Reinforcement Learning in Pac-Man

Allen Xu: xu.alle@northeastern.edu
Luc Ferrara: ferrara.l@northeastern.edu
Iba Baig: baig.i@northeastern.edu
Vishy Kamalapuram: kamalapuram.v@northeastern.edu

Northeastern University, Boston, 02115

December 4, 2024

Abstract

This project explores applying advanced reinforcement learning (RL) techniques in the classic Pac-Man game, focusing on training ghost agents using a Deep Q-Network (DQN) algorithm with a feedforward neural network (FNN), and training a Pac-Man agent using a double-DQN algorithm with a convolutional neural network (CNN). A pre-existing Gymnasium environment was utilized for training Pac-Man, and a new environment was created from scratch for training the ghosts. This project demonstrates multi-agent RL training as well as adversarial AI training. Results indicate the viability, but also the various challenges associated with DQN-based RL in complex environments.

Introduction

Pac-Man has been a popular medium for testing AI algorithms, offering a rich environment for adversarial and search-based challenges. In traditional implementations, RL has been used primarily to enhance Pac-Man's performance, such as in the Pacmancode repository that was used as a template. This project shifts focus by training ghost agents using reinforcement learning to dynamically improve their strategies against an already-intelligent Pac-Man agent.

1 Methods

1.1 Pac-Man Training

The Reinforcement Learning algorithm we used to train Pac-Man was a double-DQN architecture that uses a convolutional neural network for the policy network and target network. As epsilon decays after each Pac-Man movement, the agent gradually chooses less random actions, but rather actions dictated by the policy network. The grayscale images are first cropped from 250x160 to 180x160 in order to speed up computation time. This is done by removing unneeded parts of the UI such as the scoreboard. The CNN has 3 convolutional layers, and 1 dense layer with ReLU activation. From the 180x160 grayscale images, the CNN outputs a tensor of size 5, representing the probability for each action that Pac-Man can take. The model is trained at each iteration by first extracting the transition history from the replay memory. If the length of the memory surpasses the batch size, the optimization function proceeds by calculating Qopt and Vopt using the target network, computing the Huber Loss, optimizing the model, and performing gradient clipping. After the optimization function completes, the weights of the policy network and target network are updated.

Many improvements were made to the Pac-Man training algorithm to improve the ease of training. Firstly, Cuda was integrated with the algorithm to enable training on a GPU, which sped up training drastically. Additionally, helper functions for saving executions and loading executions were created, which allowed for pausing training and periodically saving policies, hyperparameters, replay memory and results, which could be resumed in a future run. Lastly, Matplotlib was utilized to plot the rewards achieved by the Pac-Man agent, which displayed whether or not a run was making progress or not.

1.2 Ghost Training

The ghost agents, on the other hand, were trained using a single-DQN architecture with a feedforward neural network. The ghosts spawn progressively into the environment, and their actions are dictated by the neural network policy or random exploration. Pac-Man's actions are generated by the policy we created during the Pac-Man training process. The input into the FNN is normalized coordinates of Pac-Man and the ghosts. Hidden layers include two fully connected layers with ReLU activations. The output is a tensor of size 16, representing the probabilities for each of the four actions for all four ghosts. The FNN minimizes cross-entropy loss by predicting actions that reduce Manhattan distance to Pac-Man, thus attempting to optimize the ghosts' search.

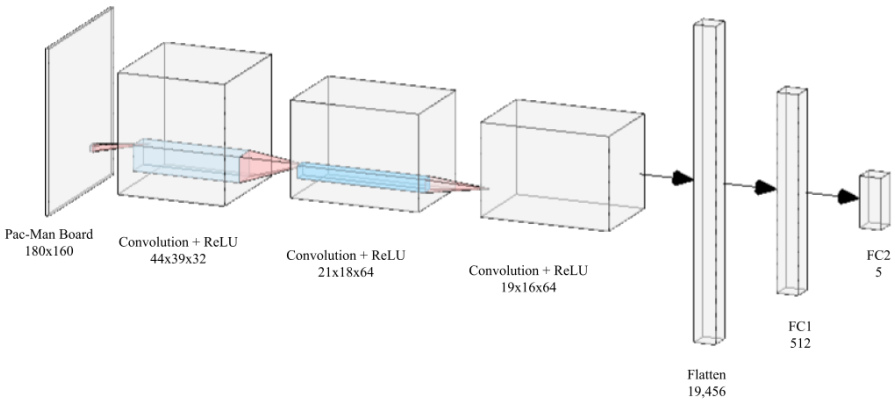
2 Data and Experiments

2.1 Pac-man

In order to train Pac-Man, we utilized the Arcade Learning Environment (ALE). This Pac-Man environment is a Gymnasium environment that is represented using 250x160 grayscale images. There is

a discrete action space of size 5, with the valid actions being (NOOP, UP, DOWN, LEFT, RIGHT). Other notable features of this environment include the tracking of Pac-Man's remaining lives, the total score achieved, and rewards for collecting pellets, power-pellets, and eating ghosts. The environment also keeps track of the frame number at each step.

The Pac-Man neural network architecture was designed to process the game's visual inputs. The input frames were first passed through three convolutional layers that extracted spatial features from the grayscale images. These features were flattened into a vector of size 19,456 and passed through a dense layer with 512 neurons, reducing the dimensionality. Finally, the output layer generated a tensor of size 5, representing the Q-values for Pac-Man's potential action: Up, Down, Left, Right, No Operation.



To create a working Pac-Man policy, a combination of hyperparameter tuning and training logic was modified over the course of over 20 training attempts spanning many weeks. The hyperparameters that were altered throughout the attempts included the batch size, the discount factor (gamma), the decay rate of epsilon, the learning rate of the target network (Tau) and the learning rate of the policy network, the capacity of the replay memory, and the number of episodes. In addition to hyperparameter tuning, some of the training logic that was altered throughout the training process was normalizing the grayscale pixel values to the range 0 to 1, cropping the observation from 250x160 to 180x160, normalizing rewards to the range -1 to 1, pausing model optimization during frames where Pac-Man was spawning, and frame-skipping.

2.2 Ghosts

To train the ghost agents, we created a custom Gymnasium environment where the ghosts are controllable, and Pac-Man has an existing policy. This environment takes in a tuple of 4 actions, with each action corresponding to Blinky, Pinky, Inky, and Clyde respectively. Each ghost can move left, right, up, or down. Pinky, Inky, and Clyde also contain an additional field that trackss whether they have spawned in or not, as all the ghosts except for Blinky spawn in on a delay. The state space of this environment is a 180x160 grayscale image (Image 1), and the initial state is rendered by copying the starting state of the ALE environment. The environment keeps track of the coordinates of all four ghosts and Pac-Man, and movement is implemented by shifting pixel colors (Image 2) corresponding to the character being moved. Moreover, collisions with walls/pellets are hard-coded, thus ensuring that only Pac-Man removes pellets from the grid, and neither ghosts nor Pac-Man can phase through walls. The reward system is currently set to +10 for winning the game, and -0.01 for each step where Pac-Man is not captured.

Ghost training involved a feedforward neural network to predict strategic movement actions to defeat Pac-Man. The FNN's input layer consists of 10 neurons, representing the normalized x and y coordinates of all four ghosts and Pac-Man. The hidden layer with 138 neurons, leverages high-dimensionality to extract meaningful features from the spatial data, using ReLU activation function to introduce non-linearity. The output layer compressed the data into 16 dimensions, corresponding to the movement

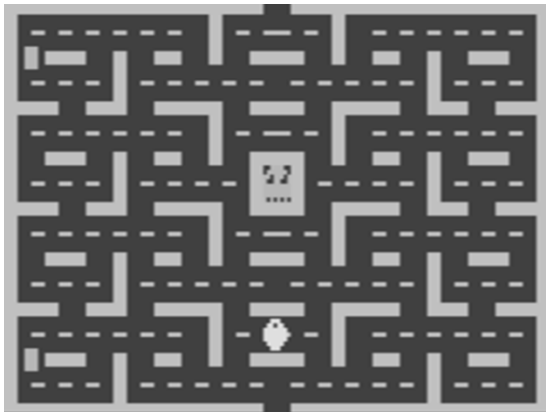


Image 1: Cropped + Grayscale Environment

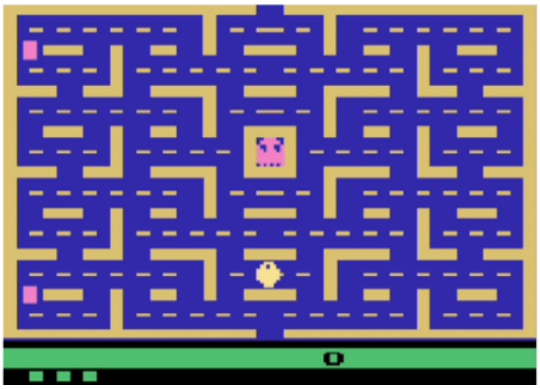


Image 2: RGB Full Environment

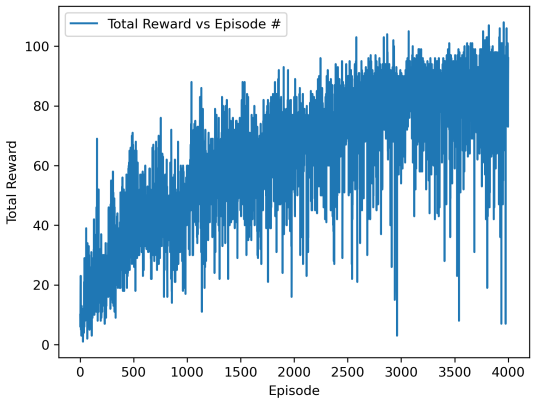
probabilities for each ghost's four possible actions: Up, Down, Left, Right.

3 Results

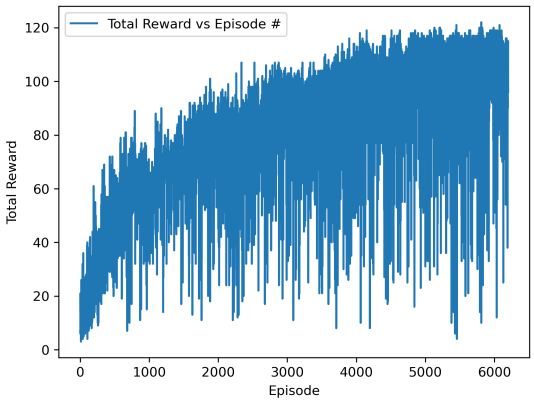
3.1 Pac-Man Training Results

The Pac-Man training involves multiple iterations to refine the reward system and encourage optimal exploration. Initially, the reward system penalized undesired actions too harshly, which discouraged exploration and resulted in cyclic behavior. This was addressed by normalizing the rewards, which allowed Pac-Man to explore the environment effectively. With progressively more episodes, the normalized reward system approach led to stabilized behavior, with the agent's performance levelling out at around 4000 episodes. Training results are visualized through learning curves up to 4000 and 6000 episodes (Figures 1 and 2 respectively). The learning curves demonstrate steady improvement in Pac-Man's average episode reward over time, indicating successful policy optimization.

The Pac-Man policy that we trained in attempt 18 is able to either win the game or get extremely close to collecting all the pips. However, some issues with Pac-Man's behavior were never fully resolved, such as performing much worse after losing one life, and sometimes inexplicably running into ghosts.



Training Attempt 18: 4000 episodes



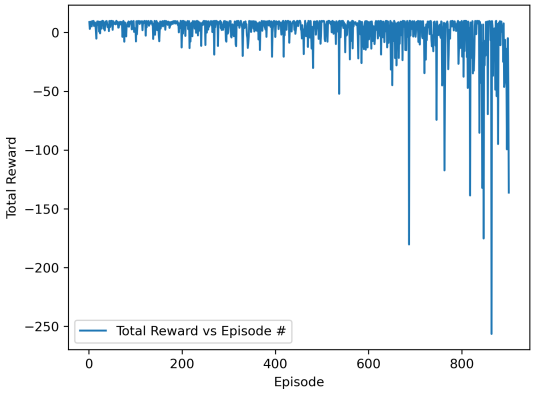
Training Attempt 19: 6000 episodes

3.2 Ghost Training Results

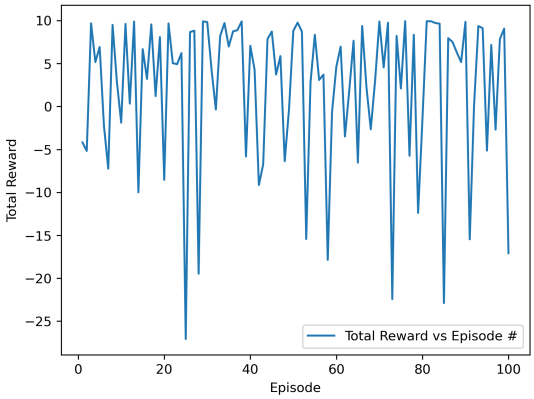
Ghost training involved a feedforward neural network to predict strategic movement actions to defeat Pac-Man. The training objective was to minimize the Manhattan distance between the ghosts and Pac-Man, in order to guide the ghosts towards Pac-Man. The success of the ghost training was not nearly as successful as the Pac-Man training for a number of reasons. Firstly, issues with the custom-made Gymnasium environment made it difficult for the ghost to traverse the entire map. For example, ghosts consistently got stuck in parts of the map where Pac-Man was not located, leading to large negative

rewards. Additionally, since the Pac-Man policy allowed for NOOP actions, the Pac-Man agent became content with sitting in one location, resulting in never-ending training episodes.

Ways to try to improve ghost training included altering the number of pixels the ghosts and Pac-Man moved per action, and only allowing the ghosts to chose legal moves. Despite these additions, the ghosts were unable to find an effective strategy to catch Pac-Man due to the lingering Gymnasium environment issues.



Training Attempt 1: 900 episodes



Training Attempt 2: 100 episodes

Given these negative results, possible remedies include reworking the collision logic in the gymnasium environment. The current implementation has hard-coded values for where all of the pixels/walls are located in an attempt to prevent the ghosts or Pac-Man from making any illegal moves. However, since many passages are very narrow throughout the grid, the collision logic should allow for some collisions with walls. Another possible solution is to check after every pixel movement whether or not an agent gained the ability to make an additional action that was previously illegal. This way, once a ghost or Pac-Man reaches a turn, their movement can be halted, thus allowing them to progress to more locations throughout the map. We also should implement a stop function in the gymnasium environment that terminates an episode that is taking too long. Lastly, the reward system can be altered to penalize the ghosts on the quantity of pips consumed by Pac-Man, rather than the number of steps it takes the ghosts to win the game.

4 Conclusion and Future Plans

This project demonstrated the effectiveness of reinforcement learning in creating dynamic, adaptive behaviors for game agents within Pac-Man. Because of this, we intend to introduce more behaviors for the ghosts to explore, particularly with cooperative behaviors among the ghosts. This would involve expanding the behavioral complexity of ghost agents. Currently, each ghost tracks its distance from Pac-Man independently, but introducing cooperative strategies informed by relative positions could enable more sophisticated interactions.

One other future plan we had was in terms of real-time difficulty scaling against human opponents. This would involve integrating a separate environment where trained ghost agents dynamically adjust their strategies based on the player's skill level during gameplay. By leveraging the existing training pipeline, this feature could create a more personalized and challenging gaming experience.

In addition to resolving lingering issues with the custom environment, we also hope to expand the environment to include more features that are present in the ALE environment, such as power pellets, fruit, multiple lives, and exiting and re-entering the other side of the map through the openings in the top and bottom. We believe that the closer this environment is to the ALE environment, the more successful the ghost training will be in the future.

5 Github Repo:

Access the repo here: `https://github.com/allenyxu2004/CS4100_Pac_Man`

6 Contributions

Luc: Designed the Pac-Man and Ghost training implementation and documented results. Created the custom Gymnasium environment for ghosts training. Also, contributed to the README, documentation, the project presentation and the final report.

Allen: Worked on initial Pac-Man RL implementation, README documentation, project presentation and report.

Iba: Worked on initial custom environment implementation, project presentation and report.

Vishy: Worked on initial custom environment implementation, project presentation and report.

7 References

Farama Foundation. 2024. *Arcade Learning Environment: Pac-Man.* Available at: `https://ale.farama.org/environments/pacman/`.

FYT3RP4TIL. 2024. *Deep Convolutional Q-Learning OpenAI Gymnasium Pac-Man.* Available at: `https://github.com/FYT3RP4TIL/Deep-Convolutional-Q-Learning-OpenAI-Gymnasium-Pac-Man/blob/main/Deep_Convolutional_Q_Learning_Pac_Man.ipynb`.

OpenAI. 2024. *PyTorch Reinforcement Learning (Q-Learning) Tutorial.* Available at: `https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html`.

PacmanCode. 2024. *Pacmancode Repository.* Available at: `https://pacmancode.com/`.