Deep RL

Simon Svensson, Lukas Martinsson

May 2022

1 Introduction

For this assignment deep Q-learning (DQN) and Actor-critic (A2C) were the algorithms chosen to analyze and compare. Since it would be interesting to have one DQN algorithm more suited for discrete tasks and one capable of solving both discrete and continuous, DQN and A2C fulfilled this requirements. Also since Q-learning have been used in previous assignment, learning a more complex deep learning version of it would be both interesting and practical. A2C was also chosen since it has been mentioned in lectures which is why an implementation of it to get a hands on experience would be interesting.

This paper used *https://github.com/DLR-RM/rl-baselines3-zoo* and implemented it with *https://colab.research .google.com/github/Stable-Baselines-Team/rl-colab-notebooks/blob/sb3/rl-baselines-zoo.ipynb*. [1] [2]

2 Deep Q-learning

In this section the Deep Q-learning algorithm is explained as well as the environments this paper has used to test it on and the results from them.

2.1 a)

Deep Q-learning is a more advanced version of the well known Q-learning algorithm. Q-learning in theory enables an agent to act optimally in each state to maximize its rewards. However its drawback is that for large state and action spaces this is either too time consuming or even impossible. This is especially true in continuous tasks, or even in highly complex games such as chess, which has an enormous state space. To solve this, deep q-learning utilizes a neural network that updates its weight instead of a q-table like normal q-learning would, which enables it to produce reliable approximate rewards for each action. Furthermore, deep Q-learning utilizes two neural networks instead of just one (the second one usually called the target network). The purpose of the target network is to provide a stationary target for the bellman equation, and after every N steps it updates itself and becomes a copy of the main network which altogether leads to both stability and improved learning efficiency. [3] [4]

2.2 b)

The environment MountainCar-v0 can be seen in figure 1. It is a classic control environment where an underpowered car must climb a one-dimensional hill to reach a target. The goal is to get to the

target as fast as possible, and to do so it may use the left hill as a swing. The observed state consists of its one-dimensional position and velocity. The possible actions are discrete, either push left, push right or do nothing. It accumulates -1 reward each time step and an average reward of -110 over 100 consecutive runs is considered a solution. The environment ends after 200 time steps if the goal is not reached. [5]

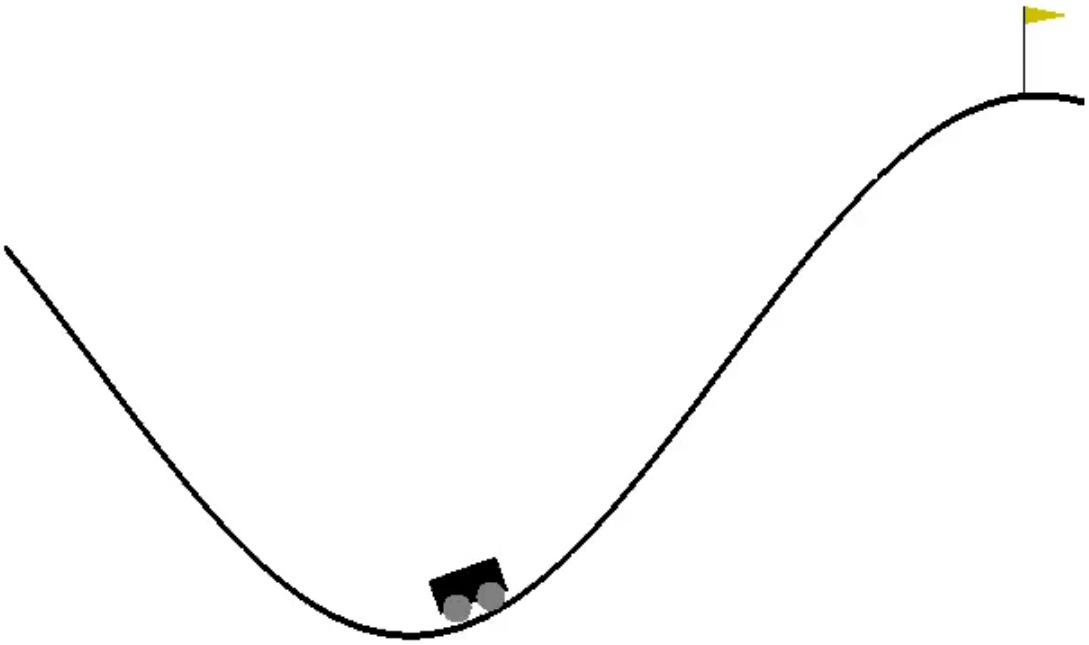

Figure 1: The MountainCar-v0 environment.

The environment Acrobot-v1 can be seen in figure 2. The goal of this environment is to get the links to reach above the line. This goal is reached through applying torque to the joint connecting the two links. The possible actions are applying -1, 0 or 1 torque to the joint. The observation space consists of 6 trigonometrical functions giving information regarding each joint's position relative in the state space and each other. It accumulates -1 reward each time step with a threshold of -100 and is rewarded 0 if the goal is reached. There is no amount of reward that is considered a solution. The environment ends after 500 time steps even if the goals has not been reached. [6]

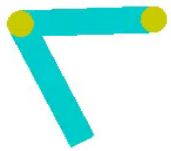

Figure 2: The Acrobot-v1 environment.

The configurations used on those two environments are given in tables 1 and 2, which were the default ones in the linked Github. We also decided to train the DQN for 350000 timesteps because that was the lowest number of timesteps that did solve the MountainCar environment a couple times.

Learning rate	0.004
Gamma	0.98
Gradient steps	8
Exploration fraction	0.2
Exploration final epsilon	0.7

Table 1: Configuration for DQN running on MountainCar-v0. Gamma is the discount of future rewards.

Learning rate	0.00063
Gamma	0.99
Gradient steps	-1
Exploration fraction	0.12
Exploration final epsilon	0.1

Table 2: Configuration for DQN running on Acrobot-v1. Gamma is the discount of future rewards.

The resulting plots for running DQN on the environments are displayed in figures 3 and 4. It took 10 minutes to train DQN on MountainCar-v0 and 18 minutes to train it on Acrobot-v1. The plots are created from our code used in the previous assignment.

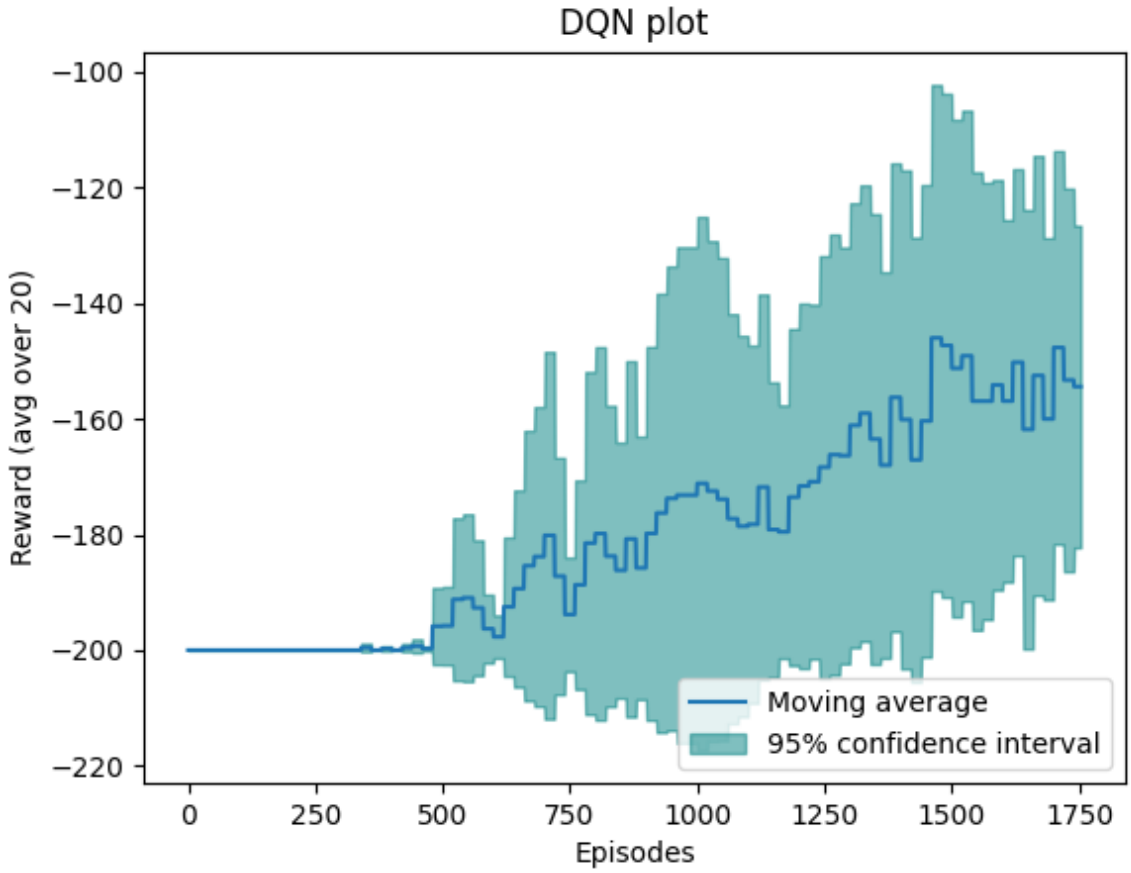

Figure 3: Average reward with 95% confidence interval over 3 runs in the MountainCar-v0 environment.

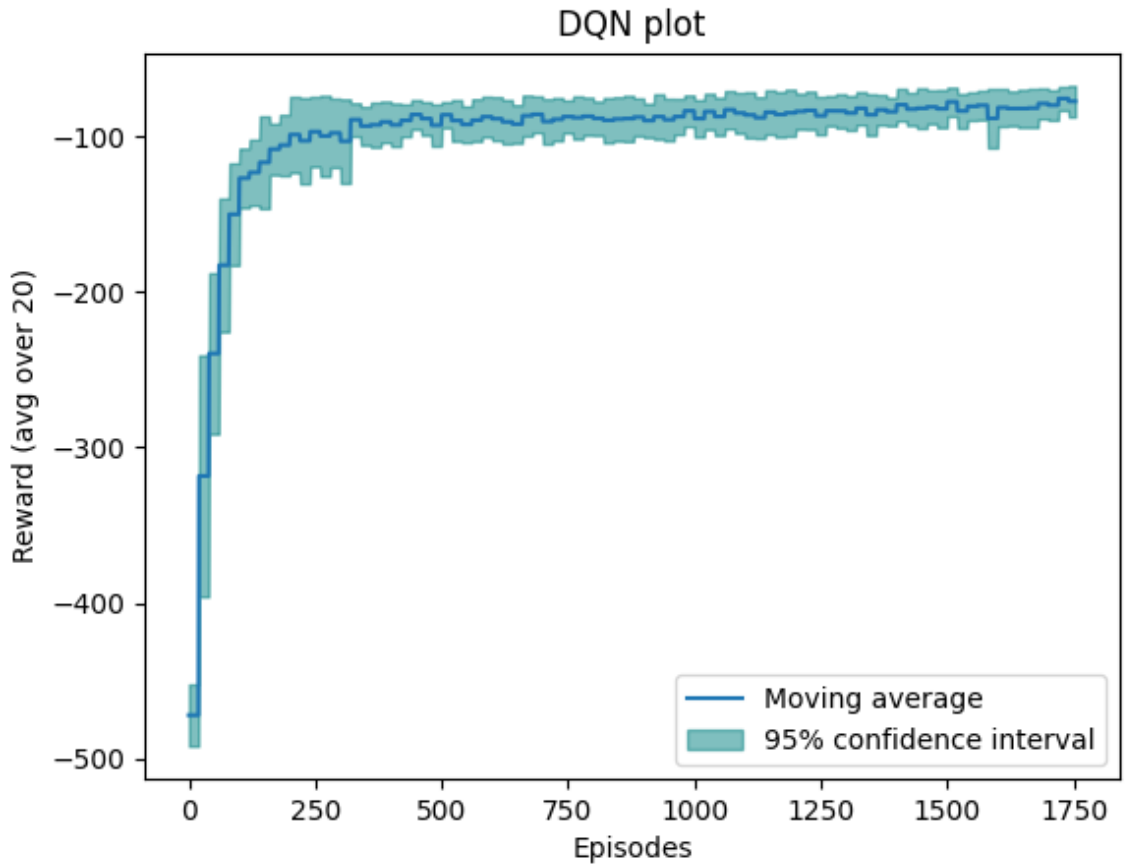

Figure 4: Average reward with 95% confidence interval over 3 runs in the Acrobot-v1 environment.

3 Actor-Critic

In this section the Actor-critic algorithm is explained as well as the environments this paper has used to test it on and the results from them.

3.1 a)

Actor-critic or A2C for short is a deep RL algorithm that utilized both value based methods with policy based methods. The main pros with value based methods are that they are excellent for finite action and state spaces since are able to calculate a reward after each step. However the drawback is that value based methods have difficulties with extremely larger action and state spaces, like in continuous task. Policy based methods on the other hand are able to handle these continuous task significantly better since they do not need to compute the maximum action every step (which would take a significant amount of time in a continuous setting). Furthermore it can, unlike the value based method use stochastic methods like Monte-Carlo which useful for exploring the environment. Therefore it has no need for the greed-epsilon policy that value based method use. However one drawback is that it can only calculate the rewards after an entire episode where the rewards can have high variance which slows down the learning process. A2C solves these problems through using both a value based network and a policy based network simultaneously. It uses the policy based method (the actor) to make each action and then the value method (the critic) for evaluation each action (which a policy method would be unable to). Through this combination the algorithm can utilize the strength of both methods while minimizing the drawbacks. [7] [8]

3.2 b)

MountainCarContinuous-v0 is the same as MountainCar-v0 except the action space is continuous and the reward function is different. The reward for reaching the goal is 100 minus the squared sum of actions from start to goal and an average reward of 90 is considered a solution. There is also no maximum number of steps. [9]

Acrobot-v1 is already described in section 2.2 b.

The configurations used on those two environments by A2C are given in tables 3 and 4, which were the default ones in the linked Github. We also decided to train the A2C for 350000 timesteps.

Learning rate	0.0007
n_envs	4
Normalize	True

Table 3: Configuration for A2C running on MountainCarContinuous-v0.

Learning rate	0.0007
n_envs	16
Normalize	True

Table 4: Configuration for A2C running on Acrobot-v1.

The resulting plots for running A2C on the environments are displayed in figures 5 and 6. It took 2 minutes to train A2C on MountainCarContinuous-v0 and 1 minute to train it on Acrobot-v1.

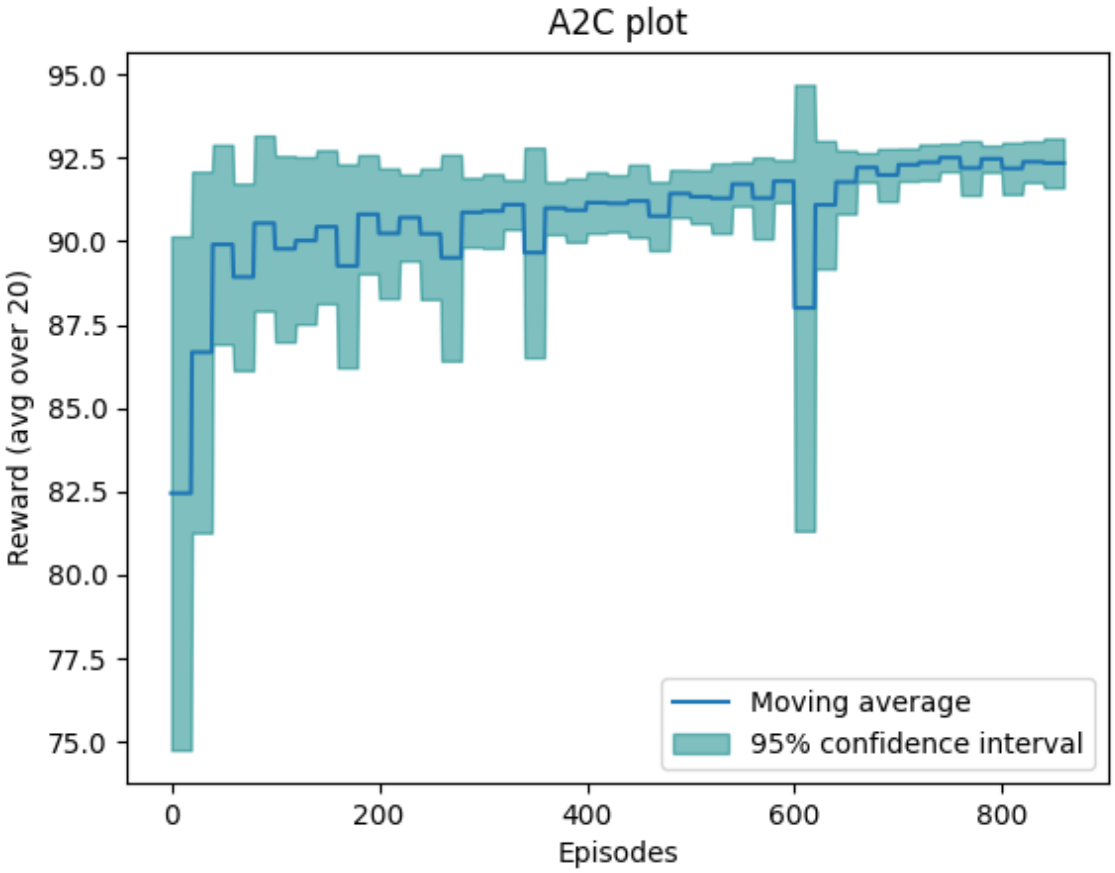

Figure 5: Average reward with 95% confidence interval over 3 runs in the MountainCarContinuous-v0 environment.

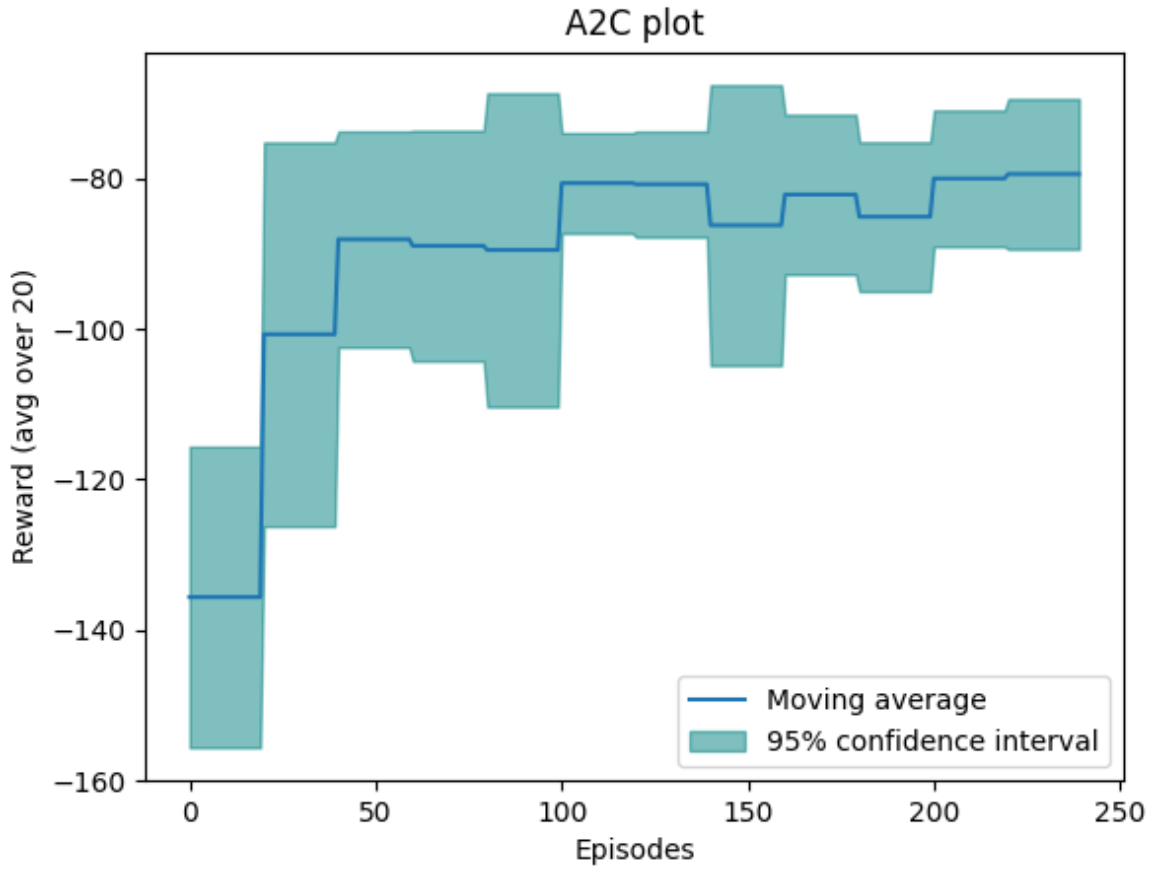

Figure 6: Average reward with 95% confidence interval over 3 runs in the Acrobot-v1 environment.

4 Comparison

In this section a comparison will be presented in regards to their differences, both in theory and in practice. The latter will be compared through the results presented above.

Although DQN can be used in continuous action spaces, it requires a discretizing of the action space which would further slow down the DQN learning speed which is already by its design slow. The A2C is an algorithm created to solve this problem, since it utilizes a combination of value based and policy based methods it is able to be utilized with good performance on continuous tasks. A similarity is that they both utilize a value based method to calculate rewards. Both algorithms relate to previous algorithms mention in the course such as the Q-learning algorithm (or possibly SARSA) since their value based method are based on them.

The first thing to notice about the results is that all algorithms are working as they should. The second thing is that the A2C algorithm was training in multiple environments at once, which made it about 10 times faster. The plots for the A2C solutions have fewer episodes because we only plotted the training from one of the environments in each run, which is because there was a different sample size of each environment making it hard to aggregate them. DQN on MountainCar-v0 had 2 unlucky runs, with that considered the results are expected and relate to the benchmark from the Github. [10]

It is hard to compare the mountain car environments since their reward functions are different. For the acrobot, we can see that both algorithms reach around -80 average reward, so their final performance is equal. Remember that both algorithms were trained for the same amount of timesteps. It appears in the acrobot plots that A2C reaches -80 reward in only 100 episodes while it takes DQN 1700 episodes. This difference is largely due to the fact that the A2C is learning in 16 environments at once and the knowledge is distributed among them. So we can multiply 100 by 16 to get 1600 which is about the same amount of episodes as DQN.

References

[1] A. Raffin, *Rl baselines3 zoo*, GitHub, 2020. [Online]. Available: https://github.com/DLR-RM/rl-baselines3-zoo (visited on 05/23/2022).

[2] *Google colaboratory*, colab.research.google.com. [Online]. Available: https://colab.research.google.com/github/Stable-Baselines-Team/rl-colab-notebooks/blob/sb3/rl-baselines-zoo.ipynb (visited on 05/23/2022).

[3] A. Vidhya, *Introduction to deep q-learning for reinforcement learning (in python)*, Analytics Vidhya, Apr. 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

[4] M. Wang, *Deep q-learning tutorial: Mindqn*, Medium, Feb. 2021. [Online]. Available: https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc.

[5] *Openai/gym*, GitHub. [Online]. Available: https://github.com/openai/gym/wiki/MountainCar-v0.

[6] *Openai/gym*, GitHub, 2022. [Online]. Available: https://github.com/openai/gym/blob/master/gym/envs/classic_control/acrobot.py (visited on 05/23/2022).

[7] C. Yoon, *Understanding actor critic methods*, Medium, Jul. 2019. [Online]. Available: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f.

[8] *Actor critic algorithms*, www.youtube.com. [Online]. Available: https://www.youtube.com/watch?v=w_3mmm0P0j8 (visited on 05/23/2022).

[9] *Mountaincarcontinuous v0 · openai/gym wiki*, GitHub. [Online]. Available: https://github.com/openai/gym/wiki/MountainCarContinuous-v0 (visited on 05/23/2022).

[10] *Rl baselines3 zoo benchmark*. [Online]. Available: https://github.com/araffin/rl-baselines-zoo/blob/master/benchmark.md (visited on 05/24/2022).