Double-DQN and One-Step Actor-Critic for "Balloon Protection Game"

Farshid Mahmoudabadi
Department of Computer Science and Engineering
University of Bologna
Bologna, Italy
farshid.mahmoudabadi@studio.unibo.it

Abstract

In this project, I investigate the use of reinforcement learning techniques, specifically Double Deep Q-Learning and One-Step Actor-Critic methods to train an autonomous playing agent in a novel game environment, developed as part of the project. Results demonstrate the agents' ability to learn expected strategies and adapt to the game dynamics. This work contributes to the broader field of autonomous and adaptive systems, demonstrating how artificial intelligence can optimize decision-making processes in novel, complex environments.

1 Introduction

This project is a component of the "Autonomous and Adaptive Systems" course, emphasizing the exploration and mastery of Reinforcement Learning (RL) in controlling intelligent machines. The course aims to provide a comprehensive understanding of contemporary RL techniques, focusing on the design, implementation, and evaluation of learning-based autonomous systems. Additionally, it addresses the ethical, societal, and philosophical dimensions associated with these technologies, underscoring their broader impact. Unibo [2023]

The main educational objective of this project was to deepen the comprehension of the control processes, with a specific emphasis on improving efficiency. To achieve this, I utilized advanced features of PyTorch, including CUDA for GPU acceleration. This project served as an exploration into a less-explored branch of Machine Learning within the master's program, offering valuable practical insights into the applications of RL.

Pygame was chosen for this project to create a unique and challenging environment to apply Reinforcement Learning (RL) principles. The game's design, with episodic gameplay, frequent punishments, a limited action set, continuous coordinations, asymmetric environment and finally an obvious set of increasingly more intelligent policies, make it perfect for exploring RL intricacies. Its complexity ruled out tabular representation, making it suitable for Deep RL approaches.

2 Background

I chose to explore both off-policy and on-policy reinforcement learning methods, specifically Double DQN and One-step Actor-Critic, to broaden my experience and understanding of different approaches to solving complex problems. Off-policy methods like Double DQN are known for their stability and effective handling of experience replay, allowing for more efficient use of past experiences in training. On the other hand, on-policy methods like One-step Actor-Critic offer a different perspective by directly learning from the most recent experiences, which can be advantageous in certain scenarios.

Preprint. Under review.

2.1 Double Q-Learning

Deep Q-Networks Mnih et al. [2013] mark a pivotal advancement in reinforcement learning, effectively addressing the challenges in generalization faced by traditional Q-learning in environments with continuous and complex state spaces. However, DQNs can overestimate Q-values due to the maximization step in the Q-learning update, which can negatively impact the learning process. Double Q-Learning Van Hasselt et al. [2015] mitigates this issue by using two separate networks: a selection network to choose the best action and a target network to evaluate the action's Q-value. This decoupling of action selection from action evaluation helps to reduce overestimations by ensuring that the selection of the best action is not biased by the same estimates used to evaluate it. The target value they use is

$$r + \gamma Q(s', \text{argmax}_a Q(s', a; \theta); \theta^-) \tag{1}$$

where $Q(s, a; \theta^-)$ denotes the target network parameters which are only updated to the current networks every C time steps.David [2020] In case of this project, at the end of each episode.

2.2 One Step Actor-Critic

The Actor-Critic method introduced in Sutton and Barto [2018] involves two separate components: the actor and the critic. The actor's goal is to learn a policy that maximizes the expected reward, while the critic's goal is to learn an accurate value function that can be used to evaluate the state. The One-Step Actor-Critic method introduces an update at each time step, which can potentially lead to faster convergence compared to methods that wait until the end of an episode such as REINFORCE.

The time difference error can be defined by:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2}$$

We expect training to improve the TD error as well; Having that goal, we can use the square of the time difference as a part of the loss. To update the actor network, we use the log probability because the math works out nicely. One loss function that can satisfy this criterion is the following:

$$\text{loss} = \delta^2 + (-\log(\pi_\theta(a|s)) * \delta) \tag{3}$$

2.3 Pygame

Pygame was selected for its seamless integration with Python and its capability to manage essential game development aspects like rendering and timing. Although it presents challenges such as lack of documentation; its straightforward approach to game loop creation and graphics management makes it a practical choice for projects where the focus is on AI development rather than intricate game design.

2.4 GRU

The Gated Recurrent Unit (GRU) Chung et al. [2014] featuring only two gates - a reset gate and an update gate - and omitting an output gate, GRUs simplify the architecture compared to Long Short-Term Memory (LSTM) units. By adjusting the reset and update gates dynamically, GRUs can selectively retain or discard information, allowing them to highlight and emphasize the most relevant features in a sequence.

3 Implementation

In the game, the player guides a shield to safeguard a balloon from randomly launched spikes, bouncing around the environment. The shield moves either clockwise or counterclockwise between eight possible positions to defend the balloon at the center. The game adds complexity with black holes that alter the path of spikes by repositioning them near the other black hole, and an obstacle to disrupt the symmetry of the environment. Represented in Figure 1

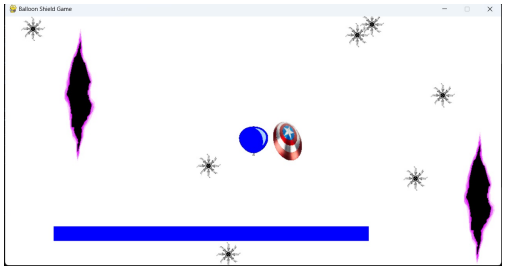

Figure 1: Game environment

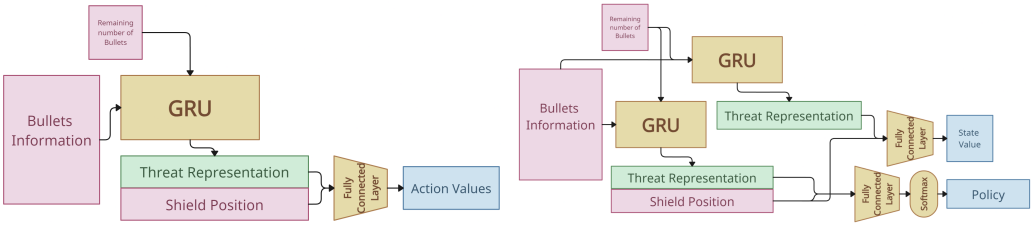

Figure 2: Double DQN network on the left and One Step Actor-Critic network on the righ; Inputs indicated in pink, Layers indicated in yellow, Mid-level representation indicated in green and Outputs indicated in blue.

3.1 Design Choices

Input representation The state representation at each time step, passed to the Neural Network consists of:

- One Hot encoding of the Shield position
- Remaining number of bullets
- Relative Coordinates of bullets w.r.t. the balloon
- Direction of movement of bullets
- Number of pixel displacement per step (speed) of bullets

Network Architectures I'll elaborate on two distinct architectures employed for different methods. In both cases, the primary input consists of information about the bullets, the shield's position, and the remaining number of bullets. The initial step in both networks involves a GRU layer responsible for filtering threatening information, producing a tensor of length 8 corresponding to possible shield positions. Subsequently, this tensor is concatenated with the encoded shield position. The two networks then diverge in their subsequent steps.

For the Double DQN network, a fully connected layer is employed, leading to the output of three values representing three possible actions. Represented in Figure 2

On the other hand, the Actor-critic network utilizes two fully connected layers after the initial concatenation, after which it outputs state value and policy seperately. Represented in Figure 2

Evaluation Occurs every 50 episodes of learning, encompassing 100 games. The metrics utilized for both training and evaluation phases are outlined below:

- **Winning Fraction**: This metric signifies the proportion of episodes concluding with either successful defense against all spikes or the agent surviving for 2000 episodes.
- **Actions Distribution**: This metric captures the distribution of actions taken by the agent within a specific episode during training moments. During evaluation, it represents the distribution across all 100 games.

- **Reward Sum**: The reward sum denotes the cumulative sum of all step rewards within an episode during training moments. In evaluation moments, it is the median of reward sums over all 100 episodes.

Optimizer The AdamW optimizer Loshchilov and Hutter [2017] is employed; The learning rate remains mostly constant throughout, with an exception during the initial training phase when the experience buffer lacks sufficient samples to fill a batch. During this phase, a lower learning rate is used to mitigate potential variance introduced by the limited samples.

Epsilon Greedy During training moments, epsilon value is set to 1 and reduced by 0.001 after each episode. Towards the end of 500 training episodes, it reaches to 0.5. This relatively high epsilon encourages exploration, postponing exploitation. During evaluation moments, this value is set to 0.01 since the goal is to exploit the agent's knowledge.

Discount Factor The discount factor (gamma) maintains a constant value of 0.99, encouraging the agent to look ahead to optimize its strategy considering the game's nature.

Smooth L1 Loss Sreekumar [2018] Can be interpreted as a combination of L1 and L2 losses. It behaves as L1-loss when the absolute value of the argument is high, and it behaves like L2-loss when the absolute value of the argument is close to zero. This combination allows the function to be less sensitive to outliers than the Mean Squared Error (MSE) loss, while retaining the benefits of being differentiable at the origin, unlike the Mean Absolute Error (MAE) loss.

Experience Replay Buffer The choice of experience buffer size presents a trade-off. A larger buffer, while accommodating an off-policy approach in Double DQN, reintroduces older trajectories that might not align with more recent model policy. Conversely, a smaller buffer emphasizes recent trajectories, enforcing the training less stable. To address this, an experimental approach involves filling a medium-sized experience buffer using multiple games simultaneously. This strategy aims to ensure a diverse set of experiences derived from a more recent model version, facilitating effective training.

Training process in Double DQN Each step of every episode involves passing the observed state (bullets information), along with the number of bullets and shield position, to the neural network model. The model then provides values for each of the three possible actions. Utilizing the epsilon-greedy method with the current value of epsilon, an action is selected. The game environment is advanced by one step, and the new state information is obtained.

All relevant information, including the current state (S), current action (A), action reward (R), next state (S'), and an indicator of whether the next state is terminal (done), is appended to the experience buffer. This buffer is then utilized within the neural network fitting function.

In the fitting function, a random batch of experiences (the tuple mentioned above) is sampled. Based on (1) a target for the current state-action value is computed. Subsequently, a loss is calculated, and the backpropagation process occurs. This function is called one time [or more] at the end of each step of the training process.

Training process in One-step Actor-Critic Each step of every episode involves passing the observed state (bullets information), along with the number of bullets and shield position, to the actor neural network model. The model then provides probability distribution over three possible actions, based on we can sample the action. The game environment is advanced by one step, and the new state information is obtained.

Then according to (3) with outputs of the critic model on the next and the current state, and also the actor model, a loss for each step of the process is kept, which then will be accumulated by losses of other steps (batch); and finally an update over the neural networks parameters happens.

4 Results

Training time As depicted in Figure 3, it is apparent that the Double DQN methods exhibit a relatively slow learning curve, whereas Actor-Critic methods demonstrate a clear advantage.

Figure 3: Training time reward sum per episode.

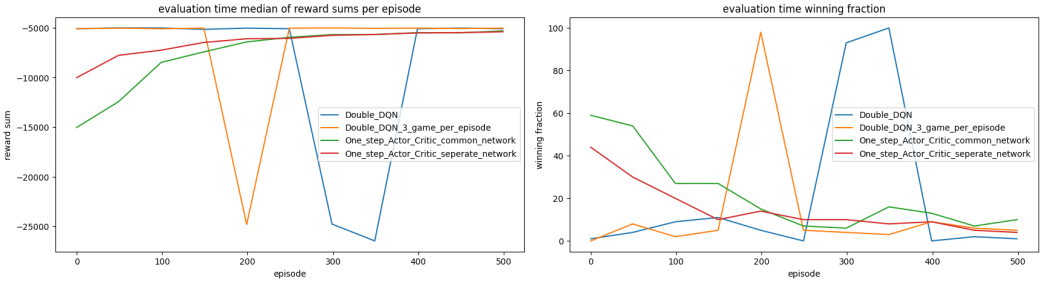

Figure 4: Evaluation time reward sum in the left and evaluation time winning fraction on the right.

Furthermore, the results of Double DQN display a notable variance in behavior, potentially attributable to the elevated epsilon level. A faster decaying epsilon might address this issue.

Evaluation Time As illustrated in 4, it is clear that all models prioritize the reward sum over winning the game. This inclination may be traced back to the reward design. To potentially mitigate this, reducing the penalty for moving the shield could be a viable solution. Furthermore, examining the action fraction during the evaluation time, as seen in 5, reveals the effective adaptation of sub-optimal policies by the Actor-Critic methods. In contrast, it is evident that Double DQN models exhibit a tendency to oscillate between two policies—namely, underactivity and overactivity. This behavior is somewhat expected due to the off-policy nature of these models.

5 Conclusion

This project effectively showcased the adaptability and proficiency of Double DQN and One Step Actor-Critic methods in navigating a dynamic game environment, developed using Pygame. The models demonstrated their ability to evolve from naive policies to effective strategies, underscoring their potential in addressing complex game environments. The project also offered valuable experience in comprehensive hyperparameter tuning, including the architecture of neural networks. For future work, it would be beneficial to compare the outcomes of this project with those obtained using other reinforcement learning techniques. Such a comparison would enhance understanding of the relative strengths of different methodologies.

References

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

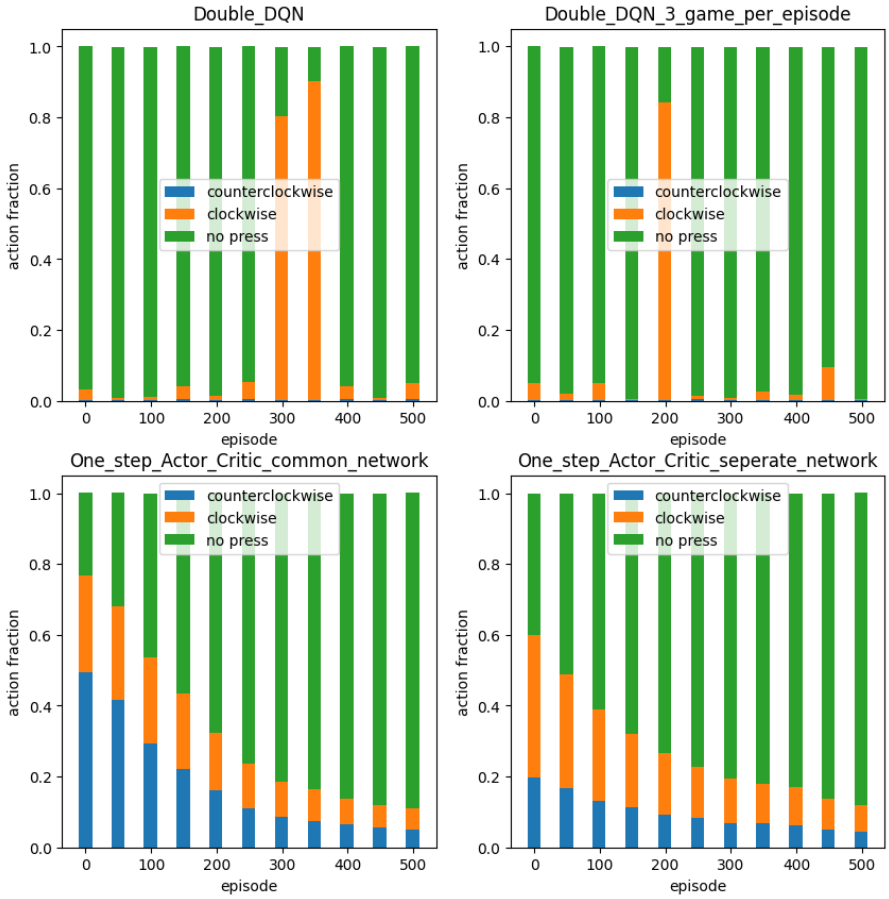

Figure 5: Evaluation time action fractions.

David. What exactly is the advantage of double dqn over dqn? AI Stack Exchange, 2020. URL `https://ai.stackexchange.com/questions/22776/what-exactly-is-the-advantage-of-double-dqn-over-dqn`. Accessed on January 19, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101v3*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Gautam Sreekumar. How to interpret smooth l1 loss? Stack Exchange, 2018. URL `https://stats.stackexchange.com/questions/351874/how-to-interpret-smooth-l1-loss`. Accessed on January 26, 2024.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Unibo. Autonomous and adaptive systems m, university of bologna website. `https://www.unibo.it/en/teaching/course-unit-catalogue/course-unit/2022/468076`, 2023. Accessed: January 25, 2024.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.