ME5406: Deep Learning for Robotics

Project for Part II: Quadrupedal Robot Standing-up using Deep Reinforcement Learning

Garen Haddeler

A0208496U

e0444217@u.nus.edu

18 November 2020

Contents

1 Introduction

Quadrupedal robots can be widely used in many robotic applications. Since they can overcome larger obstacles than their body. Showcase of standing in rear-up position can give major advantage to presenters since it has an artistic posture and shows costumers that their robot can take robust actions. Traditionally, in robotic applications, four legged robots can be stand-up by giving pre- defined joint torques values in time stamp and by using whole body kinematic controller, robot can control center of mass to balance on the equilibrium point. Controlling multiple joints to stand-up and self-balance is a complex control problem due to high dimensionality and it can be hard to solve with traditional control methods. To overcome this issue, Reinforcement algorithms can be proposed since agents can overcome non-linearity of system by learning from the environment and model.

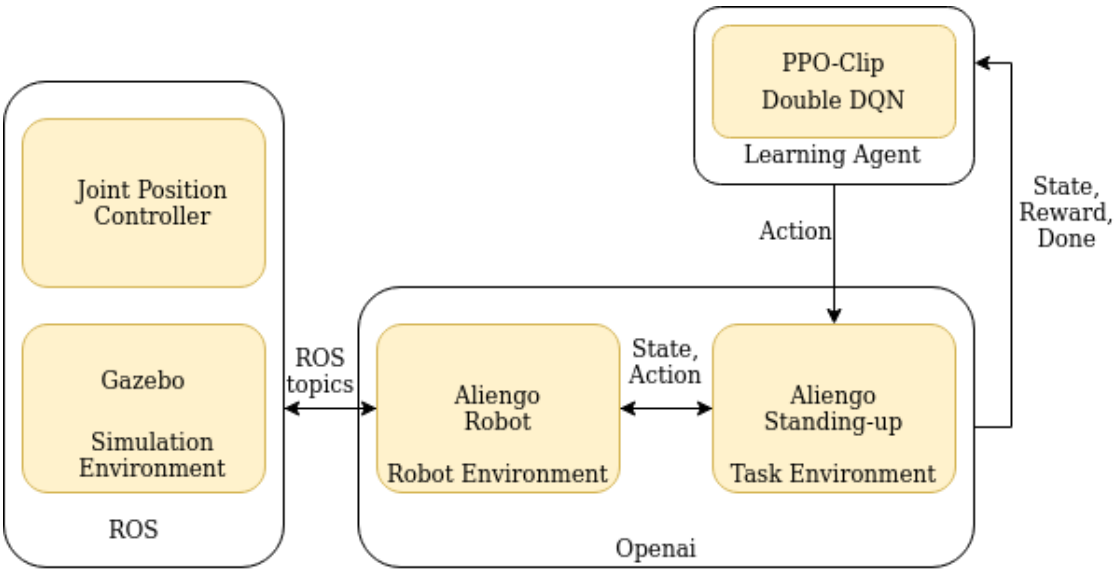

Figure 1: General Structure of Our Learning Framework

2 RL Algorithms and Their Implementations

In this project, we used Robot Operating System (ROS) and ROS-based libraries to simulate legged robot dynamics and control joints realistically (by limiting joint

torques and positions). Figure 1, shows general framework of our learning platform. We connected Openai platform with ROS-Gazebo simulator environment by defining "robot" and "task" blocks. We used Aliengo quadrupedal robot as four legged robot model. Thereafter, we implemented the learning agent as Proximal Policy Optimization (PPO-Clip) and define our robot's states,actions and rewards according to achieve desired standing-up task. Additionally, we implement Double Deep Q Network (DDQN) RL agent to compare results that we obtained from PPO-Clip agent.

2.1 Proximal Policy Optimization (PPO) Method

PPO is a RL training method and widely used for directly estimating policy instead of estimating state-value $Q(s, a)$. This method can be seen as an optimization problem since our aim is minimizing the cost function which is function of policy. PPO trains a stochastic policy in an on-policy way. This indicates that it explores by sampling actions according to the latest version of its stochastic policy. The amount of randomness in action selection depends on both initial conditions and the training procedure. After a while in training, the policy becomes progressively less random, as the update rule encourages it to exploit rewards that it has already found. This may cause the policy to get trapped in local optimal however in our case we observed that our agent can partially learn to reach goal.

In this work, we used PPO-Clip which relies on specialized clipping in the objective function to remove urge for the new policy to get far from the old policy. The update policy rule for PPO-Clip is shown Eq. (1), referred to Schulman et al. 2017. Where E is empirical expectation over time, θ_k ,θ_{k+1} current and next network parameters.

$$\theta_{k+1} = argmax_{\theta} E_{s,a\sim\pi_{\theta_k}}[L(s, a, \theta_k, \theta)] \tag{1}$$

Where epsilon ϵ is hyperparameter usually 0.2 or 0.1, $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ denotes the ratio of the probability under the new and old policies, $A(s, a)$ is estimated advantage function according to negative and positive advantage value cases. The

KL penalty L will control the change of policy and will reduce kicks related to unrelated actions. Therefore, new policy does not benefit by going far away from the old policy Schulman et al. 2017.

$$
\begin{aligned}
&L(s, a, \theta_k, \theta) = min(r(\theta)A^{\pi_{\theta_k}}(s, a), g(\epsilon, A^{\pi_{\theta_k}}(s, a))) \\
&\text{where } g(\epsilon, A) = \begin{cases} (1+\epsilon)A, A \geq 0 \\ (1-\epsilon)A, A < 0 \end{cases}
\end{aligned} \tag{2}
$$

The policy network simply has three layer with tanh activation where first layer have 3 input(observation), 20 unit, second layer has 20 input and unit , third layer 20 input, 8 unit (action space). Action probabilities are obtained using soft-max and Boltzmann distribution. Lastly, if action is scholastic, action is randomly chosen otherwise, if agent is deterministic then, highest probability of action is chosen.

2.2 Double Deep Q Network (Double-DQN) Method

In Temporal Difference (TD) Learning, state-action values, Q(s,a) is updated according bootstrapping from the current estimate of the value function. However, if number of states increases, TD learning would not be preferred since it needs to update each state-action pairs. To overcome this issue Deep Q Network learning (DQN) is proposed where Q values are estimated according to Neural Networks. DQN and TD based Q update rule can be represented as Eq. 3. Where α scalar step size, $\gamma = [0, 1]$ discount factor, r_t reward, $argmax(\widetilde{Q_\theta}(s_{t+1}, a))$ maximum value of next state-action pair and $\widetilde{Q_\theta}(s_t, a_t)$ current estimated state-action pair Hasselt, Guez and Silver 2015.

$$
\widetilde{Q_\theta}(s_t, a_t) = \widetilde{Q_\theta}(s_t, a_t) + \alpha(r_t + \gamma argmax(\widetilde{Q_\theta}(s_{t+1}, a)) - \widetilde{Q_\theta}(s_t, a_t)) \tag{3}
$$

Neural network's weights θ can be updated according to loss function Eq. 4 and using backpropagation.

$$
L = \frac{1}{2}(Q_\theta(s_t, a_t) - \widetilde{Q_\theta}(s_t, a_t)) \tag{4}
$$

While calculating loss which indicates Q value's error, target value $\widetilde{Q_\theta}(s_t, a_t)$ could change from one time step to the next. This may cause unstable behaviour. To overcome this issue Double Deep Q Network (Double-DQN) is proposed. Double-DQN uses two identical neural network to estimate Q_1 and Q_2 and only update one estimator (randomly) after each action. This prevents overestimation of Q values and agent learns with higher performance than DQN learning. Update rule similarly can be shown as Eq. 5.

$$\begin{aligned} \widetilde{Q_1}(s_t, a_t) &= \widetilde{Q_1}(s_t, a_t) + \alpha(r_t + \gamma argmax(\widetilde{Q_2}(s_{t+1}, a)) - \widetilde{Q_1}(s_t, a_t)) \text{ or} \\ \widetilde{Q_2}(s_t, a_t) &= \widetilde{Q_2}(s_t, a_t) + \alpha(r_t + \gamma argmax(\widetilde{Q_1}(s_{t+1}, a)) - \widetilde{Q_2}(s_t, a_t)) \end{aligned} \tag{5}$$

Our neural network model simply has four layer with relu activation and 100 unit where first layer have 3 input(observation) and last layer has 8 output action.

2.3 Defining States, Actions and Rewards

To make problem solvable we observe and represent the centre of mass (CoM) with two dimension. X-axis (C_x), Z-axis (C_z) and pitch angle(C_{pitch}) are chosen as states where we rounded to two digits after the comma so that our observation space became more sparse and reduced. Observation space O and each state's boundaries can be shown as Eq. 6.

$$\begin{gathered} O = (C_x, C_z, C_{pitch}) \\ \text{where } C_x = [-0.8, 0.8] \text{ , } C_z = [0.2, 2.0] \text{ , } C_{pitch} = [-\pi, \pi] \end{gathered} \tag{6}$$

Performance of agent is not main focus on this work therefore, we decided our action type as discrete and increasing-decreasing angle positions for each joint instead of torque since we observed that agent can learn faster if actions are incremented positions. Our four legged robot's has 3 joints in one leg and totally 12 joints which corresponds 12 increasing angle position and 12 decreasing angle position (24 action). To agent learn faster we can remove some of the parameters from action space. Front Right (FR) Shoulder ,Front Left (FL) Shoulder, Rear Right

(RR) Shoulder , Rear Left Shoulder(RR) are assigned as fixed proper angle positions (0°). This allow agent to not move foot in direction of y axes. Additionally since we simplify our stand-up position in 2D space, we have assigned symmetric values to left and right free joints. Thus, we reduced to 4 joints corresponds to 8 action and it can be shown as Table 2.3. Note that these action restriction might reduce performance of our desired balancing algorithm however agent might learn faster since we reduced the complexity.

action numbers	action definitions
0	Increment FR and FL thigh
1	Decrease FR and FL thigh
2	Increment FR and FL calf
3	Decrease FR and FL calf
4	Increment RR and RL thigh
5	Decrease RR and RL thigh
6	Increment RR and RL calf
7	Decrease RR and RL calf

Our task is teaching robot to stand up and stay in stand up position. To reach a specific position we have defined a desired goal pose which includes desired X-Z position and pitch angle (G_x, G_z, G_{pitch}). We structured our rewards as following items, while iteration is not done and;

1. if robot's front leg is up, back leg is in the ground and have higher than specific pitch angle (refers to standing pose), give reward 30 points
2. if $d - d_{prev}$ and $\Delta h - \Delta h_{prev}$ has negative value (decrease in distance good), give reward 5 point, where $d = (G_x - C_x)^2 + (G_z - C_z)^2$ and d_{prev} are current and previous distance between goal and CoM, $\Delta h = |G_z - C_z|$ and Δh_{prev} are current and previous height difference.
3. if d and $\Delta\theta$ are in smaller than thresholds (in our case less than 0.25m and 0.3 rad), give reward 50 point, where d distance and $\Delta\theta = |G_{pitch} - C_{pitch}|$ is pitch difference

The terminal states are exceeding maximum time duration (10 second), falling to

the ground, getting out from bounded area and reaching goal pose. If iteration is done and;

1. if goal is reached, give reward 300 point
2. if goal is not reached, penalize -100 point

3 Results

In this section, quadrupedal robot named Aliengo is used and spawned in Gazebo simulation environment. According to our structured states, rewards, actions and RL agents, we trained and test two methods; PPO-Clip and Double-DQN.

3.1 Training

Training algorithm keeps training until consecutively 10 iteration's reward have higher than a specific threshold. Thereafter, training can be terminated, agent's neural network model can be saved for further testing.

PPO-Clip training graph can be shown as in left Figure 2 where episode number ~ 260 and highest reward was ~ 3000. Double-DQN training graph can be shown as in right Figure 2 where episode number $\sim$260 and highest obtained reward as ~ 1750. It can be seen that even though reward graphs have noise (due to unstable position), both agent's accumulated rewards are increased with in further iterations. Also, both agents are learns how to stand-up in similar episode numbers. This indicates our agent is able to converge higher rewards and learn to achieve it's objective.

3.2 Testing

Obtained neural network models are stored and used as testing. In left Figure 3, shows PPO-Clip testing result, out of 100 episodes, 94 of them successfully collected high reward while staying in stand-up position until timing ends. In right Figure 3, shows Double-DQN reward-episode graph from 100 episode 88 of them robot directly reached goal state thus, it was successful. Double DQN agent

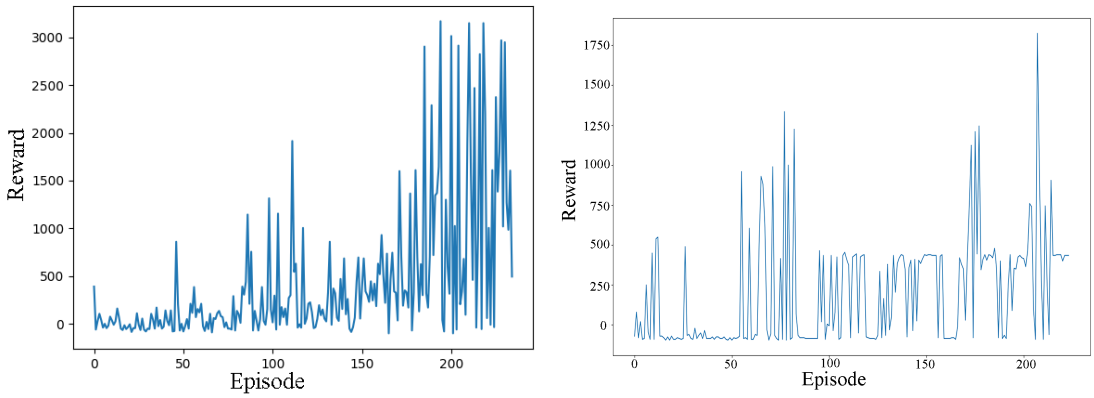

Figure 2: Training Reward-Episode Graphs: (Left) PPO-Clip Agent, (Right) DDQN Agent

can get higher reward if it would stay in stand-up position however due to stucking in local minimum, our agent couldn't get higher reward and preferred to end it's episode by directly reaching the goal. Aliengo's standing up position in Gazebo

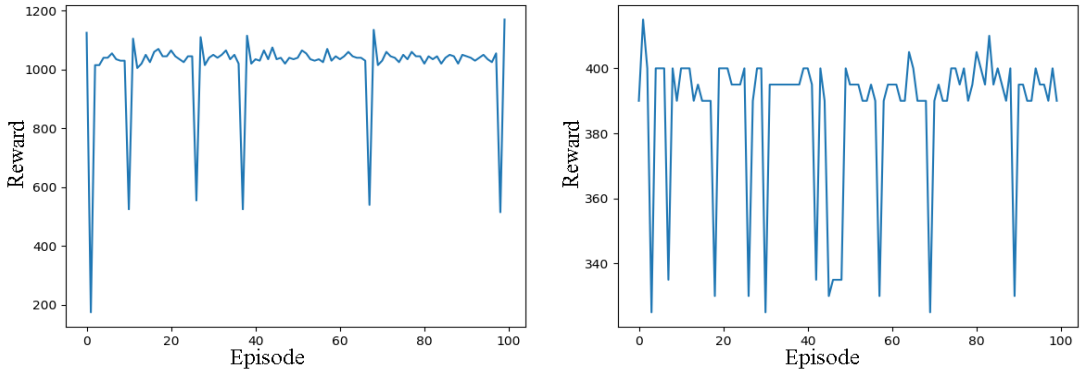

Figure 3: Testing Reward-Episode Graphs: (Left) PPO-Clip Agent, (Right) DDQN Agent

and in Rviz environment presented in Figure 4. It can be seen that Aliengo can stand up on support of it's calf due to our given low height goal pose. We have tried to give the goal in higher z location, but we observed that our agent couldn't learn to reach using it's feet.

Main limitation that we confronted is agents doesn't always collect high reward which can also referred as being stuck in local minimum. To overcome this issue,

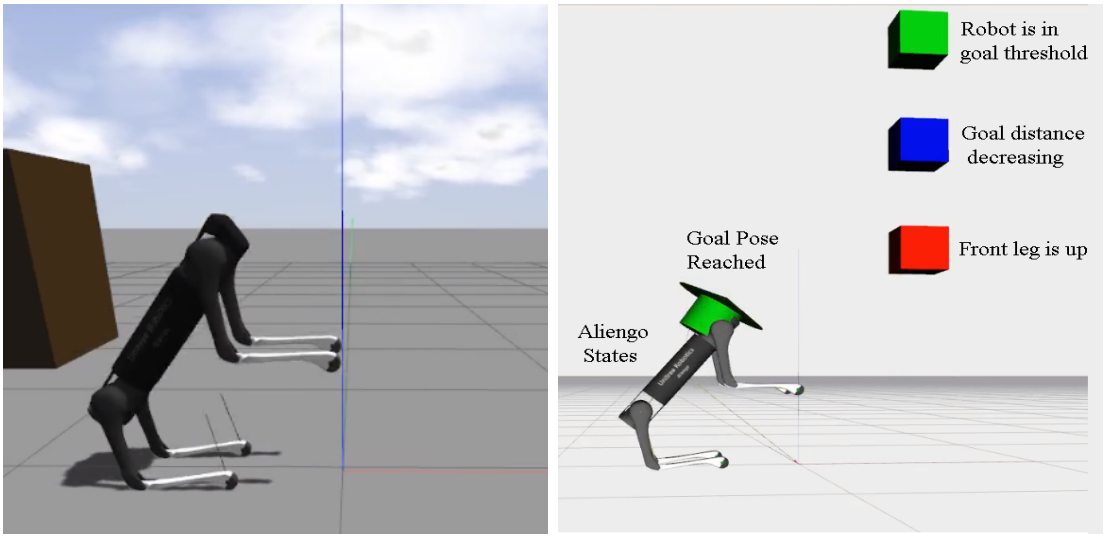

Figure 4: Snapshots of Standing-up Position in Testing: (Left) Gazebo Physic Simulator (Right) Rviz Sensor-Reward Visualization where green arrow indicate robot reached the goal pose , red cube indicate 1st reward, blue cube 2nd reward and green cube 3th reward

we have tried several training and combination of different weight of reward until agents are able to collect high rewards (converges).

Comparing two methods, both agents increases their mean reward thus, RL methods taught our robot to stand-up and stay in stand up position. PPO-Clip shows better performance by collecting more reward than Double-DQN. Because in PPO agent, robot is able to stand up and keep it's position until episode terminates. Double-DQN only perform standing-up action and then finishes by achieving terminal states. This may indicate that Double-DQN stuck more in local minimum than PPO-Clip. In testing environment PPO-Clip agent has higher success rate than Double-DQN. One of the main reason that PPO suppress Double-DQN, is unstability of our platform and estimating actions instead of Q values. Estimating action from states might perform better result since PPO's network can find better relationship between actions and states.

4 Conclusion

In conclusion, we simulate four legged robot named Aliengo by using Gazebo-ROS and Openai environment. Thereafter, we defined our states as 2D robot's center of mass, actions as joint angles and reward as achieving a goal pose in order to teach robot to stand-up and stay in stand-up position. We implement PPO-Clip and Double-DQN Deep learning algorithm to compare performances of standing-up result. It was observed that in generally, PPO-Clip outperforms Double-DQN algorithm. Since, in training process, PPO-Clip is able to collect more reward by staying in standing up position and in testing process PPO-Clip had more success rate than Double-DQN. Nevertheless, both algorithm is successfully teach Aliengo to reach standing-up position. For future work, we can re-structure the terminal states, rewards and actions so that robot can stand-up and stay in standing position on it's feet instead of its calves.

References

Hasselt, Hado van, Arthur Guez and David Silver (2015). 'Deep Reinforcement Learning with Double Q-learning'. In: *CoRR* abs/1509.06461. arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

Schulman, John et al. (2017). 'Proximal Policy Optimization Algorithms'. In: *CoRR* abs/1707.06347. arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

5 Appendix

https://github.com/hgaren/RL_Aliengo