Cart-Pole Balancing using Model-Free Reinforcement Learning

Aimen Shahid
aimen.shahid@studium.uni-hamburg.de

Alina Munir
alina.munir@studium.uni-hamburg.de

Knowledge Processing in Intelligent Systems: Practical Seminar
Knowledge Technology, WTM, Department of Informatics, University of Hamburg

***Abstract*—Cart Pole Balancing is a benchmark simulation to test the effectiveness of a reinforcement model. Balancing is a repetitive task which requires learning from previous experience to be able to apply enough force in an appropriate direction to keep an object upright. In this paper we apply multiple deep reinforcement neural networks for the cart-pole balancing task and compare their performances in terms of the durations the pole was balanced and the epsilon values throughout the training process. We implemented the simple balancing problem on two discrete and two continuous state-reward algorithms. We study the Episodes vs. Duration graphs of the algorithms and discuss the results as well as possible reasons for their performance.**

I. Introduction

Our motivation for this topic stems from our interest in robotic motion in the real world. Whenever a robot navigates through the real world or interacts with another object, the biggest obstacle is often balance. This might be whether it can balance itself or the object it is holding. The Cart-Pole balancing task is a good simulation of that.

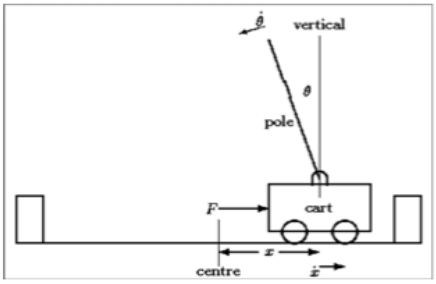

Figure 1. The Cart-Pole system. [8]

A pole is pivoted to a cart and has one degree of freedom along the horizontal axis. The aim of the Cart-Pole Balancing task is to balance the pole on the cart and keep it in an upright position using bi-directional forces. θ represents the angular position of the pole, and x represents the velocity of the cart. It is assumed that the cart is moving along a frictionless track in order to reduce the number of variables that must be dealt with. The system itself is controlled by applying a force of either -1 or +1 to the cart which causes it to move left or right respectively. The reinforcement learning algorithm gets a reward of +1 for each time step that the pole remains upright. The episode is terminated as soon as the pole is more than 15 degrees from the vertical or if the car moves more than 2.4 units from the center. [8][3]

II. Related Work

A lot of work has been done in the field of Reinforcement Learning, and one of the easier applications for using Reinforcement Learning has been the Cart-Pole Balancing problem. Even though most research papers and internet blogs have used Keras we would like to approach the problem using PyTorch. However, we will not be starting from scratch as the official PyTorch platform has released a tutorial [4] to get started with this problem and build a DQN model from scratch.

Research [6] has been done on balancing the cartpole using Q-learning, Deep Q-Network (DQN), Double-DQN (DDQN) and other complex algorithms and it has been found that DQN performs much better and faster than Q-learning algorithm. Although it is found that DQN basically learns the balancing in 150 episodes hence there is no need to use complex algorithms for such simpler tasks. This gave us the path to rather further compare the duration of how long our cart could be balanced by DQN and other complex discrete and continuous space algorithms.

III. Environment

For the environment we are using the OpenAIGym also referred to as simply gym. It is a toolkit that is used for developing and comparing reinforcement learning algorithms, and is compatible with most machine learning frameworks such as Tensorflow, Keras and PyTorch. In order to run any algorithm in the cart-pole simulation we can simply run an instance of the CartPole-v0 environment for 1000 timesteps.

The aim of our experiment is not to have the reinforcement learning algorithm simply take random actions at each step, we need it to know how each action affects the environment. In order to do this we implement the classic "agent-environment loop".

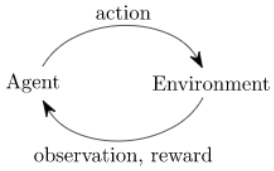

Figure 2. The Agent affects the Environment by performing an action and the Environment affects the Agent by returning an observation and a rewards. [3]

IV. Approaches

The objective of Reinforcement Learning models is to maximize the reward an agent receives for actions it performs in a dynamic environment. Based on past experiences the algorithm will then make the optimal decision, and it does this by following a few simple steps. It begins by observing the environment and then plans out a strategy for how to act. Once the best strategy has been decided, it will act on it and then receive either a reward or a penalty depending on how useful its action was. Using the reward and penalty the algorithm can now interpret which of its actions in the past were good for that environment and which weren't. It can then learn from these experiences and refine its strategies. Finally it will iterate until the optimal strategy is found.

There are two main types of Reinforcement Learning algorithms: model-free and model-based. A model-free algorithm is one that does not use environment dynamics to estimate the optimal policy. A model-based algorithm is one that uses the transition function and the reward function to estimate the optimal policy. In this paper we will focus on model-free algorithms.

A. Q-learning

The Q stands for quality, to be able to represent the quality of an action based on how much reward it would gain. Q-learning is a model-free reinforcement learning algorithm. It is also a value-based learning algorithm, which means that it uses an equation, in our case it would be the Bellman equation, to update the value function. A policy-based reinforcement learning algorithm, on the other hand, estimates the value function by using a greedy policy. It is also an off-policy learner, which means that it is able to learn the value of the optimal policy without having to rely on the actions of the agent.

Q-Learning makes use of a Q-Table, which is a data structure that is used to calculate the maximum expected reward for an action at each state. The Q-function consists of the Bellman equation and takes two inputs, the action (a) and the state(s).The steps that the Q-learning algorithm goes through are shown in Figure 3.

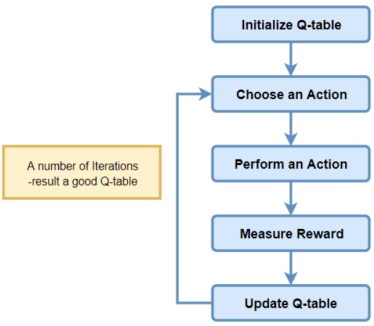

Figure 3. Q-Learning Algorithm [1]

B. Deep Q-Network

Deep Q-Network (DQN) is based on Q-learning and in fact follows most of the same steps. The main difference is that instead of a Q-Table it uses a Deep Q-Network instead. The biggest advantage of this is that the Deep Network is not limited by memory like the table-based system, and hence can learn an efficient function output for large amounts of data. Another big improvement in the DQN is the addition of a target network and experience replay.

C. Double Deep Q-Network

The regular DQN is more likely to overestimate the Q-values of a potential action in a given state. It also does not tend to overestimate all actions equally, so it is possible that the optimal action be given a lower Q-value than another sub-optimal action. In order to resolve this issue in the Double DQN the action choice is separated from the generation of the target Q-value. This is done by using the primary network to choose an action and the target network to generate the target Q-value for that action instead of choosing the max over Q-values to calculate the target Q-value.[5]. This allows us to reduce the overestimation and have a faster and more reliable training.

D. Q-NET

Q-Net is a neural network based implementation of Q-Learning that can be applied in a continuous space. It can also be considered a version of the Deep Q-Network mentioned previously, as the biggest difference between them is the space they operate in.

E. Deep Deterministic Policy Gradient

DQN can only be used to solve problems in a discrete and low-dimensional action space. It cannot be directly applied to continuous spaces. One way of resolving this is by discretizing the action space, however many limitations can arise, such as the problem of dimensionality. [7] presented a model-free, off-policy actor-critic algorithm that can work in

continuous space. The deterministic policy gradient (DPG) [10] is combined with the Deep Q-Network to create the Deep DPG (DDPG).

The DDPG utilizes two networks like the Actor-Critic method:

1) **Actor:** An action is proposed for a given state.
2) **Critic:** Given an action and a state, it predicts whether the action is good or bad, giving it a positive or negative value respectively.

DDPG also differs from the original DQN in two main ways. Firstly, it utilizes two target networks instead of just one. This allows for more stability during training as the target networks are updated slowly, the estimated targets remain stable. Secondly, it utilizes experience relay, which means that instead of only learning from the most recent experience, we learn from all our accumulated experiences by storing a list of tuples. These tuples include data like state, action, reward and the next state.[2]

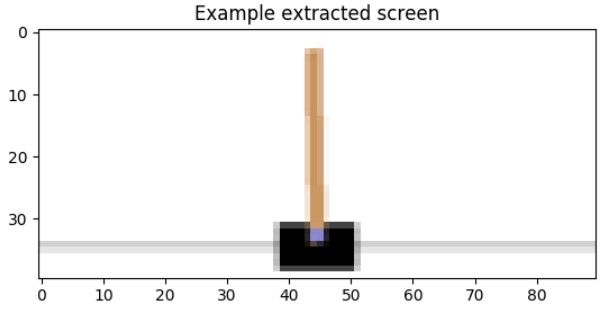

Figure 4. Sample image of Cart Pole balancing environment. [4]

V. Results/Analysis

While the Cart Pole balancing problem is a continuous one, it can be solved in both a discrete and continuous space. Discretization, however, requires that an approximation be used which can degrade the quality of learning. The Deep Q Network (DQN) and the Double Deep Q Network (DDQN) can only work in a discrete space. While both models are able to solve the problem, DDQN vastly outperforms the DQN model for two main reasons. Firstly, since discretization degrades the learning quality, for better performance further discretization is required if a continuous space cannot be used, DDQN allows us to do that. Secondly, both models make use of a replay function which makes a minibatch of randomly sampled elements from full memories, the size of which is equal to the batch size specified. The aim of this function is to ultimately converge, however, sometimes this does not easily happen in a DQN, this is however resolved in DDQN. DDQN trained significantly faster and could manage to balance the cart-pole for almost 350 time steps whereas DQN could at maximum achieve 70 time steps for mean of every 100 episodes.

For the continuous space we used the QNet and DDPG models. The QNet model used the first 100 episodes in its exploration phase where it interacted with the environment to figure out what the next state and reward should be. It also took advantage of the replay memory similar to the previous two implementations of the QLearning algorithms. Since the only major difference between QNet and the previous two models was the fact that it could operate in a continuous space and hence did not need to discretize the space first, it was able to run much faster than DQN and outperform it as well. DDPG was able to get more time steps at its maximum, more than 200, whereas QNet was only able to get a maximum of 120 timesteps before it plateaued. However, DDPG was not able to maintain it, keeping its average duration around 100 time steps.

There are several hyperparameters that can be tuned to improve the performance of these models, but in order to do a fair comparison between them they were run with their default parameters. All models were allowed to use the first 100 episodes for their exploratory phase, this means that for the first 100 episodes they chose actions randomly and simply recorded how this would affect the next state and reward in the replay memory. This is set by the epsilon parameter, and is then altered using the epsilon_decay parameter. The models do not have to keep the same number of explorations throughout, they can choose to decrease them as they start to learn and get more reward, hence getting better at the game.

The best way to evaluate the quality of learning is by measuring the duration of each episode, the longer an episode lasts the longer the model was able to keep the pole on the cart. All four algorithms, DQN, DDQN, QNet and DDPG were run for 1000 episodes. In Figure 5, Figure 6 and Figure 7, x-axis shows every 100th episode and y-axis shows the mean of durations achieved by every 100 episodes. Figure 8 shows the duration of every episode for the first 1000 episodes.

According to the durations DDQN outperformed all models but by checking its epsilon values we can also deduce that it had a few exploratory actions for quite a large number of episodes, only completely stopping exploration after 700 episodes. As a comparison, the QNet model completely stops exploration after only 200 episodes.

VI. Conclusion

We performed the Cart Pole Balancing problem through two different discrete space reinforcement learning algorithms: DQN and DDQN, and two different continuous reinforcement learning algorithms: QNet and DDPG. Overall the models for continuous space perform better, which is to be expected as the Cart Pole balancing problem is inherently a continuous problem. However, the model with the best performance was DDQN, which means that even thought discretization of the continuous space was unfavorable, through finer discretization it does not pose as much of a problem, hence removing the need for using continuous RL models. In the discrete space, even though both models are able to solve the problem DDQN has an advantage over DQN due

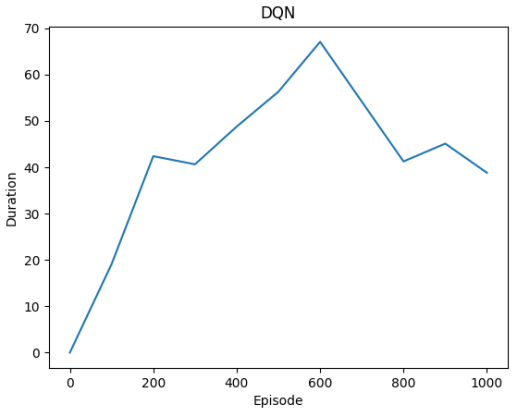

Figure 5. The graph between every 100 episodes and their corresponding mean duration the cart-pole was balanced by Deep Q-Network model

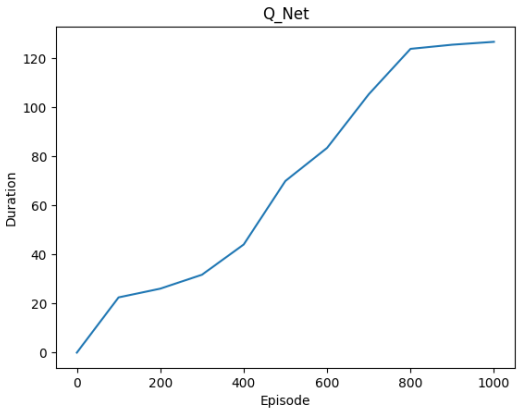

Figure 7. The graph between every 100 episodes and their corresponding mean duration the cart-pole was balanced by Q-Net model

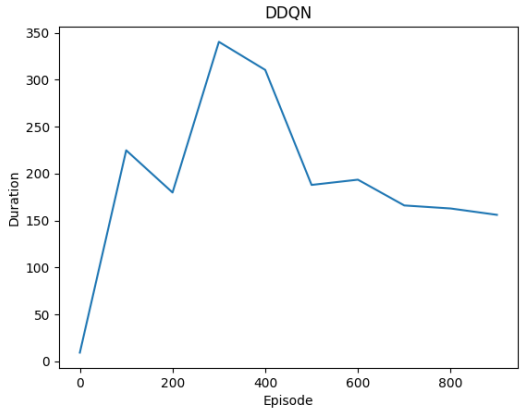

Figure 6. The graph between every 100 episodes and their corresponding mean duration the cart-pole was balanced by Double Deep Q-Network model

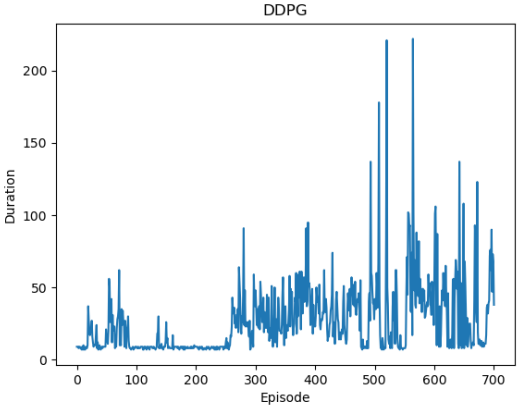

Figure 8. The graph between every episode and their corresponding duration the cart-pole was balanced by DDPG model

to this feature. In the continuous space, again both models are able to solve the problem but QNet outperforms DDPG by both duration and faster training time. These results can vary if all these models are further optimized such as by further parameter tuning or adding Prioritized Experience Replay (PER) [9]. This was not performed for our paper as we used all algorithms with their default parameters and without performing any optimizations to allow for a more equal comparison.

REFERENCES

[1] A beginners guide to q-learning. model-free reinforcement learning — by chathurangi shyalika — towards data science. https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c. (Accessed on 12/20/2021).

[2] Deep deterministic policy gradient (ddpg). https://keras.io/examples/rl/ddpg_pendulum/. (Accessed on 12/21/2021).

[3] Gym. https://gym.openai.com/envs/CartPole-v1/. (Accessed on 12/03/2021).

[4] Reinforcement learning (dqn) tutorial — pytorch tutorials 1.10.1+cu102 documentation. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html. (Accessed on 11/03/2021).

[5] Simple reinforcement learning with tensorflow part 4: Deep q-networks and beyond, by arthur juliani, medium. https://awjuliani.medium.com/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df. (Accessed on 12/18/2021).

[6] Swagat Kumar. Balancing a cartpole system with reinforcement learning – a tutorial, 2020.

[7] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[8] Savinay Nagendra, Nikhil Podila, Rashmi Ugarakhod, and Koshy George. Comparison of reinforcement learning algorithms applied to the cart-pole problem. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 26–32, 2017.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

[10] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR.