Flappy Bird Agent (Assignment 5) - Technical Report

Alexandru-Valentin Bașagă

January 2025

1 Introduction

This is the technical report for a Flappy Bird AI agent, implemented and trained through a neural network using the Q-learning algorithm, done for the Neural Networks course.

2 Applied Technologies

- **Flappy Bird Environment:** In order to train the AI, "FlappyBird-v0" for Gymnasium environment was used, which has inbuilt methods to yield simple numerical information about the game state as observations or RGB-arrays (images) representing the game's screen. In this project, LIDAR sensor readings (180° ahead) are given as information to the agent. Everything needed for Q-learning is provided as well; an action space (flap/do nothing), as well as a reward system (`+0.1` for every frame alive, `+1.0` for every pipe passed, `-1.0` for dying, `-0.5` for touching the top of the screen).
- **Neural Network:** A neural network is implemented using PyTorch's `nn` module. This architecture is specifically designed to approximate the Q-value function, which guides the agent's decision-making process in the Flappy Bird environment. The core components of the neural network includes a fully connected feedforward structure:
  1. an input layer that accepts the observations made by the agent
  2. two hidden layers thatintroduce non-linearity using the `ReLU` as activation function
  3. an output layer that produces Q-values representing the two actions the agent can take (flap/do nothing).
- **Replay Buffer:** A replay buffer is used to store and sample past experiences, which improves the stability of neural network training. This

random sampling breaks the correlation between consecutive experiences and helps the network generalize better across different game states.

- **Q-learning algorithm with epsilon greedy policy:** A Q-learning algorithm is implemented using an epsilon-greedy policy for action selection and a training loop to update the model. The epsilon-greedy policy balances exploration (finding out new things about the environment) and exploitation(doing what it knows already) by randomly selecting an action with probability epsilon or choosing the action with the highest Q-value from the model. The train function implements the Q-learning update rule by calculating the time difference error between the predicted Q-values and the target Q-values from a target network, sampled from a replay buffer. `SmoothL1Loss` is used to minimize loss, while epsilon decays linearly from $\epsilon_{\text{start}} = 1$ to $\epsilon_{\text{end}} = 0.01$, reaching the end value at around 80% of training epochs in, at which point the agent focuses more on exploitation rather than exploration.

3 History of Project

In the making of this project, many different implementations and hyperparameter tunings have been attempted:

- varied training times were tested, to see which performed the best in reinforcement learning;
- preprocessing the LIDAR data to aid in the deep learning neural network, most attempts of which did not provide to be very meaningful since the data offered by the environment is already in quite a concise and easy to work with form;
- altering the rewards per frame of staying alive in order to further motivate the agent to stay alive, and also some very noticeably failed attempts in trying to normalize the rewards by clipping, which (obviously, in hindsight) had a negative effect on the learning process;
- altering the decay for the epsilon-greedy policy - originally, a fixed amount was substracted from epsilon, which was not flexible enough for varied training episode counts, and so it was then modified to an exponential decay (which was too fast at the start and too slow at the end), and then finally settled up as a linear decay with parameters based off the amount of training loops, such that the desired 80%/20% exploration/exploitation split;
- attempts were made to implement a double DQN (Deep Q-Learning Network), where the main model selects the best action, and the target model evaluates that action's Q-value, but the end result uses only a standard, single network where the same model that selects an actiona, evaluates it as well;

4 Conclusions

Unfortunately, the results of the agent have been quite lackluster, due to a combination of lack of time and misguided attempts to improve that ended up hurting the performance and training, with the agent still barely being able to score more than a few pipes in a run. While this project may lay the foundations for what could be a successful instance of reinforcement learning using a neural network accompanied by a Q-learning algorithm, at the time being, it is most certainly not.