Flappy Bird Reinforcement Learning Report

Nofal Jaber, Vilcu-Razvan Valeriu

1. Introduction

This project implements a Reinforcement Learning (RL) agent capable of playing "Flappy Bird" using raw pixel inputs. The agent utilizes a **Dueling Deep Q-Network (Dueling DQN)** architecture implemented in PyTorch. The training process involves preprocessing game frames to reduce complexity and using a replay buffer to stabilize learning.

2. Architecture and Implementation

2.1 Training Based on Pixels

- **Preprocessing**: The `FlappyBirdPreprocess` class captures the raw RGB render of the game.
- **Grayscale & Resizing**: High-definition frames are converted to grayscale and resized to an 84x84 matrix to reduce computational load.
- **Frame Stacking**: To allow the agent to perceive motion (velocity and acceleration), 4 consecutive frames are stacked, resulting in an input tensor shape of `(4, 84, 84)`.

2.2 Neural Network Architecture (Dueling DQN)

The model uses a **Dueling DQN** architecture, which separates the estimation of the state value V(s) and the advantage of each action A(s, a). This helps the agent learn which states are valuable regardless of the action taken.

Network Layers:

1. **Convolutional Layers**: The feature extractor consists of three convolutional layers:
    a. Conv2d: 32 filters, kernel size 8, stride 4.
    b. Conv2d: 64 filters, kernel size 4, stride 2.
    c. Conv2d: 64 filters, kernel size 3, stride 1.

2. **Dueling Streams**: The output is flattened and split into two separate fully connected streams:
    a. **Value Stream**: Estimates the scalar value of the current state.
    b. **Advantage Stream**: Estimates the advantage for each possible action.

2.3 Q-Learning Algorithm Implementation

The implementation follows standard Deep Q-Learning practices to ensure stability:

- **Experience Replay**: Transitions (`state`, `action`, `reward`, `next_state`, done) are stored in a `ReplayBuffer` with a capacity of 50,000. Random batches are sampled for training to break correlations between consecutive frames.
- **Target Network**: A separate target network is used to calculate the expected Q-values. Its weights are updated with the policy network's weights every 500 steps (TARGET_UPDATE) to prevent oscillation.
- **Loss Function**: The model optimizes the `Smooth L1 Loss` (Huber Loss) between the current Q-values and the target Q-values.
- **Reward Shaping**: To encourage survival, a small positive reward (`+0.1`) is added for every step the agent stays alive, in addition to the standard game rewards.

3. Hyperparameters

The following hyperparameters were configured for the training session:

Hyperparameter	**Value**	**Description**
Batch Size	128	Number of samples per training step
Gamma (γ)	0.95	Discount factor for future rewards
Learning Rate	3×10^{-4}	Step size for the Adam optimizer
Epsilon Start	1.0	Initial exploration rate (100% random)
Epsilon End	0.01	Minimum exploration rate
Epsilon Decay	5000	Rate at which exploration decays
Memory Size	50,000	Capacity of the replay buffer
Target Update	500	Steps between target network updates

4. Experimentation and Performance

The training logs indicate the agent's progression from random actions to proficient gameplay.

4.1 Score Progression

The `best_score.txt` file tracks the maximum total reward achieved during training. The scores reflect the custom reward shaping (survival points + pipe points).

- **Early Stage (Episodes 0-100)**: The agent struggled significantly, with scores fluctuating between `-3.80` and `4.00`. This corresponds to the high epsilon phase where actions are mostly random.
- **Learning Phase (Episodes 300-700)**: The agent began to grasp the mechanics, achieving scores like `10.30` (Ep 679) and `11.10` (Ep 697).
- **Proficiency (Episodes 1000+)**: A sharp increase in performance occurred after episode 1000. The score jumped to `19.20` at Episode 1009, then rapidly increased to `55.60` (Ep 1380) and `100.70` (Ep 1590).
- **Best Recorded Performance**: Out of 2000 episodes, the agent achieved a high score of **188.50** at Episode 1752.