robotics.
[2]	To enable sample-efficient learning from rewards which are sparse and binary and therefore avoid the need for complicated reward engineering.	Introduces Hindsight Experience Replay (HER). It can be used along with existing state of the art off-policy RL algorithms, which not only speeds up the process of learning, also enables training possible in challenging environments with no reward engineering.	Simple Idea. Many RL algorithms make use of a replay buffer to store experiences/memories which are later played back to update the network. HER adds additional goals for replay.	Success rate in MuJoCo Environments	Intuitive, novel and state of the art. OpenAI released a new set of Robotics Environments for Gym this year and most of them fail to learn with DDPG but learn with DDPG+HER.
[3]	Improve performance on high-dimensional discrete/continuous action environment.	The author introduces a methodology to effectively combine parameter space noise with off the shelf DeepRL algorithms for both on/off policy methods.	To achieve structured exploration, the policy is sampled from a set of policies by applying additive Gaussian noise to the parameter vector of the current policy. This perturbed policy is sampled at the beginning of each episode and kept fixed	Success Rate in ALE	This technique is quite useful in environments with sparse-rewards and can be used to along with HER to achieve better performance.

Literature Survey

Deep Reinforcement Learning Techniques

SNo.	Objective	Main Contribution	Methodology	Performance Measure	Remarks
[4]	To learn in challenging domains using Hybrid Rewards	In domains where the optimal value function cannot easily be reduced to a low-dimensional representation, learning can be very slow and unstable.Contributes towards tackling such challenging domains, by proposing a new method, called Hybrid Reward Architecture (HRA).	The main strategy for constructing an easy-to-learn value function is to decompose the reward function of the environment into n different reward functions. Each of them is assigned a separate reinforcement-learning agent. All these agents can learn in parallel on the same sample sequence by using off-policy learning.	Score on ALE	Most humanoid robots have 18-20 motors, and each motor can be assigned a different agent. The aggregator can then take inputs from each of these motor agents and produce a stable control algorithm.
[5]	To enable learning of complex locomotion behaviors with simple reward functions.	In this paper, agents are trained in rich, diverse environments which leads to robust behaviors that generalize to across a variety of tasks using a simple reward function based on forward progress. Also, a distributed implementation of PPO is introduced to achieve better performance.	The Distributed PPO (DPPO) algorithm makes a few augmentations which include normalization of inputs and rewards as well as an additional term in the loss that penalizes large violations of the trust region constraint.	Success Rate in Mujoco Environments.	This is a seminal paper that demonstrates how complex locomotion behavior can be learned without hand-engineering a reward function and thus is quite useful.

Literature Survey

Reinforcement Learning Application to humanoid robots

SNo.	Objective	Main Contribution	Methodology	Performance Measure	Remarks
[6]	To introduce sample-efficient model-based RL as a valid approach for Robot Learning	Model-free RL algorithms are computationally cheap, but ignore the dynamics of the world, and need a lot of experience. Suggests, Model-based methods are thus more apt for robot learning and introduces a new model-based RL algorithm, Reinforcement Learning with Decision Trees (RL-DT).	The RL-DT is a novel algorithm that incorporates generalization into its model learning. Its two main features are that it uses decision trees to generalize the effects of actions across states, and that it has explicit exploration and exploitation modes.	1) Cumulative Reward and 2) Success rate in real world experiment where robot has to kick a ball	This is a rather old paper, but makes an argument for model-based RL in contrast to the currently popular model-free RL and thus could be an alternative approach that may be incorporated in the project.
[7]	To successfully apply reinforcement learning to humanoid robots.	The paper presents a probabilistic reinforcement learning approach, which is derived from the framework of stochastic optimal control and path integrals. The algorithm, called Policy Improvement with Path Integrals (PI^2).	The experiment is conducted with the Simulation Lab software. The main method is to create desired trajectories for a motor task using Dynamic Movement Primitives (DMP)s. The joint trajectories are represented by a 34-dimensional DMP. The duration of the movement is 3.0s. The initial 'movement' of the robot before learning is that it stands still at its	Trajectory cost in simulated environment	It is a rather old paper but demonstrates how a very basic RL approach can be taken, but the main drawback this requires a lot of domain knowledge about humanoid robotics.

Literature Survey

Transfer of policies from simulation to hardware

SNo.	Objective	Main Contribution	Methodology	Performance Measure	Remarks
[8]	To bridge the reality gap between physics simulators and the real world.	Discrepancies between simulators and real world make transferring behaviors from simulation challenging. Thus, introduces a method called Domain Randomization to address this challenge.	Main Concept: if the variability in simulation is significant enough, models trained in simulation will generalize to the real world with no additional training. This is implemented in practice by randomizing various aspects of the simulation for each iteration.	Object detection accuracy + Grasping success rate	This technique would play a major role for transferring the learnt policy from our project to the actual real-world humanoid robot.
[9]	To train better policies that generalize/transfer more easily to the real world.	Introduces Asymmetric Actor Critic that exploits the full state observability in the simulator to train better policies which only need partial observations as input.	The method builds upon actor-critic algorithms. For the critic, the exact position for an object is given whereas for the actor only an RGB+Depth (RGBD) Image is given. This is further combined with Domain Randomization and Hindsight Experience Replay (HER).	Success rate in MuJoCo Environments.	This technique allows training of polices that both trains fast and generalize to the real world and it also incorporates other techniques such as DDPG + Domain Randomization + HER and it is the current state of the art for continuous control tasks.

Literature Survey

Transfer of policies from simulation to hardware

SNo.	Objective	Main Contribution	Methodology	Performance Measure	Remarks
[10]	Extend upon the Domain Randomization technique and further bridge the reality gap.	Introduces methodology of randomizing the dynamics of the simulator during training, which helps to develop policies that are capable of adapting to very different dynamics, including ones that differ significantly from the dynamics on which the policies were trained.	At the start of each episode, a random set of dynamics parameters are sampled and held fixed for the duration of the episode. The parameters include Mass of each link, damping of each joint etc., for a total of 9 parameters. An extension of DDPG, called RDPG is then used to train along with HER.	Success Rate in Simulation and Real-world.	The paper is an extension to Domain Randomization for generating policies that are more robust and will be useful if we try to generalize our policy to the real world.

Proposed System

The aim of this dissertation is to explore a variation of Hierarchical RL in which two distinct policies, *PolicyA* and *PolicyB* will be trained independently. The main objective is to explore ways to compose *PolicyA* and *PolicyB* to achieve a new *PolicyC* that encapsulates the behaviour/functionality of both these policies.

To motivate this, we will tackle the specific problem of **teaching a humanoid to solve a maze**. More precisely,

PolicyA will teach a humanoid to walk in a specific direction.
PolicyB will be a general maze solving agent.
Objective : Obtain a new *PolicyC* that enables a humanoid to solve a maze by composing *PolicyA* and *PolicyB*.

Modules

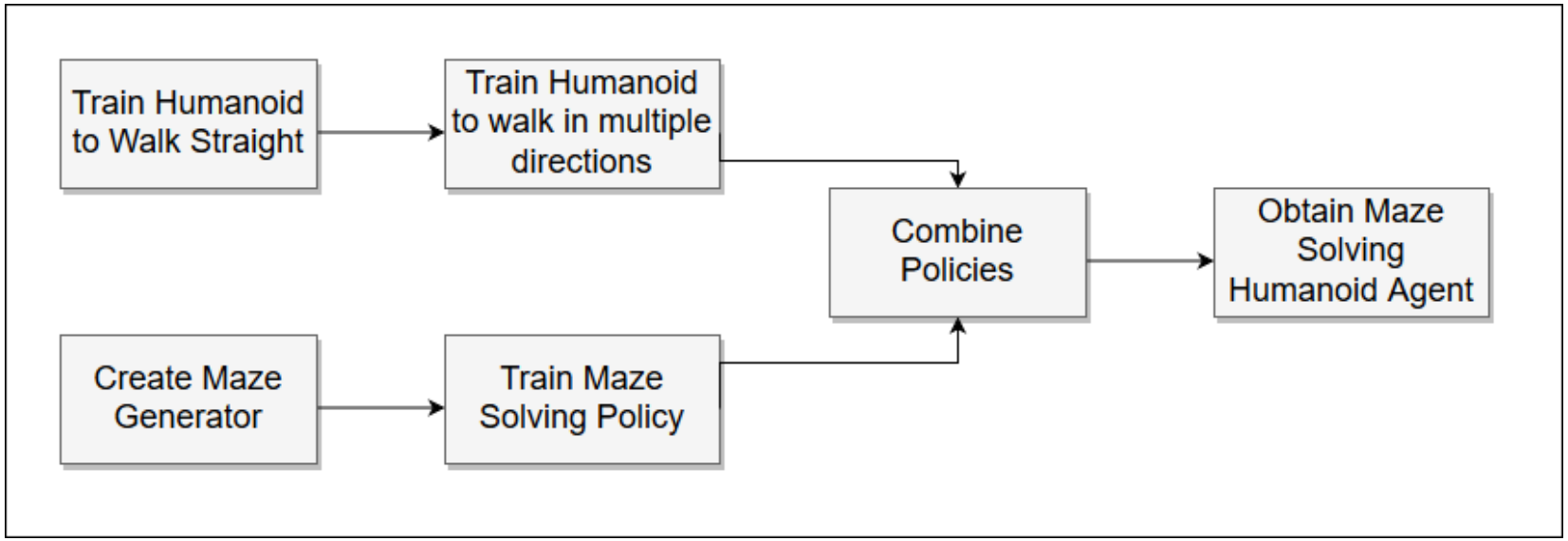
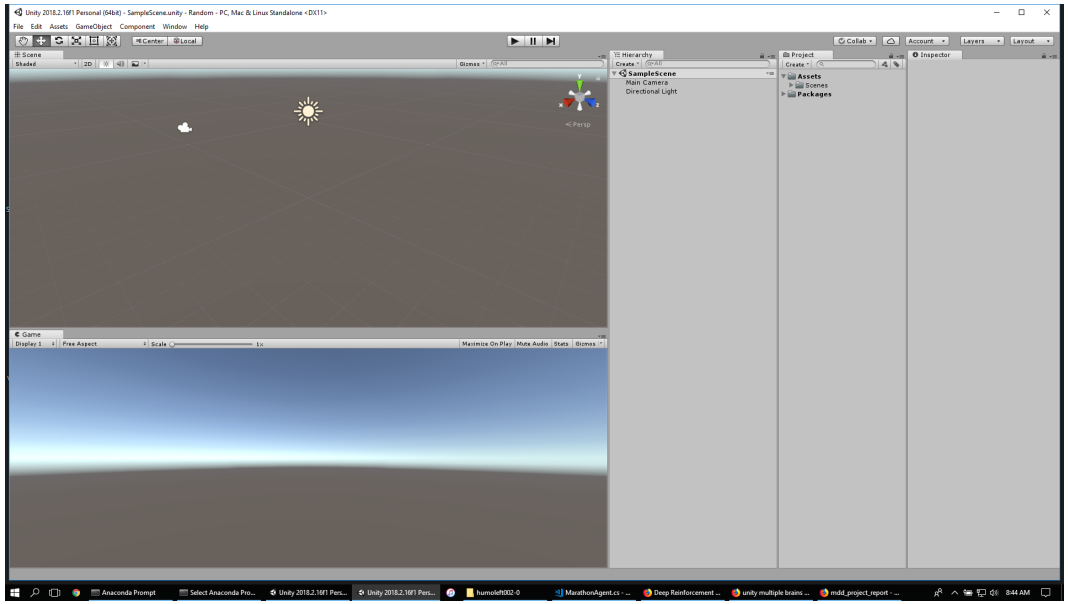


Figure 2: Modules

Software
Simulation Environment: Unity

Unity will be used. Unity is a 3D game engine that implements the Nvidia PhysX SDK for realtime accurate physics. Although it is traditionally used for designing games, Unity has recently released the **ML-Agents** Library which allows it to be used for various Reinforcement Learning Applications. It also comes with **MarathonEnvs** which is a port of DeepMind Environments including the DeepMind Humanoid to Unity.



Architecture Diagram

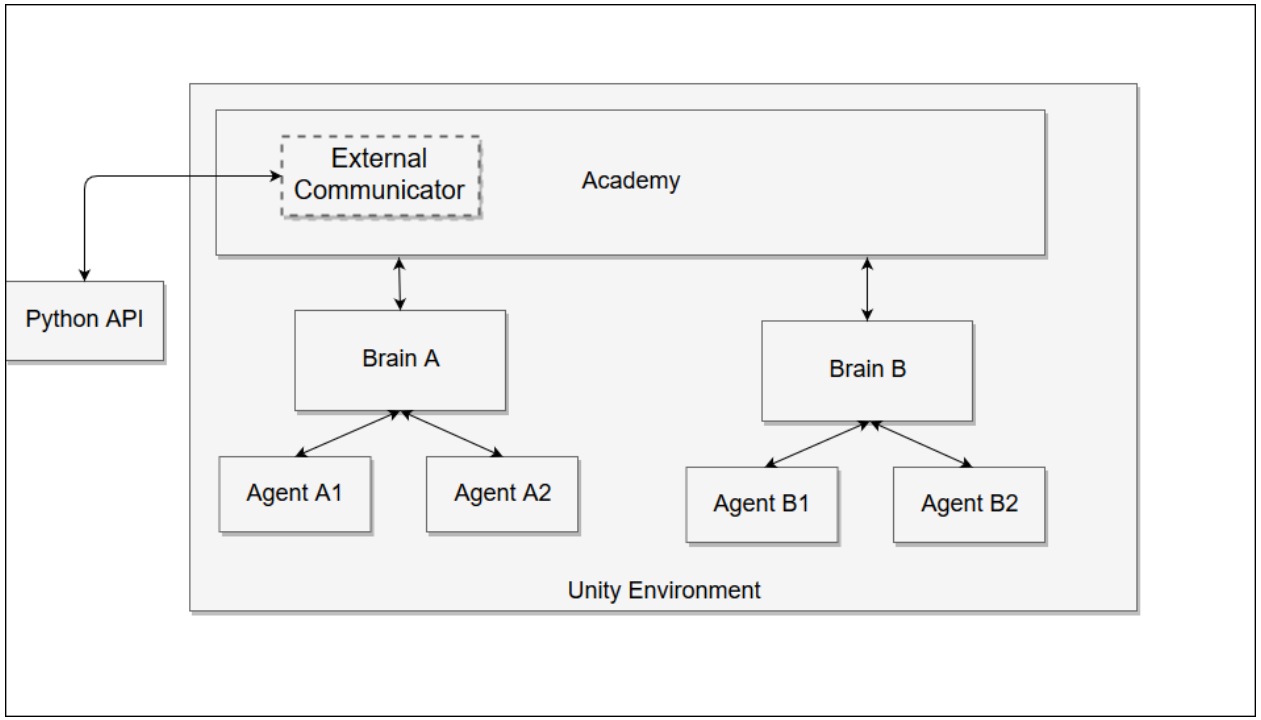


Figure 3: System Architecture

ER Diagram

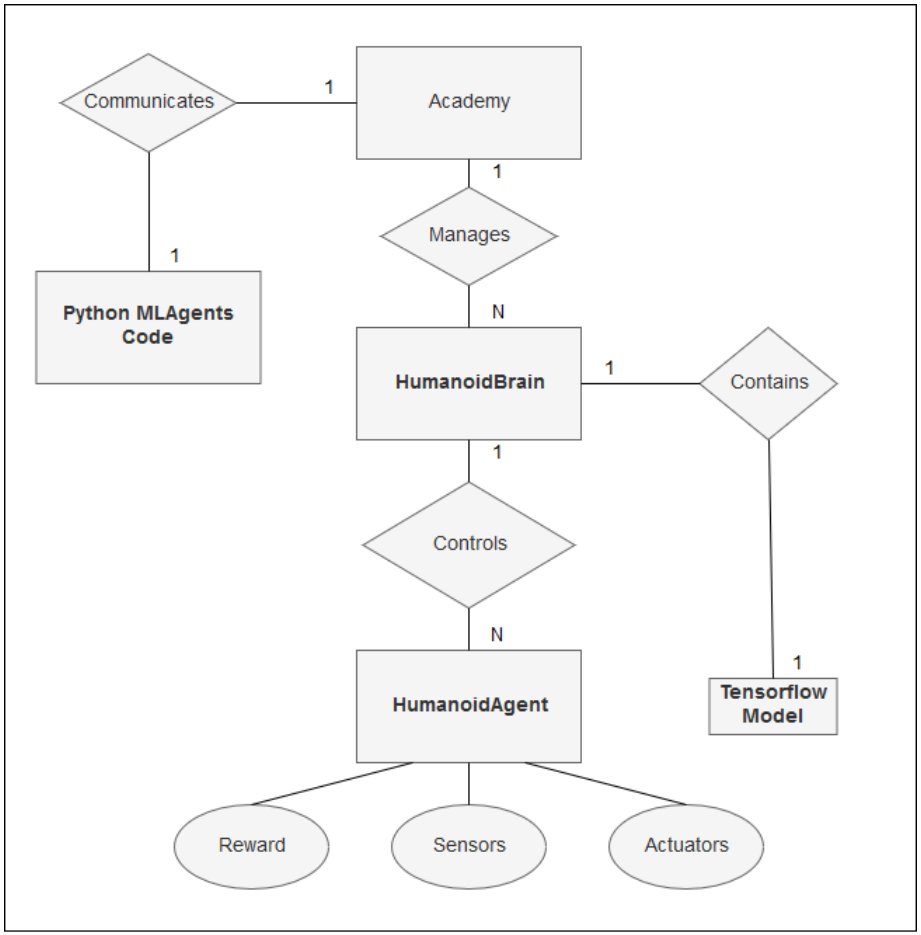


Figure 4: ER Diagram of the system

Algorithm

Proximal Policy Optimization (PPO) Loss Function

Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, so $r_{old} = 1$. Trust Region Policy Optimization (TRPO) maximizes a "surrogate" objective

$$L_t{}^{CPI}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\tilde{A}_t\right] = \hat{E}_t\left[r_t(\theta)\tilde{A}_t\right]$$

In PPO, the loss function is defined as follows -

$$L_t{}^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\tilde{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\tilde{A}_t)]$$

Algorithm

Proximal Policy Optimization (PPO)

for *iteration=1,2...* **do**
  for *actor=1,2...* **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
    Compute Advantage estimates A_1,....,A_T
  end
  Optimize surrogate loss L wrt θ
  $\theta_{old} \leftarrow \theta$
end

Modules

Module 1: Forward Motion

The Marathon Envs default reward function is defined as follows -

$$Reward = Velocity + UprightBonus + ForwardBonus - LimitPenalites$$

When trained with such a reward function for 1M steps, as expected the agent runs forward -

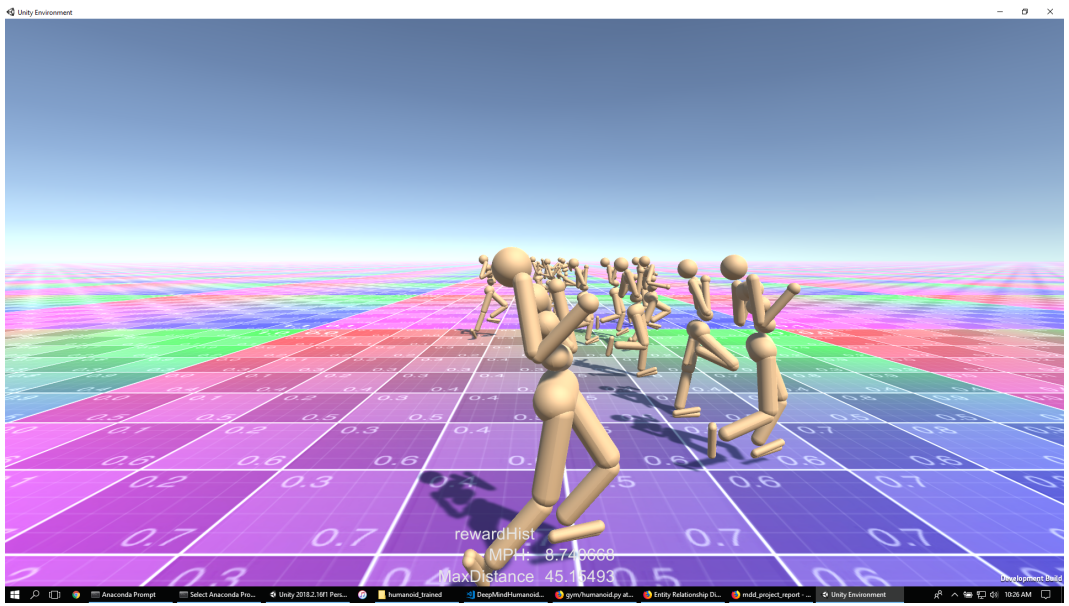


Figure 5: Forward Motion

Modules

Module 2: Directional Motion

To make it move towards a particular direction, the reward function needs to be tweaked to elicit the required response. To move the agent to the left, the reward function was modified to -

$$Reward = Velocity + UprightBonus + \textbf{LeftBonus} - LimitPenalites$$

But instead of moving towards the left, it only makes the torso twist to the left, while still moving forward as seen in the figure below -

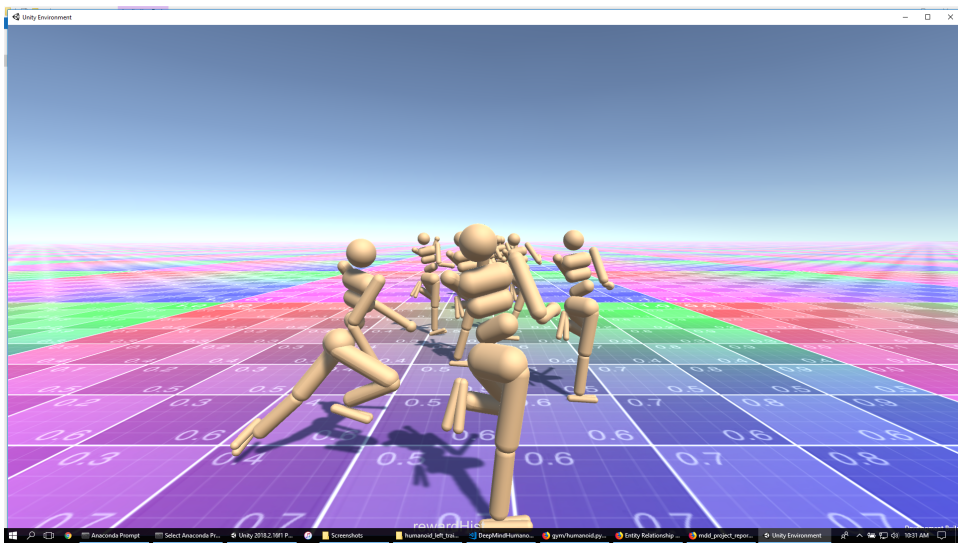


Figure 6: Forward Motion while Twisting Left

Modules

Module 2: Directional Motion

As shown by the results, the primary objective at this point is to enable the humanoid to be able to walk in any direction. This is important as the humanoid eventually needs the capability to walk in multiple directions to solve a maze. As in the Reinforcement Learning, the agent behaviour is driven solely by the reward function, the primary challenge is to correctly engineer a reward function that will elicit the desired behaviour.
The solution is to use another simulator called Roboschool, which has more realistic physics. Roboschool has a standard environment known as RoboschoolHumanoidFlagrun that is designed to walk towards a particular flag. The reward function here is

$$Reward = Alive + Electricity + Limits + FeetContact + Progress$$

Modules

Module 3: Maze Generation

Humanoid needs the ability to navigate any randomly generated maze and thus a random maze generation module needs to be implemented in Unity. Many maze generation algorithms exist which provide varying degrees of performance and we decided to use the Growing Tree Algorithm, which is one of the more popular algorithms.

Modules

Module 3: Maze Generation

Let there be a grid of dimension h, w
  Let C be an empty list
  Add one random cell to C
  while *C is not empty* **do**
    Choose a cell, x from C
      Carve a passage to any unvisited neighbor of x
      Add this neighbour to C
      if *No unvisited Neighbours* **then**
        Remove x from C
    end
end

Algorithm 1: Growing Tree

Modules

Module 3: Maze Generation

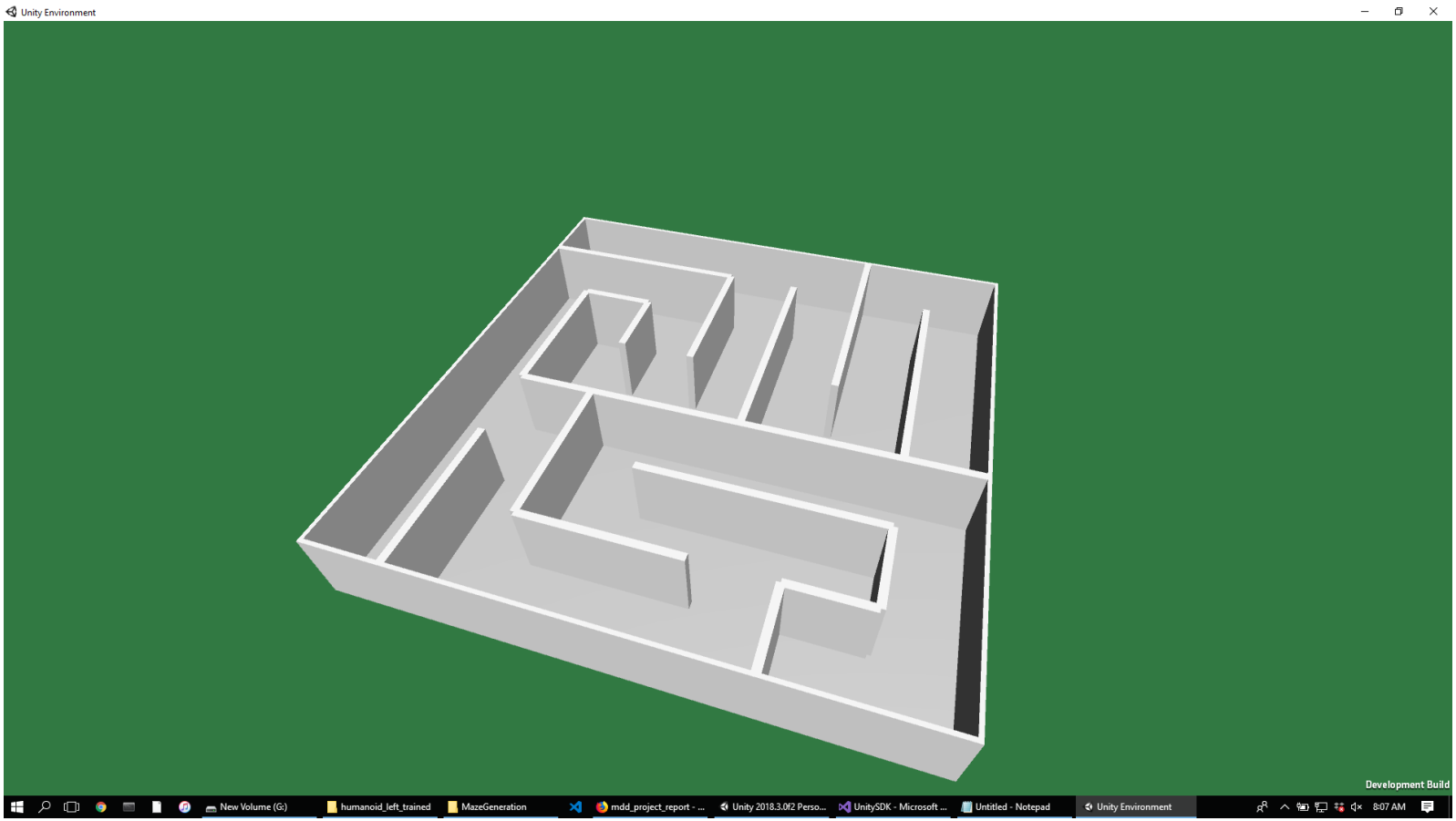


Figure 7: Randomly Generated Maze

Modules

Module 4: Maze Navigation (continuous)

The main objective of this module is to train a continuous maze solving agent. To achieve this, we make use of Proximal Policy Optimization (PPO) Algorithm, discussed earlier, to train an agent with continuous action space. For the sake of simplicity, we take a red sphere as our agent which navigates the randomly generated maze to reach target (green cube). We make use of a dense reward function which is defined as -

$$Reward = -Distance(Target, Agent) - TimePenalty$$

where the value $Distance(Target, Agent)$ is equal to the Euclidean distance between Target and the Agent.
It is observed that the model training in the continuous state/action space can be a difficult task and require longer training period. Thus, we shift to Discrete state/action space.

Modules

Module 4: Maze Navigation (continuous)

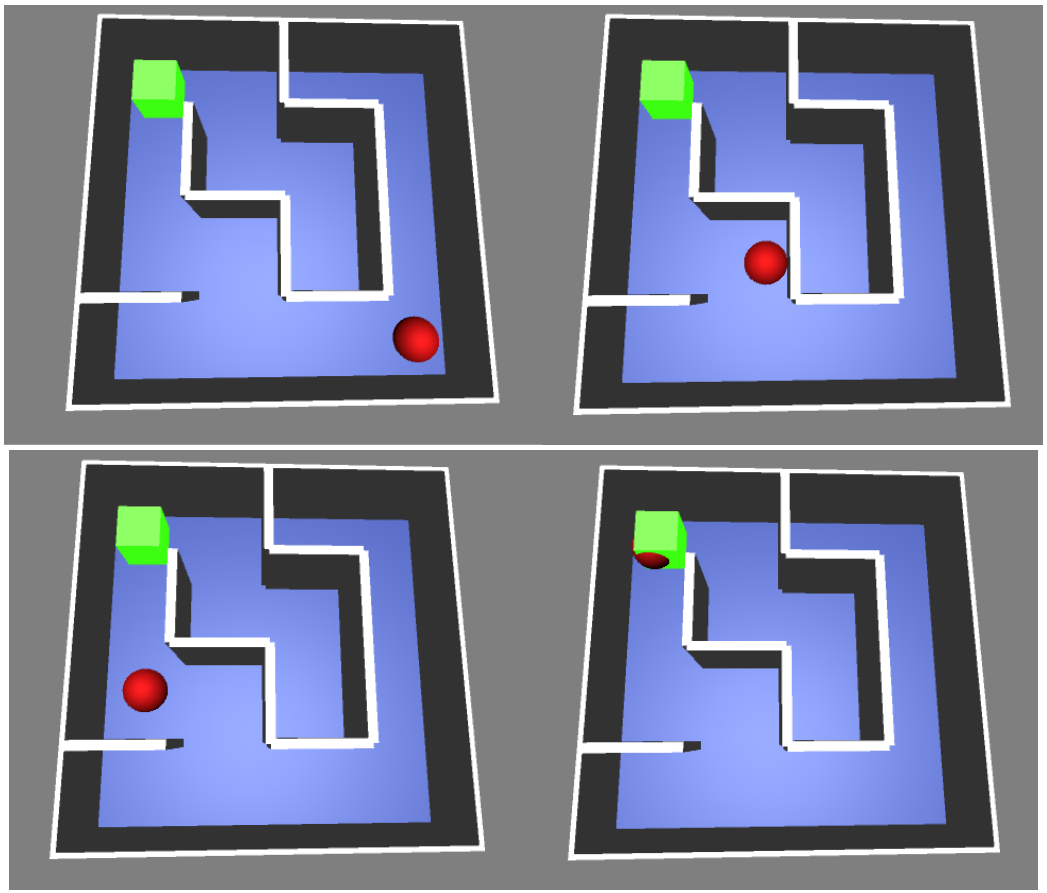


Figure 8: Agent solving the maze (continuous)

Modules

Module 4: Maze Navigation (Discrete)

We create a discrete maze solver agent whose state space consists of the individual cells in the maze while it's action space consists of motion in 4 possible directions (North, South, East and West). We design the reward function for the agent as follows:
At each time step t,

if $(distanceToTarget < thresh)$ then

$Reward_t = 1.0$

else

$Reward_t = -0.1/(mazeSize * mazeSize))$

where, the maze consists of mazeSize x mazeSize cells.
We make use of Q Learning algorithm to train the maze solver.

Modules

Module 4: Maze Navigation (discrete)

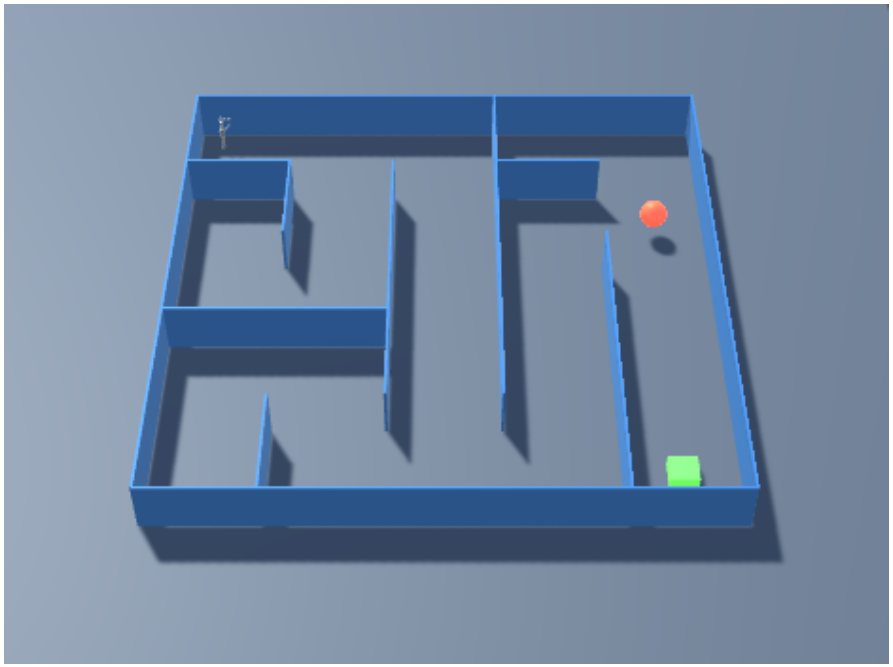


Figure 9: Agent solving the maze (discrete)

Modules

Module 5: Combination

The final part is using the maze solver policy as a high level selector policy and using it to control the direction motion of a quad/humanoid. This is done as follows -
1) Initially, the humanoid is at (0,0) and the maze agent is at (1,0)
2) The position of the maze agent is given to the RoboschoolHumanoidFlagrun policy.
3) The humanoid starts approaching toward the maze agent and the distance is constantly checked.
4) When the distance goes below a threshold, the current state is passed to the maze agent policy that outputs a new direction to move in and updates the maze agent position.
5) The humanoid now starts moving to the updated position.
6) In this manner, the higher level maze policy slowly directs the lower level humanoid to solving the maze.

Modules

Module 5: Combination

Let π_h, π_l denote the higher level and lower level trained policies respectively
Let S_h be the current state of the higher level policy
Let g_h be the final goal
while S_h *is not* g_h **do**
  New state $S_h^{'} = \pi_h(a_h|S_h)$
  Set goal of low level policy $g_l = S_h^{'}$
  Let S_l be the current state of the low level policy
  while S_l *is not* g_l **do**
    $S_l^{'} = \pi_l(a_l|S_l)$
    $S_l = S_l^{'}$
  end
end

Algorithm 2: Hierarchical Policy Integration

Modules

Module 5: Combination

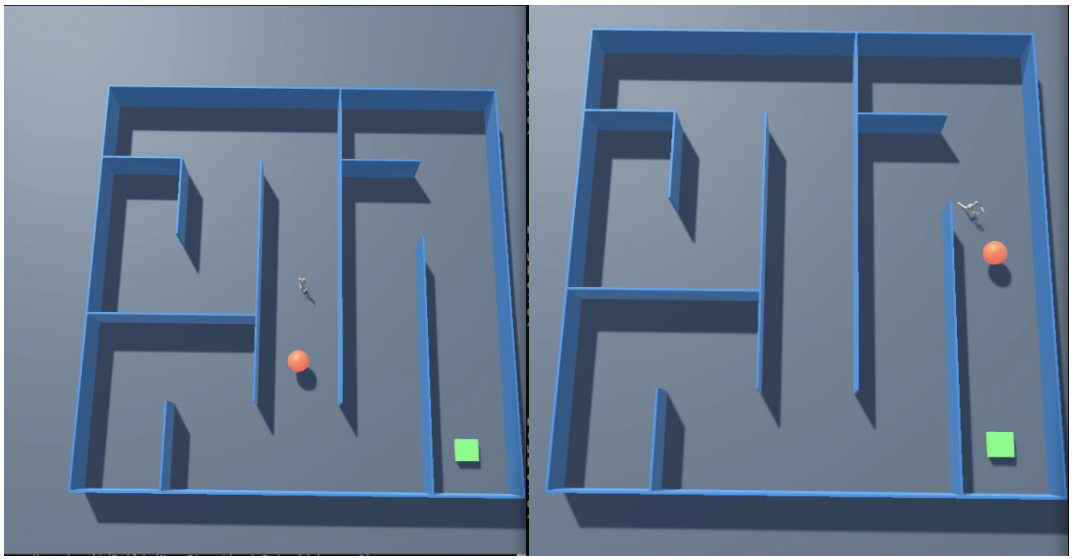


Figure 10: Maze Agent Combination

Results

1) We observed that the policies are effectively combined and the hierarchical behaviour is as intended.
2) Switching from continuous maze to a discrete maze env does radically improve performance.
3) Both of these continuous/discrete versions of MLAgents-PPO is outperformed by our q-learning implementation.
4) But the overall performance and success rate varies widely as the individual policies have not been trained to perfection and it is only occasionally able to solve the maze so there is still scope for environment.

Results

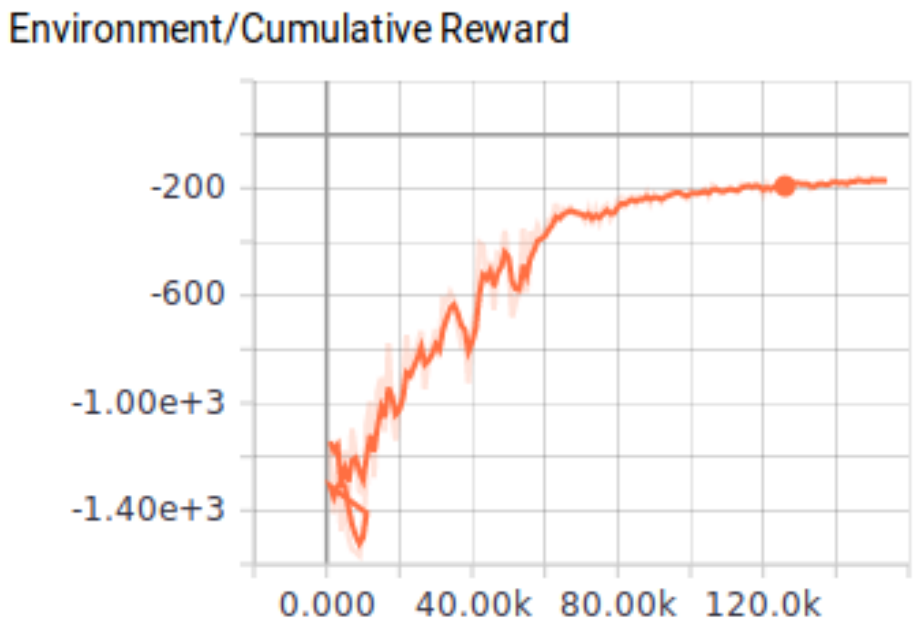


Figure 11: Cumulative Reward vs Number of Episodes using PPO on continuous state-action space

Results

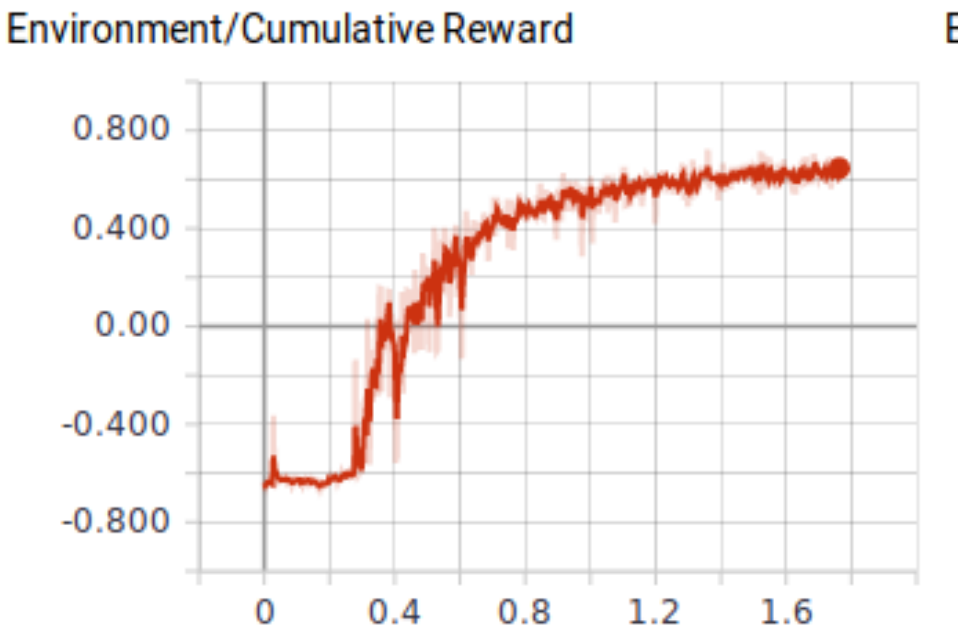


Figure 12: Cumulative Reward vs Number of Episodes for PPO on discrete state-action space

Results

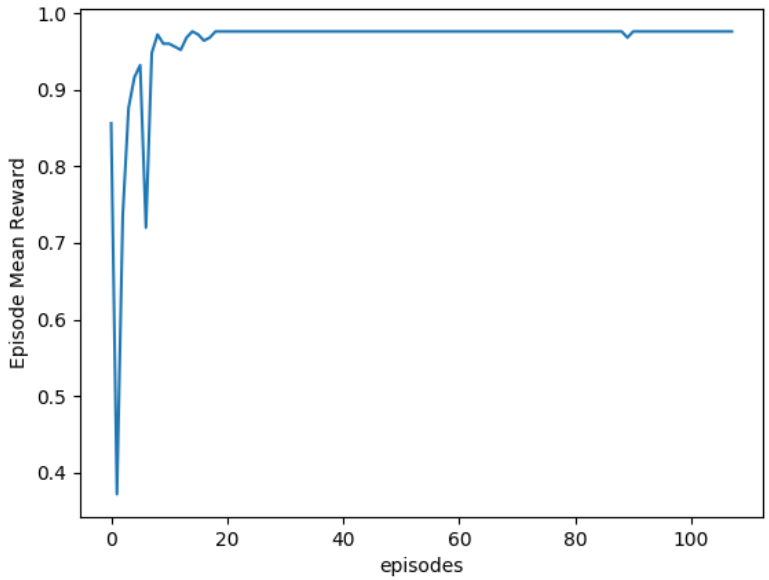


Figure 13: Cumulative Reward vs Number of Episodes for Q-Learning on Discrete state-action space

Results

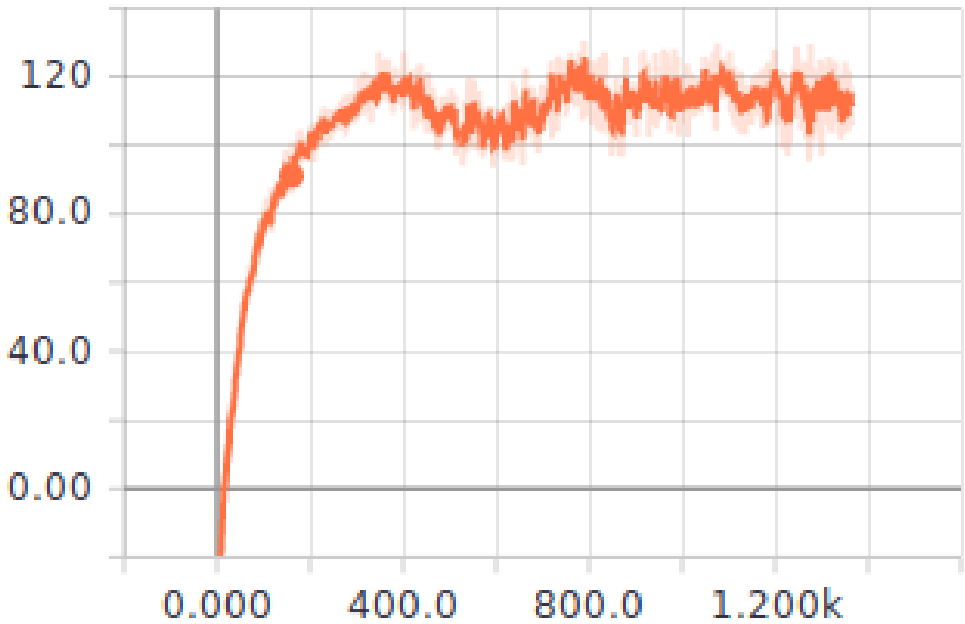


Figure 14: Cumulative Reward vs Number of Episodes for HumanoidFlagRun

Conclusion

The paper has demonstrated how a continuous high level task can be tackled using a hierarchical divide and conquer approach. The fact that the lower level policy is trained separately makes the overall task modular in nature. This means that the lower level policy could simply be replaced by an Ant/Hopper/Walker or any of the other continuous control agents to obtain respective maze solving agents. But yet in an ideal scenario, it would be better if efficient algorithms are developed where the entire hierarchy can be learned together at once.
Although in this paper, a humanoid was trained to solve a maze, it was not a humanoid robot. More precisely, the humanoids which are generally used in RL for benchmarking, are abstract dummy humanoid that do not correspond to any existing real humanoid robots. So the next logical task would be to create and experiment with simulations of popular humanoid robots such as Atlas, Darwin-OP, Nao etc. and study/analyze their relative performance.

References

[1] 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.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel and W. Zaremba, "Hindsight experience replay", in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

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

[4] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes and J. Tsang, "Hybrid reward architecture for reinforcement learning", in *Advances in Neural Information Processing Systems*, 2017, pp. 5392–5402.

References

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

[6] T. Hester, M. Quinlan and P. Stone, "Generalized model learning for reinforcement learning on a humanoid robot", in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, 2010, pp. 2369–2374.

[7] F. Stulp, J. Buchli, E. Theodorou and S. Schaal, "Reinforcement learning of full-body humanoid motor skills", in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, IEEE, 2010, pp. 405–410.

References

[8] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world", in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 23–30.

[9] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba and P. Abbeel, "Asymmetric actor critic for image-based robot learning", *arXiv preprint arXiv:1710.06542*, 2017.

[10] X. B. Peng, M. Andrychowicz, W. Zaremba and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization", *arXiv preprint arXiv:1710.06537*, 2017.