International Journal of Control and Automation
Vol. 11, No. 3 (2018), pp.193-200
http//dx.doi.org/10.14257/ijca.2018.11.3.17

Deep Reinforcement Learning Agent for Playing 2D Shooting Games

Dongcheul Lee[1] and Janise McNair[2]

[1]*Department of Multimedia Engineering, Hannam University, Korea*
[2]*Department of Electrical and Computer Engineering, University of Florida, USA*
[1]*jackdclee@gmail.com*, [2]*jymcnair@ufl.edu*

Abstract

Reinforcement learning has been very challenging in learning how to play video games without human interaction. However, recent studies on deep reinforcement learning were very successful to master the learning with only high-dimensional raw input data. In this paper, we build the general agent for playing 2D shooting games using various deep reinforcement algorithms. By using this agent, we can compare the performance of the algorithms in multiple aspects. The architecture of the agent is modularized to allow sharing common parts and training each model independently. Also, the hyper-parameters for training the models are fully described. We compare the performance of the algorithms using various metrics provided by the proposed agent. It shows that the agent is very efficient for training each model and comparing their performance.

Keywords*: Deep learning, 2D shooting game, reinforcement learning, general agent*

1. Introduction

Learning to play video games has been a very challenging problem in reinforcement learning (RL) [1]. The goal of RL is to learn an optimal policy for choosing an action on a given environment, which results in maximizing a cumulative future reward. Previously, most RL agents had been relied on manually-selected features which was usually time-consuming and incomplete. Whereas recent agents, which is called deep RL agents, learn feature representations directly from high-dimensional raw input data such as images and videos using a convolutional neural network (CNN) architecture [2]. However, deep RL still has challenging issues. Since the reward is sparse and delayed, it is very hard to associate the chosen action with the given reward. Also, sequences of states of a video game are highly correlated, which leads to overfitting and falling into a local minimum.

To be successful in deep RL, the agent should derive main features from raw video input and use these past experiences to new situations to gain the maximum reward. There have been many improvements in deep RL to solve these problems. Deep Q-Network (DQN) used a separated target neural network and an experience replay technique [3]. Asynchronous Advantage Actor-Critic (A3C) took advantage of a multi-threaded agent while asynchronously updating a global neural network using policy gradient [4]. [4] also introduced a recurrent agent which combines A3C with Long Short Term Memory (LSTM) [5]. It uses LSTM for interpreting the features which were extracted from CNN since it can remember previous states and use them for making predictions.

In this paper, we provide a general agent that can make use of various RL algorithms and compare their performance metrics in multiple aspects. The agent can play 2D shooting games using only raw pixels on the screen and tries to gain the maximum score.

The next section introduces related works on common environment for playing games and the RL algorithms that our agent will use, which have been most successful in the last

Received (December 18, 2017), Review Result (February 6, 2018), Accepted (February 8, 2018)

ISSN: 2005-4297 IJCA

several years. In Section 3, we propose the architecture of the agent for playing 2D shooting games. All the hyper-parameters that were used in our models are described in detail. In Section 4, we compare the performance metrics of each RL algorithm in various aspects. The paper concludes with a short discussion.

2. Related Works

2.1. OpenAI Gym

OpenAI Gym is an Atari emulator for comparing various kinds of RL algorithms [6]. Since it gives you standardized set of environments for play Atari games, users can write a general algorithm and compare it from other algorithms. When the agent takes an action in the environment, it receives a state, reward, and other information related to the current step. The initial state of the environment is randomly chosen, and the agent interacts with the environment until it reaches a terminal state. The goal of the agent is maximizing the expectation of total reward per episode and achieving a high level of performance in as few episodes as possible. We used OpenAI Gym as a standard environment for our agent just as other RL researches did [7,8,9].

2.2. Deep Q-Networks (DQN)

[3] suggested DQN which trains CNN from high-dimensional sensory input using RL. An agent uses the network to maximize the expected return R_t

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k} \tag{1}$$

from time step t where γ is a discount factor and T is the time at which the game terminates and r_t is a reward from a state s_t. The output of the network is a Q-value function

$$Q^*(s,a) = \max_{\pi} \mathrm{E}\left[R_t \mid s_t = s, a_t = a, \pi\right], \tag{2}$$

which estimates maximum R_t when observing a state s and taking an action a where π is a policy mapping sequences to actions. The agent selects an action a_t that maximize a Q-value to interact with an unknown environment. The agent updates the network using following loss function:

$$L_i(\theta_i) = \mathrm{E}\left[\left(r_t + \gamma \max_{a'} Q\left(s_{t+1}, a' \mid \bar{\theta}_i\right) - Q(s_t, a_t \mid \theta_i)\right)^2\right] \tag{3}$$

where θ_i is a Q-Network parameter and $\bar{\theta}_i$ is a target network parameter at iteration i. $\bar{\theta}_i$ is only updated with θ_i every C steps while θ_i is updated every time step.

2.3. Asynchronous Advantage Actor-Critic (A3C)

While DQN uses a single agent that interacts with a single environment, [4] uses multiple agents which each have their own copy of the environment. Each agent selects an action a_t according to its policy π at each time step t. The value function of state s_t under policy π is defined as

$$V_\pi(s_t) = \mathrm{E}\left[R_t \mid s_t\right], \tag{4}$$

which is the expected return for following policy π from state s_t. To determine how good it is to take an action a_t in a state s_t compared to average, an advantage function

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t) \tag{5}$$

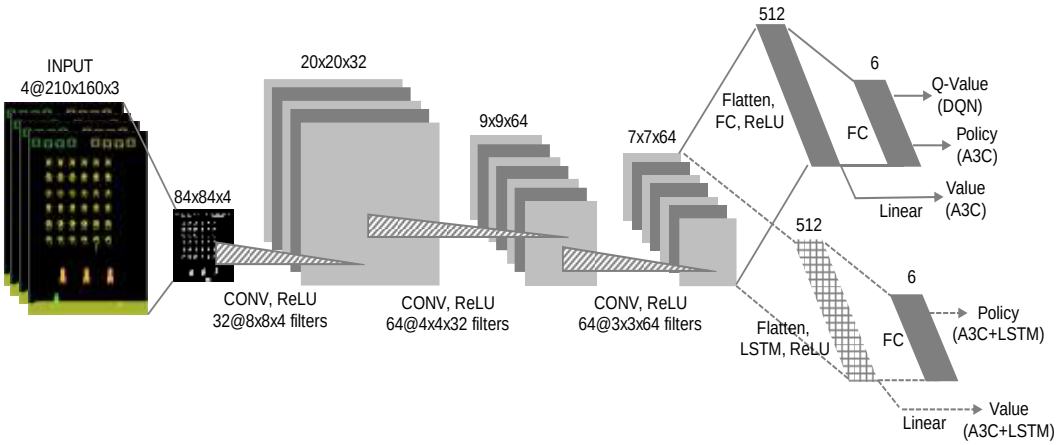

Figure 1. Illustration of the Network Architecture of Our Models We Used for Our Experiments

is used where a state-action value function is

$$Q_\pi(s_t, a_t) = \mathrm{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t, a_t]. \quad (6)$$

Also, to express how good the policy π is, an objective function

$$J(\theta) = \mathrm{E}[V_\pi(s_o)] \quad (7)$$

is defined. The gradient of the objective function $J(\theta)$ is approximately derived as

$$\Delta_\theta J(\theta) = \Delta_\theta \log \pi_\theta(a_t \mid s_t) \cdot A_\pi(s_t, a_t), \quad (8)$$

which can be used to optimize the objective function $J(\theta)$.

3. Proposed Architectures and Training Method

We now present a deep RL agent for playing 2D shooting game. The purpose of this agent is adopting various RL algorithms in a modular manner and comparing their performance in multiple aspects. In this paper, we compare three RL algorithms which are DQN, A3C, and A3C combined with LSTM (A3C+LSTM). Using the recurrent neural network, particularly the LSTM network, is helpful to remember information for an arbitrary long amount of time. To provide a common environment to the agent, we used OpenAI Gym.

Figure 1 shows the architecture of our model that the agent uses. To reduce input dimensionality, raw input is preprocessed as follows: A 210×160 pixel image with RGB color is converted to a gray-scale and down-sampled and cropped to a 84×84 image. Also, four subsequent frames are stacked up together to produce vectorized input to the network. The model consists of 3 convolutional layers, 3 fully-connected (FC) layers and 1 recurrent layer. All convolutional layers are shared, whereas 2 FC layers are used for DQN and A3C, and the other is used for A3C+LSTM. The number of filters in the three convolutional layers are 32 (8×8, with stride 4), 64(4×4, with stride 2), 64(3×3, with stride 1) and the Rectified Linear Unit (ReLU) is used for an activation function. The final hidden layer of DQN and A3C is FC with 512 cells, whereas the final layer of A3C+LSTM is LSTM with 512 cells. Both output layers are FCs with 6 outputs for each action.

In a shooting game, rewards are normally delayed upon a specific action, which makes it difficult for the agent to learn which action is responsible for what reward. Also, the scale of scores varies depending on the types of enemies hit. To limit the scale of the error derivatives, we fixed all positive rewards to be 1 while leaving 0 reward unchanged. This can improve the performance of our agent since it does not differentiate between various rewards.

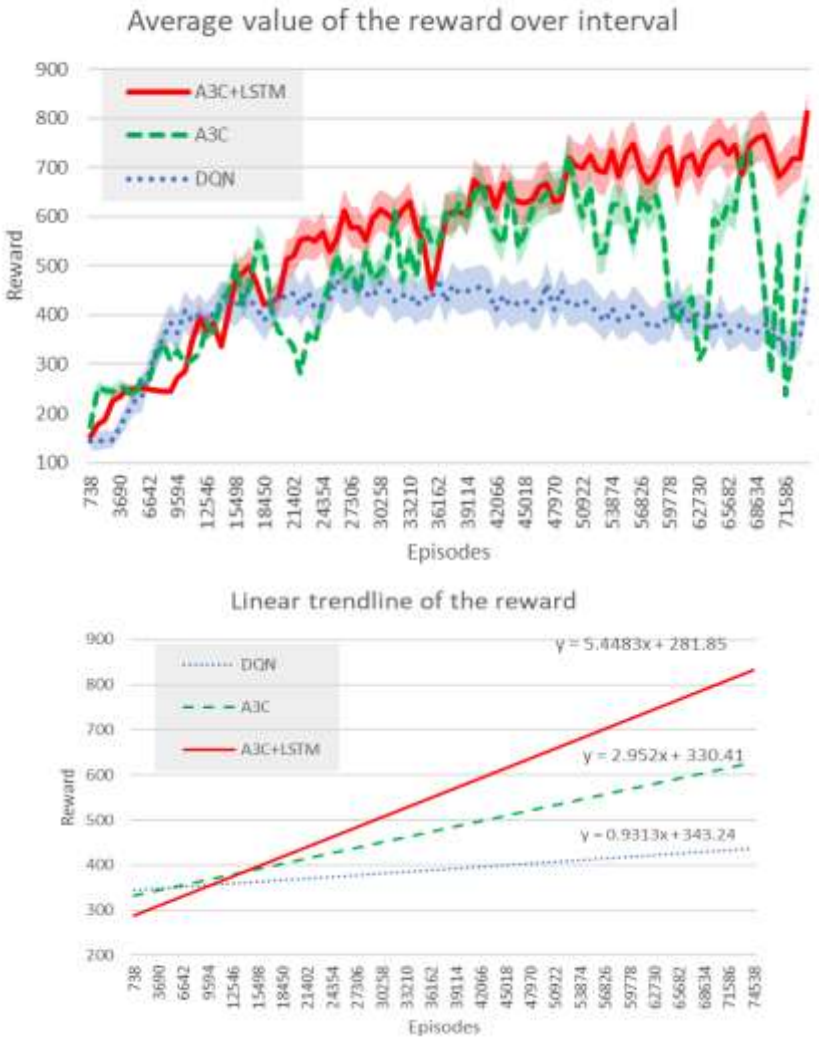

Figure 2. Learning Speed Comparison for DQN, A3C and A3C+LSTM on Space Invaders

If the agent uses consecutive samples while learning, this leads to strong correlations between samples, resulting network parameters get stuck in a poor local minimum, or even diverge. The agent breaks the correlations by random sampling. To explore a wide range of states in the domain space, the agent uses ε-greedy strategy that selects a random action with a probability ε and follows the optimal policy with a probability 1-ε. The value of ε decreases linearly from 0.5 to 0.1 until 70,000 time steps. Also, when using DQN, the replay memory and the separated target Q-network are used for breaking the correlation. The agent stores 500,000 most recent frames in the replay memory, and randomly samples a minibatch of size 32 from the replay memory. The agent uses this minibatch to train the DQN parameters and updates the target network every 10,000 step. Whereas, when using A3C or A3C+LSTM, the agent uses 8 threads interacting with their own copies of the environment. To prevent overwriting each thread's update, the samples are accumulated over 8 time steps before they are used for updating network parameters. Also, to make the exploration policies differ, each thread uses ε-greedy strategy when selecting an action.

Table 1. Score Comparison for DQN, A3C and A3C+LSTM

Algorithm	Average Score	Max Score
DQN	390.1	3058.5
A3C	479.4	2884.5
A3C+LSTM	557.3	3286.5

4. Experiments

We have performed experiments on Space Invaders from OpenAI Gym to compare the performance of DQN, A3C and A3C+LSTM. Our models were implemented using Keras [10] with Tensorflow backend and trained on a NVIDIA GTX 1070. All networks were optimized using the RMSProp optimizer [11] with the learning rate 1×10^{-4} until the maximum number of episodes reached, which was 74,000. The discount factor γ was set to 0.99.

Figure 2 compares the learning speed of the three algorithms using the total reward the agent collects in an episode. When we compared the reward, it was not clipped, which means it is same with the total score of the game. The plot on the above shows the average value of the reward over 738 episodes interval. After 15,000 episodes, the reward in DQN did not increased anymore, while the rewards in A3C and A3C+LSTM kept increasing. The reward in A3C was noisy compared to the one in A3C+LSTM. The plot on the below shows the linear trendline of the reward. It is clear that A3C+LSTM learned faster than others, while DQN was the slowest. Table 1 shows the actual average score

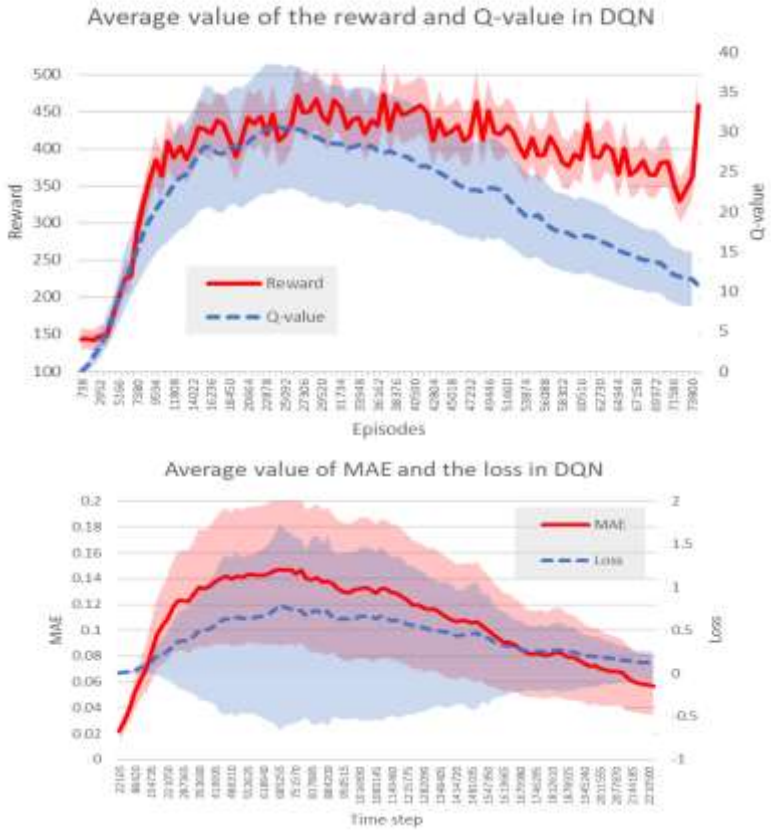

Figure 3. Plot of the Performance Metrics Obtained by DQN on Space Invaders

and the average of the top 10 highest scores of the game over the entire episodes. Again, A3C+LSTM gained the top average score and the top maximum score in three algorithms.

Figure 3 compares the performance metrics obtained by DQN while training. The plot on the above compares the reward with the Q-value as a function of episode. Two metrics were averaged over 738 episodes interval. The Q-value function, which estimates how much discounted reward the agent can get by following its policy, was more stable than the reward plot. The plot of two metrics showed similar curves over the entire episodes. The plot on the below compares the mean absolute error (MAE) with the loss as a function of the time step while optimizing network. Two metrics were averaged over 22105 time steps interval. We could not observe any divergence issues and their plots were similar smooth curves.

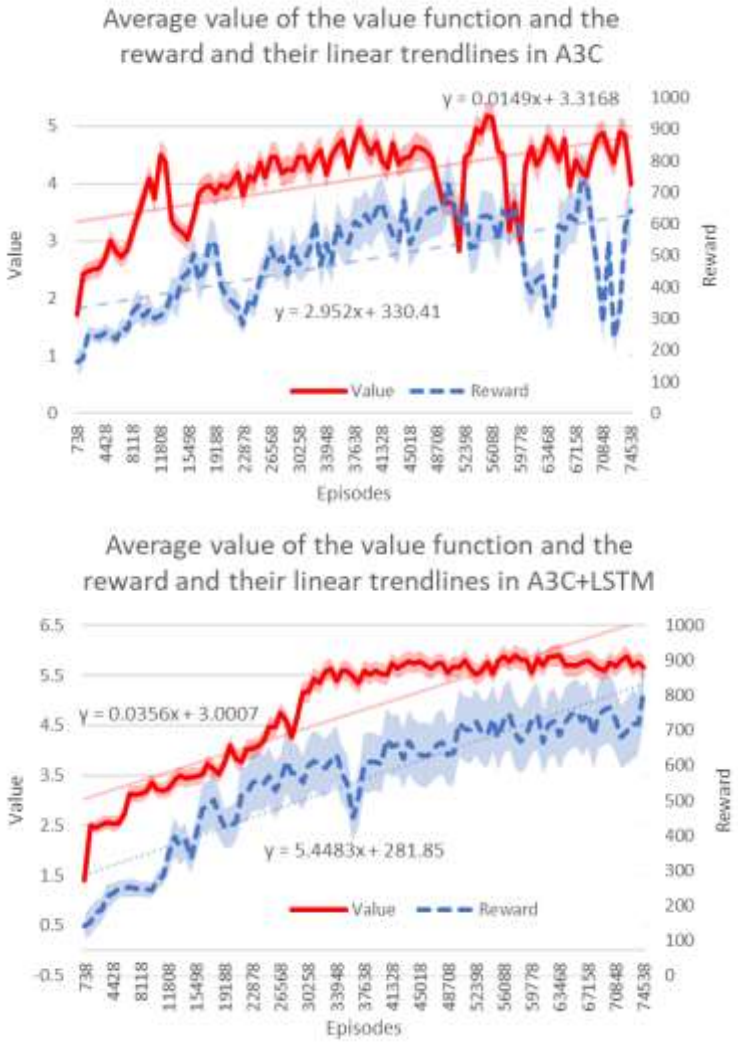

Figure 4. Plot of the Value and the Reward Obtained by A3C and A3C+LSTM on Space Invaders

Figure 4 compares the output of the value function with the reward obtained by A3C and A3C+LSTM while training. Both metrics were averaged over 738 episodes interval. The plots on the above compares them in A3C. Both plots were noisy but showed very similar curves. When we plotted the reward, the plot of the value followed similar curves after a few episodes later. The plot on the below compares them in A3C+LSTM. Both plots were more stable than that of A3C. Also, both values of the metrics grew faster than

that of A3C, but we could not observe any similarity between the reward and the value just as in A3C.

5. Conclusion

In this paper, we have presented a complete architecture of the RL agent for playing 2D shooting games. By using the agent, we could compare the performance of recent successful RL algorithms in various aspects. Experiment results showed the correlation between the performance metrics, such as the reward and Q-value in DQN, and MAE and the loss in DQN, and the value and the reward in A3C and A3C+LSTM. The experiment also showed that A3C+LSTM was fastest in terms of the learning speed and achieved a highest score among other algorithms.

For future work, we plan to expand the scope of adopted RL algorithms using various hyper-parameters on wide-range of environment. By doing this, we can make a new algorithm that combines advantages of the existing algorithms.

References

[1] J.N. Tsitsiklis and B.V. Roy, "An analysis of temporal-difference learning with function approximation", IEEE Transactions on Automatic Control, vol. 42, no. 5, **(1997)**, pp. 674–690.
[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing atari with deep reinforcement learning", Proceedings of the Neural Information Processing Systems (NIPS) Deep Learning Workshop, **(2013)**.
[3] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski and S. Petersen, "Human-level control through deep reinforcement learning", Nature, vol. 518, no. 7540, **(2015)**, pp. 529-533.
[4] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning", Proceedings of the International Conference on Machine Learning (ICML), **(2016)**, pp. 1928-1937.
[5] F.A. Gers, J. Schmidhuber and F. Cummins, "Learning to forget: continual prediction with LSTM", Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN), **(1999)**, pp. 850-855.
[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI Gym", arXiv preprint arXiv:1606.01540 **(2016)**.
[7] Y. Hou, L. Liu, Q. Wei, X. Xu and C. Chen, "A novel DDPG method with prioritized experience replay", Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), **(2017)**, pp. 316-321.
[8] A. Rajeswaran, K. Lowrey, E.V. Todorov and S.M. Kakade, "Towards Generalization and Simplicity in Continuous Control", Proceedings of the Advances in Neural Information Processing Systems (NIPS), **(2017)**, pp. 6553-6564.
[9] D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T.A. Runkler and V. Sterzing, "A benchmark environment motivated by industrial control problems", Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), **(2017)**, pp. 1-8.
[10] K. Choi, D. Joo and J. Kim, "Kapre: On-GPU Audio Preprocessing Layers for a Quick Implementation of Deep Neural Network Models with Keras", Proceedings of the International Conference on Machine Learning (ICML) machine learning for music discovery, **(2017)**.
[11] T. Le, J. Kim and H. Kim, "An Effective Intrusion Detection Classifier Using Long Short-Term Memory with Gradient Descent Optimization", Proceedings of the International Conference on Platform Technology and Service, **(2017)**.

Authors

Dongcheul Lee, he has been an assistant professor in the Department of Multimedia at Hannam University, Korea since 2012. He received his B.S. and M.S. degrees in Computer Science and Engineering from POSTECH, Korea in 2002 and 2004, respectively, and a Ph.D. degree in Electronics and Computer Engineering from Hanyang University, Korea in 2012.

Janise McNair, she is an Associate Professor in the Department of Electrical & Computer Engineering at the University of Florida, where she leads the Wireless And Mobile Systems Laboratory. She earned her B.S. and M.S. in electrical engineering from the University of Texas at Austin in 1991 and 1993, respectively, and her Ph.D. in electrical and computer engineering from the Georgia Institute of Technology in 2000.