Enhancing Flappy Bird Performance With Q-Learning and DQN Strategies

Yanqin Guo *

High School Affiliated to Fudan University, Qingpu Campus, Shanghai, China

* Corresponding Author Email: Yanqin.Guo@calhoun.edu

Abstract. Flappy Bird, a classic single-player game, boasts a deceptively simple premise yet proves to be a formidable challenge in achieving high scores. Various algorithms have been employed to improve its performance, yet a comprehensive assessment of Q-Learning and Deep Q-Network (DQN) in the context of this game remains elusive. This study undertakes the task of training Flappy Bird using both Q-Learning and DQN methodologies, showcasing the potency of reinforcement learning within the realm of gaming. Through meticulous comparisons and analyses, the paper uncovers the inherent strengths and weaknesses embedded within these algorithms. This exploration not only fosters a nuanced grasp of Q-Learning and DQN but does so by leveraging a simplistic gaming environment as the proving ground. Strikingly, the experimental results unveil an initial disadvantage for DQN during training, followed by a rapid surge in performance surpassing Q-Learning in mid-training. Conversely, Q-Learning demonstrates an aptitude for swiftly reaching its performance zenith. Both algorithms tout distinct merits: Q-Learning's adeptness in simpler tasks and DQN's reliability in tackling complex states. In conclusion, this study not only discerns algorithmic prowess but lays a foundational framework for broader application across diverse gaming scenarios. By delving into the nuances of Q-Learning and DQN, the paper establishes a clearer path for harnessing the advantages in shaping the future landscape of game optimization.

Keywords: Q-Learning; DQN; Flappy Bird.

1. Introduction

"Flappy Bird" was created by Dong Nguyen, a solo game developer hailing from Vietnam [1]. The game involves controlling a bird's flight using touch controls. Upon tapping the screen, the bird ascends, while releasing the touch makes it descend. The goal is to guide the bird safely through openings between pipes, demanding precise timing and control to prevent collisions. Fig. 1 is the screenshot of Flappy Bird. Deep reinforcement learning, which combines deep learning and reinforcement learning, has become prominent in tasks like playing games [2, 3]. Through training, this technique can autonomously derive optimal game strategies by grasping game rules and reward signals. By implementing deep reinforcement learning into "Flappy Bird," its advantages can be harnessed to enhance game performance. Although prior research has utilized algorithms like deep learning and reinforcement learning to train "Flappy Bird," few studies have systematically compared distinct algorithms, let alone offered intuitive analysis explaining their effectiveness. This study addresses these gaps. This paper focuses on applying Q-Learning and DQN strategies to enhance the performance of the game's main character [4, 5]. In doing so, it also highlights the pros and cons of these two algorithms during training and evaluation. This analysis aids in guiding future algorithm selection for similar applications. Furthermore, the paper delves into the underlying rationale behind these two different algorithms, elucidating their purposes and motivations.

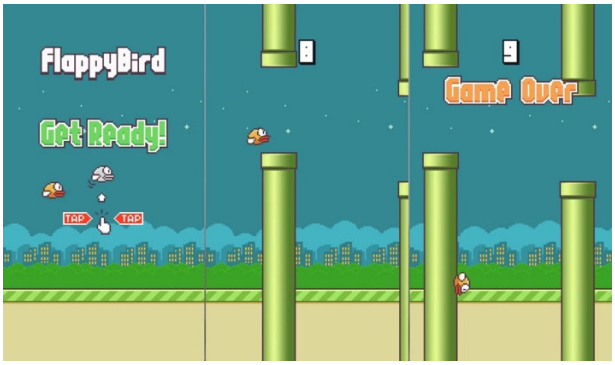

Fig. 1 Screenshot of Flappy Bird (Photo/Picture credit: Original)

2. Related Work

In this paper by Tai Vu and Leon Tran, reinforcement learning takes center stage, empowering an agent to gauge the expected utility of its state. This capability enables the agent to make optimal decisions in unfamiliar and dynamic environments [6]. Tai Vu and Leon Tran implemented SARSA and Q-Learning, but with some fine-tuning, including the introduction of an ε-greedy policy, discretization, and backward updates. These adjustments yielded impressive results, with their agent consistently achieving scores averaging 1400+ and even reaching the game's pinnacle score of 2069. Furthermore, the results under various variables were discussed, providing a comprehensive analysis. In another paper by Kevin Chen, the challenge of training an agent using only pixel information and the game score is addressed [7]. This is a demanding task, and in this study, the bird within the game must learn these representations and interactions to generalize effectively given the very large state space. Specifically, Kevin Chen employed DQN in the training process and thoroughly analyzed the results under different situations. Additionally, the paper explores various scenarios, including alterations in training time, training with an initial pre-trained network, and removing the reward for staying alive. In a third paper by Cedrick Argueta, Austin Chow, and Cristian Lomeli, the concept of transfer learning, a prevalent technique in machine learning, involves repurposing a model initially designed for one task as a foundation for another [8]. It is particularly favored in deep learning, where pre-trained models from specific tasks are utilized to expedite training and markedly enhance performance. In this study, Cedrick Argueta, Austin Chow, and Cristian Lomeli applied transfer learning to Flappy Bird and conducted a comparative analysis with the traditional reinforcement learning algorithm DQN.

While the two aforementioned papers delve into algorithms like DQN and Q-Learning, providing results and analyses under various conditions, there are also papers that exclusively focus on either Q-learning or DQN. For instance, "Applying Q-Learning to Flappy Bird" employs Q-Learning as their chosen approach for enhancing Flappy Bird's gameplay. On the other hand, "Playing Flappy Bird with Deep Reinforcement Learning" exclusively focuses on the application of DQN [9, 10].

3. Methodology

The training of Flappy Bird involves the implementation of Q-Learning and DQN Strategies. Concurrently, performance feedback encompassing actions, rewards, and scores are recorded throughout the training process, anticipating subsequent assessment of distinct performances aided by these two reinforcements learning algorithms.

3.1. Q-Learning

Q-Learning holds a pivotal role in the domain of reinforcement learning algorithms. Algorithm 1 provides a detailed depiction of Q-Learning's underlying mechanisms. Expressed as Q (s, a), it quantifies the reward acquired by executing action "a" in state "s" at a specific moment. The

environment responds by providing corresponding rewards based on action outcomes. This underscores the central principle of the Q-Learning algorithm: organizing states and actions into a Q-table to store Q-values and selecting actions that maximize benefits based on these values.

Algorithm 1: Q-Learning

parameters: $\alpha \in (0, 1]$, $\varepsilon > 0$
Initialize Q(s, a), for all $s \in S^+$, $a \in A(s)$, except that Q(terminal, ·) is arbitrarily set to 0
while Q does not converge:
  Initialize the bird's position S, start a new round of play
  While S != dead state:
    Use strategy π, get action A=π(S)
Play with action A, get the bird's new position S' and reward R(S, A)
Update Q(S, A) with $Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
Update S with S'

The crux of Q-Learning lies in two tables: the R-table and the Q-table. These tables share matching dimensions, with rows representing states and columns signifying actions. Through repeated learning leading to Q-table convergence, the optimal action's maximum return value can be executed in the current state as per the Q-table. Table R records rewards garnered from diverse actions in each state, established and fixed artificially. The entry at row n and column m in Table R signifies the immediate reward from action m in state n. The Q-table captures experiential learning, with its value at row n and column m representing the future long-term return attainable by executing action m in state n after a learning phase. This return value factors in both the immediate return post-action execution and the effect on the game's future progress. Nonetheless, Q-Learning has drawbacks due to inherent traits. It necessitates a Q-table, and with numerous states, the Q-table expands significantly, demanding substantial time and space for search and storage. Furthermore, the actions within these tables are discrete, not continuous. Moreover, real-world scenarios involve numerous environmental states, each with an array of actions, which undermines the effectiveness of the Q-table.

3.2. Off-policy algorithm DQN

DQN can be perceived as the union of Q-Learning and neural networks. Unlike Q-Learning which uses a table for Q-value maintenance, DQN employs a neural network to directly generate Q-values from input, which in this case comprises 4 image frames.

Q-Learning's core revolves around the Q-table, which directs actions based on its structure. However, this method is specifically tailored for scenarios featuring discrete state and action spaces. Moreover, the Q-Table's size grows exponentially with the number of states and actions, rendering it impractical for maintenance and search. Conversely, the DQN algorithm is generally unsuited for continuous action spaces. Q-Learning fundamentally maps a (state + action) to the (maximum reward from executing an action in that state). Abstractly, this equates to a mapping between input and output parameters, a domain where neural networks excel. The thought arises as to whether a neural network can replace Q-function calculations. This gave rise to a method that directly maps the current state to optimal action, known as the Deep Q Network.

3.3. Loss function and Hyperparameter

The loss function's role is to assess the disparity between the model's predicted value, denoted as f(x), and the actual value Y. Typically, this function is represented as L (Y, f(x)) and is a non-negative real-valued metric. A smaller value of the loss function signifies a greater level of model robustness. When training a neural network, the objective is to optimize this loss function. The process involves obtaining samples and subsequently updating the neural network's parameters using the gradient descent method through backpropagation. Thus, the target Q value is employed as a label, resulting in the following Formula 1 for the Q network's training loss function:

$$L(x)=E\ [(y + maxm'Q(t',m';x)-Q(t,m;x))*2] \quad (1)$$

Upon careful consideration, adjustments are made to the algorithm's parameters to optimize Flappy Bird's performance. The decay rate of past observations is set at 0.99. Additionally, Flappy Bird is allowed 1000 timesteps for observation before commencing training. The initial epsilon value is established as 0.01, with a final epsilon value of 0.001. To expedite the training process while conserving computational resources, the minibatch size is modified to 32. Furthermore, the number of previous transitions to retain is adjusted to 50000, and the frames over which to anneal epsilon are set to 200000.

3.4. Differences Between DQN and Q-Learning

DQN employs a neural network to substitute the Q-table for calculating Q-values, addressing the issue of dimension explosion. It incorporates a memory bank to store past experiences, featuring two key aspects: Random extraction which eliminates data correlation. Without random extraction, the acquired data might exhibit continuous correlation, potentially leading to a training dataset dominated by a specific state and hindering successful training. By employing random extraction, this correlation is disrupted to ensure a uniform distribution of training data samples. Rolling update, involving the replacement of older memories with more recent ones. Due to memory limitations, a constant update is essential, and newer memories are generally more valuable than older ones (although exceptions exist, as some crucial experiences could reside in older memories). This aspect falls under DQN's later optimization efforts. DQN temporarily freezes the Q-target parameter to maintain a fixed target. This allows for enhanced convergence of the neural network.

In Q-Learning, each row corresponds to a state, and each column represents an action. This configuration allows for the storage and retrieval of action values for each state-action pair. For the game Flappy Bird, assuming an input frame picture's pixel matrix is 80x80, with each pixel having 256 possible values, a single frame image leads to 256^(80x80) potential states. Additionally, the in-game birds have two actions available (jumping and taking no action). If Q-Learning were employed, the program would require a Q-table and an R-table each with a size of 2x256^(80x80), imposing considerable costs in terms of maintenance and lookup operations.

Given the inherent capacity of neural networks to handle complex representations, adopting them to manage Q-tables is a more suitable approach. DQN emerges from the integration of Q-learning and neural networks. While Q-Learning maintains Q-values using a table, DQN inputs a state (consisting of 4 image frames) and directly generates Q-values through a neural network. Consequently, DQN proves better equipped to address challenges posed by intricate state spaces. By evaluating the outcomes during and after the training process, the distinct performance of Flappy Bird becomes evident, effectively highlighting the strengths and weaknesses associated with both algorithms. With a comprehensive understanding of the theoretical pros and cons of both algorithms, delving deeper into their operational modes and conducting a comparative assessment of their merits and drawbacks enables a more lucid grasp of the strategies underpinning Q-Learning and DQN.

4. Results and Analysis

4.1. Initial and Intermediate Stages of Q-Learning and DQN

Fig. 2 illustrates the initial phases of Q-Learning and DQN. As the training iterations reached approximately 7000 to 8000, both algorithms began to exhibit improved performance. However, DQN displayed a tendency towards accumulating zero scores more frequently within the range of 0 to 10. On the other hand, Q-Learning demonstrated a propensity to achieve higher scores compared to DQN, even surpassing a score of 36.

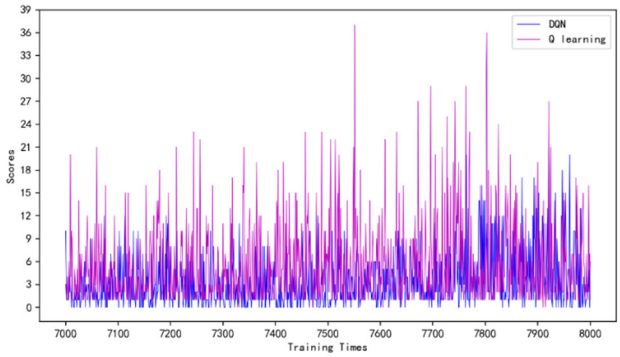

Fig. 2 Beginning part of Q-Learning and DQN (Photo/Picture credit: Original)

Fig. 3 provides insight into the intermediate stages of Q-Learning and DQN. Around 8000 to 9000 training iterations, a notable divergence in performance became evident. In the initial 400 rounds, the performances of the two algorithms closely resembled each other. However, as the rounds progressed, DQN started achieving notably higher scores, while Q-Learning's performance improved as well, albeit at a slower pace.

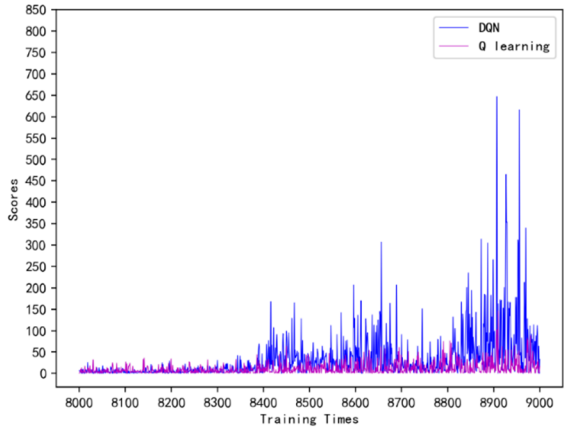

Fig. 3 Middle part of Q-Learning and DQN (Photo/Picture credit: Original)

In Fig. 4 below, the loss decreases significantly at the outset and remains close to zero as the episode progresses, indicating improved Flappy Bird performance. However, around episode 7000, the loss experiences a sudden increase and fluctuates between 0 and 0.3. This could be attributed to inadequate labeling of the training data or possibly due to continuous warming up, causing the learning rate to increase excessively. A large learning rate can result in substantial gradients, leading to an abrupt rise in loss.

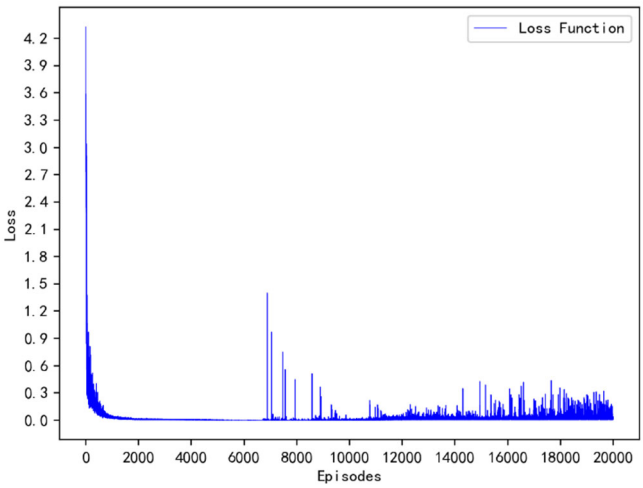

Fig. 4 Loss Function (Photo/Picture credit: Original)

4.2. Analysis and Comparison

Fig. 5 showcases the distinct differences in Q-Learning and DQN performance. A remarkable and dramatic shift occurred between 10000 and 10100 training iterations. Q-Learning reached a mastery level, overturning the situation and outperforming DQN. Conversely, DQN's performance deteriorated significantly during this period. This contrast could be attributed to Q-Learning's earlier attainment of a saturation point, while DQN had not yet reached such a stage.

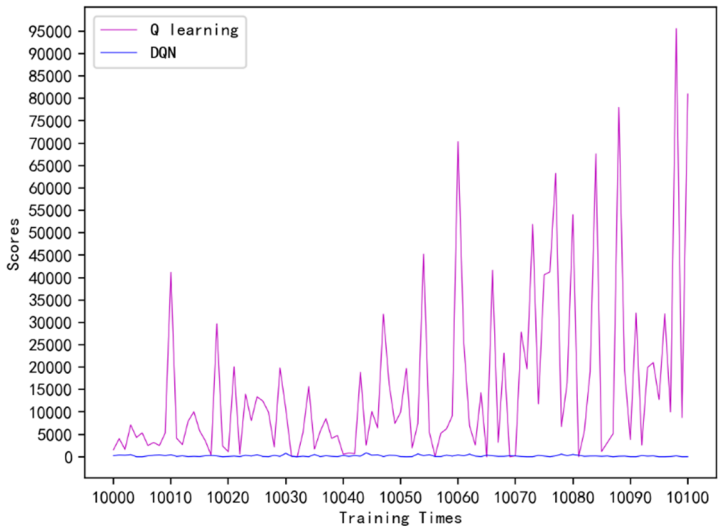

Fig. 5 Distinct different performances of Q-Learning and DQN (Photo/Picture credit: Original)

Fig. 6 outlines the challenges encountered during the training process. With increasing training iterations, sporadic drops in performance were observed, transitioning from favorable to worse outcomes. As illustrated in Fig. 6, at around 10600 training iterations, scores peaked above 8,000,000. However, with further iterations, the results regressed, at times resulting in the bird navigating only a couple of obstacles. This could be attributed to insufficient training iterations or the potential overshadowing of crucial experiences within the new memory.

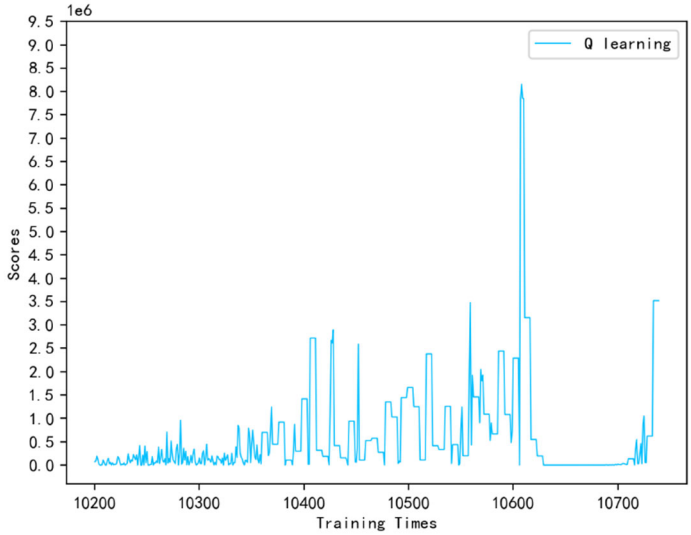

Fig. 6 Q-Learning Training Results (Photo/Picture credit: Original)

Considering the entire dataset, it can be inferred that DQN initially lagged in performance but exhibited faster progress in the intermediate phase compared to Q-Learning. Nonetheless, Q-Learning displayed a tendency to reach its saturation point more swiftly, achieving mastery before DQN. Analyzing the core principles of the two algorithms, DQN's progress can be accelerated and stabilized due to its neural network advantage. However, this doesn't undermine Q-Learning's efficacy. In

scenarios with simpler states, Q-Learning can attain high proficiency with fewer training iterations, effectively trading space for time.

5. Conclusion

Flappy Bird presents a seemingly simple gameplay yet poses challenges in achieving high scores. Numerous algorithms have been employed to optimize Flappy Bird's performance, yet a comprehensive comparison between Q-Learning and DQN in the context of the game remains lacking. This study delves into the utilization of both Q-Learning and DQN to train Flappy Bird, showcasing the efficacy of reinforcement learning in game enhancement. Through meticulous comparisons and analyses, the paper uncovers the strengths and weaknesses of these two algorithms. This process enables a gradual comprehension of Q-Learning and DQN through a simple game implementation. Experimentally, DQN exhibited a slower start during training but displayed accelerated progress in the mid-phase, contrasting Q-Learning's quicker saturation point attainment. Thus, each algorithm possesses its unique advantages. While DQN may demonstrate superior early performance, it lags in swiftly reaching a mastery level compared to Q-Learning. In essence, Q-Learning proves efficient for simpler tasks, while DQN excels in managing complex states. These insights simplify future algorithm selection in practical applications. Further experimentation can explore the upper limits of scores achievable by both algorithms. Additionally, this analysis can extend to encompass different algorithms and diverse games, contributing to the evolution of a more robust programming landscape.

References

[1] J Gu, Y Guo, Y Lam, et al. Flappy Bird Game Based on the Deep Q Learning Neural Network[J]. Highlights in Science, Engineering and Technology, 2023, 34: 191-195.

[2] V Mnih, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[3] R S Sutton, A G Barto. Reinforcement learning: An introduction[M]. MIT press, 2018.

[4] C J C H Watkins, P Dayan. Q-learning[J]. Machine learning, 1992, 8: 279-292.

[5] V Mnih, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[6] T Vu, L Tran. FlapAI bird: training an agent to play flappy bird using reinforcement learning techniques[J]. arXiv preprint arXiv:2003.09579, 2020.

[7] K Chen. Deep reinforcement learning for flappy bird[J]. CS 229 Machine-Learning Final Projects, 2015.

[8] C Argueta, A Chow, C Lomeli. Deep Reinforcement Learning and Transfer Learning with FlappyBird[J]. Machine learning, 1992, 8(4): 279-292.

[9] M Ebeling-Rump, M Kao, Z Hervieux-Moore. Applying q-learning to flappy bird[J]. Department Of Mathematics And Statistics, Queen’s University, 2016.

[10] L S Pilcer, A Hoorelbeke, A D Andigne. Playing Flappy Bird with Deep Reinforcement Learning [C][J]. IEEE Transactions on Neural Networks, 2015, 16(1): 285-286.