Training a Neural Network using Q-learning Algorithm to control an agent in the Flappy Bird game

Ioana-Delia Blendea
Faculty of Computer Science
Alexandru Ioan Cuza Univeristy of Iasi

Dragoș Mărtinaș
Faculty of Computer Science
Alexandru Ioan Cuza Univeristy of Iasi

***Abstract*—This paper details the implementation of an autonomous agent capable of mastering the Flappy Bird environment using raw pixel data. The proposed solution utilizes a Deep Q-Network (DQN) integrated with a Convolutional Neural Network (CNN) to process stacked, binary-thresholded frames. By employing experience replay and target networks, the agent successfully navigated the high-variance physics of the environment. Experimental results demonstrate a significant performance breakthrough following strategic learning rate decay, resulting in a peak score of 642 pipes and a sustained average of 115.5 pipes across 41 evaluation episodes. The study concludes that pixel-based training, when combined with temporal frame stacking and fine-tuned optimization, allows for superhuman proficiency in procedurally generated arcade environments.**

***Index Terms*—Deep Q-Network (DQN), Reinforcement Learning, Convolutional Neural Networks (CNN), Experience Replay**

I. Introduction

Reinforcement Learning (RL) has become a powerful framework for training agents to interact directly with visual environments. Among classic benchmark tasks, Flappy Bird presents a particularly challenging control problem due to its brittle dynamics: a single poorly timed action immediately terminates the episode. This work investigates whether an agent can learn a robust and high-performing policy using only pixel-level observations, without access to engineered state variables. A Deep Q-Learning approach is adopted, enabling the agent to infer spatial structure and temporal dynamics directly from processed image frames.

II. Methodology

A. Environment and Rendering

The agent interacts with the environment through the *flappy-bird-gymnasium framework*. To enable real-time visualization during training and evaluation, the Gymnasium-provided *HumanRendering wrapper* was used. This wrapper allows environments that only support *rgb_array* rendering to display frames in a human-readable window without modifying the underlying observation or interaction interface.

B. Visual Preprocessing and State Representation

Each observation undergoes a multi-stage preprocessing pipeline designed to reduce input complexity while retaining task-relevant information:

1) **Spatial Cropping:** The lower region of each frame corresponding to the ground is removed, allowing the network to focus on the bird and obstacle geometry while eliminating visually irrelevant information.
2) **Binary Thresholding:** Each frame is converted into a high-contrast binary image using the predefined outline color of the game objects, effectively suppressing background noise and reducing input complexity.
3) **Rescaling and Normalization:** The processed frames are resized to 84×84 pixels and normalized to the range $[0, 1]$ to ensure numerical stability during training.
4) **Temporal Frame Stacking:** Four consecutive frames are stacked along the channel dimension to form a $(4, 84, 84)$ state tensor, providing the agent with implicit temporal information such as motion and velocity.

This representation enables the agent to reason about both spatial structure and short-term temporal dynamics using visual input alone.

C. Neural Network Architecture

The Q-network is implemented as a convolutional neural network designed for efficient spatial feature extraction. It consists of three convolutional layers with kernel sizes of 7×7, 5×5, and 3×3, and corresponding strides of 3, 2, and 1, respectively. This configuration progressively reduces the spatial dimensionality of the input while preserving salient visual features. ReLU6 activation functions are employed throughout the network to improve numerical stability and mitigate activation saturation during extended training. The convolutional feature maps are subsequently flattened and passed to a fully connected layer with 256 units, which outputs the Q-values associated with the two available actions, *Wait* and *Flap*.

D. Q-Learning Algorithm Implementation

Training stability is achieved through the incorporation of several established Deep Q-Learning techniques. Experience replay is employed via a replay buffer containing up to 50,000 transitions, enabling the agent to sample past experiences uniformly and thereby reducing temporal correlations between consecutive training samples. To further stabilize learning, a

separate target network is maintained for the computation of Q-value targets, which mitigates oscillations and divergence during training. Additionally, frame skipping is applied such that each selected action is executed over two consecutive frames, aligning the agent's decision frequency with the underlying game physics and resulting in smoother control behavior.

III. Experiments and Hyperparameters

Training was conducted using GPU acceleration on an *NVIDIA GTX 1650* via CUDA. Key hyperparameters are summarized below:

TABLE I
Training Hyperparameters

Hyperparameter	**Value**
Gamma (γ)	0.99
Learning Rate (α)	1×10^{-4}
Batch Size	32
Replay Buffer Size	50,000
Observation Steps	1
Exploration Steps	10,000
Initial Exploration Rate (ϵ_{init})	1.0
Final Exploration Rate (ϵ_{final})	1×10^{-4}

IV. Performance and Results

During evaluation, exploration was disabled and actions were selected greedily with respect to the learned Q-function ($\epsilon = 0.0$).

A. High-Score Episodes

The highest-performing evaluation episodes are summarized below:

- 642 pipes (Episode 15)
- 348 pipes
- 326 pipes
- 324 pipes
- 321 pipes

B. Analysis of Performance Improvement

During training, agent performance initially plateaued at approximately 150-200 pipes. A significant improvement was observed following a reduction of the learning rate at epoch 420, which enabled the network to refine previously learned representations. This adjustment allowed the agent to better accommodate rare and challenging pipe configurations that had previously resulted in catastrophic failures, leading to a marked increase in both maximum and average performance.

V. Conclusion

This study demonstrates that raw pixel-based input, when combined with targeted visual preprocessing and temporal frame stacking, constitutes a sufficient state representation for mastering the Flappy Bird environment. The proposed Deep Q-Learning framework achieves reliable and superhuman performance despite the environment's highly sensitive dynamics. These results underscore the effectiveness of convolutional architectures and stabilized Q-learning methods for vision-based control tasks in procedurally generated environments.