**Capstone Project: Train an AI Agent to Play Flappy Bird**

**Alisha D. Rush. Muhammad Abdullah Farooq, Saima Sano, Zaid Tahir**

**Patricia McManus**

**ITAI 1378**

**The environment of Flappy Bird**

**Flappy Bird Game Environment:**

Flappy Bird is a popular mobile game developed by Dong Nguyen in 2013. It received a massive influx of players and achieved great success quickly. To play the game, players tap the screen to navigate the bird and make it jump at the right moment to get through a set of pipes. This game showcases a simple 2D environment with an infinite scrolling background of clouds and sky. The main character of this game is a small sprite bird of 32x32 pixels with a flapping animation. The bird needs to navigate through green pipes of various heights, positioned in pairs with gaps. The ground perpetually scrolls at the bottom of the screen.

The bird is continually influenced by downward gravity. Tapping the screen or clicking the mouse makes the bird "flap," granting it upward velocity. The bird's movement follows a basic physics engine:

* Vertical speed is updated every frame: vertical\_speed += gravity
* Bird's Y position is updated accordingly: bird\_y += vertical\_speed
* Flapping assigns a positive value to the vertical speed, causing an upward movement (*Can Someone Explain Flappy Bird’s Physics to Me?*, n.d.).

Players accumulate points by navigating the bird through the gaps between pipes. The score is usually shown at the top of the screen. The game concludes if the bird hits a pipe, the ground, or the top edge of the screen.

A screenshot of a video game

Description automatically generated

**Libraries or Tools:**

To successfully program an AI agent to play Flappy Bird, I would use PyGame and OpenAI Gym. PyGame provides an accessible framework for 2D game development in Python. It includes user-friendly functions for managing graphics, sound, and input. Whereas OpenAI Gym offers a standardized environment for reinforcement learning. It provides various AI algorithms to train and evaluate the AI agent. It ensures a consistent API for interacting with the environment.

**AI interaction:**

For AI interaction setup, we have choosen a pixel-based approach, which directly captures the game screen, including the bird's vertical position and the horizontal distance to the gap in the next pipe, the positions of the pipes, the bird’s vertical velocity, and the background. The action space can be discrete, consisting of a limited set of actions like "flap (1)" or "do nothing (0)." The reward system includes positive rewards for successfully passing through a pipe gap (+1 point) and negative rewards for colliding with a pipe or the ground (-10 point) to discourage crashing. As these rewards are infrequent, make the learning process more challenging.

**Preprocessing Game Frame:**

To decrease computational cost, we must preprocess game frames for AI input. By preprocessing the game frames, we can create a suitable input format for the AI agent to learn and make accurate decisions. First, we must resize the game resolution to either 84x84 or 64x64 pixels. Then, we must convert the game frame to greyscale to simplify the input from 3 to 1. We would also stack 4 frames so that AI can learn better. Normalize the pixel values from 0 to 1 to improve training.

**Pre-trained Model Usage**

**Transfer learning** is a machine learning technique where a model trained on one task is re-used as a starting point for a related task (*What Is Transfer Learning? - Transfer Learning in Machine Learning Explained - AWS*, n.d.). To train an AI agent to play Flappy Bird, we can leverage pre-trained models from computer vision tasks to extract relevant features from the game frames.

Using pre-trained models offers several advantages for implementing Flappy Bird. These models have already acquired general visual features, significantly reducing the amount of training data and time needed. Additionally, leveraging the knowledge from large datasets can enhance performance, particularly when training data is limited, as pre-trained models enable the AI agent to recognize game environment patterns more swiftly. Moreover, transfer learning can mitigate overfitting, with the pre-trained weights functioning as a form of regularization, resulting in more robust and generalizable models (Vaishnavi et al., n.d.).

**Choosing the right pre-trained model:**

Although ResNet and MobileNetV2 are powerful models, they may be excessive for a simple task like playing Flappy Bird. This is because both models are deep neural networks with numerous layers, which might not be necessary for such a basic game. A simpler model like VGG16 can be trained faster and with fewer computational resources. Additionally, there is a higher risk of overfitting with complex models and limited training data. A simpler model is less prone to overfitting. In terms of computational efficiency, a simpler model requires fewer computations during training and inference, which is especially important when running the model on devices with limited processing power. For Flappy Bird, the primary task is to extract relevant visual features, such as the position of the bird and the pipes. A simpler convolutional neural network (CNN) can effectively extract these features without the need for the complex architectures of ResNet or MobileNetV2. VGG16 can provide a better balance of performance, efficiency, and ease of training.

**Modifying the Pre-trained Model:**

To modify a pre-trained model for feature extraction in Flappy Bird, we will typically follow these steps:

**1. Freeze the Convolutional Base:**

* The convolutional base of the pre-trained model, consisting of convolutional and pooling layers, is responsible for extracting features from images.
* To prevent these layers from being updated during training, we will freeze them. This ensures that the model retains its learned knowledge about image features (*Transfer Learning and Fine-tuning*, n.d.).

**2. Replace the Top Layers:**

* The top layers of the pre-trained model are usually the fully connected layers specific to the original task. Therefore, we would remove these layers and replace it with a fully connected layer of two output neurons, one for the "flap" action and one for the "do nothing" action.

**3. Add a Custom Output Layer:**

* For the final layer of the modified model, we would apply a softmax activation function, so that the output probabilities sum to 1.

**Difference between Input and Output Layer**

The output layer and the top layer are not the same, even though they both have two neurons. Here's a breakdown:

**Top Layer:**

* This layer is the last layer of the pre-trained convolutional base.
* It produces a high-dimensional feature representation of the input image.
* The number of neurons in this layer depends on the architecture of the pre-trained model.

**Output Layer:**

* This layer is added on top of the pre-trained base to produce the final output.
* It takes the high-dimensional feature representation from the top layer as input.
* It has two neurons, each corresponding to one of the two possible actions: "flap" or "do nothing".
* The output of this layer is a probability distribution over the two actions.

The key difference between these layers lies in their purpose and the type of output they produce. The top layer extracts features, while the output layer decides based on those features.

**Challenges:**

|  |  |  |
| --- | --- | --- |
| Domain Gap | One challenge in using pre-trained models for Flappy Bird is the **domain gap**, where the image distribution in the pre-trained model's dataset (such as ImageNet) differs significantly from Flappy Bird's game frames. | To solve this issue, we are going to use data augmentation techniques like random cropping, flipping, rotation, and color jittering to diversify the training data. Additionally, exploring domain adaptation techniques can help bridge the gap between the source and target domains, ensuring the model adapts better to the specific characteristics of Flappy Bird frames. These approaches can improve the model's performance by making it more robust and versatile in recognizing game-specific features(Pilcer et al., n.d.). |
| Noise | Random Pipe Generation: The game generates pipes at random intervals and heights, adding unpredictability. | **Robust Feature Extraction:** Train the AI agent to learn features invariant to small variations and apply noise reduction techniques like filtering or smoothing. |
|  | Physics Engine Variations: The physics engine might introduce slight variations in the bird's movement or the pipe's speed. | **Reinforcement Learning Techniques:** Balance exploration and exploitation, use policy gradient methods, and employ experience replay to improve learning stability and efficiency. |
|  | Sensor Noise: The AI agent's perception of the game state might be imperfect due to factors like screen resolution, frame rate, or rendering artifacts. | **Adaptive Learning:** Allow continuous learning and implement meta-learning for the agent to quickly adapt to new situations. |
|  |  | **Robust Control:** Use model-predictive control to predict future states and plan optimal actions and adaptive control to adjust parameters based on the game state and noise level. |
| Computational Load | The computational constraints of running a complex model in real-time on a device. Model compression techniques such as pruning, quantization, and knowledge distillation can be employed to reduce the model's size and computational cost. | To resolve this issue, employing VGG 16 can reduce the computational load. Also, utilizing specialized hardware like GPUs or TPUs can further accelerate inference. |

**Reinforcement Learning Implementation**

A screenshot of a video game

Description automatically generated

Reinforcement learning provides a powerful approach to training an AI agent to play complex games like Flappy Bird. The Deep Q-Network (DQN) algorithm offers a sophisticated method for enabling the agent to learn through interaction with the game environment, transforming the challenge of navigating through obstacles into a structured learning problem.

## **Core Components of the DQN Approach**

The implementation relies on several critical components that work together to facilitate effective learning. The neural network serves as the core decision-making mechanism, predicting Q-values (expected rewards) for potential actions based on the current game state. This architecture enables the agent to make intelligent choices about when and how to navigate through the game environment.

To address the challenge of correlated data in sequential learning, the implementation incorporates a replay memory system. This technique stores past experiences, allowing the model to train on batches of uncorrelated data. By random sampling from this memory, the agent can learn more effectively and prevent overfitting to recent experiences.

Stability is crucial in reinforcement learning, which is why the target network mechanism periodically updates a separate Q-network. This approach helps stabilize the learning process and prevent rapid, potentially destructive oscillations in value predictions.

## **Algorithm Implementation**

The learning process follows a systematic approach that mimics how an intelligent agent might learn to play a challenging game. It begins with observing the current game state, capturing critical parameters like bird position, pipe gaps, and velocity. The neural network then generates Q-values for available actions, using these predictions to select an action through an epsilon-greedy policy that balances exploration and exploitation.

Each action is carefully executed, with the agent observing the resulting state and reward. These experiences are meticulously recorded in replay memory, creating a rich dataset for future learning. Periodically, the Q-network is trained by minimizing the difference between predicted and target Q-values, gradually refining its decision-making capabilities.

## **Addressing Sparse Rewards**

Flappy Bird presents a significant challenge due to sparse rewards. The learning process must be carefully designed to handle infrequent and binary feedback. This requires sophisticated techniques like reward shaping and experience replay to guide the agent's learning process, transforming minimal feedback into meaningful learning signals.

## **Training Methodology**

The training loop follows a structured yet adaptive approach. Each episode begins by resetting the game environment, with the agent navigating through multiple time steps. It observes states, selects actions, and stores experiences in replay memory. Periodic training of the Q-network ensures continuous improvement, allowing the agent to progressively refine its strategy.

## **Hyperparameter Optimization**

Careful selection of hyperparameters proves to be a delicate art. The learning rate controls the gradient descent step size, while the batch size determines the number of experiences sampled per training iteration. The discount factor balances immediate and long-term rewards, requiring meticulous tuning to achieve optimal performance.

## **Testing and Evaluation**

The training process involves comprehensive performance tracking. Loss and accuracy metrics are continuously monitored, with performance compared against baseline strategies. Visualization plays a crucial role, with replay videos and probability plots offering deep insights into the agent's decision-making processes.

## **Potential Improvements and Future Work**

While the current implementation provides a solid foundation, the journey of machine learning is never complete. Future work might explore advanced algorithms like Proximal Policy Optimization, conduct extensive hyperparameter tuning, and experiment with alternative neural network architectures.

## **Conclusion**

This reinforcement learning project demonstrates the remarkable potential of adaptive AI systems to learn complex tasks through iterative interaction. By carefully designing the learning process, addressing challenges like sparse rewards, and implementing a robust DQN algorithm, we create an agent capable of making intelligent decisions in challenging game environments.

The implementation offers more than just a solution to playing Flappy Bird. It provides profound insights into the intricate process of teaching an AI to navigate complex, dynamic scenarios through systematic learning and adaptation. It stands as a testament to the growing capabilities of machine learning techniques in solving real-world challenges.

**Learning Log**

***Challenges Faced & Solution Implemented***

1. **Challenge: Understanding Flappy Bird’s Physics**
   1. **Issue:** Initially, it was difficult to translate the game mechanics (e.g., gravity, bird velocity) into a clear mathematical framework for AI interaction.
   2. **Solution:** Researched game development forums and Stack Exchange for explanations of Flappy Bird’s physics. This clarified how vertical speed and bird position are updated frame by frame.
2. **Challenge: Selecting a Suitable Pre-trained Model**
   1. **Issue:** Deciding between powerful models like ResNet and simpler architectures like VGG16 was challenging due to concerns about overfitting and computational efficiency.
   2. **Solution:** Chose VGG16 after analyzing its balance of simplicity, efficiency, and effectiveness for this task. This decision aligned with the project’s requirements for feature extraction without unnecessary complexity.
3. **Challenge: Sparse Rewards in Reinforcement Learning**
   1. **Issue:** Sparse rewards made it harder for the agent to learn the optimal strategy.
   2. **Solution:** Implemented reward shaping by assigning intermediate rewards for staying alive longer and using experience replay to diversify the training data.
4. **Challenge: Balancing Exploration and Exploitation**
   1. **Issue:** The agent needed to explore actions early in training while exploiting learned strategies later.
   2. **Solution:** Used an epsilon-greedy policy with a decay schedule to gradually reduce exploration as the agent’s performance improved.
5. **Challenge: Preprocessing Game Frames**
   1. **Issue:** Designing a preprocessing pipeline to simplify input without losing important information was tricky.
   2. **Solution:** Resized frames to 84x84 pixels, converted them to grayscale, normalized pixel values, and stacked four frames to capture motion.

**Work Cited**

*Can someone explain Flappy Bird’s physics to me?* (n.d.). Game Development Stack

Exchange. [https://gamedev.stackexchange.com/questions/70268/can-someone-explain-](https://gamedev.stackexchange.com/questions/70268/can-someone-explain-flappy-birds-physics-to-me)flappy-birds-physics-to-me

*What is Transfer Learning? - Transfer Learning in Machine Learning Explained - AWS*. (n.d.). Amazon Web Services, Inc. [https://aws.amazon.com/what-is/transfer-](https://aws.amazon.com/what-is/transfer-learning/#:~:text=Transfer%20learning%20(TL)%20is%20a,for%20a%20new%2C%20related%20task)learning/#:~:text=Transfer%20learning%20(TL)%20is%20a,for%20a%20new%2C%20related%20task.

Vaishnavi, P., Eykholt, K., & Rahmati, A. (n.d.). *A study of the effects of transfer learning on adversarial robustness*. OpenReview. <https://openreview.net/forum?id=T6RygOFZ6B>

*Transfer learning and fine-tuning*. (n.d.). TensorFlow. https://www.tensorflow.org/tutorials/images/transfer\_learning#:~:text=Freeze%20the%20convolutional%20base,-It%20is%20important&Freezing%20(by%20setting%20layer.,from%20being%20updated%20during%20training.

Pilcer, L.-S., Hoorelbeke, A., & D’andigne, A. (n.d.). Playing Flappy bird with deep reinforcement learning. *Research Gate*, 324066514. <https://doi.org/10.13140/RG.2.2.13159.96165>