Solving CartPole-v1 using Deep Q-Learning Methods and Actor Critic Algorithm

Letiushev Nikita

2022 Spring Semester

Abstract

This paper is based on my research regarding Deep Reinforcement Learning techniques. The report will compare reinforcement learning algorithms on OpenAI gym's CartPole-v1 environment, which are: Deep Q-Learning, Double Deep Q-Learning, Dueling Double Deep Q-Learning and Actor Critic method. The above mentioned algorithms will be implemented in Python on given environment, the research will study the results of algorithms and compare them with each other.

Part I

Introduction

CartPole environment corresponds to the version of the cart-pole problem described in Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem[1]. A pole is attached by an un-actuated joint to a cart, which moves along a friction less track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.

Part II

Related work

The theoretical implementation of algorithms are belong to the book Reinforcement Learning: An Introduction[2]. This book describes detailed algorithm implementation and pseudo code. In addition, how to apply these algorithms to the game is precisely described in the Hands-On Reinforcement Learning for Games [3]. The analysis of results was implemented by using Plotty Express tools and default Matplotlib libraries.

Part III
Problem description

Environment

Action space

The action is a ndarray with shape (1,) which can take values {0, 1} indicating the direction of the fixed force the cart is pushed with. Action 0 pushes cart to the left, action 1 pushes cart to the right. The velocity that is reduced or increased by the applied force is not fixed and it depends on the angle the pole is pointing. The center of gravity of the pole varies the amount of energy needed to move the cart underneath it.

Observation space

The observation is a ndarray with shape (4,) with the values corresponding to the following positions and velocities:

- Cart Position $\in [-4.8; +4.8]$
- Cart Velocity $\in (-\infty; +\infty)$
- Pole Angle $\in (-24°; +24°)$
- Pole Angular Velocity $\in (-\infty; +\infty)$

The cart x-position (index 0) can be take values between (-4.8, 4.8), but the episode terminates if the cart leaves the (-2.4, 2.4) range. The pole angle can be observed between (-.418, .418) radians (or ±24°), but the episode terminates if the pole angle is not in the range (-.2095, .2095) (or ±12°)

Rewards

Since the goal is to keep the pole upright for as long as possible, a reward of $+1$ for every step taken, including the termination step, is allotted. The threshold for rewards is 475 for v1.

Starting state

All observations are assigned a uniformly random value in (-0.05, 0.05)

Episode Termination

The episode terminates if any one of the following occurs:

- Pole Angle is greater than ±12°

- Cart Position is greater than ± 2.4 (center of the cart reaches the edge of the display)
- Episode length is greater than 500

Agent

The main goal of the agent is to solve the problem effectively - which means that it should has the most reward in the long run. The agent follows policy, it is the connection between action and state. The solution to this problem is to estimate the Q(s,a) value by using some approximation methods. These methods using neural network for the prediction of state-action value function. All of the using algorithms decide which action should the agent perform, the decision is improving by past experience. The main difference is how the agent will solve the problem. The paper will be focused on analysis of theirs performance.

Part IV
Methodology

Deep Q-Learning

The environment is deterministic, so all equations presented here are also formulated deterministically for the sake of simplicity. They would also contain expectations over stochastic transitions in the environment.

This algorithm trains a policy that tries to maximize the discounted, cumulative reward $R_{t0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$ where R_{t0} is also known as the return. The discount γ, should be a constant between 0 and 1 that ensures the sum converges. It makes rewards from the uncertain far future less important for our agent than the ones in the near future that it can be fairly confident about.

The main idea behind Q-learning is that if we had a function $Q* : State \times Action \to R$, that could tell us what our return would be, if we were to take an action in a given state, then we could easily construct a policy that maximizes our rewards:

$$\pi * (s) = argamaxQ * (s, a)$$

However, we don't know everything about the world, so we don't have access to $Q*$. But, since neural networks are universal function approximators, we can simply create one and train it to resemble $Q*$.

For our training update rule, we'll use a fact that every Q function for some policy obeys the Bellman equation:

$$Q\pi(s, a) = r + \gamma Q\pi(s\prime, \pi(s\prime))$$

The difference between the two sides of the equality is known as the temporal difference error, δ:

$$\delta = Q(s,a) - (r + \gamma amaxQ(s\prime, a))$$

To minimize this error, we will use the Huber loss. The Huber loss acts like the mean squared error when the error is small, but like the mean absolute error when the error is large - this makes it more robust to outliers when the estimates of Q are very noisy. We calculate this over a batch of transitions, B, sampled from the replay memory:

$$L = \mid B \mid 1(s,a,s\prime,r) \in B \sum L(\delta)$$

$$\text{where } L(\delta) = \begin{cases} 0.5\delta^2 for \mid \delta \mid \leq 1 \quad \mid \delta \mid -0.5 otherwise \end{cases}$$

Double DQN

The idea of double Q-learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation.

In the vanilla implementation, the action selection and action evaluation are coupled. We use the target-network to select the action and estimate the quality of the action at the same time. What does this mean?

The target-network calculates Q(s, a_i) for each possible action a_i in state s. The greedy policy decides upon the highest values Q(s, a_i) which selects action a_i. This means the target-network selects the action a_i and simultaneously evaluates its quality by calculating Q(s, a_i). Double Q-learning tries to decouple these procedures from one another.

In double Q-learning the TD-target looks like this:

$$Y_t^{DDQN} = r_{t+1} + \gamma Q(s_{t+1}, argmax_a Q(st+1, a, \theta_t), \theta^-)$$

Dueling DQN

Dueling DQN or DDQN extends the concept of a fixed target or fixed Q target and extends that to include a new concept called advantage. Advantage is a concept where we determine what additional value or advantage we may get by taking other actions. Ideally, we want to calculate advantage so that it includes all the other actions. We can do this with computational graphs by separating the layers into a calculation of state value and another that calculates the advantage from all the permutations of state and action.

$$Q(s,a,\theta,\alpha,\beta) = V(s,\theta,\beta) + (A(s,a,\theta,\alpha) - max_{a\prime \in |A|} A(s,a,\theta,\alpha))$$

Actor Critic method

Actor-Critic methods are temporal difference (TD) learning methods that represent the policy function independent of the value function.

A policy function (or policy) returns a probability distribution over actions that the agent can take based on the given state. A value function determines the expected return for an agent starting at a given state and acting according to a particular policy forever after.

In the Actor-Critic method, the policy is referred to as the actor that proposes a set of possible actions given a state, and the estimated value function is referred to as the critic, which evaluates actions taken by the actor based on the given policy.

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

Part V
Results

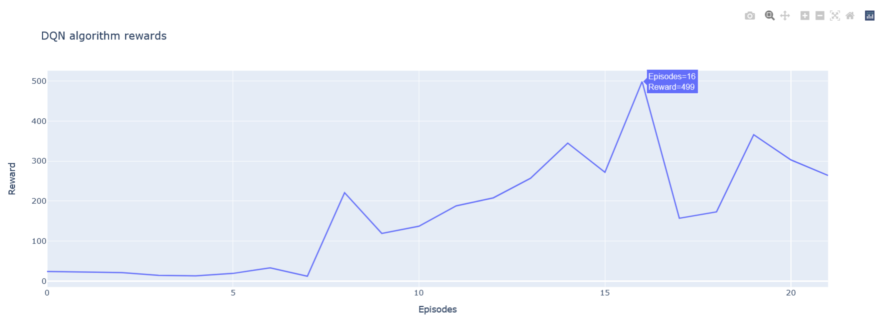
In my research the task can be considered solved when the agent reaches the reward around 500 several times with average reward more than 195. The most important issue in my implementation was the computation power, I had to find optimal proportion between time of training and condition for termination of training. Therefore, with this approach we can ensure that the model is training successfully and the agent found optimal policy.

Episodic rewards

One way of analyzing is to take a look on the graph of rewards for each episode in each algorithm.

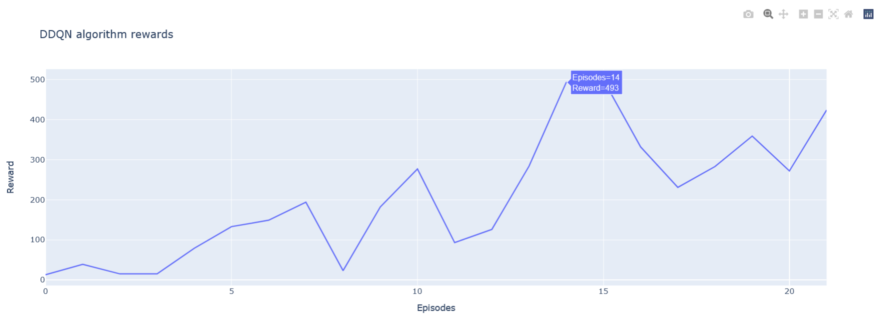
DQN

DQN was the most wasteful algorithm in my implementation. On the other hand, the learning was started on the 7th iteration with the maximal reward on 16th iteration:



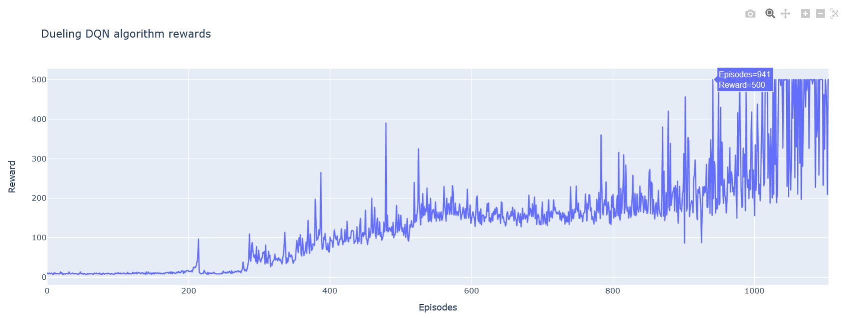
DDQN

Another wasteful algorithm was the DDQN. But in the overall it performs better than DQN. The learning was started on 3rd iteration with the maximal reward reaching on the 14th and 15th episode:



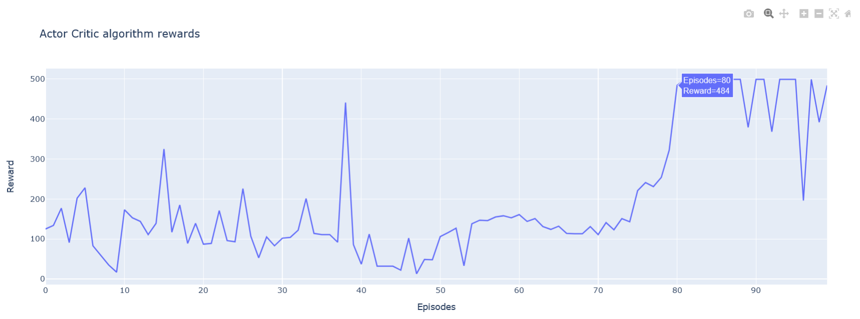
Dueling DQN

Dueling DQN shows the average time compilation with 1100 iteration, the learning was started approximately from 270th step, it was started to reaching maximal reward from 941th episode. The reason I was continuing to run it because of average results plotting, which I will explain in next section.



A2C

This algorithm had the minimal running time along with the fastest optimal policy finding. The learning was started from 73th episode, and the maximal reward was starting to reach from 80th episode:

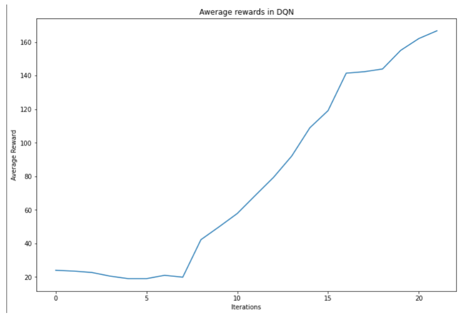


Mean rewards

The mean result is another tool which I used for analysis. I used iterative averaging method it calculates the mean result at each time step and plot it as a continuous function for each of the learning algorithm

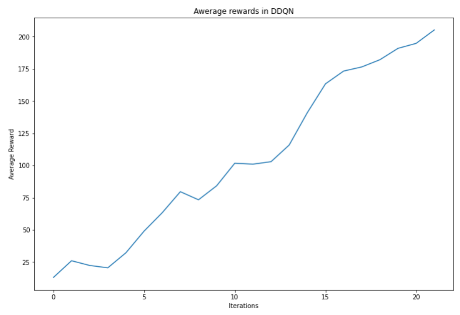
DQN

The mean result for DQN is the following. It seems like the learning was started with some delay, but from the 7th iteration it was started to grow almost linearly:



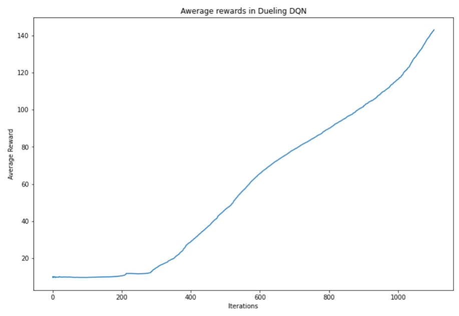
DDQN

The mean result for DDQN is slightly different: the average rewards were started to grow immediately with the average reward more than 195 at 19th iteration



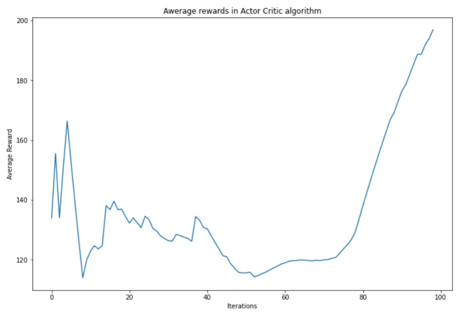
Dueling DQN

The mean result of Dueling DQN is the most slight and smooth with the average more that 145 on 1100th iteration:



A2C

The average reward for A2C was the most fleshly, but at the same the reward greater than 195 was achieved on the 100th step:



Part VI
Conclusion

In this project multiple deep Q-Learning and Actor Critic algorithms were implemented and tested on the OpenAI's CartPole-v1 environment. It was shown that multiple factors can make great influence on performance, e.g. network architecture. DQN and DDQN methods turned out to be the most wasteful and slow algorithms, but they found the optimal solution faster in terms of steps. Although Dueling DQN and A2C turned out to be the most flexible and universal algorithm with the optimal balance between time cost and learning rate. These algorithms still can be improved by tuning number of neural layers or dimension of above mentioned layers.

References

[1] Andrew G. Barto; Richard S. Sutton; Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems, 1983.

[2] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. Second edition, in progress, 2014, 2015.

[3] Micheal Lanham. Hands-On Reinforcement Learning for Games, 2020.