COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Applying Deep Q-Learning Network to Solve Cart-pole Control Problem

ELEN-6885 | Kangrui Li kl3350 Yunhe Liu yl4879
Xiaohang He xh2509 Zheyuan Song zs2527

***Abstract*—In this paper, we provide a variety of reinforcement learning (RL) algorithms to control the Cart-Pole system. What's more, we explore some RL concepts such as Q-learning, Deep Q Networks (DQN), Double DQN, Dueling networks, (prioritized) experience replay. Focus on the deep Q-learning network architecture in our project. Based on OpenAI gym library, we use the 'CartPole-v0' and 'CartPole-v1' environment to test our designed Double-DQN algorithm performance.**

***Keywords*—Reinforcement learinng, Q-learning, Deep Q-learning, Cart-pole Control system, DQN, DDQN.**

1. Introduction

Reinforcement learning could be considered as a semi-supervised learning approach where the supervision signal required for training the model is made available indirectly in the form of rewards provided by the environment. Reinforcement learning is more suitable for learning dynamic behaviour of an agent interacting with an environment rather than learning static mappings between two sets of input and output variables. Over the years, a number of reinforcement learning methods and architectures have been proposed with varying success. However, the recent success of deep learning algorithms has revived the field of reinforcement learning finding renewed interest among researchers who are now successfully applying this to solve very complex problems which were considered intractable earlier [1].

Reinforcement learning is a general framework where agents learn to perform actions in an environment so as to maximize a reward. The two main components are the environment, which represents the problem to be solved, and the agent, which represents the learning algorithm. The agent and environment continuously interact with each other. At each time step, the agent takes an action on the environment based on its policy $\pi(a_t | s_t)$, where s_t is the current observation from the environment, and receives a reward r_{t+1} and the next observation s_{t+1} from the environment. The goal is to improve the policy so as to maximize the sum of rewards (return).

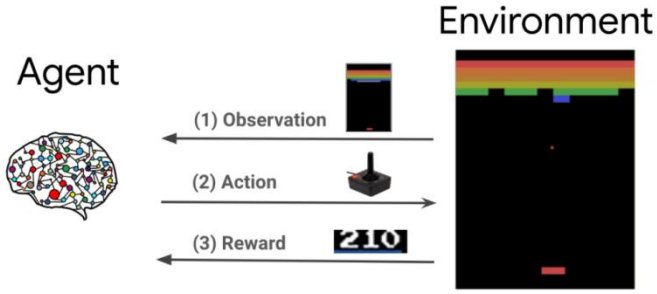

Figure1-1: A general framework which can model a variety of sequential decision making problems such as games, robotics etc.

Reinforcement learning is a key paradigm to build such intelligent systems which can learn from its experience over time. Reinforcement algorithms are now being increasingly applied to Robotics, healthcare, recommender system, data centres, smart grids, stock markets and transportation [2].

The Cartpole environment is one of the most well known classic reinforcement learning problems. It is just like the RL's "Hello, World!". A pole is attached to a cart, which can move along a frictionless track. The pole starts upright and the goal is to prevent it from falling over by controlling the cart. CartPole problem is one of the most common problems for testing reinforcement learning algorithms. If you have ever tried to balance a pen on its tip in the palm of your hand, then you will be familiar with the CartPole problem.

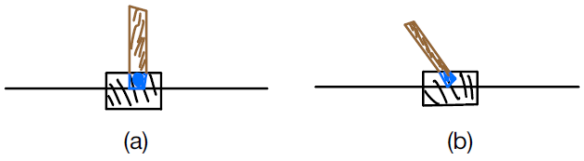

Figure1-2: A Cart-Pole System: (a) Balanced state, (b) Unbalanced state.

We will learn Q-learning [3] and then using Deep Q network (DQN) [4] and DDQN to control a Cart Pole system. Nowadays，these reinforcement learning methods are very well known. This project is very useful for it can improve our

practical skills. We can implement several reinforcement learning concepts such as DQN, experience replay and DDQN by using Python, OpenAI/Gym.

2. Background

Among of some foundation ideas of deep learning and reinforcement learning algorithms, Deep Q-network is a great and powerful model for us. We will review some reinforcement learning's basic algorithms and the learn about the Q-network and DDQN. Then we can deal with Cart Pole problem in the Gym platform by using deep Q-network and DDQN methods. We can test our understanding in this case study.

2-1. Review of Reinforcement Learning

A machine learning paradigm where an agent learns the optimal action for a given task through its repeated interaction with a dynamic environment that either rewards or punishes the agent's action is called Reinforcement Learning (RL). Agent which means a decision maker will be in an environment will have an observation of the environment. These actions called the state of the environment.

Agent can affect the environment by taking actions based on the observation. Each agent's behaviour will under each behaviour policy's control, which maps the observation of states into probabilities of the agent taking each action. What's more, agent always chooses actions which can maximize total reward.

2-2. DQN

Begin with the lookup table, neural network comes from here. What's more, Deep Q-Learning belonged to one of the Reinforcement Learning (RL) methods only worked with a lookup table when time comes to nearby 2013. Due to the great success of neural networks in computer vision, people gradually became interested in the idea of trying neural networks in RL. Scientist V.Mnih presented the first successful Deep Learning model using a neural network function approximation. Two years after 2013, DeepMind claimed that the Deep Q-network agent, receiving only the row pixel data and the game score as inputs, was able to exceed the performance of all previous algorithms. After these two DeepMind works has been done, the lookup tables were replaced by neural networks and a Q-Learning became a Deep Q-Learning, which equals to the Deep Q-Network (DQN). In fact, DQN is meaningful for the RL agent training for it achieved a new breakthrough.

The DQN is the algorithm that combines Q-learning with neural networks.

Q-learning + Neural Networks = DQN

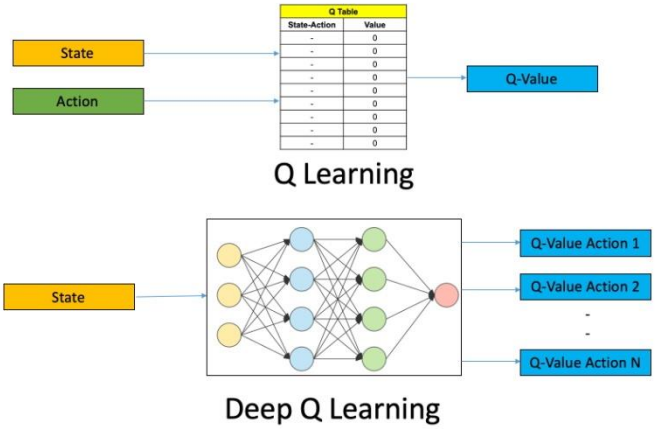

Figure2-1: Comparison between Q-learning & deep Q-learning

3. Approach: Algorithm Development

In this section, we will discuss the deep Q-learning network architecture in our project. Based on OpenAI gym library, we use the 'CartPole-v0' and 'CartPole-v1' environment to test our designed Double-DQN algorithm performance. We completed agent training on Jupyter Notebook, the graphics card used is RTX 2060 and the Tensor Flow version is 2.5.

3-1. Environment and System

CartPole is a very famous control problem in Reinforcement Learning topics and it is also an environment in OpenAI. In such a situation, a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every time step that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

3-2. Q-Learning Network

The DQN is the combination of deep network and Q-learning. In previous experience, the training of neural networks is an optimization problem. First, we need a large amount of labeled data to keep the neural network training values close to the true values during the training process. In DQN, it is to make the value obtained from training constantly close to the optimal Q value.

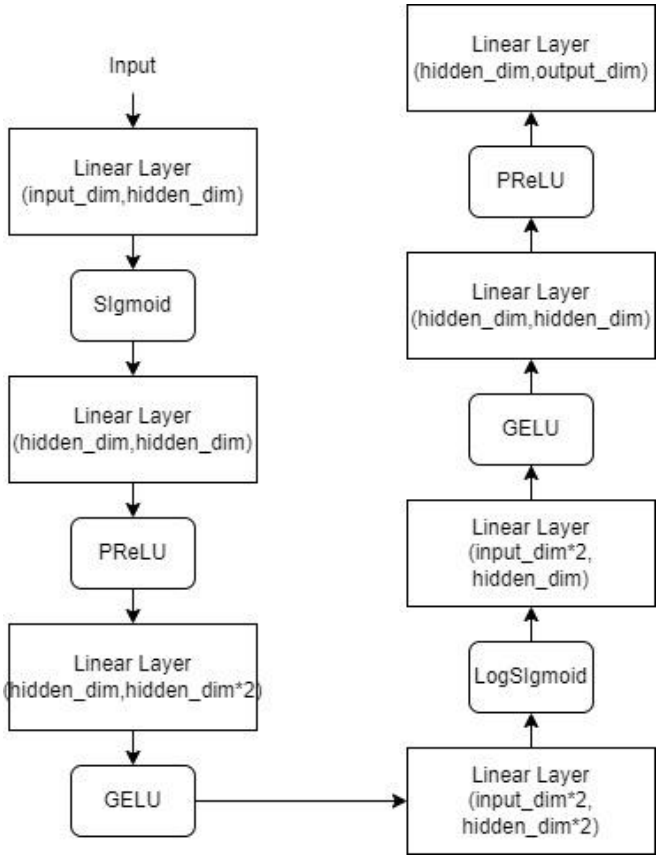

Figure3-1: Q-learning Network Architecture.

3-3. DDQN-Agent

In our project, we designed a DDQN-agent to learn to keep control of the simple cartpole. There are mainly three parts in this agent, initial parameters and variables definition, action function and learning approach. In initialization part, a memory is established for storing data. Old Q value and new value are initialized with Q-learning network. Loss function like 'Mean-Square Entropy' function and optimizer like 'Adam' is defined for learning part.

In 'get_action' part, the agent simply follows epsilon-greedy policy to take action. The DDQN epsilon-greedy mechanism promotes a reasonable proportion between exploration and exploitation, which means, that the agent finds the following action by maximizing the Q-value over all possible actions for the given state, see the relevant lines from the function.

DQN components applied the targeted network, which is the same as local network except that its parameters are copied every step from the local network. In the learning function, tuples of actions, states and other elements are obtained and Q value in neural network is calculated. The distance between 'Q_targets' and 'Q_expected' is calculated by 'MSELoss' function and the loss value is minimized by gradient descend method.

Compared with DQN, DDQN involves decoupling process by applying another network to deal with the set of weights. There is a problem in Q-learning that for all actions, each Q(st,at) update simultaneously causes an update of Q(st+1,at), which causes a change in the target value. In contrast, in the newly proposed method, after each update of C, the Q network is first copied and noted as Q hat. After C updates Q, Q hat is used to update the target network. In this way, by using separate networks, the stability of the algorithm is guaranteed.

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Figure3-2: DDQN pseudo code.

3-4. Training Algorithm

In our code implementation, first we import some key library like 'gym', 'torch', etc. Then we initialize the GPU to make training faster. We also set some hyper parameters like learning rate, hidden dimensions, and target update. For environment, we choose both 'CartPole-v0' and 'CartPole-v1', from which we obtained some constraints on our input, output, and score threshold. We also define a save function so that we can save the weights after training and use them to control the balance of the cartpole smartly.

In our training we use epsilon decreasing to lower epsilon value while using epsilon-greedy policy to exploit. In each episode, when the state is not done, action value, next state, and reward are obtained and the agent will use experience replay to push data by being randomly selected from the memory storage.

For training process (in each training iteration), starting from the largest epsilon value, score is obtained for running 50 episodes every time. For every training process, the average score of all the episodes in that process is calculated and compared with threshold. If it is beyond the threshold value, agent training is finished and this cartpole can learn to keep balance by itself.

4. Experiment Results

In the project, we conducted experiment on two cartpole environments with different hyper parameter settings. And we saved training weights to show how smart the agent could be after training.

4-1. DDQN with CartPole-v0

Setting optimizer to be ‘Adam’ and learning rate = 0.001, we get the Score and Average Score results of this training:

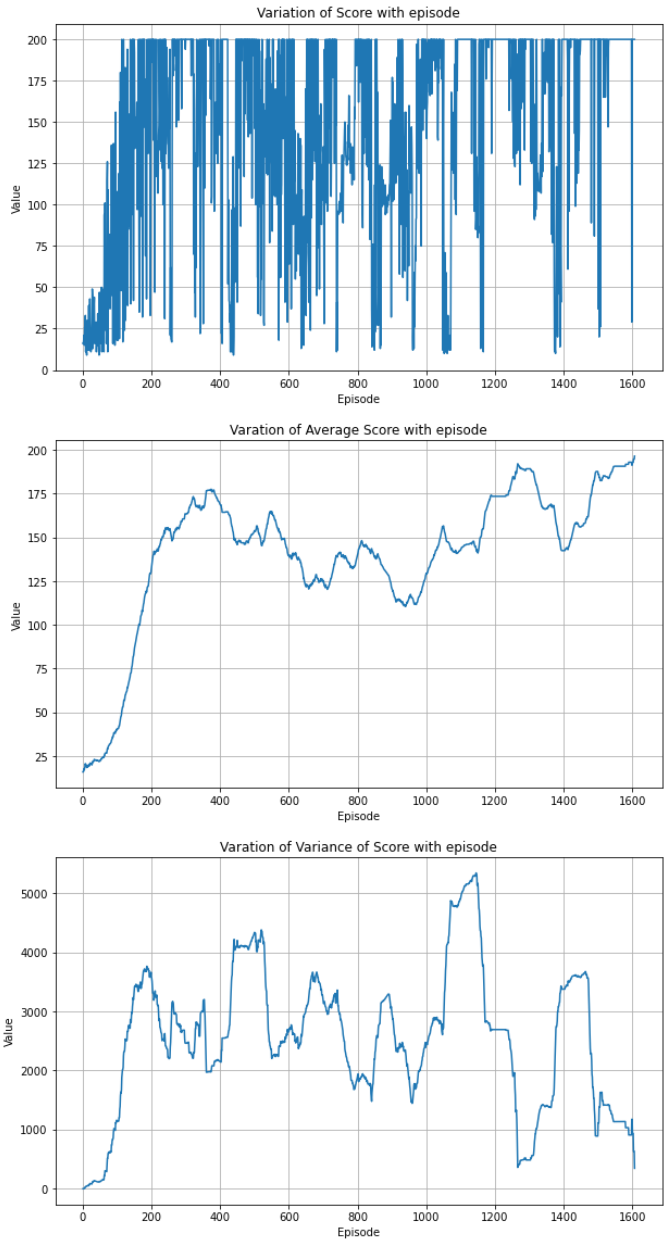

Figure4-1: Score and Average Score results when optimizer is ‘Adam’ and learning rate is 0.001.

Setting optimizer to be ‘Adam’ and learning rate = 0.0001, we get the Score and Average Score results of this training:

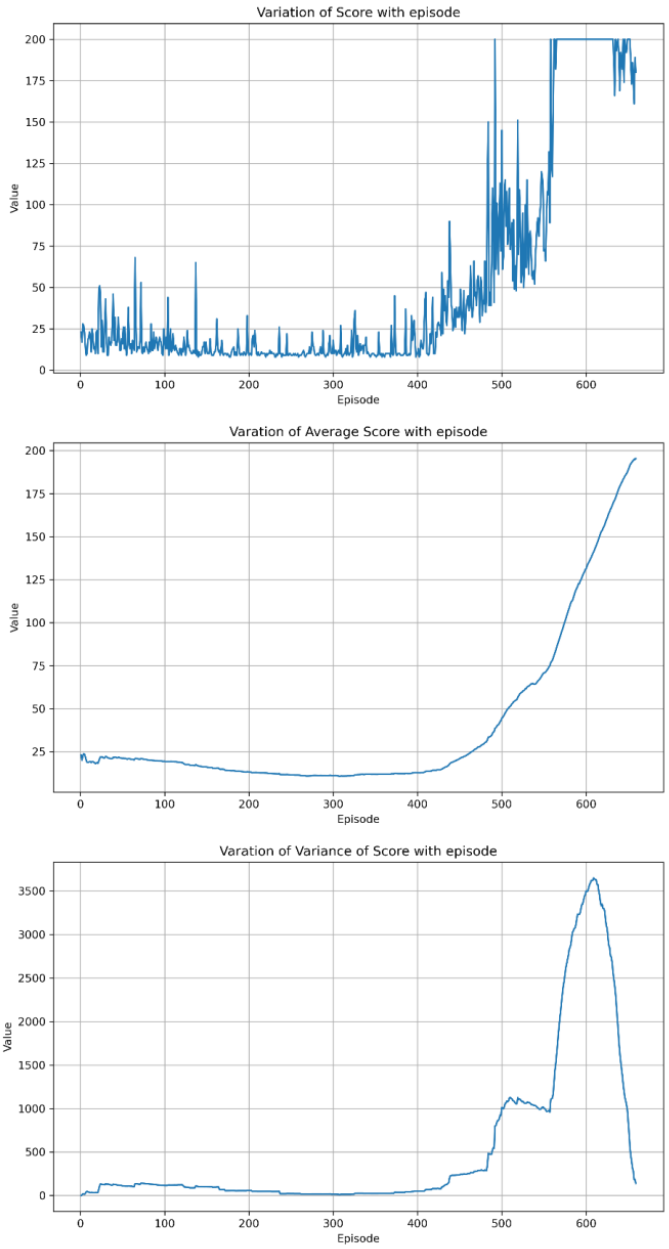

Figure4-2: Score and Average Score results when optimizer is ‘Adam’ and learning rate is 0.0001.

4-2. DDQN with CartPole-v1

Setting optimizer to be ‘RMSprop’ and learning rate = 0.001, we get the Score and Average Score results of this training:

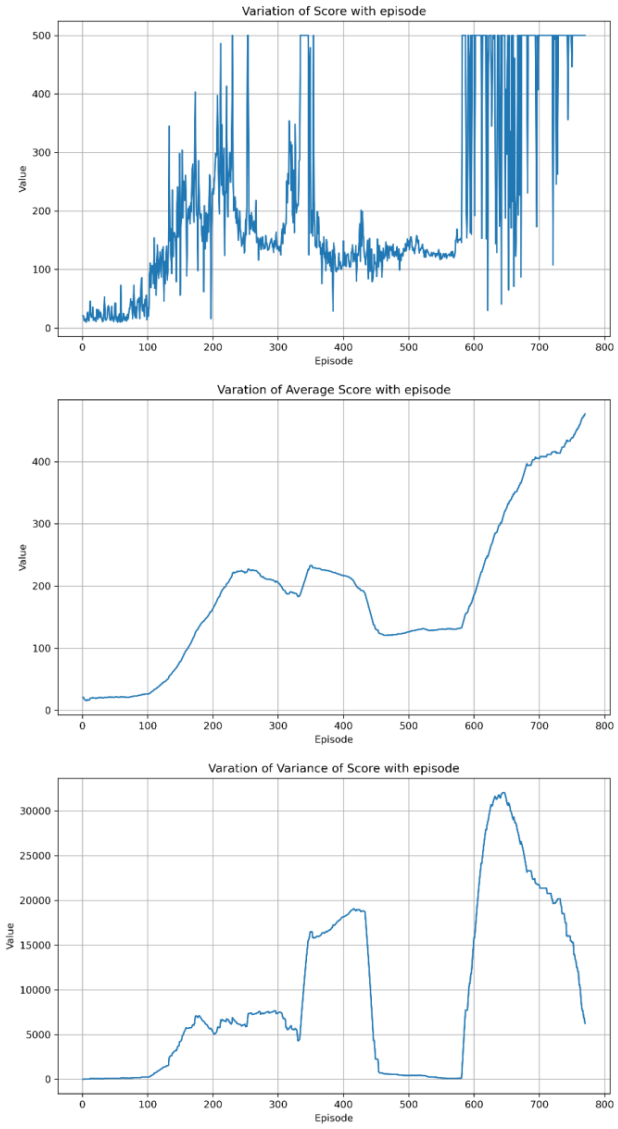

Figure4-3: Score and Average Score results when optimizer is ‘RMSprop’ and learning rate is 0.001.

\

Setting optimizer to be ‘RMSprop’ and learning rate = 0.0001, we get the Score, Average Score and Variance results of this training:

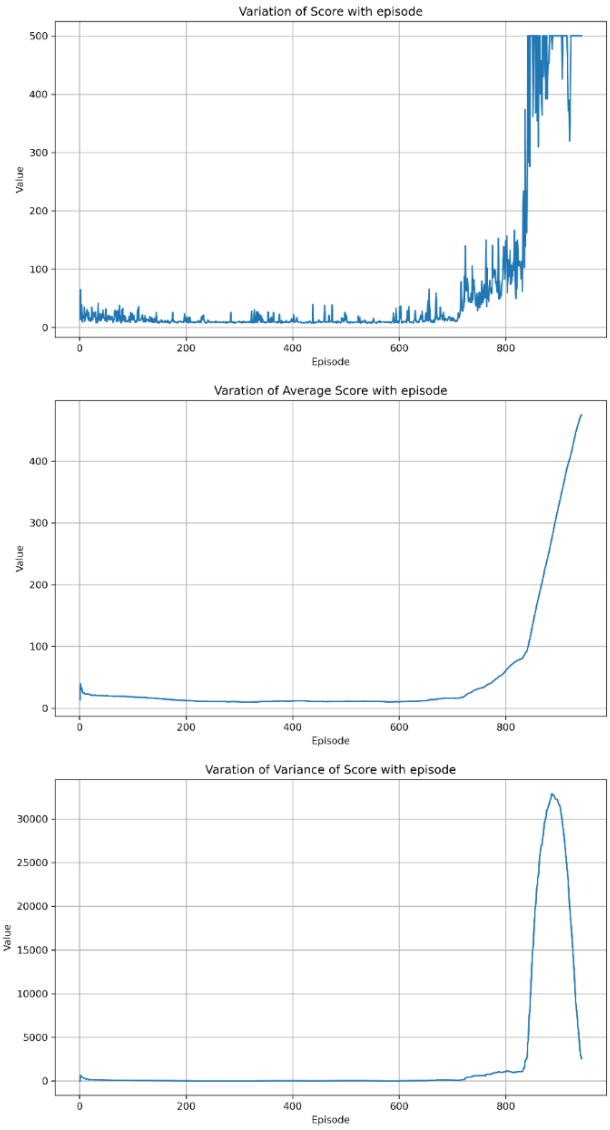

Figure4-4: Score and Average Score results when optimizer is ‘RMSprop’ and learning rate is 0.0001.

4-3. Saved Weights Performance

With pretrained weights, the agent plays well and can get the best mark in every episode in both v0 and v1 environments:
In Cartpole-v0:

We selected the trained weights obtained when lr is 0.0001 and the training converges at 658 episodes, here we get such results:

```
Episode 1       Average Score: 200.00,   Timesteps: 200        Time: 00:00:02
Episode 2       Average Score: 200.00,   Timesteps: 200        Time: 00:00:01
Episode 3       Average Score: 200.00,   Timesteps: 200        Time: 00:00:01
Episode 4       Average Score: 200.00,   Timesteps: 200        Time: 00:00:01
Episode 5       Average Score: 200.00,   Timesteps: 200        Time: 00:00:01
```

Figure4-5: Results when lr is 0.0001 and the training converges at 658 episodes.

In Cartpole-v1:

We used trained weights obtained when lr is 0.0001 and length of episodes is 769, here we get the play performance by the agent:

```
Episode 1       Average Score: 500.00,   Timesteps: 500        Time: 00:00:03
Episode 2       Average Score: 500.00,   Timesteps: 500        Time: 00:00:03
Episode 3       Average Score: 500.00,   Timesteps: 500        Time: 00:00:03
Episode 4       Average Score: 500.00,   Timesteps: 500        Time: 00:00:03
Episode 5       Average Score: 500.00,   Timesteps: 500        Time: 00:00:03
```

Figure4-6: Results when lr is 0.0001 and length of episodes is 769.

5. Conclusion

In our project, we implemented reinforcement learning to solve control problem for cartpoles and we achieved good performance. For 'cartpole-v0', we set learning rate to 0.001 and 0.0001 respectively and results show that training converged at 1606 episodes, which is a little slow and 658 episodes, which is an optimal solution with fast convergence speed. And agent could achieve high score and low convergence speed when using high learning rate.

For 'cartpole-v1', a higher threshold score with 475 is needed and we got training convergence at 769 episodes and 991 episodes respectively, showing that a higher learning rate increased training speed while a lower learning rate converges slowly but the trained weights can successfully guide the agent to make the right move. With all the weights obtained from training, we make agent move smarter and agent can keep balance in both two environments at the very beginning of its play.

The objective of this course is to provide students with required knowledge to conduct research in reinforcement learning and basic skills to apply reinforcement learning techniques to real-world applications.

Reinforcement learning becomes much more interesting in the past few years and it is very powerful when it combined with deep learning and modern hardware. DQN plays a significant part during the simpler environments in OpenAI's gym.

Double DQN Networks, Dueling DQN and Prioritized Experience replay are more advanced Deep RL techniques and they can improve the learning process efficiently. These techniques can help us to gain more useful and meaningful results. Furthermore, we can try more environments other than Cart Pole with DQN or DDQN algorithm to get better or the best model.

Appendix

Contribution:

Kangrui Li(25%)： Building & training agent +writing
Yunhe Liu(25%)： Building & training agent +writing
Xiaohang He(25%): Building & training agent +writing
Zheyuan Song(25%): Building & training agent +writing

Acknowledgements

We would like to express our gratitude here to Prof. Chong Li and Prof. Chonggang Wang in the Electrical Engineering Department of Columbia University, who provided amount of Reinforcement Learning knowledge and significant suggestions for us.

References

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6):26–38, 2017.
[2] Y. Li. Reinforcement learning applications. arXiv preprint arXiv:1908.06973, 2019.
[3] C. J. Watkins and P. Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.
[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.
[5] Reinforcement Learning: An Introduction, by Richard S. Sutton and Andrew G. Barto. MIT Press, Cambridge MA. (Second edition).
[6] Reinforcement Learning for Cyber-physical Systems with Cybersecurity Case Study, by C. Li and M. Qiu, CRC Press, 2019.
[7] Dynamic Programming and Optimal Control, by D.P. Bertsekas, 2 Vols., Athena Scientific Press, 2005 .
[8] Algorithms for Reinforcement learning, by Csaba Szepesvari, Morgan & Claypool Publishers, 2010.
[9] REINFORCEMENT LEARNING (DQN) TUTORIAL- Adam Paszke
[10] V.Minh et. al., Playing Atari with Deep Reinforcement Learning (2013), arXiv:1312.5602
[11] H.van Hasselt et. al., Deep Reinforcement Learning with Double Q-learning (2015), arXiv:1509.06461
[12] S.Karagiannakos, Taking Deep Q Networks a step further, (2018), TheAISummer
[13] F.Mutsch, CartPole with Q-Learning — First experiences with OpenAI Gym (2017), muetsch.io
[14] T.Seno, Welcome to Deep Reinforcement Learning Part 1 : DQN, (2017), TowardsDataScience
[15] Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Thirtieth AAAI conference on artificial intelligence.