Playing DOOM with Deep Reinforcement Learning

Yueqiu Sun
New York University
Center for Data Science
ys3202@nyu.edu

Ruofan Wang
New York University
Center for Data Science
rw2268@nyu.edu

Abstract

Deep reinforcement learning has been widely used in video game playing. In this project, we trained agents to play scenario 'search and hit' and scenario 'health gathering' in game DOOM with deep reinforcement learning model including Deep Q Learning with prioritized experience replay and policy gradient. Additionally we implemented and evaluated the performance of the agents that were trained with several different exploration strategies including random policy, ϵ-greedy, and Boltzmann policy.

To conclude, the agent for the 'search and hit' scenario achieved almost perfect performance, and the agent for the 'health gathering' scenario performed far better than the baseline model. We also conclude that randomness in exploration policy enables agents to gather more information about the environment and is important in training agents.

1 Introduction

DOOM is the first shooter video game in the world. In the most basic version of this game, the player can choose to move left, move right, or shoot. The goal for the player is to hit the target. Within the game, there exist many different scenarios , including *health gathering*, where the player survive by picking up some certain items called 'medkits' that can heal some portions of the player's health, and *search and hit*, where the player aims at hitting the target as soon as possible.

In our project, We train an agent to perform well in the above scenarios in DOOM with Deep Reinforcement Learning. Our agent will obtain information from the environment by taking action and receiving feedback from the environment, namely the next state and the rewards. The feedback can help agent to improve their policy to generate actions(policy-based algorithm) or improve their understanding of the Q values of the state(value-based algorithm). An example of the two scenarios can be shown in Figure 1 and Figure 2,

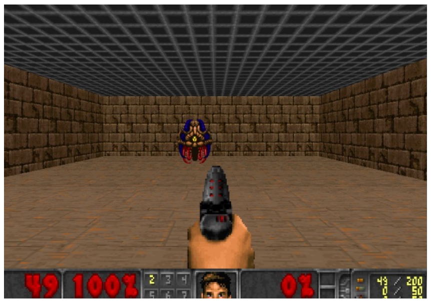


Figure 1: search and hit

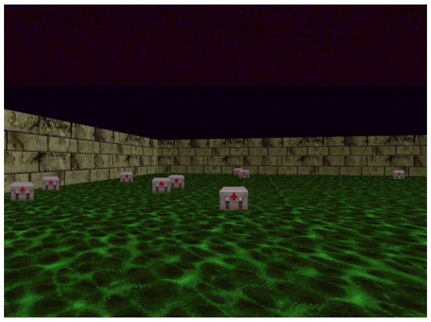


Figure 2: health gathering

For the two scenarios, we implemented different algorithms to solve the problem. The model architecture, results, and analysis are covered in the next several parts.

2 Related Work

2.1 Reinforcement Learning

Reinforcement learning(1) is a framework containing an environment and an agent. The agent interacts with the environment according to the current state. During the interaction, the agent makes an action and a reward term is generated immediately. During the training process, the agent learns to take actions to maximize the expected total reward. The difference between traditional supervised learning and reinforcement learning is, in supervised learning, the machine learn from the 'teacher', while in reinforcement learning, the machine(i.e. agent)learn from past experience.

There are two basic prototypes of reinforcement learning: policy-based and value-based. Roughly speaking, in policy-based learning, the machine learns an actor to make actions, in value-based learning, the machine learns a critic to do evaluation. There are also some more advanced types proposed these years. For example, A3C(Actor + Critic) (7) is a combination of these two.

2.2 Reinforcement Learning for Game Playing

In the field of deep reinforcement learning, one of the most significant work was DeepMind's paper in 2013(2). In this paper, they proposed to train an agent to play deep Atari games with reinforcement learning. This is the very beginning of incorporating computer vision with reinforcement learning, where we can use the visual data as input to the model. The performance of the reinforcement learning algorithm surpasses the human performance in most Atari game . From then on, people have started to use deep reinforcement learning to play more and more complicated video games, like Flappy Birds(3), Geometry Dash(4), etc. and have obtained very good results.

2.3 Deep Reinforcement Learning for playing DOOM

The beginning of using deep reinforcement learning to play DOOM is the publishment of ViZDoom (5) in 2016. ViZDoom is an AI platform for reinforcement learning from raw visual information. It is primarily intended for machine learning and deep learning research in the video game playing field. The main aim of it was to create an easy-to-use environment (6)with which people can build machine learning models to train agents that act and learn based only on visible data from the game.

We trained our agents using the environment from ViZDoom with DQN and policy gradient. Additionally, we made improvements to the baseline model to get better results.

3 Dataset

As mentioned in the previous section, the environment in the ViZDoom package can provide us with the frames of each step as the agent plays the game, which can be used as the input to a neural network to extract high-level features.

In Seita's work(9), a new idea called 'stacking frames' was raised. Stacking frame is a fixed size deque consisting of the most recent frames. The number of frames is determined by the size of the deque. Stacking frame is powerful in this case because it can offer a sense of motion to the agent during the game.

This is how it works, assuming the size of the deque is 4:

- In the beginning of each episode, we initialize a deque of fixed size 4 and feed in 4 identical initial frames of size $84 * 84$.
- Within each time step of this episode, we firstly process the picture to get a $84 * 84$ pixel matrix, then feed the new matrix into our deque and remove the oldest one from our deque.

4 Methods

4.1 Scenario 'Search and Hit': Deep Q-Learning

4.1.1 Deep Q Learning

For this task, we used value-based reinforcement learning model: Deep Q Learning(2). As mentioned earlier, in value-based models, the machine will learn a critic to evaluate every actions.

In Q Learning, the neural network aims to approximate a function:

$$Q^*(s,a) = max_\pi E[r_+ \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi],$$

where s_t represents the state at time step t, r_t represents the reward at time step t, and γ is the discount rate. Therefore, the Q function at the time step t represents the highest expected discounted sum of future rewards.

The Q function can be written in the recursive fashion:

$$Q^*(s,a) = E_{s'}[r + \gamma max_{a'} Q^*(s', a') | s, a],$$

where s represents the current state and a is the action, s' is the resulting state and r is the corresponding reward. The result of the recursive calculation is denoted as Q-value of the corresponding state and action.

4.1.2 Model Architecture

The architecture of our model is as follows:

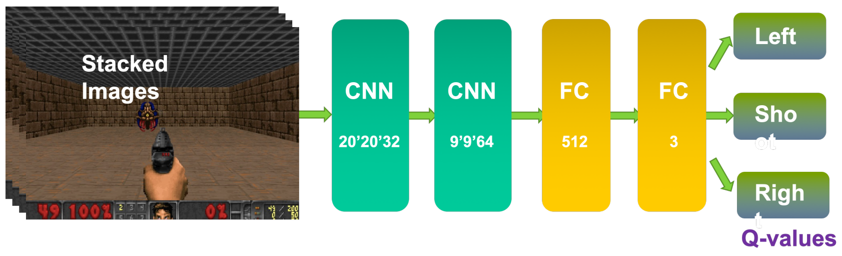


Figure 3: Deep Q Network Architecture

As we can see, we firstly stack the images, then feed the stacked images into convolution layers. After the the convolution layers, we flatten the output and feed it into fully-connected layers. We calculate the Q-value for each of the action: left, right, and move. Then we make a decision and continue to the next step.

4.1.3 Exploration v.s. Exploitation

In reinforcement learning, exploration(8) and exploitation are two important components in decision making. Exploration allows the agent to gather more information via obtaining feedback from the environment by executing non-optimal actions, while during exploitation agent choose the optimal action(the action with largest Q value) given the current environment state. How to balance between this two is very interesting and critical.

Generally speaking, we should try to make the best overall long-term strategy, which means that we need to involve short–term sacrifices along the way.(10)

There are many strategies about this, including random policy, ϵ-greedy, and Boltzmann policy.

- Random-Policy: Under Random Policy, the agent will take an action sampled from an uniform distribution during training. In other words, the policy(action) is purely random.
- ϵ-greedy: In this case, we have a threshold ϵ range from 0 to 1. We have two different strategies with probability ϵ and $1 - \epsilon$. With probability ϵ, we still take an action randomly(uniform distributed). In contrast, with probability $1 - \epsilon$, we choose the action with the largest Q value.
- Boltzmann Policy: As we can see, in the ϵ-greedy approach, when making a decision using the Q value, the result is deterministic, which mean the agent will always make the same decision when facing a single state. This is not what we want because we also expect our agent to explore the new environment and take new actions to gather more information. In order to solve this problem, we use Gibbs(Boltzmann) distribution here. The distribution is as follows:

  $$P(a) = \frac{e^{Q(a)\theta^{-1}}}{\sum_{a':allactions} e^{Q(a')\theta^{-1}}}$$

  Essentially, we feed the Q values of each action into a softmax distribution. There is also another parameter called gain factor β, which is also called the temperature. It is a positive number representing the randomness in the given system(action making procedure).
  If β goes to infinity, it means the system is totally random and the Boltzmann distribution will converge to an uniform distribution. If β converges to 0, the decision making will be like the deterministic way as we mentioned before.
  Every time step, we sample from the Boltzmann distribution to get an action from our action space.

In our work, we implemented the Deep Q Learning network as well as all of the exploration strategies. The results, comparison and our analysis will be covered in the next section.

4.2 Scenario 'Health Gathering': Policy Gradient

In the health gathering scenario, we implemented a policy gradient(7) algorithm to train the agent. In policy gradient algorithm, the agent will learn a policy function which can map a environment state to the probability of each action.

The idea behind it is that we parameterize the policy function π with parameter θ . We use the expected reward as our objective function and optimize the function using gradient ascent to find the local maximum.

When making a decision, there are two methods: deterministic and stochastic(11). In deterministic setting, the agent will make a single action. The result is certain and it is kind of like the "hard assignment". In contrast, in stochastic setting, the agent will make actions according to a probability distribution, which means that the result is uncertain. So it is more like the "hard assignment".

In policy gradient, the agent uses stochastic method to make a decision each time. Then we will talk more about the specific details in policy gradient networks.

4.2.1 Objective Function

As mentioned earlier, we use the expected total rewards as our objective function and attempt to maximize it.

$$E(R_\theta) = \sum_\tau R_\theta(\tau)P(\tau|\theta),$$

where each τ is an episode containing a sequence of scenes.

In practice, it is very hard to calculate the expectation of the summed rewards since it is impossible to integrate over all τs. So what we actually do is sampling a lot of τs and use the empirical result to estimate the expectation.

So our objective function becomes:

$$E(R_\theta) = \frac{1}{N}\sum_{n=1}^{N} R_\theta(\tau^n),$$

where we use π_θ to play N games to obtain N episodes: $\tau^1, \tau^2, ..., \tau^N$

4.2.2 Gradient Ascent

In order to maximize the objective function(empirical expected total rewards), we use the gradient ascent method.

We can calculate the gradient as follows:

$$\begin{aligned}\nabla R_\theta &= \sum_\tau R_\tau \nabla P(\tau|\theta)\\ &= \sum_\tau R_\tau P(\tau|\theta)\nabla logP(\tau|\theta)\\ &= \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\nabla logP(\tau|\theta)\end{aligned}$$

For the log probability, we can expand it as:

$$logP(\tau^n|\theta) = logP(s_1) + \sum_{t=1}^{T} P(a_t^n|s_t^n, \theta) + \sum_{t=1}^{T} logP(r_t, s_{t+1}|s_t, a_t)$$

Only the second term is related to θ, so we can get our gradient as:

$$\nabla logP(\tau|\theta) = \sum_{t=1}^{T} \nabla P(a_t^n|s_t^n, \theta)$$

Therefore,

$$\nabla R_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} R(\tau^n)\nabla P(a_t^n|s_t^n, \theta)$$

5 Experiments and Results

5.1 Input Image Prepossessing

For each input image, we perform the following input image prepossessing steps,

- The input image is converted from RBG image to greyscale image.
- The input image is cropped to remove the roof as it contains no information to help agents play the game.
- The input image is normalized by compressing the pixel value to be between 1 and 0.
- The input image is reshaped to 84x84.

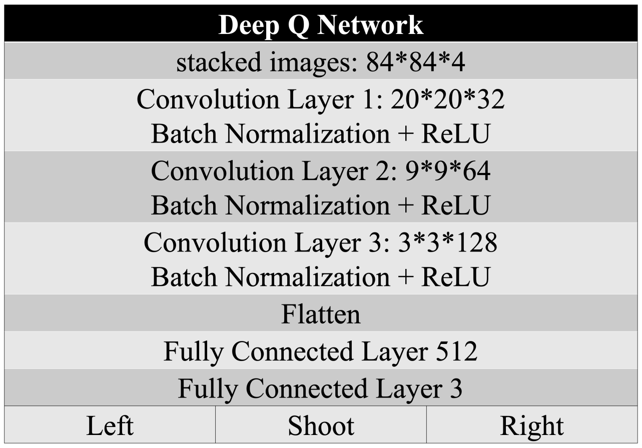
5.2 Evaluation

We evaluate the performance of our agents with the average rewards they obtained in the testing settings. While evaluating performance, the agents will not execute exploration policy. In the case of DQN, the agent will choose the action with the maximum Q value in condition of the state.

5.3 Scenario 'Search and Hit': Deep Q-Learning

This scenario assigns 101 rewards for successfully killing the monster, -1 rewards for each frames past, and -5 rewards for shooting without hitting the target. Basically, this configuration encourages the agent to kill the monster as fast as possible and hit the monster accurately.

We take the stacked image as our input into our Deep Q Network. The stack size we use is 4. In each episode, we run the agent to obtain the information from the environment, namely the triplet of state, action and rewards. The experience is added the memory buffer. We backpropagate the model using the loss calculated using the samples from the memory buffer. The detailed Deep Q Network are plotted as in Figure 4.



Deep Q Network		
stacked images: 84*84*4		
Convolution Layer 1: 20*20*32 Batch Normalization + ReLU		
Convolution Layer 2: 9*9*64 Batch Normalization + ReLU		
Convolution Layer 3: 3*3*128 Batch Normalization + ReLU		
Flatten		
Fully Connected Layer 512		
Fully Connected Layer 3		
Left	Shoot	Right

Figure 4: Deep Q Network

Different exploration strategies including random-policy, ϵ-greedy-policy with different decay rate, Boltzmann -policy with different temperatures are deployed to train our agents.

5.3.1 Random Policy

We plotted the rewards curve during the training process with random policy in Figure 5.

The value of the rewards when executing random policy is constantly low since the agent does not utilize Q values corresponded to the state. We can also see the curve fluctuates a lot since every time the agent randomly takes an action. We are surprised to see that test average rewards in the table 2 looks good. It suggests that our agent is not constrained by exploiting while training and learns from the information obtained while exploring.

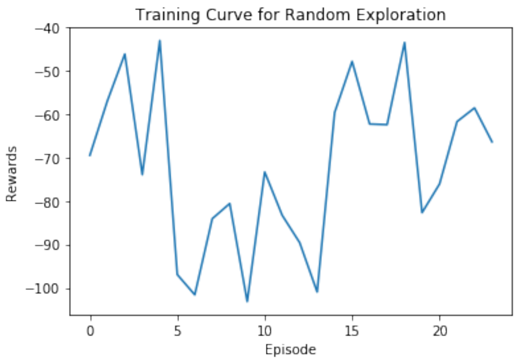


Figure 5: Training curve for Random-Policy

5.3.2 ϵ-greedy

For ϵ-greedy, we fix the starting ϵ values to be 1 and the minimum ϵ to be 0.01. We tested two different decay rate: 1e-4 and 1e-5. The training curves for these two configurations are plotted in Figure 6 and Figure 7.

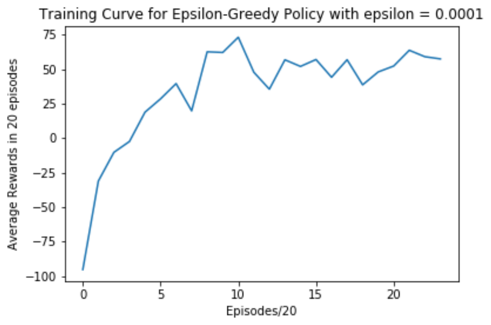


Figure 6: Epsilon-Greedy-Policy with ϵ = 1e-4

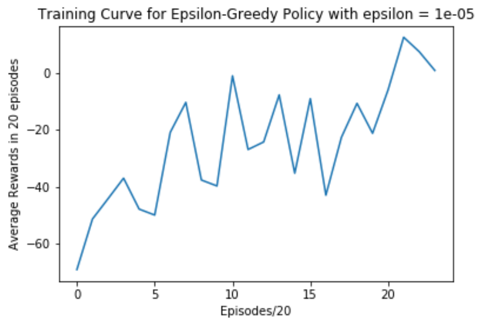


Figure 7: Epsilon-Greedy-Policy with ϵ = 1e-5

The results in table 2 indicate that slow decay rate in exploration possibility helps to improve the performance of the agents, because slower decay can leave the agent more space to explore new actions to gather information, which is beneficial to its learning process.

5.3.3 Boltzmann-Policy

During training, Boltzmann-Policy assigned different possibility to the actions according to the Q values. If the temperature is very low(close to 0), the decision making process with will converge to a random policy since the Boltzmann distribution will converge to an uniform distribution. The training curves for Boltzmann-Policy with temperature $\beta = 2, 1, 0.5$ are plotted in Figure 8, 9 and Figure 10. The results in table 2 also indicate that randomness in the exploration policy helps agent to achieve better performance.

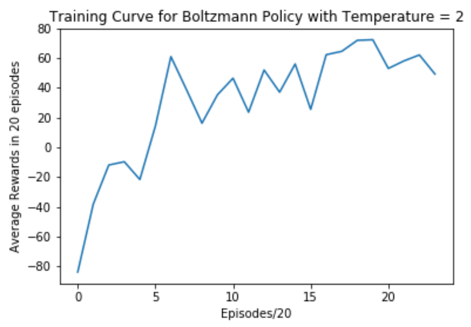


Figure 8: Boltzmann-Policy w. temperature = 2

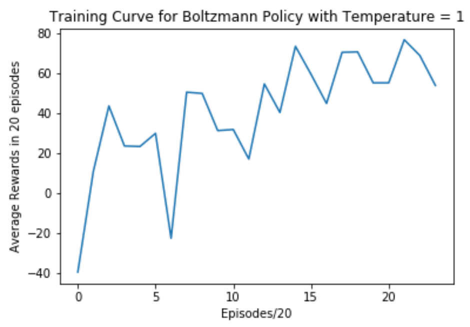


Figure 9: Boltzmann-Policy w. temperature = 1

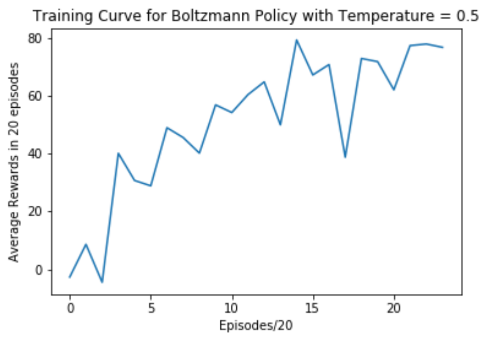


Figure 10: Boltzmann-Policy w. temperature = 0.5

5.4 Test Performance for Each Configuration

In table 2, we summarized the test average rewards of each configuration obtained in the test settings. The test average rewards are calculated from the average of the rewards of 1000 episodes of game play using the trained agent.

Baseline model refers to the agent that execute random action in the test setting. Note that the Random-Policy model is different from baseline model in the way that Random-Policy agent learn from exploration and execute the action with the highest Q value in the test setting.

Because it is a rather easy game, human can achieve perfect performance. The performance achieved by our best model DQN under Boltzmann-Policy with temperature = 2 is very close to the perfect human performance.

Table 1: Test average rewards for different exploration scheme

Model	Configuration	Test Average Rewards
Baseline	None	-75.18
Human	None	78.21
Random-Policy	None	55.11
ϵ-Greedy-Policy	decay rate = 1e-4	33.76
ϵ-Greedy-Policy	decay rate = 1e-5	70.83
Boltzmann-Policy	temperature β = 0.5	56.7
Boltzmann-Policy	temperature β = 1	66.20
Boltzmann-Policy	temperature β = 2	75.42

5.5 Scenario 'Health Gathering': Policy Gradient

This configuration of this scenario is as follows, the rewards for living for one frame is 1, and the reward for death(health = 0) is -100. The agent will constantly lose health and will eventually die. The agent in this scenario will learn to survive by finding the "medkits". However, we don't explicitly tell the agent that medkits will help the agent to survive. The agent need to learn the logic themselves by getting feedback from the environment.

We implemented the policy gradient algorithm with batch size = 1000 and discounting rate = 0.05 to train the agent to play the 'Health Gathering' scenario. The training curve is shown in figure 11.

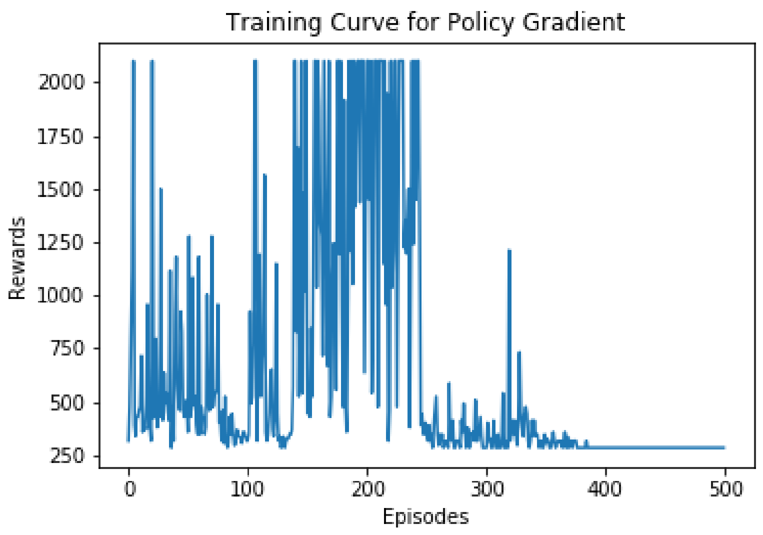


Figure 11: Training curve for policy gradient applied to the Scenario 'Health Gathering'

Table 2: Test average rewards for the scenario 'Health Gathering'

Model	Test Average Rewards
Baseline	476.9
Human	2100
Policy Gradient	1533.12

The average test score for our agent is 1533.12 after testing for 1000 episodes. The maximum score for the game is 2100, that is you survive for a minute in the game. Therefore, 1533.12 is a decent score, but it can still be improved by experimenting with other algorithms.

6 Conclusions

- The balance between exploration and exploitation is important in reinforcement learning models. We should leave the agent some space to explore new things in order to gather more information. We need to sacrifice short-term benefits sometimes to achieve long-term success.
- In our case, stochastic approach(Boltzmann-policy) outperforms deterministic approach(always picking the one with largest Q value). This suggests it important to solving machine learning problems in a probabilistic way. Introducing more uncertainty can make the machine learn better.

7 Future Work

We plan to implement more advanced models for the 'search and hit' scenario in the future, including Double Q Learning(12) and Deuling Q Learning, and Deep Recurrent Q Learning(13).

We also plan to try more scenarios in DOOM to solve more complicated and interesting problems using deep reinforcement learning.

References

[1] https://en.wikipedia.org/wiki/Reinforcement_learning#Introduction

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Playing Atari with Deep Reinforcement Learning, 2013

[3] https://flappybird.io/

[4] https://scratch.mit.edu/projects/105500895/

[5] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek Wojciech Jaśkowski, ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning, IEEE Conference on Computational Intelligence and Games, pp. 341-348, Santorini, Greece, 2016

[6] Michał Kempka, Grzegorz Runc, Jakub Toczek, Marek Wydmuch, VIZIA: 3D VIDEO GAME-BASED ENVIRONMENT FOR RESEARCH ON LEARNING AGENTS FROM RAW VISUAL INFORMATION

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, Asynchronous Methods for Deep Reinforcement Learning, 2016

[8] Roger McFarlane, McGill University, A Survey of Exploration Strategies in Reinforcement Learning

[9] https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing.-for-deep-q-networks-on-atari-2600-games/

[10] http://home.deib.polimi.it/restelli/MyWebSite/pdf/rl5.pdf

[11] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller, Deterministic Policy Gradient Algorithms

[12] Hasselt et al,Deep Reinforcement Learning with Double Q-learning, , 2015

[13] Hausknecht and Stone, Deep Recurrent Q-Learning for Partially Observable MDPs, 2015