Deep Q-Networks

Sivashanmugam Saravanan Omprakash

Department of Mechanical & Aerospace Engineering

University at Buffalo (SUNY)

Buffalo, NY 14260

saravan3@buffalo.edu

Abstract

The goal is to explore & understand the value function approximation algorithms by working with OpenAI Gym environments. The first part involves the implementation of vanilla Deep Q-learning Network (DQN) and the second part involves the implementation of an improved version of DQN (i.e.) Double DQN in order to stabilize training & achieve better performance.

1. DQN & DDQN on Solving grid-world & Cart Pole environments

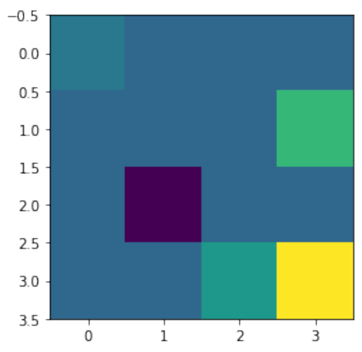
Theory

Everything the *agent* interacts with, comprising everything outside the agent, is called the *environment*. These interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent [1].

Report

	Environment	
	Number	**Set**
Action	4	{Left, Up, Right, Down}
States	16	{(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 1), (2, 2), (2, 3), (3, 0), (3, 1), (3, 2), (3, 3)}
Rewards	5	{-5, 1, 3, 5, 10}

Figure 1 shows the layout of the environment. There are 9 states, 4 actions allowed, & 4 reward signals placed in the environment.



🟦	-	Agent's position (+1)
🟦	-	Empty states
🟩	-	Coin position (+3)
🟩	-	Diamond position (+5)
🟪	-	Monster position (-5)
🟨	-	Goal position (+10)

Figure 1: Default environment layout

The size of the experience replay buffer can drastically affect the performance of the agent even in such simple settings. Irrespective of the buffer size, the memorization happens in a way that there will always be a relation between the 2 data. But when sampling the size of the buffer has a non-monotonic effect on the RL performance, meaning the random sampling will break the cohesion that exists in the data.

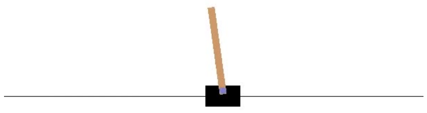
DQN: The main difference between analytical Q-learning method & DQN is that the Q-table is replaced by a neural network that is trained to approximate the Q-values. The input for this network will be the state to the observation & the number of neurons would be the number of actions that the agent can perform. For training purposes the targets (i.e.) the Q-values of the actions are used in correspondence to the agent's state.

DDQN: The improved version I've chosen is the Double DQN. In this method, there are 2 identical neural networks, where one learns from the experience replay and the other updates its weights from the previous model. But the Q-value prediction actually happens in the second network.

The main improvement over the Vanilla DQN is that, in the case of DQN, when maximum value is chosen over and over again, the difference between each output value is high. DDQN provides a solution to bring down the difference using the second model's Q prediction.

The OpenAI Gym's CartPole environment is a simpler gym environment in which a cart moves along a 1 dimensional floor while trying to keep the pole upright. The environment's parameters are shown in the following table.

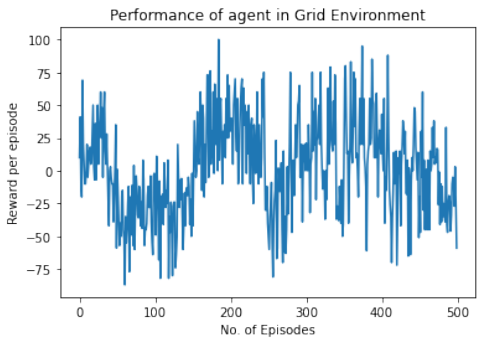
Actions	:	[-1, +1]
Rewards	:	1
States	:	Box[4,]

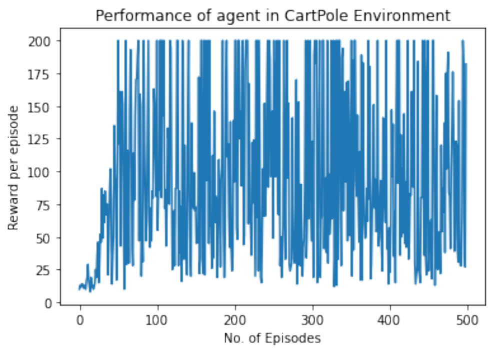


The complex environment (image-based) I've chosen is the Atari Breakout-v0. The objective in this game is to maximize our score by training a convolutional neural network to extract features from a preprocessed set of frames and reward accordingly.

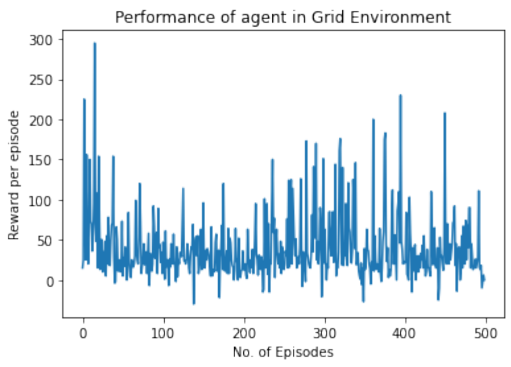
Actions	:	Discrete(4)
Rewards	:	Depends on break sequence
States	:	Box(0, 255, (210, 160, 3), uint8)

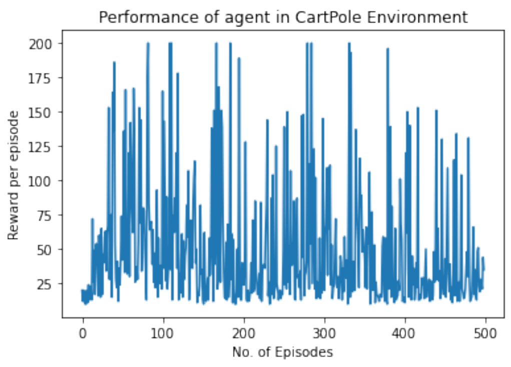
Performance of DQN agent





The above images show the performance of agent in grid and Cart Pole environments.The agent tends to start to converge in Cart Pole environment, and maybe when run for many more episodes the agent we could have seen the convergence. Hyper-parameter tuning in this case may prove effective. In the grid environment, the agent performs poorly and the reward obtained oscillates wildly.





The above image shows the performance of DDQN agent in grid & Cart Pole environments. In both cases, the agent is performing poorly and more iterations of execution is necessary to understand if the problem is with the update function as the overall logic seems to be in order.

2. References

1. Reinforcement Learning: An Introduction by Richard S. Sutton & Andrew G. Barto - http://incompleteideas.net/book/RLbook2020.pdf
2. Reinforcement Learning (DQN) Tutorial - https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html
3. Vanilla Deep Q Networks [Medium] - https://towardsdatascience.com/dqn-part-1-vanilla-deep-q-networks-6eb4a00febfb
4. Playing Atari with Reinforcement Learning - https://arxiv.org/abs/1312.5602
5. OpenAI Gym Environment [Breakout] - https://gym.openai.com/envs/Breakout-v0/
6. Deep Reinforcement Learning. Introduction. Deep Q Network (DQN) algorithm. [Medium] - https://medium.com/@markus.x.buchholz/deep-reinforcement-learning-introduction-deep-q-network-dqn-algorithm-fb74bf4d6862
7. Rendering OpenAi Gym in Google Colaboratory - https://star-ai.github.io/Rendering-OpenAi-Gym-in-Colaboratory/ and Notebook - https://colab.research.google.com/drive/1flu31ulJlgiRL1dnN2ir8wGh9p7Zij2t#scrollTo=8nj5sjsk15IT