**Group 2:**

Lionel DeLuna • Silas Caporal • Morgan Germany

Masson Lopez • Mohammad Sufyaan

Assignment: Capstone Project

Train an AI Agent to Play Flappy Bird

ITAI 1378

Professor: Patricia McManus

Capstone Project (Conceptual Path):

Train an AI Agent to Play Flappy Bird

By undertaking a capstone project to train an AI agent to play Flappy Bird, we can gain a comprehensive understanding of various machine learning and reinforcement learning concepts. We learned to design and implement neural network architectures, train models using techniques, and apply computer vision techniques to process game frames. We also delve into reinforcement learning concepts such as Q-learning, experience replay, and exploration-exploitation trade-offs. The project included problem-solving, critical thinking, and project management skills as we break down complex problems, design and implement algorithms, and collaborate with team members. (Sutton & Barto, 2018)

Environment Setup

The Flappy Bird game environment is a 2D side-scrolling game where the player controls a bird by flapping its wings to avoid colliding with pipes. The game's primary visual components include the bird, pipes, and the background. The bird's physics is relatively simple, involving gravity and a vertical velocity that can be increased by flapping. The scoring system is based on the number of pipes the bird successfully passes through. To set up this environment for AI interaction, we'll leverage the PyGame Learning Environment (PLE). PLE provides a convenient framework for creating reinforcement learning environments based on classic games like Flappy Bird. It handles the game's mechanics, rendering, and interaction with the AI agent. The state representation for the AI agent will be a processed frame from the game. This frame will typically be a grayscale image, resized to a smaller dimension to reduce computational complexity. The image will capture the bird's position, velocity, and the positions and gaps of the upcoming pipes. The action space for the agent is discrete and consists of two actions: Flap: The bird flaps its wings, increasing its vertical velocity. No-op: The bird does nothing, allowing gravity to take effect. The reward system is designed to incentivize the agent to maximize its score. A positive reward is awarded when the bird successfully passes through a pipe. A negative reward is a penalty when colliding with a pipe or the ground. A small negative reward is a penalty for spending too much time in the air without making progress. The frame preprocessing allow the game frames suitable for input to the neural network, we'll apply the following preprocessing steps. A grayscale conversion converts the RGB image to grayscale to reduce the number of input channels. Resizing the images to a smaller dimension, such as 84x84 pixels, to reduce computational cost. Normalize the pixel values to a specific range, such as 0 to 1.

By carefully designing the state representation, action space, and reward system, we can create a challenging and rewarding environment for training an AI agent to play Flappy Bird effectively. (Goodfellow & Courville, 2016)

Pre-trained Model Usage

Transfer learning is a technique where a model trained on one task is repurposed for a different but related task. In the context of training an AI agent for Flappy Bird, we can leverage pre-trained models to accelerate the learning process and improve performance.

A suitable pre-trained model for this task is MobileNetV2. This model is designed for efficient mobile and embedded vision applications, making it ideal for real-time gameplay. MobileNetV2 excels at feature extraction, which is crucial for recognizing patterns in the game frames (e.g., the bird's position, the pipes' positions and gaps).

To adapt the pre-trained MobileNetV2 model for Flappy Bird, we'll follow these steps:

1. Feature Extraction: We'll freeze the convolutional base layers of the model, preventing them from being updated during training. These layers have learned powerful feature representations from a massive dataset of images.
2. Custom Top Layers: We'll add custom layers on top of the frozen base layers to suit the specific requirements of Flappy Bird. This typically involves adding fully connected layers to process the extracted features and output the action probabilities.
3. Fine-tuning: We can optionally fine-tune the top layers of the model using a small learning rate to further improve performance. However, this step requires careful consideration to avoid overfitting.

Some of the potential challenges in adapting a pre-trained model for Flappy Bird include:

* Domain Gap: The pre-trained model might not be perfectly suited for the specific visual characteristics of Flappy Bird. This can be mitigated by careful data augmentation and hyperparameter tuning.
* Computational Cost: While MobileNetV2 is efficient, training a large neural network can still be computationally expensive. Techniques like model quantization and knowledge distillation can help reduce the model's size and computational requirements.

By leveraging a pre-trained model like MobileNetV2 and carefully adapting it to the Flappy Bird environment, we can significantly improve the performance and training efficiency of our AI agent. Google AI. (2024)

Reinforcement Learning Implementation

Reinforcement learning (RL) is a machine learning paradigm where an agent learns to make decisions by interacting with an environment. Key components of RL include states, actions, rewards, and policies. States represent the current situation, actions are the choices the agent can make, rewards are the feedback for actions, and policies are strategies for selecting actions.

To train an AI agent to play Flappy Bird, we'll use the Deep Q-Network (DQN) algorithm. DQN combines deep learning with Q-learning, using a neural network to approximate the Q-value function, which estimates the expected future reward for actions in specific states. A replay memory stores past experiences, and a target network stabilizes training.

To balance exploration and exploitation, we'll use an epsilon-greedy policy. This policy selects random actions with probability epsilon and the best-known action with probability 1-epsilon. Over time, epsilon decreases, shifting the agent's behavior from exploration to exploitation.

The training process involves initializing the DQN, replay memory, and target network. The agent interacts with the environment, taking actions and storing experiences. A batch of experiences is sampled from the replay memory to train the DQN, minimizing the temporal difference error. The target network is periodically updated to stabilize training. By tuning hyperparameters and following these steps, we can train an effective Flappy Bird agent.

Model Training

The DQN training process involves iteratively updating the network's weights to minimize the difference between estimated and actual future rewards. This process starts by initializing the DQN, replay memory, and target network. The agent then interacts with the environment, taking actions and observing the resulting states and rewards. These experiences are stored in the replay memory. OpenAI. (2024)

Next, a batch of experiences is sampled from the replay memory and used to calculate the temporal difference (TD) error. This error measures the difference between the immediate reward and the discounted future reward. The DQN's weights are then updated using gradient descent to minimize this error. To stabilize training, the weights of a target network, which is a copy of the main network, are periodically updated.

Hyperparameters like learning rate, discount factor, and exploration rate are tuned to optimize performance. The exploration-exploitation trade-off is managed using an epsilon-greedy policy, which balances exploring new actions with exploiting known good actions.

To evaluate the agent's performance, metrics like average reward, survival time, and success rate are tracked. Learning curves can be plotted to visualize the training progress. By addressing challenges like catastrophic forgetting and reward sparsity and carefully monitoring the training process, we can train an AI agent to effectively play Flappy Bird. Google AI. (2024)

Testing and Evaluation

A comprehensive testing strategy for the trained Flappy Bird agent involves evaluating its performance across various scenarios and metrics. Key metrics to consider include the average score. The average score achieved by the agent over multiple episodes. Another metric includes the survival time, which is the average time the agent survives in each episode before colliding with a pipe or the ground. The success rate, which is the percentage of episodes in which the agent successfully passes a certain number of pipes. To interpret the results, we can compare the agent's performance to human players or other AI agents. A good benchmark is to compare the agent's performance to a random agent that selects actions randomly.

To visualize the agent's performance, we can use tools like Matplotlib to plot learning curves, which show the evolution of the average reward over time. Additionally, we can visualize the agent's decision-making process by rendering the game and highlighting the selected actions.

Based on the evaluation results, several areas can be explored to improve the AI agent's performance. Hyperparameter tuning, which involves adjusting parameters like learning rate, discount factor, and exploration rate, can significantly impact the training process. Experimenting with different neural network architectures, such as convolutional neural networks with varying numbers of layers or activation functions, can enhance the agent's ability to learn complex patterns from the game environment.

Additionally, refining the reward function to better incentivize desired behaviors can lead to more effective learning. Employing diverse exploration strategies, like curiosity-driven exploration or prioritized experience replay, can accelerate the learning process. Finally, increasing the diversity of training data through data augmentation techniques, such as random rotations, flips, and color jittering, can improve the agent's generalization ability. By continuously monitoring the agent's performance and making informed adjustments, we can strive to improve its capabilities and achieve even better results. OpenAI. (2024)

**Bibliography**

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning

Google AI. (2024). Gemini. [Large language model]

OpenAI. (2024). ChatGPT. [Large language model]