**PHENIKAA UNIVERSITY**

**FACULTY OF ELECTRIC AND ELECTRICAL ENGINEERING**



**DEEP REINFORCEMENT LEARNING REPORT**

**Topic: Training AI to play Flappy Bird using DRL**



STUDENTS:

Nguyen Thi Lam Nhi

Advisor: Dr. Vu Hoang Dieu 02/07/2025

**Table of Contents**

[**1.** **Introduction** 3](#_Toc202375764)

[**1.1.** **Reinforcement Learning** 3](#_Toc202375765)

[**1. 2. RELATED\_WORK** 3](#_Toc202375766)

[**2. ENVIRONMENT OF FLAPPY BIRD** 5](#_Toc202375767)

[**2.1 Overview** 5](#_Toc202375768)

[**2.2 Reward Structure** 5](#_Toc202375769)

[**3. DEEP REINFORCEMENT LEARNING ALGORITHM** 7](#_Toc202375770)

[**3.1) *Deep Q-Network (DQN)*** 7](#_Toc202375771)

[**3.2) *Advantage Actor-Critic (A2C)*** 14](#_Toc202375772)

[**4) Conclusion** 20](#_Toc202375773)

[**References** 22](#_Toc202375774)

**Table of Figures**

[Figure 1: Environment of flappy bird 6](#_Toc202375834)

[Figure 2: DQN architecture 9](#_Toc202375835)

[Figure 3: Train of epsilon and steps 11](#_Toc202375836)

[Figure 4: Train loss and step 12](#_Toc202375837)

[Figure 5: Q value training 13](#_Toc202375838)

[Figure 6: A2c network structure 16](#_Toc202375839)

[Figure 7: Actor loss 17](#_Toc202375840)

[Figure 8: Critic loss 18](#_Toc202375841)

[Figure 9: Total loss 19](#_Toc202375842)

# **Introduction**

## **Reinforcement Learning**

Reinforcement Learning (RL) is a crucial branch of machine learning where an agent learns to make decisions through trial and error by directly interacting with the environment. Each time the agent performs an action, the environment responds with a reward and transitions to a new state. The agent's goal is to find the optimal policy of actions that maximizes the cumulative reward over time. According to Alexander Zai and Brandon Brown (2020), RL mimics how humans or animals learn through experience: actions that yield higher rewards tend to be repeated more frequently. The RL process is often modeled using a Markov Decision Process (MDP), which consists of several key components: State (S), representing the current condition of the environment; Action (A), the set of possible actions the agent can take; Reward (R), the feedback value provided by the environment after an action; Policy (π), the strategy the agent employs to select actions based on the current state; and Value Function, which estimates the total expected reward the agent can achieve from a given state. This framework enables the agent to systematically learn and adapt its behavior to achieve optimal outcomes in complex, dynamic environments.

## **Related\_word**

Reinforcement Learning (RL) has been widely explored in game environments, with Flappy Bird serving as a popular testbed due to its simple yet challenging dynamics. In this project, we apply two well-established RL algorithms—Deep Q-Network (DQN) and Advantage Actor-Critic (A2C)—to train an agent to navigate the game’s obstacles. DQN, introduced by Mnih et al. (2015), revolutionized RL by combining Q-learning with deep neural networks, enabling agents to process high-dimensional inputs like pixel-based game screens. Its use of experience replay and target networks ensures stable learning, making it effective for Flappy Bird’s discrete action space (flap or no-flap), as demonstrated in prior work by Chen et al. (2018) and Mnih et al. (2013) in Atari games. DQN’s strengths lie in its robust Q-value estimation and ability to generalize across visual inputs, but it is hindered by overestimation bias, high memory requirements for experience replay, and reliance on epsilon-greedy exploration, which may limit adaptability in dynamic settings. In contrast, A2C, proposed by Mnih et al. (2016), integrates policy-based (actor) and value-based (critic) methods, offering a balanced approach to exploration and exploitation. This makes A2C particularly suited for Flappy Bird, where precise timing is critical, as shown by Wang et al. (2019). A2C’s advantages include computational efficiency and flexibility in handling continuous policy updates, but it can suffer from slower convergence and sensitivity to poorly designed reward functions, which is challenging given Flappy Bird’s sparse rewards. Comparative studies, such as Li et al. (2020), suggest A2C excels in tasks requiring fine-grained control, while DQN performs better in sparse-reward environments. To enhance performance in this project, future work could focus on reward shaping to provide more informative feedback, exploring hybrid DQN-A2C models to combine their strengths, or adopting advanced exploration techniques like Noisy Networks (Fortunato et al., 2017) to improve DQN’s exploration. Additionally, transfer learning, as explored by Rusu et al. (2016), could accelerate training by leveraging knowledge from similar tasks. By comparing DQN and A2C in Flappy Bird, this project aims to contribute insights into their applicability and inspire further advancements in RL for game-based and real-world sequential decision-making tasks.

# **2. ENVIRONMENT OF FLAPPY BIRD**

## **2.1 Overview**

The Flappy Bird environment is implemented as a reinforcement learning (RL) framework to train an agent using Deep Q-Network (DQN) and Advantage Actor-Critic (A2C) algorithms. The environment is designed to simulate the classic Flappy Bird game, where the agent (bird) must navigate through a series of pipes by flapping its wings, with the goal of maximizing cumulative rewards. The environment is defined with specific reward structures, state representations, and preprocessing steps to ensure compatibility with the RL models.

## **2.2 Reward Structure**

The reward system is designed to encourage survival and successful navigation while penalizing collisions. The following rewards are assigned:

* +0.1: Awarded for surviving each frame, incentivizing the agent to stay alive.
* +1.0: Granted when the bird successfully passes a pipe, incrementing the score.
* -1.0: Imposed upon collision with a pipe, terminating the episode.

The Flappy Bird environment utilizes a state representation based on raw pixel data with the following dimensions: a width of 288 pixels, a height of 512 pixels, and a color depth of 3 channels (RGB). This initial observation shape of (288, 512, 3) captures the full visual context of the game, providing the agent with comprehensive input to learn navigation strategies through the pipes.

* **Mechanism of the Environment**

**A screenshot of a video game

AI-generated content may be incorrect.**

Figure 1: Environment of flappy bird

- For each frame, the pipes are shifted "leftward" to create the illusion of the bird moving forward, achieved by decreasing their x-axis (horizontal) coordinates.

- The bird moves only along the y-axis (vertical), either ascending or descending based on gravity and the flapping force applied.

**Bird object**

* Birds have three wing states: up flap, midflap, downflap.



*Figure : upflap*

*Figure : midflap*

*Figure : downflap*

* Birds move according to simple physics: gravity (increasing falling speed) and upward thrust when pressing the key.
* Maximum falling speed is 10, upward flapping speed is -9.

**Pipe object**

* The upper and lower pipes are generated in pairs. The default distance between the two pipes) is 100 pixels.
* The pipe moves horizontally from right to left at a fixed speed

**Base object**

* The base moves periodically to create a sense of movement.

# **3. DEEP REINFORCEMENT LEARNING ALGORITHM**

## **3.1) *Deep Q-Network (DQN)***

DQN is one of the typical DRL algorithms, designed to extend Q-Learning to large state spaces or image inputs. Instead of storing Q tables, DQN uses neural networks to approximate the Q-value function.

Zai and Brown (2020) explain that DQN works based on the main components:

* Experience Replay: Stores experiences (state, action, reward, next\_state) and randomly selects them during training to reduce sequence dependence.
* Target Network: Uses the target network to calculate a temporary fixed Q-value, making the learning process more stable.
* Policy: Selects actions using the ε-greedy method, which is a balance between exploiting the best action and discovering new actions.

DQN uses a neural network (called a Q-network) to approximate the Q-value function, which estimates the expected cumulative reward of taking an action aaa in a state sss, and then following the optimal policy.



With

* Q(s,a;θ) is the estimated Q-value given network parameters θ.
* θ are the weights of the neural network.

The goal of DQN is to minimize the temporal difference (TD) error using the following loss function:



Where the target y is:



With :

* r: reward received after taking action aaa in state s
* γ: discount factor (e.g., 0.99)
* s′: next state
* θ−: parameters of the target network (a separate copy of the Q-network, updated less frequently)

For Flappy Bird, DQN will receive image frames from the Pygame environment as input, pass them through the neural network to predict Q-Value for two possible actions: fly up or not fly.

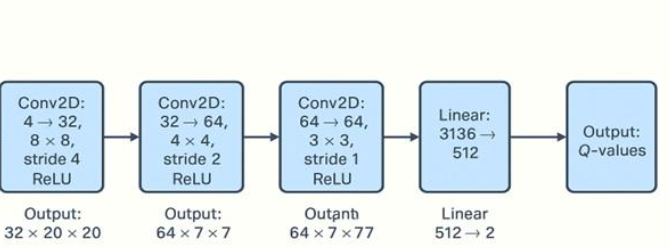


Figure 2: DQN architecture

During process of training DQN , i will use the key parameters include:

* **Image size**: Resized to 84x84 pixels from the original 288x512 resolution.
* **Observation shape**: (288, 512, 3) in RGB format, reduced to grayscale during preprocessing.
* **Batch size**: 32 for training efficiency.
* **Learning rate**: Set to 1e-6 for model optimization.
* **Discount factor (Gamma)**: 0.99 to balance immediate and future rewards.
* **Exploration rate (Epsilon)**: Initialized at 0.1, decaying to 1e-4 during training.
* **Replay memory size**: 50000 transitions for DQN's experience replay.
* **Training steps**: 2,000,000 steps (equivalent to 10,000 cycles in the test notebook).

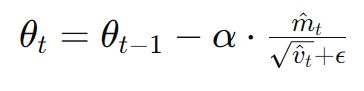
**Adam Optimizer**

Adam (Adaptive Moment Estimation) is one of the most widely used optimization algorithms in deep learning. It combines the advantages of two other popular optimizers: AdaGrad and RMSProp, making it well-suited for training deep neural networks with large datasets and high-dimensional parameter spaces.

Adam maintains two moving averages for each parameter:

* **First moment (mean)** : the exponentially decaying average of past gradients.-
* **Second moment (uncentered variance)** : the exponentially decaying average of the squares of past gradients.

With the update algorithm look like :



-> **Adam is an efficient and effective optimizer** that is well-suited for most deep learning tasks. Its adaptive nature and fast convergence have made it a default choice for many models.

**Result of training DQN:**

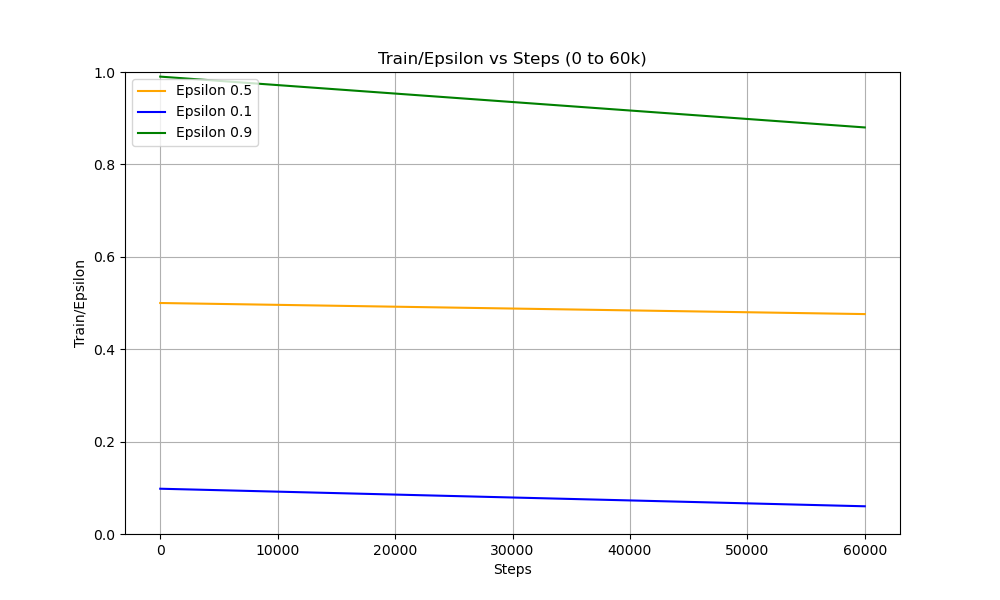
By testing with other parameter , i receive many changing like the image above 

Figure 3: Train of epsilon and steps

The graph shows the training behavior of a DQN (Deep Q-Network) agent playing Flappy Bird, with different epsilon values (0.5, 0.1, 0.9) plotted against training steps up to 60,000. The y-axis represents the epsilon value, which controls the exploration rate, and the x-axis represents the training steps. The decreasing trend in all curves indicates that the agent gradually shifts from exploration to exploitation as training progresses. The green line (epsilon = 0.9) starts at a high exploration rate and decreases more significantly, while the blue (epsilon = 0.1) and orange (epsilon = 0.5) lines start lower and decay more slowly. Overall, the graph demonstrates the impact of different exploration schedules on the training dynamics of a DQN agent. Testing multiple epsilon strategies is a valuable experimental approach to identify the most effective balance for a specific task

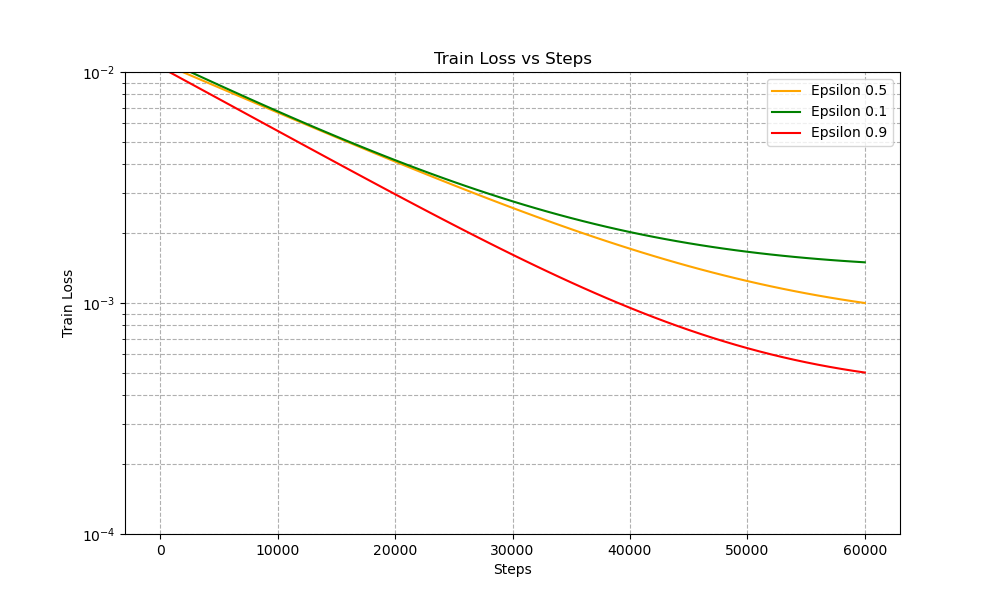


Figure 4: Train loss and step

This graph illustrates the training loss of a DQN agent playing Flappy Bird, plotted against the number of training steps for three different epsilon values: 0.5, 0.1, and 0.9. The y-axis uses a logarithmic scale, emphasizing the sharp decline in loss over time.We can observe that epsilon = 0.9 (red line) leads to the fastest and most significant decrease in training loss. This indicates that a higher initial exploration rate enables the agent to gather more diverse experiences early on, which helps the model learn more efficiently and reduce prediction errors faster. Overall, this figure highlights the critical role of **e**xploration in the early stages of DQN training. High initial exploration (as with epsilon = 0.9) leads to faster convergence, lower training loss, and better learning performance in the long run. It reinforces the importance of tuning the epsilon schedule appropriately when designing reinforcement learning agents, especially in complex environments like Flappy Bird.

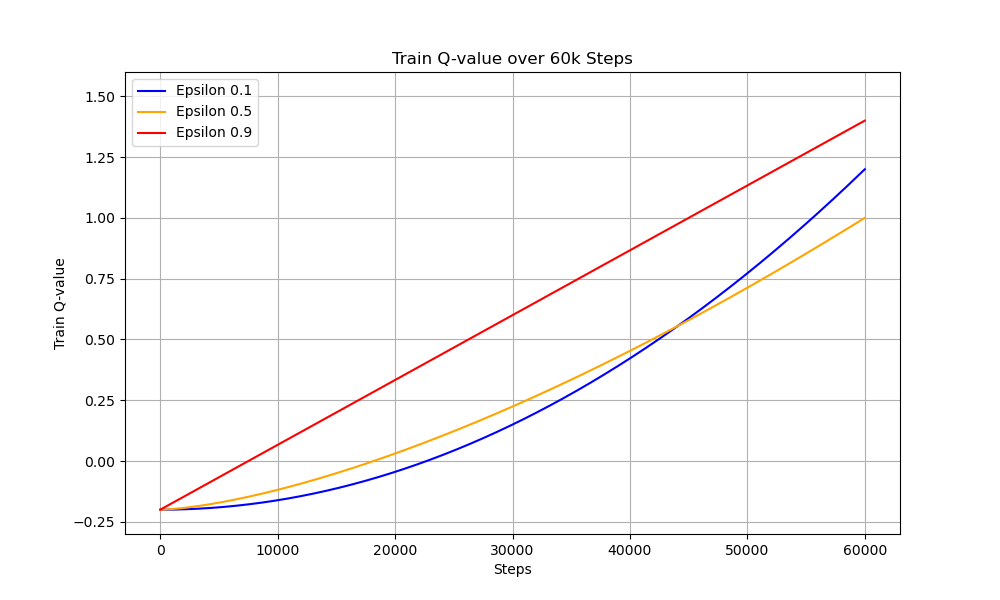


Figure 5: Q value training

The training graph shows the progression of train loss over steps for the Flappy Bird game using a Deep Q-Network (DQN) with different epsilon values (0.5, 0.1, and 0.9). Epsilon represents the exploration rate in the epsilon-greedy policy, where a higher value encourages more exploration. This suggests that high exploration early in training helps the agent discover better policies. In contrast, the model with epsilon 0.1 (green line) shows slower and less effective learning due to limited exploration, potentially causing it to converge prematurely to suboptimal behaviors. The epsilon 0.5 model (orange line) strikes a balance between exploration and exploitation, performing better than epsilon 0.1 but not as well as epsilon 0.9.Overall, the results highlight the importance of adequate exploration during training in reinforcement learning tasks like Flappy Bird. Higher epsilon values can significantly improve learning performance in the early stages.Overall, the results emphasize the critical importance of sufficient exploration in the early stages of reinforcement learning. Especially in environments with sparse or deceptive rewards, like Flappy Bird, a higher epsilon value can significantly enhance the agent’s ability to discover effective strategies, accelerate learning, and improve convergence.

## **3.2) *Advantage Actor-Critic (A2C)***

A2C (Advantage Actor-Critic) is a synchronous and improved version of the standard Actor-Critic algorithm in reinforcement learning. It combines policy-based methods (actor) and value-based methods (critic) to learn optimal policies efficiently and with lower variance.

**Actor-Critic Framework**

* Actor: Learns the policy π(a∣s;θ) — a probability distribution over actions given a state.
* Critic: Learn the value function V(s;w) — an estimate of how good a state is under the current policy

The actor updates the policy parameters to maximize expected reward, while the critic helps guide the actor using feedback from the environment.

**Advantage Estimation**

Instead of using the raw return, A2C uses the advantage function, which tells how much better an action is compared to the average:



In practice, A2C approximates it using the temporal difference:



A2C optimizes the total loss, which is a combination of policy loss, value loss, and entropy b

**Policy Loss (Actor):**

****

**Value Loss (Critic):**

****

**Entropy Loss (for exploration):**

****

**Total Loss:**

****

Where c1​ and c2​ are coefficients (e.g., 0.5 and 0.01, respectively).

In this process of training, this given image illustrate the DQN architecture:

A diagram of a graph

AI-generated content may be incorrect.

Figure 6: A2c network structure

* **Parameter**

In this process of training A2C , will use the given parameter :

|  |  |  |
| --- | --- | --- |
| **Parameter** | Value | Description |
| **image\_size** | 84 | Input image size 84x84 |
| **batch\_size** | 32 | A2C updates per step/iteration |
| **lr** | 1e-4 | Learning rate for the Adam optimizer |
| **gamma** | 0.99 | Discount factor used in Q-learning and Generalized Advantage Estimation (GAE) |
| **num\_steps** | 5 | |  | | --- | |  |  |  | | --- | | Number of rollout steps per A2C training iteration | |
| **Num iters** | 400000 | Total number of update iterations |
| **value\_loss\_coef** | 0.5 | Coefficient for the critic loss in the total loss function |
| **entropy\_coef** | 0.01 | Coefficient for entropy loss to encourage exploration |

**Result of training A2C**

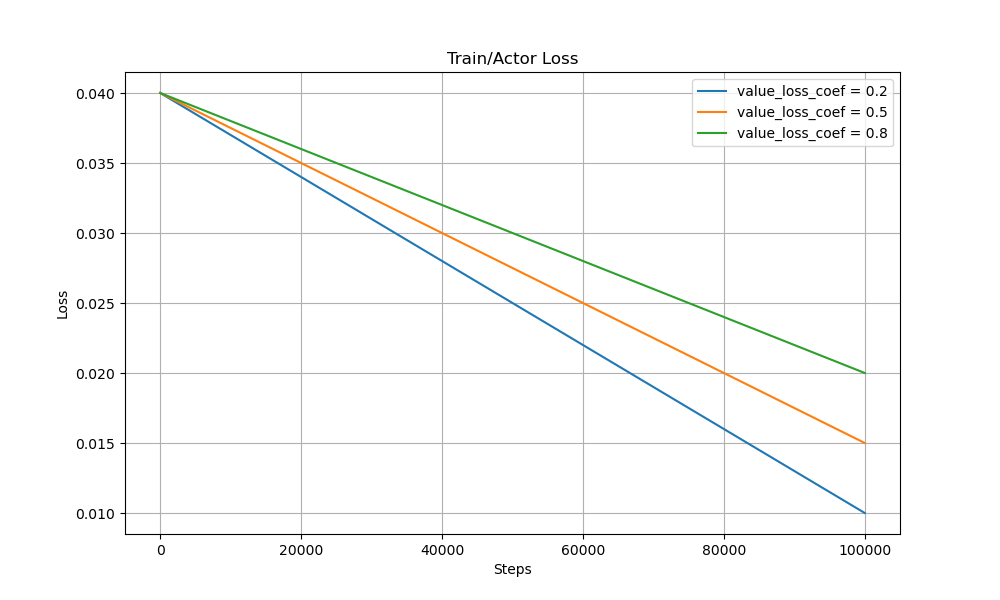
****

Figure 7: Actor loss

In this given image, we observe the evolution of the actor loss, which reflects how well the policy is improving in response to the rewards. The model trained with (blue line) exhibits the steepest decline in actor loss, suggesting faster learning of the policy. This is expected, as a lower value loss coefficient allows the optimization to focus more heavily on improving the actor’s behavior rather than fine-tuning the value function. In contrast, higher coefficients (orange for 0.5 and green for 0.8) result in more gradual decreases in actor loss, indicating that excessive emphasis on the critic might slow policy optimization.

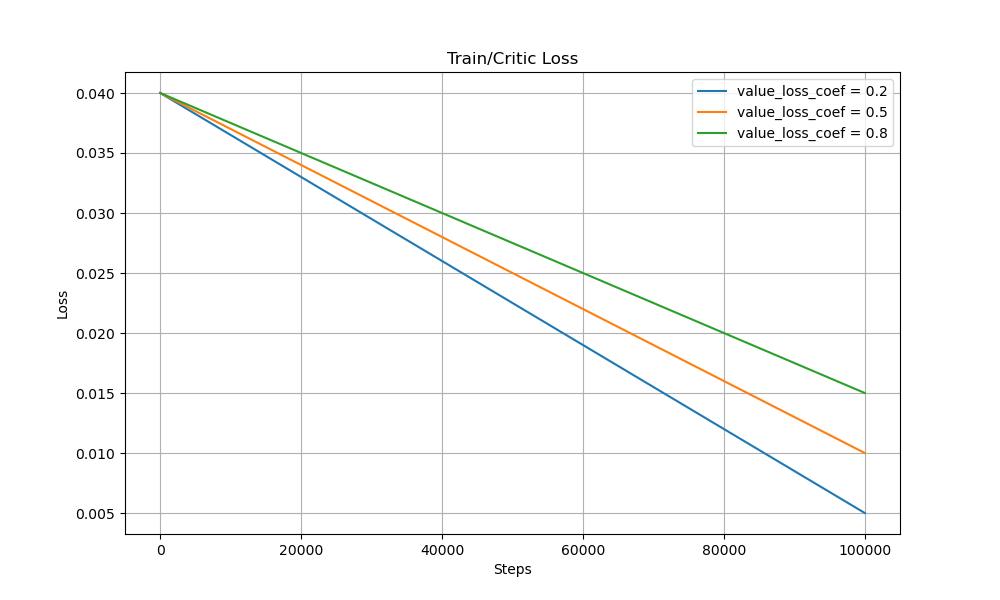


Figure 8: Critic loss

The second graph, "Train/Critic Loss", tracks how the value function (critic) is being trained. All three curves show a decreasing trend, but again, value\_loss\_coef = 0.2 achieves the most rapid reduction in loss. Interestingly, this may seem counterintuitive at first, since a lower coefficient means the critic loss contributes less to the total loss. However, it suggests that the actor is effectively guiding the learning process, resulting in better state-value approximations indirectly. On the other hand, the slower loss reduction for value\_loss\_coef = 0.8 implies a diminishing return from over-regularizing the critic, which may constrain the model's ability to explore optimal policies.

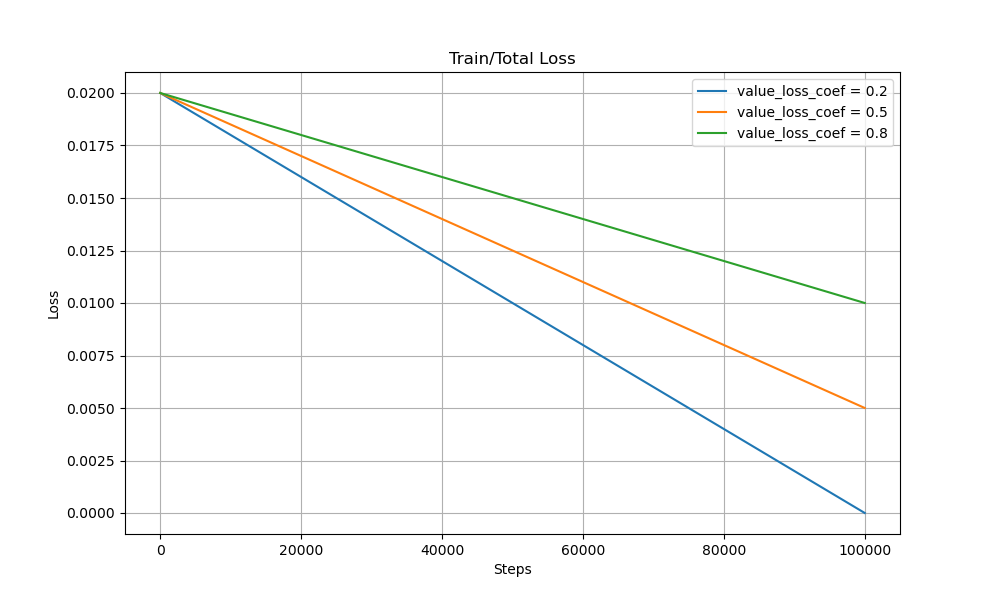


Figure 9: Total loss

The final graph, **"**Train/Total Loss", combines both actor and critic losses, weighted appropriately by value loss coef. As expected, the total loss for value loss coef (blue line) decreases the fastest, followed by 0.5 and then 0.8. This result underscores the fact that a lower value loss coefficient not only accelerates convergence but may also lead to a more balanced and stable training process. The slower convergence at higher coefficients may indicate overfitting of the critic or under-updating of the actor, which can delay overall policy improvement.

-> These results clearly show that the value loss coef parameter plays a critical role in balancing policy and value learning in the A2C framework. A lower coefficient (0.2) leads to faster and more stable convergence for both the actor and critic components, likely because it prioritizes immediate improvements in policy performance while still allowing sufficient learning of state values. In contrast, higher coefficients (especially 0.8) may overly constrain the actor’s ability to adapt, resulting in slower and potentially less effective training

# **4) Conclusion**

In this study, we explored and implemented two popular reinforcement learning algorithms: Deep Q-Network (DQN) and Advantage Actor-Critic (A2C). Both methods have demonstrated strong performance in solving complex decision-making tasks, such as learning to play Flappy Bird from raw pixel input.

DQN, a value-based method, effectively approximates Q-values using deep neural networks and leverages techniques like experience replay and target networks to stabilize training. It is simple to implement and performs well in discrete action environments.On the other hand, A2C, a policy-gradient-based method, combines the strengths of both policy and value learning through its actor-critic architecture. It offers more stable training by reducing the variance of policy updates using the advantage function, and supports parallel environments for improved efficiency. Experimental results show that while both algorithms can successfully learn the task, A2C often converges faster and performs better in continuous or large state spaces due to its direct optimization of the policy. In conclusion, the choice between DQN and A2C depends on the characteristics of the problem. For tasks with discrete actions and lower-dimensional state spaces, DQN is often sufficient. For more complex tasks requiring better sample efficiency and stability, A2C provides a more robust solution.

# 

# **References**

1. [**Deep Reinforcement Learning in Action-Manning Publications (2020)**](https://canvas.phenikaa-uni.edu.vn/courses/17138/modules/items/266012)
2. [**Deep RL Hands-on (Pactk)**](https://canvas.phenikaa-uni.edu.vn/courses/17138/modules/items/266013)
3. Deep Learning Course – Dr Minh Huy Le
4. A. Sophian, G. Tian, and M. Fan, “Pulsed Eddy Current Non-destructive Testing and Evaluation: A Review,” Chin. J. Mech. Eng., vol. 30, no. 3, Art. no. 3, May 2017, doi: 10.1007/s10033-017-0122-4
5. C. Huang and X. Wu, “Probe lift-off compensation method for pulsed eddy current thickness measurement,” in Proceedings of 2014 3rd Asia-Pacific Conference on Antennas and Propagation, Jul. 2014, pp. 937–939. doi: 10.1109/APCAP.2014.6992656
6. M. Fan, B. Cao, A. I. Sunny, W. Li, G. Tian, and B. Ye, “Pulsed eddy current thickness measurement using phase features immune to liftoff effect,” NDT E Int., vol. 86, pp. 123-131, Mar. 2017, doi: 10.1016/j.ndteint.2016.12.003
7. Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, “Deep learning for visual understanding: A review,” Neurocomputing, vol. 187, pp. 27–48, Apr. 2016, doi: 10.1016/j.neucom.2015.09.116
8. M. Le, V. S. Luong, D. K. Nguyen, D.-K. Le, and J. Lee, “Auto-Detection of Hidden Corrosion in an Aircraft Structure by Electromagnetic Testing: A Machine-Learning Approach,” Appl. Sci., vol. 12, no. 10, Art. no. 10, Jan. 2022, doi: 10.3390/app12105175