Capstone Project: Train an AI Agent to Play Flappy Bird

Houston Community College

Mary Balemba

ITAI 1378: Computer Vision

Professor Anna Devarakonda

April 1st, 2025

1. Introduction

Flappy Bird emerged as a prominent mobile game characterized by its deceptively simple premise: guiding a bird through gaps in a series of green pipes. The core mechanic involves tapping the screen to provide a slight upward impulse to the bird, while ceasing input results in the bird's descent due to gravity. This seemingly straightforward gameplay belies a significant challenge, demanding precise timing and rapid reflexes from players. The game's unexpected surge in popularity underscores the engaging nature of this difficulty, making it a compelling environment for the application of artificial intelligence. This report explores key RL concepts relevant to Flappy Bird, including Deep Q-Networks (DQNs), transfer learning with Convolutional Neural Networks (CNNs), game mechanics, and tools like PyGame and PyGame Learning Environment (PLE). It also discusses CNN architectures (e.g., MobileNetV2, ResNet), state and action spaces, reward function design, training challenges, and evaluation metrics. Flappy Bird's difficulty necessitates sophisticated AI strategies despite its limited action space. Prior applications of both Q-Learning and DQNs demonstrate the relevance of classical and deep RL approaches in solving this problem.

2. Detailed Analysis of the Flappy Bird Game Environment

The visual presentation of Flappy Bird adopts a 2D, pixel art style, reminiscent of classic arcade games. Key visual elements within the game world include the bird itself, a series of green pipes featuring equally sized gaps at varying heights, a backdrop of a blue sky, and the ground at the bottom of the screen. The bird exhibits a constant horizontal motion from left to right, with its vertical movement being the primary aspect controlled by the player or the AI agent. The simplicity of these graphical elements suggests that an AI agent can focus on the fundamental game mechanics without requiring complex visual processing.

The game's physics engine governs the bird's movement. A constant downward acceleration, closely approximating Earth's gravity, perpetually pulls the bird towards the ground. The player's or agent's input, typically a tap or a similar action, provides an upward impulse or sets a fixed upward velocity, counteracting the force of gravity. Notably, the upward impulse often exhibits an unrealistic characteristic, frequently resetting the bird's vertical velocity to a predetermined value rather than applying a consistent force. The game concludes immediately upon the bird colliding with either the green pipes or the ground. Despite the simplified nature of the physics, the need for precise control over the bird's trajectory to successfully navigate the narrow gaps between the pipes presents a significant challenge. The deterministic nature of the game's physics, where each action invariably leads to a predictable outcome, makes it a suitable domain for the application of learning-based AI techniques.

The scoring system in Flappy Bird is straightforward: the player or agent earns one point for each successful passage through a pair of pipes. A point is typically awarded once the bird has completely cleared the set of pipes, often determined by the bird's horizontal position relative to the pipes. Upon the game's conclusion, human players are also recognized with medals (bronze, silver, gold, and platinum) based on their final score, serving as a benchmark for performance. This clear and quantifiable scoring mechanism is invaluable for guiding the reinforcement learning process, as the reward signal for the AI agent can be directly linked to the achieved score.

3. Environment Setup for AI Interaction

To enable an AI agent to play Flappy Bird, a suitable environment must be created using appropriate libraries. PyGame, a popular Python library for 2D game development, offers essential functionalities like game windows, graphics, and user input management. Its simplicity, cross-platform compatibility, and integration with AI libraries make it ideal for building a custom Flappy Bird environment. Additionally, extensive PyGame resources ease development.

OpenAI Gym (transitioning to Gymnasium) provides a standardized API for reinforcement learning (RL), but Flappy Bird is not included, requiring a custom implementation or wrapper. The PyGame Learning Environment (PLE) facilitates RL-agent interaction with PyGame-based games. Using Gym or PLE allows defining state space, action space, and reward systems, simplifying RL integration.

State representation helps the AI make decisions. One approach uses raw pixel data, offering full visual information but demanding high computation. A more efficient method is feature engineering, selecting key parameters like the bird's vertical position, velocity, and distances to pipes. Alternatively, discretizing states categorizes distances into finite groups. A history of frames provides temporal insights. A feature-engineered state is often more practical than raw pixel data for initial implementations.

The action space in Flappy Bird is discrete and simple: the agent either flaps or does nothing. More complex controls could be explored, but a binary action space is intuitive and makes learning the optimal flap timing more manageable.

The reward system guides the AI's learning. A typical reward function includes small rewards for survival, larger rewards for passing pipes, and penalties for collisions. Sparse rewards can slow learning, so a well-balanced system incentivizes survival, scoring, and avoiding failure.

Preprocessing raw game frames reduces input complexity and improves efficiency. Common techniques include resizing, grayscale conversion, and pixel normalization using OpenCV or Pillow. These steps streamline training, especially when employing neural networks.

4. Leveraging Transfer Learning with Pre-trained Models

Transfer learning is a powerful machine learning technique that involves utilizing knowledge gained from training a model on one task and applying it as a starting point for a model on a different but related task. Instead of training a model from scratch, transfer learning leverages pre-trained models that have already learned general features from vast datasets, such as ImageNet. In the context of developing an AI agent for Flappy Bird, transfer learning offers several potential advantages, including reduced training time, improved performance, and lower computational costs. By starting with a model that has already learned to recognize basic visual patterns, the AI agent can potentially learn the specific nuances of the Flappy Bird environment more quickly and effectively.

Two prominent pre-trained convolutional neural network architectures that could be considered for this task are MobileNetV2 and ResNet. MobileNetV2 is a lightweight and

efficient CNN designed for mobile and embedded vision applications. Its architecture incorporates features like depthwise separable convolutions, inverted residuals, and linear bottlenecks, which contribute to its high efficiency. The computational efficiency of MobileNetV2 makes it particularly appealing for Flappy Bird, as it can lead to faster training times and the potential for real-time inference. Its successful deployment in various real-world applications further underscores its generalizability.

ResNet, on the other hand, is a deep residual network architecture renowned for its ability to effectively train very deep networks using residual connections (skip connections), which help to address the vanishing gradient problem. Various versions of ResNet exist, differing in their depth and computational complexity, such as ResNet18, ResNet34, and the deeper ResNet50, ResNet101, and ResNet152.69 ResNet's strength lies in its ability to achieve high accuracy in image classification tasks 68, making it a potential choice if maximizing the agent's performance is the primary goal. It has been widely adopted in diverse computer vision applications that demand robust feature extraction.

Model Name	Architecture Type	Key Features	Strengths	Potential Suitability for Flappy Bird
MobileNetV2	CNN	Depthwise Separable Convolutions, Inverted Residuals	Efficiency	High
ResNet (e.g., ResNet50)	CNN	Residual Connections	High Accuracy	Medium

To utilize a pre-trained model for feature extraction in the context of Flappy Bird, the top classification layers of the CNN, which are specific to the original training task (e.g., classifying 1000 categories in ImageNet), need to be removed.55 The remaining convolutional base of the network can then serve as a feature extractor, transforming the input game frame into a lower-dimensional representation.55 Custom layers, such as dense layers, can be added on top of this feature extractor to form the Q-network that will be used in the reinforcement learning implementation.

Adapting a pre-trained model to the Flappy Bird environment presents certain challenges, primarily due to the domain mismatch between the real-world images on which these models are typically trained and the synthetic game frames. Several strategies can be employed to overcome this. Fine-tuning involves unfreezing some of the layers of the pre-trained model and retraining them on the Flappy Bird game frames (along with the newly added custom layers) using a small learning rate. This allows the model to adapt the

learned features to the specific characteristics of the game. Initially freezing the convolutional base and only training the added layers can also be beneficial to prevent drastic changes to the pre-trained weights during the early stages of training. Data augmentation techniques, such as applying small rotations or translations to the game frames during training, can further enhance the model's robustness and generalization ability. It is also important to consider the computational resources required for fine-tuning, as larger pre-trained models can be demanding. Opting for a more lightweight model like MobileNetV2 can help to mitigate these computational concerns.

5. Reinforcement Learning Implementation

Reinforcement learning is a computational approach where an agent learns to make decisions by interacting with an environment. The core elements of an RL system include an agent, an environment, states representing the current situation, actions the agent can take, rewards providing feedback on the agent's actions, and a policy that guides the agent's action selection. The goal of the agent is to learn an optimal policy that maximizes the cumulative reward received over time.

For the Flappy Bird game, the Deep Q-Network (DQN) algorithm is a suitable choice. DQN utilizes a deep neural network, known as the Q-network, to approximate the optimal Q-value function, which estimates the expected future reward for taking a specific action in each state. In the context of Flappy Bird, the state of the game (e.g., a preprocessed game frame or a feature vector) is fed into the Q-network, which then outputs the estimated Q-values for each possible action (flap or do nothing). The agent typically selects the action with the highest predicted Q-value or employs an exploration strategy to discover potentially better actions. DQN is well-suited for this problem as it can handle continuous state spaces and learn complex control policies through the function approximation capabilities of deep neural networks.

Implementing the DQN algorithm requires several key components. The Q-network itself is a deep neural network whose architecture could potentially be based on a modified pre-trained model, with the top layers removed and additional dense layers added. The input shape of the network will depend on the chosen state representation, and the output layer will have a size equal to the number of possible actions. A replay memory, or experience replay buffer, is used to store the agent's experiences (state, action, reward, next state) at each time step. During training, mini-batches of these experiences are randomly sampled from the replay memory to update the Q-network's weights. This random sampling helps to break the correlation between consecutive experiences and stabilizes the training process. Finally, a target network, which is a separate and delayed copy of the main Q-network, is used to calculate the target Q-values during training. The weights of the target network are

updated periodically by copying the weights from the main Q-network, which further stabilizes the learning process by preventing rapid fluctuations in the target values.

A crucial aspect of reinforcement learning is managing the exploration-exploitation trade-off. The agent needs to explore the action space to discover new and potentially better strategies while also exploiting the actions it has already learned to be effective to maximize its immediate reward. A common strategy for balancing this trade-off is the epsilon-greedy approach.9 With a probability of epsilon (ε), the agent selects a random action (exploration), and with a probability of 1-ε, it chooses the action with the highest estimated Q-value (exploitation).9 To encourage more exploration at the beginning of training and gradually shift towards exploitation as the agent learns, an annealing schedule is often used to decrease the value of epsilon over time.

Experience replay is a vital technique in DQN implementation. At each time step, the agent's experience (current state, action, reward, next state) is stored in a replay memory buffer. During training, a mini-batch of these experiences is randomly sampled and used to update the Q-network's weights. This process of experience replay is important for several reasons. It breaks the correlation between consecutive experiences, which can lead to unstable training. It also allows for greater data efficiency, as each experience can be used multiple times during training. Furthermore, by averaging over a batch of experiences, the training process becomes more stable.

6. Model Training: A Step-by-Step Guide

Training a Flappy Bird AI agent using Deep Q-Networks (DQN) follows several steps. First, the environment, Q-network, target network (optionally pre-trained), and replay memory are initialized. An exploration rate (epsilon) is set. Training proceeds in episodes, each representing a game. At each time step, the agent observes the state, selects an action via the epsilon-greedy policy, executes it, and receives a reward. The experience (state, action, reward, next state) is stored in replay memory. A mini-batch is randomly sampled to compute target Q-values, which are used to update the Q-network via gradient descent. Periodically, the target network’s weights are updated. Epsilon decreases over time, balancing exploration and exploitation. This cycle repeats until sufficient episodes train an effective policy, after which the trained model is saved.

The training loop uses an environment library (e.g., PyGame via PLE) to reset the game, obtain states, execute actions, receive rewards, and check termination. A discount factor (gamma) weighs future rewards, influencing decision-making.

Hyperparameters crucially impact performance. These include the learning rate, discount factor, exploration rate, replay memory size, batch size, and target network update

frequency. Tuning methods include manual experimentation, grid search, random search, and Bayesian optimization.

Common training challenges include catastrophic forgetting, mitigated by experience replay, and reward sparsity, addressed through reward shaping and curriculum learning. DQN instability can be countered with a target network and gradient clipping.

Model evaluation tracks progress via average scores, survival time, epsilon decay, and Q-network loss. Visualizing gameplay provides qualitative insights into strategy improvements. These iterative processes refine the agent's ability to make optimal game decisions.

7. Testing and Evaluation

Once the AI agent has been trained, a comprehensive testing strategy is necessary to evaluate its performance. During testing, the exploration rate (epsilon) should be set to a very low value or zero to ensure that the agent primarily relies on its learned policy to make decisions. The trained agent should be run in the Flappy Bird environment for a significant number of episodes, and its performance in each episode should be recorded using relevant evaluation metrics. To assess the agent's ability to generalize, it can also be tested with slight variations in the game environment, such as different pipe spacing or speeds.

Several metrics can be used to evaluate the trained agent's performance. The average score achieved over multiple test episodes provides a general measure of the agent's ability to navigate the pipes. The average survival time indicates how long the agent can keep the bird alive. The highest score achieved in a single episode can highlight the agent's potential. The crash rate, or the percentage of episodes ending in a crash within a certain number of steps, reflects the agent's stability. Comparing these metrics to human players' scores or the performance of other AI agents (if available) can provide valuable context. The Flappy Bird game's medal system (bronze, silver, gold, platinum) can serve as a rough benchmark for comparison against human performance.

Interpreting the evaluation results involves analyzing the distribution of scores and survival times to understand the consistency of the agent's performance. Comparing the average score and survival time to human-level performance helps to gauge the agent's skill. If results from other AI approaches for Flappy Bird are available, a comparison should be made. Establishing a baseline performance, such as that of a random agent or a simple

rule-based agent, can also help to quantify the improvement achieved by the reinforcement learning agent.

Visualizing the agent's performance and decision-making process can provide valuable qualitative insights. Recording gameplay videos allows for a visual assessment of the agent's behavior and the identification of any patterns in its actions.85 If the state space is discrete and manageable, visualizing the learned Q-values for different states can be informative. Plotting the agent's trajectory and the timing of its flap actions relative to the pipe positions can also offer insights into its strategy. For agents using convolutional neural networks, techniques like saliency maps can help understand which parts of the game frame the agent is focusing on.

Based on the evaluation results, potential improvements and directions for future work can be identified. This could include exploring different reinforcement learning algorithms, experimenting with alternative state representations and network architectures, investigating more advanced exploration strategies, or incorporating techniques like curriculum learning or reward shaping to enhance training efficiency and performance. Evaluating the agent's robustness to changes in the game environment and exploring the possibility of multi-agent learning are also potential avenues for future research.

8. Conclusion

Developing an intelligent agent to play Flappy Bird involves a multifaceted approach combining reinforcement learning and transfer learning techniques. The process begins with a thorough analysis of the game environment, followed by setting up the environment for AI interaction by defining the state space, action space, and reward system. Leveraging pre-trained models through transfer learning can provide a strong starting point for feature extraction, which is then integrated into a Deep Q-Network (DQN) algorithm. The model is trained through iterative interaction with the game, storing experiences, and updating its decision-making policy. Finally, a comprehensive testing and evaluation phase assesses the agent's performance and guides future improvements. This endeavor highlights the potential of AI techniques to master challenging game environments through learning and adaptation.

References:

Sandler, M., Howard, A., Zhu, M., et al. (2018) Mobilenetv2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, 18-23 June 2018, 4510-4520. https://doi.org/10.1109/CVPR.2018.00474

Shahi TB, Sitaula C, Neupane A, Guo W (2022) Fruit classification using attention-based MobileNetV2 for industrial applications. PLoS ONE 17(2): e0264586. https://doi.org/10.1371/journal.pone.0264586

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen: MobileNetV2: Inverted Residuals and Linear Bottlenecks. CVPR 2018: 4510-4520

Zoph B., Vasudevan V., Shlens J., et al. Learning transferable architectures for scalableimage recognition. In Conference on Computer Vision and Pattern Recognition, 8697-8710,2018.

Chollet F.. Xception: Deep learning with depthwise separable convolutions. In Conferenceon Computer Vision and Pattern Recognition, 1800-1807, 2017.

Ayad, O. (2014). Learning under concept drift with support vector machines. In Wermter, S., Weber, C., Duch, W., Honkela, T., Koprinkova-Hristova, P. D., Magg, S., Palm, G., and Villa, A. E. P., editors, Artificial Neural Networks and Machine Learning - ICANN 2014 - 24th International Conference on Artificial Neural Networks, Hamburg, Germany, September 15-19, 2014. Proceedings, volume 8681 of Lecture Notes in Computer Science, pages 587-594. Springer. DOI: 10.1007/978-3-319-11179-7_74.

Belouadah, E., Popescu, A., and Kanellos, I. (2020). Initial classifier weights replay for memoryless class incremental learning. In 31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020. BMVA Press. DOI: 10.48550/arXiv.2008.13710.