Learning to play Space Invaders using Deep Q-Networks

Meghna Asthana
Department of Computer Science
University of York
[redacted]

Qi Chen
Department of Computer Science
University of Bath
[redacted]

Abstract

The project we have undertaken is the application deep reinforcement learning technique of Deep Q-Networks (DQN) on a classic Atari 2600 game, Space Invaders. This has been achieved by transforming the Q-learning algorithm such that it is compatible with high dimensional states which an image contains. We developed a Convolutional Neural Network which provided an optimize Q-policy and experimented with hyper-parameters to obtain the best model. Finally, we concluded that the best model we trained was a DQN with preprocessed image frame and experience replay batch size of 16 with training duration of one day. We furthered our understanding of deep reinforcement learning by looking into advanced methods like Double and Dueling DQNs and Inverse Reinforcement Learning but have not implemented them due to time and computation constraints.

1 Introduction

Programming and controlling an intelligent agent to learn and play a game with human-level or beyond human-level skills has been a long-standing challenge[1]. It has been traditionally solved using Reinforcement Learning which utilizes sequential data to learn policies which maximize the cumulative future record. As its performance heavily relies on the quality of the hand-crafted features (which are combined with linear value functions or policy representations[2]), researchers had to observe leading players and design feature based on their strategies.

While combining deep learning and reinforcement learning, we can encounter certain issues. From a deep learning perspective, we require large amounts of labelled training data, however, for reinforcement learning we are only able to learn from scalar rewards. Furthermore, a delay between action and reward is an undesirable feature for deep learning models where there is direct association between the inputs and targets. Finally, the assumption of data samples as independent entities can cause problems for reinforcement learning module as utilizes highly correlated states.

However, with the recent advancements and increase in use of deep learning techniques and supercomputers with heavy computational power, we are no longer required to manually design features as it allows us to extract high level features from raw data. The intervention of Convolutional Neural Networks (CNNs) in the system architecture allows the reinforcement learning algorithm to learn policies from raw image frame data. This combined architecture of CNN and reinforcement learning algorithm (Q-learning) is known as the Deep Q-Networks[1].

In this project, we will be developing and training a DQN to play the Atari game, Space Invaders and analyze our performance based on score value per episode for them. We will be implementing our DQN on The Arcade Learning Environment (ALE) Atari 2600 RL testbed which provides us with an agent and high-dimensional visual input (210 x 160 RGB video at 60Hz). The network was not

Project report for CM50270 Reinforcement Learning at the Department of Computer Science, University of Bath, UK. May 2019.

provided with any specific information about the game. The only inputs provided were the video input, the reward, the terminal signal and the set of all possible actions.

2 Background

2.1 Q-Learning

With the provided sequence of states, actions and rewards we are required to learn an optimal strategy to maximize our reward for each game-play. As we have a sequence of rewards, the rewards obtained at every step is discounted by a factor γ. This is defined as

$$R_T = \sum_{t=0}^{t=T} \gamma^t r_t \tag{1}$$

In order to learn an optimal strategy in a practical setting, we minimize the squared loss function

$$L = \sum_{s,a,r,s'} (Q_{opt}(s,a) - (r + \gamma V_{opt}(s')))^2 \tag{2}$$

and the Q-function is estimated using functional approximation. As in practice we do not generalize unseen states and actions, we minimize the loss function using gradient descent for learning weights w. It is updated as

$$w \rightarrow w - \eta(Q_{opt}(s,a) - (r + \gamma V_{opt}(s')))\phi(s,a) \tag{3}$$

2.2 Deep Q-Learning

The approach we have implemented is training directly from high dimensional raw input, using updates based on stochastic gradient descent. The large amount of data from each image frame inputted in the deep neural networks allows us to learn better representation as compared to the hand-crafted features.

The technique of experience replay helps us estimate the value function by updating the parameters of the network. In this, we store the agent's experience at each time-step $e_t = (s_t, a_t, r_t, s_{t+1})$ in a dataset $D = e_1, \ldots, e_N$, experienced over many episodes in a replay memory. When we implement Q-learning, we only update the batch of sample experiences which are chosen at random from the stored dataset D. Following completion of experience replay, the agent selects and executes an action based on the ϵ-greedy policy.

The approach has many advantages over standard Q-learning. Firstly, it enables better data efficiency as the experience at each step is used to update weights. Secondly, in Q-learning we directly learn from consecutive samples which is inefficient due to their high correlation and it reduces the variations in updates.

Another technique that has been proved to speed up the training process is frame skipping[2]. The agent only looks at the nth frame instead of all the frames. This helps the agent to play n times more episodes for the same amount of time and computation power.

3 Problem Description

In the Space Invaders game, we have an agent which we train by interacting with the environment. This is achieved by discretizing the game-play into time-steps where the agent chooses an action from a list of actions $A = 1, 2, \ldots N$. The chosen action is then applied to the current state of environment which translates to the new state. During this transition, we are provided with an updated reward r_t and new state s_{t+1}. A formal definition of each component of data is described below:

1. State s: At any given time-step t, state s_t is an observation which is a matrix representing the image frame in an RGB format with the size of 210 x 160 x 3.
2. Action a: It is the input provided by the agent to the environment and can be an integer in the range $[1, N]$ where, $N = 6$ for Space Invaders. The actions are named as {FIRE, RIGHT, LEFT, RIGHTFIRE, LEFTFIRE, NOOP}.

3. Reward r: This is the output returned by the environment after an action is taken. It has been boundaries as $[-1, 1]$.

4 Data and Experiments

4.1 DQN Implementation

The implementation has been achieved by creating two classes – one for Deep Q-Network and the other for the Agent. We have used PyTorch for implementing our Convolutional Neural Network for its simple and easy to write syntax. The Deep Q-Network class, DeepQNetwork contains the $init()$ function which is the main initializer for the CNN. The CNN encompasses three convolutional layers, two funlly connected layers and a stochastic gradient descent (SGD) optimizer. We calculate the loss after applying SGD. We also incorporate the functionality for using GPU if the CUDA parallel computing platform is available on the system. A detailed description of each layer is described below:

Layer	Type	Input	No. of Filters	Filter size	Stride	Padding	Output
Conv1		1	32	8	4	1	32
Conv2	Convolutional	32	64	4	2		64
Conv3		64	128	3			128
Fc1	Fully Connected	128×19×8					512
Fc2		512					6

Table 1: Layers of CNN

The next function for the DeepQNetwork class is the $forward()$ function which defines a multi-dimensional matrix known as tensor for the sample set of observations and passes those observations to the previously defined three convolutional layers with rectification linear unit, ReLU working at each layer. Finally, we apply the fully connected layers to the observation to get the final action for that time-step.

For the agent class, Agent we initialize the Agent object in the initializer function $init()$. The next function is the $storeTransition()$ function which stores all the rewards and states in the central memory until the maximum memory size is reached. The function $chooseAction()$ implements neural network we previously defined to evaluate the optimal Q learning policy for a sampled set of observations. This is the implementation of Experience Replay. The $learn()$ function implements the standard Q learning algorithm on the Q values learned from the neural network.

Our experimentation study has been divided into three spheres:

1. **Effect of Pre-processing on training time:** In the first phase, we develop two DQNs – one with pre-processing and one without it. The pre-processing included converting the RGB image for each time-step frame input to grayscale. This is expected to reduce the training time.
2. **Training duration analysis:** We trained our model for two different time duration – 5 hours and 28 hours and analyzed its effect on the game-play.
3. **Effect of Experience Replay batch sizes on training time and game-play:** We implemented Experience Replay for batch sizes 8, 16 and 32.

5 Discussion and Analysis

5.1 Strategies learned by agent

The agent learned various strategies for different training sessions. The most intriguing and fruitful strategies are enumerated below:

1. **Destroying Mothership:** For some of the sessions in the training, the agent learns to boost its score by 225 points by aiming at the mothership. The difficulty point for this strategy is that the mothership does not necessarily appear in every game and is a fast moving target. In order to successfully aim and hit the mothership, the agent has to retract its focus from

the other smaller ships and follow the mothership. Mastering this strategy is quite difficult and require rigorous train, thus, it has only been seen in the 28-hours training session. The biggest drawback for this strategy is that, when the agent abandons other ships, it allows the fleet of smaller ships to destroy the barriers and reach to the bottom of game-screen which automatically ends the game. Therefore, it is important to maintain a healthy balance between the mothership chase and destroying the fleet.

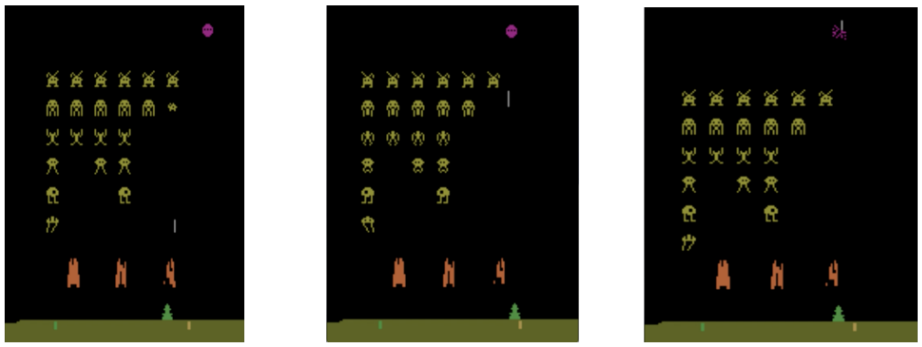


Figure 1: Destroying the mothership

2. **Destroying fleet of smaller ships column-wise:** The strategy is comparatively easier to master and efficient as it provides better reward than randomly shooting the targets. This is because, by simultaneous firing on a column of space ships, it saves precious game-time and is able to eliminate more ships from the fleet before they reach the bottom of game-screen. We have observed this training strategy in short-training duration (5 hours) and smaller batch sizes of experience replay.

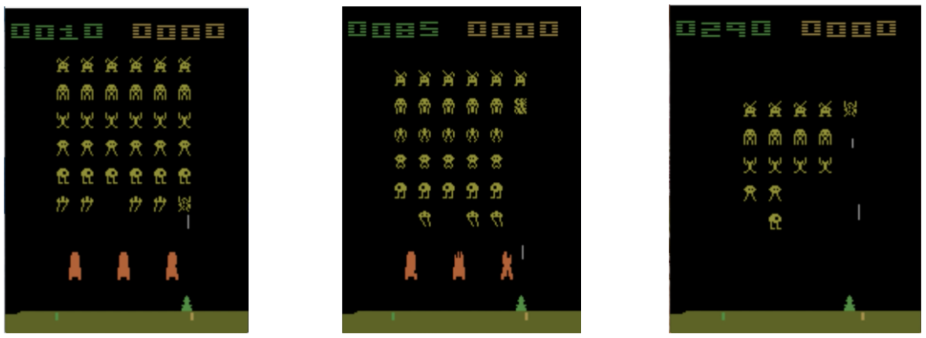


Figure 2: Destroying fleet column-wise

3. **Dodging the fire:** The strategy has been learning in both long-term (28-hours) and short-term (5-hours) training scenarios. This is because a single shot leads to loss of life which is a highly negative reward and thus, is undesirable. At the end of the long-term training, we observe that the agent has learned at which exact location the shot will be fired and is able to dodge shots in close proximity.

5.2 Hyper-parameter Selection Criteria

1. **ϵ-greedy:** In order to maintain a healthy balance between exploratory and exploitation move, we maintain the value ϵ greedy. The optimal policy is chosen with the probability $1 - \epsilon$ and the random policy with ϵ. This is altered at certain intervals of episodes from 1.0 to 0.05 over the complete course of training. This enables more exploration moves in the earlier stages of training and more exploitation moves over the end of the training.
2. **ϵ-end:** The ϵ-end value determines the lowest value to which our ϵ-greedy value can reach. This has been set to 0.05 which is usually achieved around the twentieth episode. It allows

the agent from this stage onward to concentrate on exploitation actions and acquire maximum attainable reward.

3. **Discount Factor γ:**The discount factor γ provides us a measure of the extent of distant future the algorithm looks to. We have chosen a high γ value of 0.95 as it allows us to take very distant future rewards into consideration. A new technique of choosing discount factor dynamically[7] could be implemented, however, this was not performed in order to keep things simple.

4. **Learning Rate α:** For the purpose of a more reliable training, very low value of learning rate α must be chosen. This is because a higher α value may not lead to convergence or worse make it diverging due to minima overshooting and making loss worse. We have chosen our α value as 0.003 which serves its purpose quite well. However, a better way of selecting an optimal α-value is to start training the network with low learning rate and decrease it by exponential factor when it starts reaching the gradient minima in later stages of training[6].

5. **Batch Size:** The batch size of a neural network training session should be selected as a trade-off between efficiency of training and noisiness of the gradient estimate. We have experimented with three batch sizes - 8, 16 and 32.

5.3 Comparison of DQNs with and without preprocessed raw data

Image frame preprocessing from RGB to gray-scale provides a massive boost in terms of cutting down computation time. From the figure below, we observe that we are able to play around 2000 episodes in 2-hours, providing an average game time to be 0.06 minutes whereas the non-preprocessed version is only able to play 50 episodes in 5 hours, providing an average game time to be 6 minutes. Thus, we can conclude that the image frame processing provides a 99% reduction in computation time. Furthermore, the highest reward obtained by the preprocessed DQN is 915 (obtained within 2 hours) as compared to the non-preprocessed DQN with 450 (obtained after training for 5 hours).

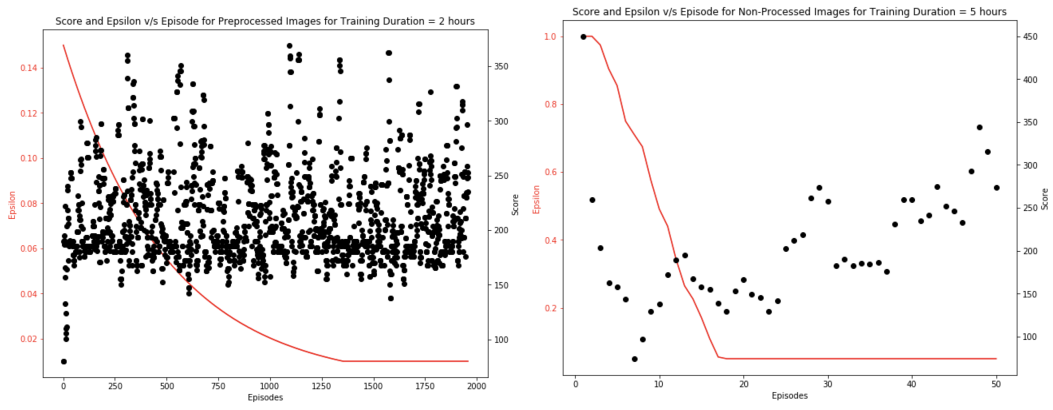
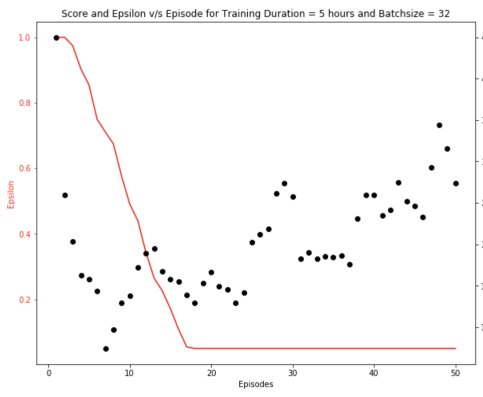


Figure 3: Effect of Preprocessing image data

5.4 Comparison of Training Duration

The figure below shows the comparison between long-term (28-hours) and short-term (5-hours) training sessions as well as the learning trend with change in epsilon. We observe that as the value of epsilon decreases from 1 to 0.05, the score steadily increases as shown in the first figure. This is an expected outcome as the agent starts to adhere to more exploitative actions in the later episodes, leading to higher reward. When we compare the outcomes of different training duration, we observe that highest reward obtained by the agent trained for 28 hours is 760 and for 5 hours is 450 however, there is no significant difference between the training outcomes in terms of final aggregated reward - 222 for long-term training and 214 for short-term training.



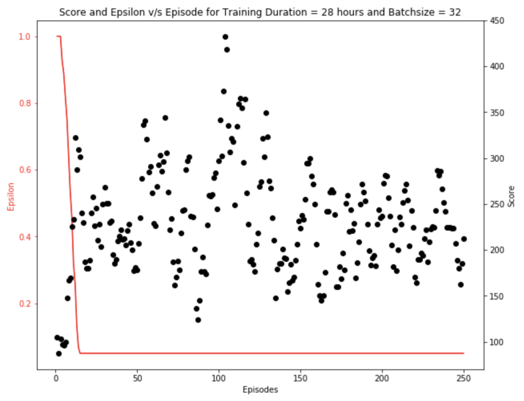


Figure 4: Comparison of Training Duration

This can be attributed to the fact that the highest reward outcomes are most likely are an outcome when the agent is successful in shooting the mothership. As discussed earlier, the instances of such occurrences is overall very low and therefore, for the majority of episodes this is rather unlikely to happen. However, the incidents of such occurrences increase when we train longer, justifying the fact that it is learning the 'destroying mothership strategy' though not efficiently.

5.5 Comparison of Batch Sizes for Experience Replay

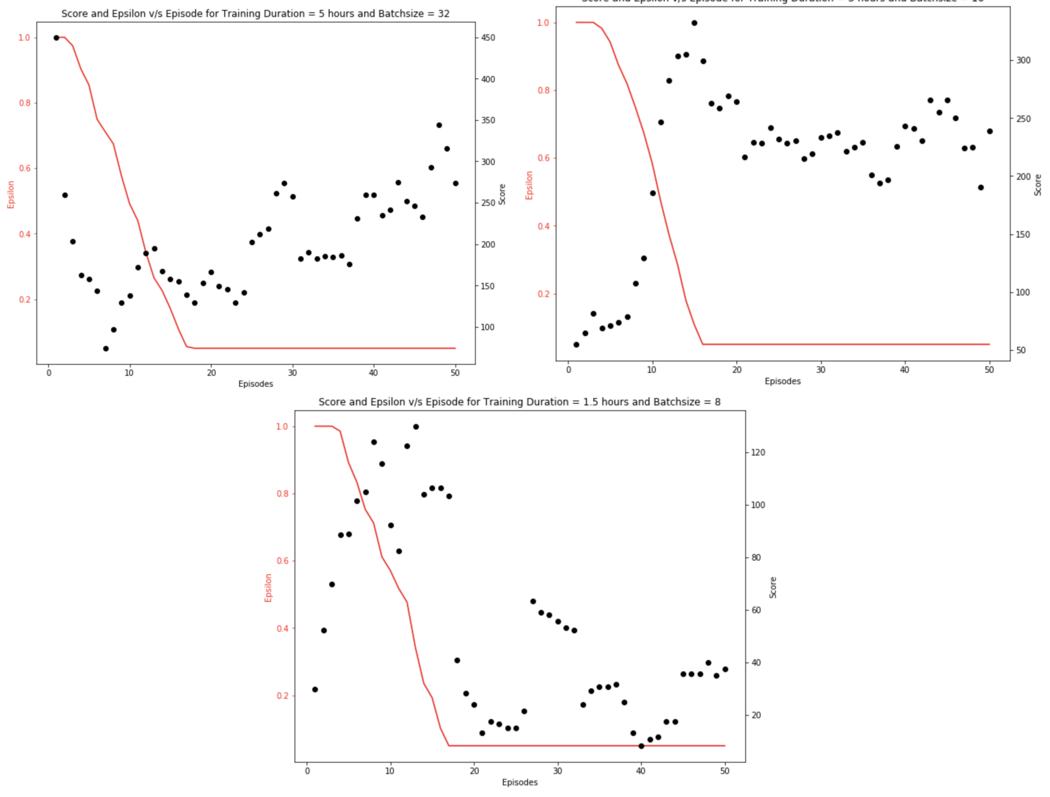


Figure 5: Comparison of Batch Sizes for Experience Replay

The figure above shows the training progression with different Experience Replay batch sizes. We observe that there is a comparable performance for batch sizes 32 and 16 with highest rewards episodes obtaining 450 and 460 respectively. The maximum reward for batch size 8 (excluding outliers) is 190, therefore, its the worst performer of the lot. We also observe that as we decrease batch size for experience replay, the training duration decreases corresponding to same number of game-play episodes. Thus, in order to balance the trade-off between training duration and maximizing reward the best batch size is 16 which provides an optimal training in less time.

5.6 Problems Encountered and Drawbacks of DQN

1. **Analyzing the strategies is difficult for the agent:** This problem has been attributed to the fact that our DQN training duration is comparatively lower than the average training duration in literature and limited computational power of the CPU in MacBook Pro.
2. **DQN training is slow:** Neural Networks take long time to learn and require high computational power. Therefore, they are very slow when it comes to training. Running speed depends on the game state updated after agent makes an action. This issue was resolved by changing the original 210x160x3 RGB image into gray-scale.
3. **More efficient exploration strategy required:** Balancing "Exploration" and "Exploring" is the essential problem of Reinforcement Learning, which requires us to design more efficient exploration strategies. We overcame this problem by defining a dynamic ϵ-greedy value which decreases from 1 to 0.05 over the course of training.
4. Reward Shaping has a huge impact on the performance of DQN. As there is a big lag between the action and reward (multiple image frames), it is difficult to obtain an instant feedback for a particular action taken by the agent. In order to mitigate this, we need to insert feedback signals during the training of the model which helps to partially overcome the problem of too sparse feedback. The application of Inverse Reinforcement Learning's GAN technique (described later) has been proven to alleviate this problem[3].

6 Conclusion and Future work

6.1 Conclusion

In this report we discussed about the methods implemented to train our agent to learn to play a classic Atari 2600 game, Space Invaders. We implemented Deep Q-Networks for training and analyzed it over various hyper-parameters. We fixated values for certain hyper-parameters like discount factor γ as 0.95 and learning rate α as 0.003 as they have been already proven to be the best in literature. The hyper-parameters we experimented with were batch size, ϵ-greedy and training time duration. Furthermore, we performed a comparative study for preprocessed and non-preprocessed data in terms of accuracy and computation cost. After successful training with variety of hyper-parameters, we observed our trained agent implementing strategies it learned over the training period. The most prominent among them were - destroying mothership, column-wise destruction of fleet of smaller ships and dodging the fire. Finally, after careful consideration of all model performance, we arrive to the solution that a model with preprocessed image frames trained with a DQN of experience replay of batch size 16 for about a day would be the most efficient model in terms of computational power and time.

6.2 Future Work

1. **Double DQN:** The one-step Q-learning algorithm implemented has a major drawback when it comes to estimating action value. It sometimes fails to meet the actual high value of an action value functions as the maximum step in it tends to be highly valued leading to overestimation. This results in overoptimistic value estimates. The reason behind is the insufficient flexibility of the function approximation and noise. This issue can be alleviated by implementing double DQN. It helps create a Q-learning function which can be generalized to arbitrary approximation functions[4].

   The Double Q-learning function corrects the overestimation by storing two sets of weights θ and θ'. Thus, for every stage of update, one set of weights are utilized to calculate the greedy policy and the other set for calculating its value. This can be represented as

$$Y_t{}^Q = r + \gamma Q(s', argmax_{a'}, Q(s', a', \theta), \theta') \quad (4)$$

2. **Dueling DQN:** Dueling-DQN improves DQN from the network structure, and the action value function can be divided into state value function and dominant function,

   $$Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a) \quad (5)$$

   using the neural network approximation for these two functions.

   The advantage function, $Q_\pi(s, a) - V_\pi(s)$ evaluates the current action value function relative to the average. Therefore, the advantage here refers to the advantage of the action value function compared to the value function of the current state. If the advantage function is greater than zero, the action is better than the average action. If the advantage function is less than zero, the current action is not as good as the average action.

3. **Prioritized Replay Buffer:** DQN's empirical playback uses a uniform distribution, while evenly distributed sampling does not make efficient use of data. Because the experience of the agent is the data that has been experienced, but the data is not equally important for training. The efficiency of the agent in some states is higher than that of other states.

   An ideal criterion is that the efficiency of the agent learning is higher and the weight is greater. Larger the TD deviation, larger is the difference between the value function and the TD target at this state, and greater is the update amount of the agent and thus, the learning efficiency is higher.

4. **Inverse Reinforcement Learning:** The domain consists of the following techniques[8]:
   (a) Linear programming: It aims to reproduce the model by enciphering possible states.
   (b) Maximum Entropy method: In this method, the features of each state are quantified and inputted in the equation for reward. It is considered a more practical solution than linear programming[9].
   (c) Hostile Inverse Reinforcement Learning: It focuses on GAN (Genetic Conflict Network) technology, which is particularly remarkable in the field of deep learning. GAN is a network in which two networks, a network to judge and evaluate (Discriminator) and a network to generate patterns (Generator), are combined. This feature is often useful for solving inverse reinforcement learning problems. The problems of conventional inverse reinforcement learning are concentrated on the point that "complex problems are difficult to imitate". Here, the “complex task” refers to a task that has a large amount of information of the state to be observed, and the variation of the change of the state is enormous. For example, in the case of not only oneself, but also the other party's state, and there are multiple opponents (Shogi or Go, a game facing multiple enemy characters, etc.), the state has a large amount of information, and the state Variations of transition from to the state will be a tremendous combination. When it comes to such “complex tasks”, we cannot catch up with the expressive and learning abilities of conventional reverse reinforcement learning[5].

References

[1] Nihit D., Abhimanyu B. Deep Reinforcement Learning to play Space Invaders. Stanford University. CS221.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[3] Liveway, 2018. Lecture 14:Initation Learning I [Online]. Beijing:Zhihu. Available from: https://zhuanlan.zhihu.com/p/34365843?fbclid=IwAR1R6VZlYPc2G1CFzwx4RfZC84f2AwHePmG f2zZ1-MlPFwxeIquPB6j3tIE [Accessed 28 April 2019].

[4] WANG, K., ZHANG, W., HE, X. and GAO, S., 2017. Deep Reinforcement Learning of the Model Fusion with Double Q-learning. *DEStech Transactions on Computer Science and Engineering*, (aiea).

[5] GMO internet GROUP,2018. AI Introduce the AI can be familliar with the skils of tal- lent "hostile inverse reinforcement learning". [Online]. Tokyo:GMO internet. Available from

https://recruit.gmo.jp/engineer/jisedai/blog/airl_with_open_ai_gym/?fbclid=IwAR1a8sK8uqjjGXEuN xDTVZ6h-AdYJAr5341hdavQ3MXajxnC4miZw7rFjps [Accessed1May 2019].

[6] Smith, L.N., 2017, March. Cyclical learning rates for training neural networks. In 2017 *IEEE Winter Conference on Applications of Computer Vision (WACV)* (pp. 464-472). IEEE.

[7] François-Lavet, V., Fonteneau, R. and Ernst, D., 2015. How to discount deep reinforcement learning: Towards new dynamic strategies. *arXiv preprint arXiv:1512.02011.*

[8] Ng, A.Y. and Russell, S.J., 2000, June. Algorithms for inverse reinforcement learning. In *Icml (Vol. 1, p. 2).*

[9] Ziebart, B.D., Maas, A.L., Bagnell, J.A. and Dey, A.K., 2008, July. Maximum entropy inverse reinforcement learning. In *Aaai* (Vol. 8, pp. 1433-1438).