Exercise 3
The implementation of Deep Reinforcement Learning on Atari games

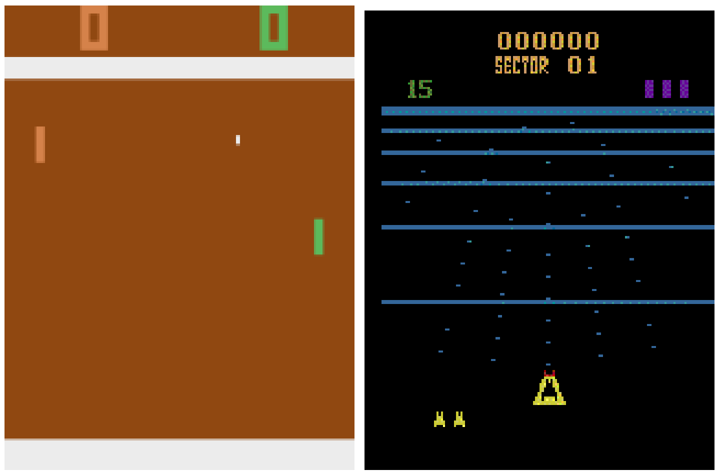
Student: Khuong G.T Diep
Advisor: Prof. Yong-Guk Kim

December 10, 2022

1 Introduction

The main point of this report is to implement Deep Reinforcement Learning on Atari games. Atari 2600 is a video game console from Atari that was released in 1977. The game console included popular games such as Breakout, Ms. Pacman and Space Invaders. Since Deep Q-Networks were introduced by Mnih et al. in 2013, Atari 2600 has been the standard environment to test new Reinforcement Learning algorithms. Atari 2600 has been a challenging testbed due to its high-dimensional video input (size 210 x 160, frequency 60 Hz) and the discrepancy of tasks between games.

The Atari 2600 environments was originally provided through the Arcade Learning Environment (ALE). The environments have been wrapped by OpenAI Gym to create a more standardized interface. The OpenAI Gym provides 59 Atari 2600 games as environments [1].



(a) Pong (b) Beam rider

Figure 1: Two Atari games used in this report [1]

2 Reinforcement Learning Background

Deep Reinforcement Learning Hands-On [2] and Reinforcement Learning: An Introduction [3] are the base for all the theories described below.

Reinforcement learning is learning what actions to do in a given situation in order to maximize a numerical reward signal. This learning framework differs from supervised and unsupervised learning [3]. Unlike supervised learning, which requires a large number of labeled data sets to train the model, reinforcement learning studies the environment and learns from its own experience. Reinforcement learning, on the other hand, can be compared to unsupervised learning, which similarly works with unlabeled data. However, they are distinct in that the purpose of unsupervised learning is to extract hidden features from data, whereas the goal of reinforcement learning is to maximize the reward signal. As a result, reinforcement learning is classified as a third machine learning paradigm.

First of all, we need to know some basic terminologies such as an agent and a state. An agent is a decision-maker that interacts with its environments and decide what to do. The state, on the other hand, is defined as a representation of the agent's current surroundings. The next step is to look at the main components of reinforcement learning. There are four essential elements in a reinforcement learning system including a policy, a reward, a value function and an environment model. Firstly, regarding to a policy, it is a strategy that determines what decisions and actions the agent will take in a specific state. Secondly, a reward is what a learning agent receives every time it makes a right decision. In contrast, it will receives a penalty when the movement is incorrect. In summary, trying to acquire as many rewards as possible is the goal of a reinforcement learning algorithm. Thirdly, whereas the reward function defines the short-term benefit, the value function denotes the long-term advantage of a state. To explain clearly, the value of a state is the total quantity of rewards that an agent is expected to get over time when it starts at that state. In reinforcement learning, a state might have a low immediate reward but a high value because it is followed by states that produce high rewards. Lastly, the model of environment is something that imitates the environmental behavior.

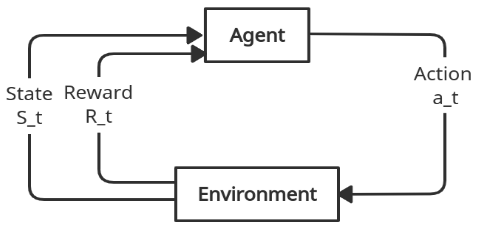


Figure 2: The interaction process between agent and environment in reinforcement learning

Fig. 2 describes a reinforcement learning process. At each time step t, the agent observes a state s_t and then follows its given policy π to decide what action a_t to do from an action space A. After executing a chosen action, the agent receives a reward or a penalty and then observes the new state s_{t+1} from the environment [4]. The agent in a RL model normally seeks to maximize the expected long-term reward at each state, so there is a terminology called expected return $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, which denotes the total discounted rewards over the future. The additional parameter that we need is the discounted rate $\gamma \in [0, 1]$ which specifies how concerned we are about the future reward. If this argument is set to zero, the agent only concerns about the maximum immediate reward of the next step. On the other hand, as γ reaches the value of one, the agent's focus shifts to maximizing the long-term rewards [5]. Furthermore, the RL algorithm can be separated into three basic ways: model-based methods, policy-based methods, and value-based methods. Model-based strategies attempt to forecast the next observation and/or reward. The agent is attempting to determine the best possible action based on this forecast. Policy-based approaches directly approximate the agent's policy [6], that is, what actions the agent should take at each state. This strategy directly trains to optimize the agent's policy. Finally, value-based techniques, in which the agent calculates the value of each available action and selects the action with the highest value rather than the probability of actions. By training the value-function, this strategy indirectly trains the agent's policy [6].

3 Methodology

3.1 Introduction

In reinforcement learning, one of the most essential algorithm is Q-Learning. This is a simple algorithm that aims to find the best action to take given the current state. But when the count of the potential states is very large, it becomes impossible to use this algorithm. For example, consider an environment with 10,000 states and 1,000 actions per state. This would result in a table with 10 million cells. Things will get out of control. In the topic of deep reinforcement learning, there have been some another well-known algorithms. They are frequently used as a base method and researchers might utilize them with some modifications to optimize the behavior. This optimization can save processing time, make the system more energy efficient, boost performance, etc.

One of the most popular algorithm is Deep Q Learning. Before diving deeply into DQN, understanding experience replay is very important.

3.2 Experience replay

Experience Replay is a replay memory technique used in reinforcement learning where we store the agent's experiences at each time-step, in a data-set , pooled over many episodes into a replay memory. We then usually sample the memory randomly for a minibatch of experience, and use this to learn off-policy, as with Deep Q-Networks. This tackles the problem of autocorrelation leading to unstable training, by making the problem more like a supervised learning problem [7].

3.3 Deep Q Network (DQN)

To begin, the Deep Q-Network algorithm (DQN) has demonstrated exceptional performance in handling complicated problems. The term "Deep" in the "Deep Q Networks" (DQN) refers to the use of "Convolutional Neural Networks" (CNN) in the DQNs. Convolutional

Neural Networks are deep learning architectures inspired by the way human's visual cortex area of the brain works to understand the images that the sensors (eyes) are receiving [8]. Deep Q-Learning, crucially, replaces the regular Q-table with a neural network. A neural network maps input states to (action, Q-value) pairs rather than mapping a state-action pair to a q-value.

$$Q_{s,a} = r(s,a) + \gamma \sum_{s' \in S} P_{ss'} V_{s'} = r(s,a) + \gamma \max_a Q'(s', a') \tag{1}$$

Several steps in the DQN algorithm are described in the Algorithm 1.

Algorithm 1 Deep Q-Learning

```
 1: Initialize Q(s,a) function with random weights;
 2: Initialize an empty replay buffer;
 3: for $i = 1$ to $N$ do
 4:     Initialize the beginning states s;
 5:     for $t = 1$ to $T$ do
 6:         Choose a random action $a_t$ based on $\epsilon$-greedy policy;
 7:         otherwise select $a(t) = \arg\max_a Q(s,a)$;
 8:         Observe reward $r_t$ and the next state $s(t+1)$;
 9:         Store the observed transition in the replay buffer;
10:         Compute loss function $(Q(s,a) - y)^2$;
11:         Update weight parameters $\omega$ of the target network;
12:         Update the Q table;
13:     end for
14: end for
```

3.4 Double DQN

According to reference [2], the original DQN tends to overestimate Q values, it might affect negatively to the training process. An algorithm called Double DQN was introduced to overcome this problem. Double DQN shows a great improvement upon DQN, especially, on the Atari 2600 domain [9]. When computing the Q target, two networks are used to decouple the action selection from the target Q value generation. DQN network is used to select what is the best action to take for the next state (the action with the highest Q value) and the target network is used to calculate the target Q value of taking that action at the next state. The new expression of Q-function is shown below:

$$Q^{Double}_{s,a} = r(s,a) + \gamma \max_a Q'(s', \arg\max_a Q(s', a)) \tag{2}$$

3.5 Dueling DQN

There is another enhanced algorithm called Dueling DQN. Reference [10] said that by dividing the network into two different streams, the dueling architecture can learn which states are or are not useful, without having to know the effect of each action at each time step. This algorithm compute a new quantity called the advantage $A(s, a)$, which is the subtraction between Q-value and state-value:

$$A(s, a) = Q(s, a) - V(s) \tag{3}$$

The value function V(s) tells us how much reward we will collect from state s. And the advantage function A(s, a) tells us how much better one action is compared to the other actions. Then we combine these two streams through a special aggregation layer to get an estimate of Q(s,a).

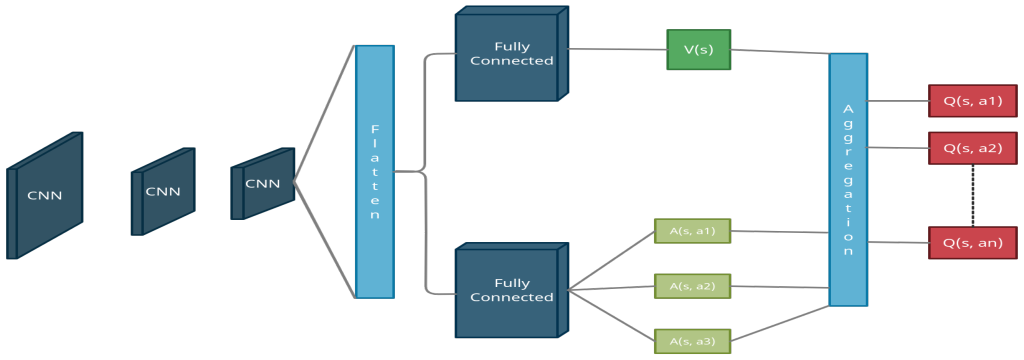


Figure 3: Dueling Deep Q-Network diagram

3.6 Dueling Double DQN

To take advantage of the Double DQN and Dueling DQN methods, a hybrid known as Double Dueling DQN (D3QN) was developed. This algorithm outperformed other techniques in the DQN family, especially in discrete action space problems [11]. The target Q-value is shown in equation below:

$$Q^{D3QN}_{s,a} = r(s', a') + \gamma Q(s', \arg\max Q(s', a; \theta, \alpha, \beta)) \tag{4}$$

3.7 Asynchronous Advantage Actor-critic (A3C)

However, one major disadvantage of DQN algorithm is the use of experience replay. Experience replay has several drawbacks it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy. In reinforcement learning, there is another method known as Actor-Critic. Using two neural networks, this technique combines value-based and policy-based methods into a single algorithm. First, there is the Actor network, which accepts state input and produces the best action. It effectively regulates the agent's behavior by learning the best policy (policy-based). In contrast, the Critic network evaluates the action by computing the value function (value based). This method was developed to take benefit of both policy-based and value-based approaches while eliminating all of their disadvantages. For example, policy-based approaches are superior for continuous and stochastic contexts, with faster convergence, whereas value-based approaches are more efficient and steady. the authors in a paper [12] used Asynchronous Advantage Actor-Critic (A3C) to asynchronously execute multiple agents in parallel, on multiple instances of the environment. This parallel reinforcement learning paradigm also offers practical benefits. Whereas previous approaches to deep reinforcement learning rely heavily on specialized hardware such as GPUs. A3C is a model-free, on-policy technique that maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate of the value function $V(s_t;\theta_v)$. It updates both the policy and the value-function in the forward view, using a combination of n-step returns. After every t_max actions or when the terminal state is reached, the policy and value function are updated. The algorithm's update can be observed in Algorithm 2.

Algorithm 2 Asynchronous Advantage Actor-Critic

```
1: Randomly initialize the global parameter $\omega$ and $\theta$;
2: Initialize the counter T with value 1;
3: **while** $T < T_{max}$ **do**
4:     Synchronize the local parameters $\omega'$ and $\theta'$ with the global ones $\omega$ and $\theta$;
5:     **while** $t < t_{max}$ or $s'$ is not terminal **do**
6:         Execute the action $a = \pi_\theta(a,s)$, then determine the appropriate state $s'$ and reward $r_t$;
7:         **if** $s'$ is terminal **then**
8:             $R = r_t$;
9:         **else**
10:            $R = r_t + \gamma R$
11:        **end if**
12:        Update the policy gradient and value gradient:
               $d\theta' = \nabla \log(\pi_\theta(a,s))(R - V_\omega(s))$    (5)
               $d\omega' = 2(R - V_\omega(s))\nabla_\omega(R - V_\omega(s))$    (6)
13:        Set $s = s'$
14:        Increase $t = t + 1$
15:    **end while**
16:    Update global $\omega$ and $\theta$ using $d\omega'$ and $d\theta'$
17: **end while**
```

4 Experiments

4.1 Pong

Initially, I implemented DQN, Double DQN, Dueling DQN and Dueling Double DQN to train the Pong game. The training parameter and results for this training are shown in Fig. 4.

Parameter	Value
batch size	32
learning rate	0.0001
gamma	0.99
epsilon	1.0
epsilon decrement	1e-5
epsilon mean	0.1
memory size	50000

Table 1: Input parameter for DQN family algorithm

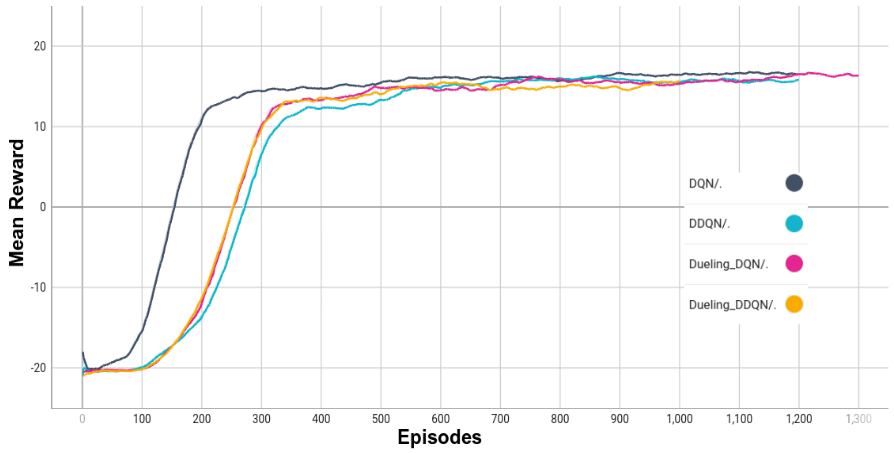


Figure 4: Mean rewards during training with DQN family algorithm

As seen in Fig.4, DQN is the most efficient algorithm in my training. It converts faster than other DQN algorithms. On the other hand, Dueling DQN tends to get higher mean score if the training episode is extended. The highest mean score for this game is 21. However, the best mean reward that my DQN model can achieve during training is between 16 and 17 score, which is imperfection. After finding and read several papers, I found another efficient algorithm in the paper [12] that implement A3C to train the Atari games. The A3C in this paper parallelly train multiple agents in one training episode. The number of parallel agent depends on the number of processor of your PC.

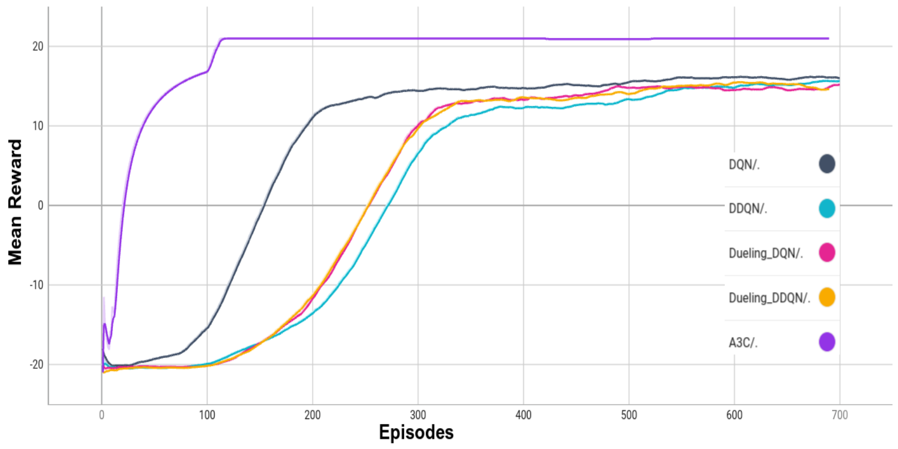


Figure 5: Mean rewards during training with DQN family and A3C algorithms

Fig. 5 demonstrates that the A3C algorithm works very well with the Pong game. The average reward for A3C converts after around 110 episodes, and the average score approaches the maximum score of 21.

4.2 Beam Rider

Beam Rider is the second game to be used to test my algorithms. It was training with the same models with Pong game.

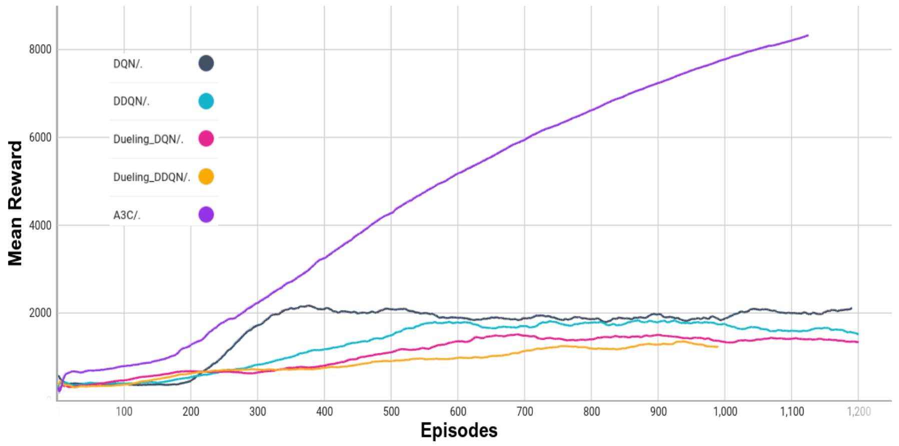


Figure 6: Mean rewards during training with DQN family and A3C algorithms

According to Fig. 6, the training for DQN family algorithm is converted during training. As a result, the conventional DQN is still the best model for this game among other DQN algorithms and it achieves the highest mean reward with approximately 2000 scores. On the other hand, the mean reward for A3C algorithm increase significantly and reaches over 8000 rewards after just 1200 episodes. As seen in the mentioned Figure, the A3C training curve has not yet converted and will continue to increase if we continue to train this algorithm. But I had to terminate the training early because this algorithm takes a long time to train.

5 Conclusion

In conclusion, this report used Deep Reinforcement Learning algorithms such as the DQN family and the A3C algorithms to play Pong and Beam Rider games. Furthermore, these algorithms have been trained and their performance has been compared in order to obtain knowledge and characteristics of each model. Within the scope of this study, the A3C algorithm outperformed the other five algorithms, and DQN is the best model within the DQN family algorithms.

References

[1] *Gym documentation*. [Online]. Available: hhttps://www.gymlibrary.dev/environments/atari/beam_rider/.

[2] M. Lapan, *Deep Reinforcement Learning Hands-On, Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and more*. Birmingham, UK: Packt Publishing, 2018, 546 pp., ISBN: 978-1-78883-424-7.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. DOI: 10.1109/MSP.2017.2743240.

[5] Y. Li, *Deep reinforcement learning: An overview*, cite arxiv:1701.07274, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274.

[6] T. Simonini, *Q-learning, let's create an autonomous taxi (part 1/2)*, Medium, Oct. 2020. [Online]. Available: https://thomassimonini.medium.com/q-learning-lets-create-an-autonomous-taxi-part-1-2-3e8f5e764358.

[7] *Experience replay*. [Online]. Available: https://paperswithcode.com/method/experience-replay.

[8] M. Sewak, "Deep q network (dqn), double dqn, and dueling dqn," in *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*. Singapore: Springer Singapore, 2019, pp. 95–108, ISBN: 978-981-13-8285-7. DOI: 10.1007/978-981-13-8285-7_8. [Online]. Available: https://doi.org/10.1007/978-981-13-8285-7_8.

[9] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, 2015. DOI: 10.48550/ARXIV.1509.06461. [Online]. Available: https://arxiv.org/abs/1509.06461.

[10] C. Yoon, *Dueling deep q networks*, Towards Data Science, Oct. 2019. [Online]. Available: https://towardsdatascience.com/dueling-deep-q-networks-81ffab672751.

[11] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, “Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot,” *Applied Sciences*, vol. 9, no. 24, p. 5571, Dec. 2019, ISSN: 2076-3417. DOI: 10.3390/app9245571. [Online]. Available: http://dx.doi.org/10.3390/app9245571.

[12] V. Mnih, A. P. Badia, M. Mirza, *et al.*, “Asynchronous methods for deep reinforcement learning,” 2016. DOI: 10.48550/ARXIV.1602.01783. [Online]. Available: https://arxiv.org/abs/1602.01783.