Learning Robotic Control for the Double Pendulum Using Reinforcement Learning

Sara Honarvar
Mechanical Engineering
University of Maryland
College Park, USA
honarvar@umd.edu

***Abstract*—In this project, we apply reinforcement learning techniques to find a control policy for the classic control task of swinging up a double pendulum (acrobot). The acrobot system includes two links and two joints, where only the joint between the two links is actuated and the goal is to swing up the second link to a certain height. The data used in this paper is obtained from the OpenAI Gym, Acrobot-v1 environment simulator. At each time step, in this environment, the agent chooses an action and receives a reward until it reaches to the desired height or the maximum number of steps is reached. The objective for the acrobot is to reach to the target height with the shortest number of steps to maximize the cumulative reward. To achieve this aim, three of well-known techniques in reinforcement learning: Q-learning, policy gradient, and actor-critic have been implemented and compared.**

I. Motivation

A double pendulum or acrobot [1] usually refers to a two link underactuated robot with one end fixed at some point in the coordinate frame. The acrobot system, which is roughly similar to a gymnast swinging on a highbar, includes two links and two joints, where only the joint between the two links is actuated. This system is a classical problem in control theory, as it represents a nonlinear and unstable dynamic system that can exhibit chaotic behavior. Control of dynamical systems usually requires detailed knowledge of the physical system, which can be difficult and time consuming as the complexity of system increases. Applying learning algorithms that can learn the optimal controllers without prior knowledge of the system is thus an attractive approach in these situations. In this project, we aim to find a control policy for swinging up a two link pendulum using reinforcement learning (RL) techniques. To address this objective, we used the Acrobot-v1 system from OpenAI Gym environment [2]. OpenAI Gym is a standard python enviroment in the field of RL. It includes a diverse collection of environments, which enables people to iterate on and improve RL algorithms. One of these environments, is Acrobot-v1 which gives us this opportunity to simulate the behavior of a two link pendulum while applying torques to it and collect corresponding rewards. For this project, we used this simulation environment [3].

Within the Acrobot-v1 environment, the episode starts with the links hanging downwards (Fig. 1 (a)), and the goal is to actuate the inter-joint (by applying a torque to it) such that the end of the lower link would swing up to a given height (Fig. 1 (b)). The state space of the environment is comprised of six values:

(a) Beginning of an episode (b) End of an episode

Fig. 1: The Acrobot-v1 environment from OpenAI Gym in the (a) beginning and (b) end of an episode

$[sin(\theta_1), cos(\theta_1), sin(\theta_2), cos(\theta_2), \dot{\theta}_1, \dot{\theta}_2]$, where θ_1 and θ_2 are the two rotational joint angles and $\dot{\theta}_1$ and $\dot{\theta}_2$ are their corresponding angular velocities. These 6 values represent the observation spce. The action space is discrete and allows applying +1, 0, or -1 that corresponds to applying a clockwise/counterclockwise/no torque to the actuated link (the second link). At each time step, the agent chooses an action from this discrete space and a reward of -1 is provided for each timestep taken, except for the termination step, where the reward is zero. The episode ends when the end of the lower link is at the given height, or in the worst case the maximum number of steps (500) is reached.

In this problem, there is no input data set that is divided into training, validation, or testing subsections but rather the input is constructed from the live observations and reward from the environemnet that depend on the agent's current action. Based on this discription, this problem fits under reinforecement learning (RL), where the process of learning is performed by interacting with an environment. In this process, the agent observes a state, s, performs an action, a, and receives a new state, s', and reward, r.

II. Technical Approach

In this problem, the goal is to swing up the lower link to a certain height in as few steps as possible. As mentioned above, in this environment, for each action taken, a reward of -1 is given untill the agent achieves the mission. So, the goal is to maximize the cumulative reward.

A. Models used

To solve double pendulum problem, we implemented three methods: 1) Q-learning (a critic method) as our benchmark, 2) REINFORCE algorithm with baseline from policy gradient (an actor method), and 3) an actor-critic method with advantage value. Finally, we compared the results of these methods in terms of averaged reward per episode.

1) Q-learning: Q-learning is a critic method, where the goal is to learn the action-value function, Q of some policy, π, and is defined as the expected reward starting from state, s, taking the action, a, and following the policy π [4]:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[R_t|s_t = s, a_t = a] \\ &= \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a] \end{aligned} \tag{1}$$

Where γ is the discount factor, specifying the relative importance of imediate reward comparing to those received in the future [4]. Using the Bellman optimality equation, the function Q can be optimized with the update equation of $Q(s,a) \leftarrow \alpha\mathbb{E}[r + \gamma max_{a'}Q(s',a')]$, where α is the learning rate. Finally, the optimal policy is defined as $\pi(s) = max_{a'}Q(s',a')$.

Q-learning is an off-policy approach as the value function learns from actions that are outside the current policy, independent of the policy being followed. In the Acrobot-v1 environment, as mentioned earlier, we are dealing with a continuous time-dependent observation/state space, which includes six values: sin and cos of two joint angles and the angular velocity for each link. Data visualization and exploration showed that the state space is multi-dimensional, with the first four dimensions ranging from -1 to 1 (sin and cos), while the final two dimensions having a larger range. For this system, a tabular q-learning approach, in which we create a table of [state, action] form, might not be practical. Since it needs to store too many states in the table and the immense memory requirement causes the training to be slow and impractical. However, it is an easy-to-implement algorithm and we use it as the benchmark in this project. In order to create the Q-table, we first discretize the observation space using tile coding [1]. Tile-coding is a practical and computationally efficient tools being used in RL problems with continuous space, where we create several overlapping tilings (grids) with different offsets [5]. Then, for any sample value, we check in which tile it lies. Finally, we encode the original continuous value by a vector of bits that identifies each activated tile.

For tile-coding, we use the lower bound (i.e., -1) and upper bound (i.e., 1)of the observation space and 5 bins for each feature (state) and 0.4 as the offset. After discretizing the observation space, we can create the Q-table and implement the Q-learning. For taking action in the Q-learning, I use both exploitation (i.e., selecting actions based on the maximum future reward) with ϵ-greedy approach and exploration (i.e., selecting random actions to explore and discover new states that may not be selected during the exploitation process) approaches. In my algorithm, the ϵ was set to 1 with a decaying rate of 0.9995. The other hyperparameters of the model are shown in (Table. I).

TABLE I: Hyper-paramteres for different methods

Method	Hyper-parameters	
	γ	α
Q-learning	0.99	0.02
REINFORCE	0.99	0.005
Actor-critic	0.99	0.002

2) Policy gradient: In policy gradient methods, the objective is to learn a policy $\pi_\theta(a_t|s_t)$ that maximizes the cumulative future reward to be received from start time to terminal time. θ is the policy parameter, a_t and s_t are the action and state at time t. For that, the probability distribution of actions is updated such that the actions with higher future expected reward (G_t) have a higher probability value for an observed state [4]. Here, we implemented the REINFORCE algorithm, which is a Monte-Carlo variant of policy gradients [4]. In this algorithm, the agent collects a trajectory of one episode using its current policy, and uses it to update the policy parameter. The update equation is as follows:

$$\Delta_\theta J \sim \sum_{t=0}^{T-1} \Delta_\theta log\pi_\theta(a_t|s_t)G_t \tag{2}$$

To implement the REINFORCE algorithm on acrobot-v1 problem, a two layer neural network was considered for the policy estimator, with ReLU as the inner activation function and softmax as the outer activation function. The actions were selected as softmax function so that the output is a probability over available actions. Different network structures with different hidden nodes (e.g 16, 32, 64, and 128) were tried. 64 found to produce the best results.

The REINFORCE algorithm was implemented in pytorch. At first, the algorithm was ran on CPU but to increase the running time, we switched to GPU using google colab, which is the google cloud platform. Each run takes less than a minute. Different learning rates: from 0.005 to 0.02 and different discounted reward form 0.9-0.99 were tested and the values that produced the best performance were selected I. Finally, to decrease the large variance produced by the REINFORCE algorithm, and to stabilize the learning, we implemented REINFORCE with a baseline, with baseline being the average of the rewards [4]. For the algorithm, we used 500 episodes with 10 batches (i.e., number of trajectories for each episode) and used adam optimizer.

3) Actor-critic: The actor-critic method can be regarded as combination of the actor and critic methods for faster convergence. In this approach, the "critic" estimates the value function, which is the action-value (the Q value) or state-value (the V value). While, the "actor" part, like the policy gradient, updates the policy distribution in the direction suggested by the Critic.

Here, we implemeted the Advantage Actor Critic (A2C) method, which is an actor-critic method with baseline [6]. The advantage term (A) demonstrates how much better it is to take a specific action comparing to the averaged general action at a given state.
In our algorithm, wrote in pytorch, for each learning step, we updated both the actor parameter (with policy gradient and advantage value) as well as the aritic parameter by minimizing the mean squared error with the Bellman update equation. So, the update equation is as follows:

$$\Delta_\theta J \sim \sum_{t=0}^{T-1} \Delta_\theta log \pi_\theta(a_t|s_t) A(s_t|a_t) \tag{3}$$

Where, again θ is the policy parameter and $\Pi_\theta(a_t|s_t)$ is the policy from which we take action. For the actor part, I used a fully connected network with three hidden layers and leaky ReLU as the activation function to directly learn the policy. For the critic part,to estimate the value function, I used a two layer neural network with ReLU as the inner activation function and softmax as the outer activation function. # of hidden nodes was set to 64. The other hyperparameters are presented in I. We used the same learning rates for both actor and critic parts. Although different network structures have been examined, the reported structure found to produce the best results. Again, Iwe used GPU provide by google colab to improve the running time.

B. Evaluation

In this problem, the goal is to swing up the lower link to a certain height in as few steps as possible. In all methods, we used the maximum of 3000 episodes for training and 1000 episodes for testing. Besides, the maximum number of steps, taken in each episode was set to 500. As mentioned above, in this environment, for each action taken, a reward of -1 is given until the agent achieves the mission. Therefore, the goal is to maximize the cumulative reward or minimize the number of steps taken for getting to the maximum reward. Consequently, for evaluation purposes, we present the cumulative reward for each episode. As a side note, the average number of steps taken to finish an episode can be implicitly derived from the averaged reward plots.

III. Results

In this section, we will show the rewards over #episodes for each of the method for training and testing episodes (Fig. 2). Furthermore, we represent the mean and standard devaition (std) of rewards across 3000 episodes of training and 1000 episodes of testing to show which method performs better than the others (Table. II).

IV. Discussion of Results

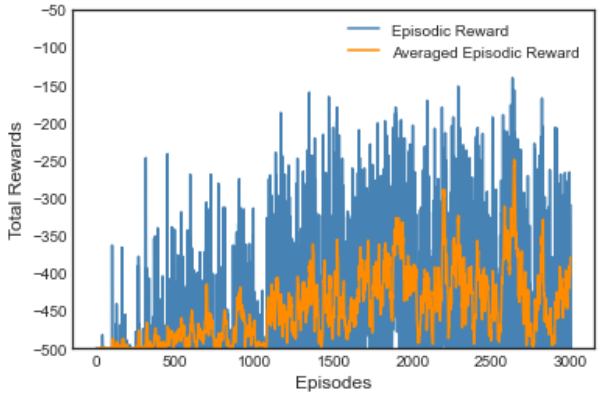
Our results showed that the REINFORCE algorithm with testing reward of -80.84 $\pm$ 33.78 outperformed Q-learning approach and actor-critic method when solving for acrobot-v1 problem. Our implementation also outperforms the state-of-the-art results reported in [7]. It obtained more rewards in a short period of time when it was trained appropriately. Sometimes in the training process, it happens that it cannot find the optimal policy, but once it achieves this level of knowledge, it performs very well. The REINFORCE algorithm was also faster than the benchmark and the A2C. The REINFORCE reaches to reward of about -100 after only about 300 episodes. It also performs better than the actor-critic method in [8]. In their algorithm, after about 2500 episodes, the cumulative reward is settled around 190. Also, our algorithm produce comparable results to the deep Q learning algorithm reported in [8]. Although adding the baseline decreased the high variance of the REINFORCE to a great extent, as it is mentioned in [4], the baseline is not used for bootstrapping. So, we used the actor-critic to guarantee achieving some level of performance by learning the value function in environment. However, the performance never reached the one with REINFORCE method. Even increasing the hidden layers, and nodes did not seem to improve the results. This is an interesting result, which shows that using a complicated method, does not necessarily result in a better performance especially when solving for a problem which does not have a complicated nature. For sure, actor-critic method would perform better for a large number of problems. However, finding an appropriate solution for a specific problem is something that should be considered when using machine learning as primary solution.

V. Conclusion

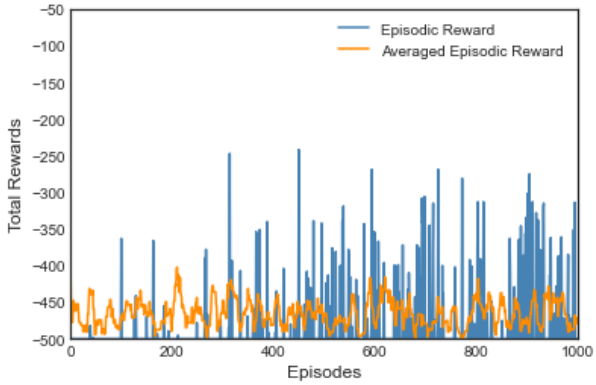
This project aimed at learning a control policy for swinging up a double pendulum (a classic control problem) from the RL perspective. Three algorithms of Q-learning, REINFORCE with baseline, and an actor-critic method was implemented. It was shown that REINFORCE method outperfomed the other two methods in terms of reaching to maximum reward in a short time. We found that having a complicated model does not always lead to the best results. It is improtant that we pay attention to the complexity and nature of the problem at hand and choose our approach for addressing the problem, accordingly. The data visualization and exploration can be a very important stage in choosing the approach.

TABLE II: Performance of different methods

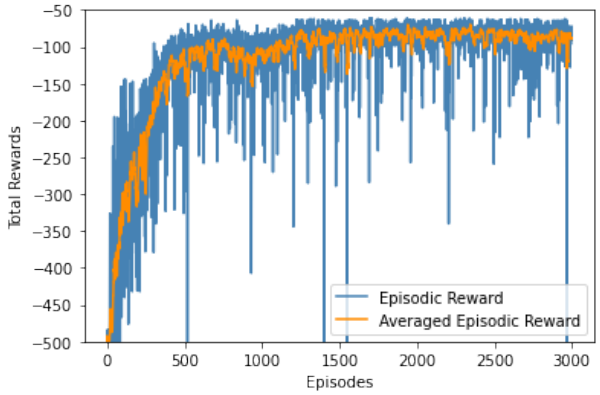
Method	Train Reward		Test Reward	
	mean	std	mean	std
Q-learning	-446.54	81.1	-465.93	59.31
REINFORCE	**-118.59**	**80.59**	**-80.84**	**33.78**
Actor-critic	-200.19	140.71	-188.46	49.12



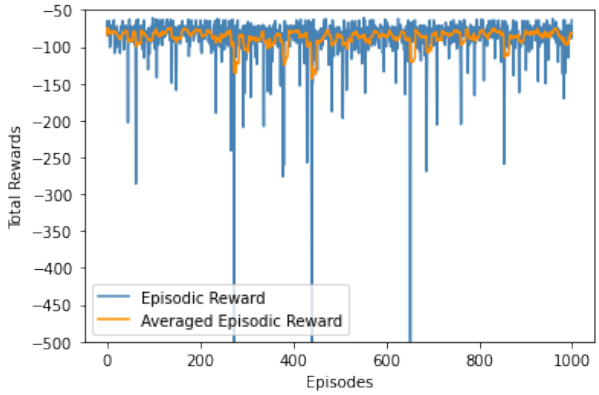
(a) Q-learning (Training episodes)



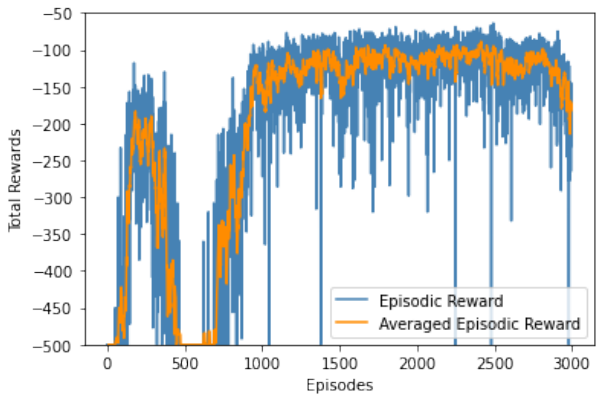
(b) Q-learning (Testing episodes)



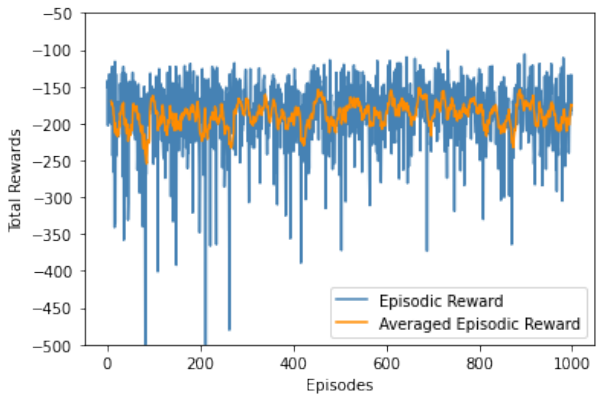
(c) REINFORCE (Training episodes)



(d) REINFORCE (Testing episodes)



(e) Actor-critic (Training episodes)



(f) Actor-critic (Testing episodes)

Fig. 2: Train and test results in terms of rewards for (a-b) Q-learning, (c-d) REINFORCE algorithm and (e-f) Actor-critic for solving the Acrobat-v1 problem.

REFERENCES

[1] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

[2] Alborz Geramifard, Christoph Dann, Robert H Klein, William Dabney, and Jonathan P How. Rlpy: a value-function-based reinforcement learning framework for education and research. 2015.

[3] Christoph Dann. Classic acrobot task.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[5] Alexis Cook. Deep reinforcement learning nanodegree.

[6] Chris Yoon. Understanding actor critic methods and a2c.

[7] Haotian Zhang, Yuhao Wang, Jianyong Sun, and Zongben Xu. Amortized variational deep q network. *arXiv preprint arXiv:2011.01706*, 2020.

[8] Ron Dorfman and Eyal Ben David. Github repository: Advanced topics in reinforcement learning.