

Learn to Play PACMAN using DQN

Faruk Tamyurek
University of California, Riverside
Riverside, CA
otamy001@ucr.edu

Abstract—Deep Learning has made substantial advancements, notably in achieving human-level efficiency in tasks, which is particularly apparent in reinforcement learning (RL) scenarios. This project aims to demonstrate how reinforcement learning algorithms can be applied to play a video game like PAC-MAN, achieving performance on par with human players. The study concentrates on evaluating the performance of Deep Q-learning algorithms in these gaming scenarios. The OpenAI Gym Atari emulator is employed to conduct these tests, which serves as an excellent platform and provides essential insights into the gaming environment. In particular, the environment provides pixel data for each game frame to our RL agent. The agent employs the above-mentioned algorithms to evaluate the environment's current state and decide on the following action to maximize its reward in the game. This investigation will enhance the understanding of how Deep Q-Network architectures and hyperparameters affect the agent's performance.

The source code and trained models will be available at: https://github.com/ftmyrk/PACMAN_Reinforcement_Learning/

Keywords—Deep Q-Learning, agent, Atari Game, Reward, State

I. INTRODUCTION

Pac-Man, originally called Puck Man in Japan, is a 1980 maze video game developed and released by Namco for arcades. In North America, the game was released by Midway Manufacturing as part of its licensing agreement with Namco America. [1]

Reinforcement learning (RL) is among the disciplines striving to narrow the performance disparity between humans and machines. Despite the challenges in distilling actionable insights from environments, progress in Deep Learning has facilitated feature identification. While the integration of deep learning with RL presents its own set of challenges, such as the absence of labeled data, learning from limited and imprecise reward signals, and the dynamic nature of data distributions as new behavioral patterns are learned, neural networks—convolutional ones in particular—have been instrumental in surmounting these obstacles.

Video games serve as excellent testing grounds for advancing RL algorithms due to their diverse levels of complexity accessible through a uniform platform. The availability of various games, each with its own rules and control strategies, is advantageous for evaluating an algorithm's adaptability across different contexts. Our research centers on Deep RL algorithms that are particularly effective at deriving control policies from raw video inputs. We employ these

algorithms to develop an agent capable of applying RL methods to learn to play games such as PAC-MAN. These algorithms leverage convolutional neural networks (CNNs) to interpret high-dimensional state spaces, such as images.

Reinforcement learning (RL) constitutes a broad paradigm in which agents are trained to make decisions within an environment to maximize a cumulative reward. There are two primary elements: the environment, which embodies the challenge to be addressed, and the agent, which symbolizes the algorithm that undertakes the learning process. There is a continuous interaction between the agent and the environment. With each progressive time step, the agent executes an action upon the environment, guided by its policy $\pi(at|st)$, with s_t being the latest data observed from the environment. Following this, the agent is given a reward r_{t+1} and another data point s_{t+1} from the environment. The overarching aim is to refine the policy to optimize the total of the rewards received.

II. PROBLEM DESCRIPTION

PAC-MAN is a grid-based game where the player navigates a maze to collect pellets while avoiding ghosts. This problem can be framed as a Markov Decision Process (MDP) with states, actions, rewards, and transitions. The state is a representation of the game board, including the positions of PAC-MAN, ghosts, and pellets. The actions are the possible moves (up, down, left, right). The reward structure includes positive rewards for collecting pellets and negative rewards for being caught by ghosts.

A. MsPacman Enviroment

PACMAN is provided by Arcade Learning Environment (ALE) which is a simple framework that allows researchers and hobbyists to develop AI agents for Atari 2600 games.

1) *Actions*: MsPacman has the action space of Discrete(9) with the table below listing the meaning of each action's meanings. To enable all 18 possible actions that can be performed on an Atari 2600, specify `full_action_space=True` during initialization or by passing `full_action_space=True` to `gymnasium.make`.

Value	Meaning	Value	Meaning	Value	Meaning
0	NOOP	1	UP	2	RIGHT
3	LEFT	4	DOWN	5	UPRIGHT
6	UPLEFT	7	DOWNRIGHT	8	DOWNLEFT

Figure 1: PAC-MAN Actions

2) Observation Space

The observation space in the Ms. Pac-Man environment can also vary based on the type of observation:

RGB: The default observation type is an RGB image with a shape of (210, 160, 3) and values ranging from 0 to 255.

Grayscale: A grayscale version of the RGB image with a shape of (210, 160) and values ranging from 0 to 255.

RAM: The raw memory of the Atari 2600, represented as a 128-byte array with values ranging from 0 to 255. [2]

III. TECHNICAL METHOD

A. Workflow

The overall workflow of the DQN implementation for PAC-MAN is depicted in Figure 1. The process begins with setting up the game environment, followed by designing the DQN architecture and preprocessing the state information. The reward system is then defined to guide the agent's learning process. An exploration strategy is employed to balance exploration and exploitation during training. The agent is trained over multiple episodes, and its performance is evaluated and tested. Finally, optimization and tuning are performed to enhance the agent's performance.

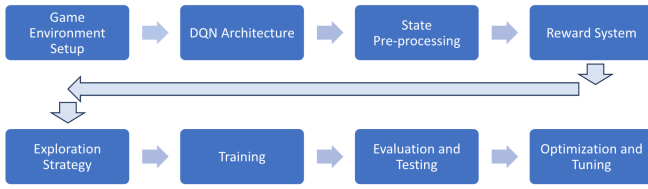


Figure 2: Workflow Chart

B. Neural Network Architecture

A convolutional neural network (CNN) is used to approximate the Q-function. The input is the game state, and the output is the Q-value for each action. The architecture of the CNN is shown in Figure 2. The network consists of the following layers:

- Input Layer: The input is a 210x160x3 image representing the game state.
 - Convolutional Layers:
 - The first convolutional layer has 32 filters of size 8x8.
 - The second convolutional layer has 64 filters of size 4x4.
 - The third convolutional layer has 64 filters of size 3x3.
- Fully Connected Layer: This layer has 512 units.
- Output Layer: The output layer provides the Q-values for each action.

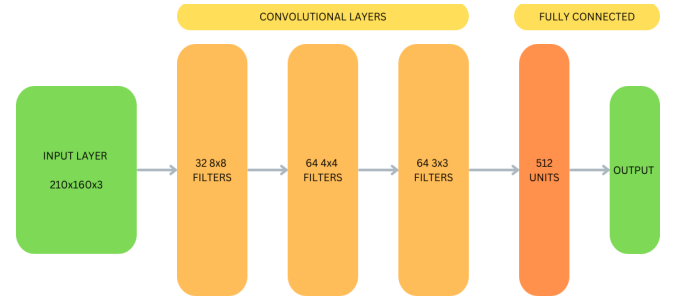


Figure 3: CNN architecture used for the DQN implementation for PAC-MAN

C. Q-Network

The DQN algorithm uses the Bellman equation to update the Q-values. The Q-value for a state-action pair (s, a) is updated using the following equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

Where:

- s is the current state [3]e.
- a is the action taken.
- r is the reward received.
- s' is the next state.
- α is the learning rate.
- γ is the discount factor.
- $\max_{a'} Q(s', a')$ is the maximum predicted Q-value for the next state.

To minimize the difference between the predicted Q-values and the target Q-values, the loss function is defined as:

$$L(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$$

Where:

- θ represents the parameters of the Q-network.
- θ^- represents the parameters of the target network, which are updated periodically to match θ

The gradients of this loss function with respect to the network parameters are then used to perform a gradient descent update. [3]

D. Epsilon-Greedy Exploration Strategy

The exploration strategy used in this implementation is epsilon-greedy, which balances exploration and exploitation. The agent selects a random action with probability ϵ and the action that maximizes the Q-value with probability $1 - \epsilon$.

The epsilon value starts high to encourage exploration and is gradually reduced to promote exploitation as the agent learns.

The epsilon-greedy policy can be formulated as:

$$\pi(a|s) = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \end{cases}$$

[4]

IV. RESULTS

The performance of the Deep Q-Learning Network (DQN) agent was evaluated using various metrics, including average reward per episode, mean reward for every 100 episodes, episode rewards, and the maximum score achieved. The following sections detail the findings.

A. Training Performance

The DQN agent was trained over 10,000 episodes, with its performance monitored throughout the training period. Figure 4 depicts the mean reward per episode, and Figure 5 shows the mean reward for every 100 episodes.

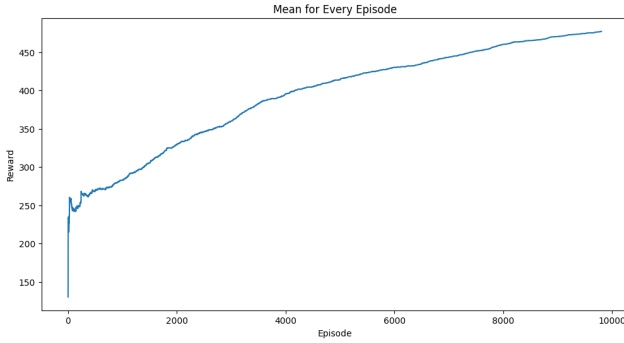


Figure 4: Mean Reward per Episode

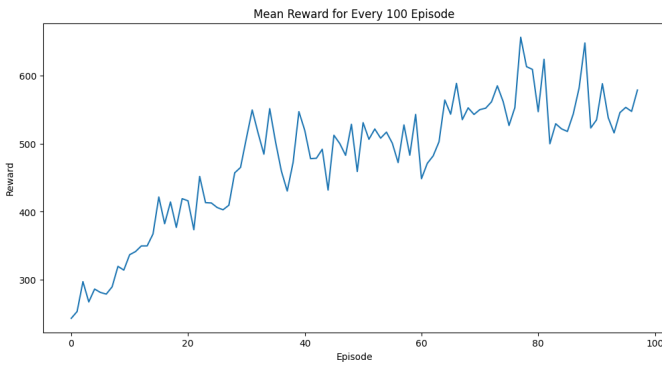


Figure 5: Mean reward for every 100 episodes

B. Reward Distribution and Maximum Score

The distribution of rewards per episode and the mean reward over the entire training period are presented in Figure 6.

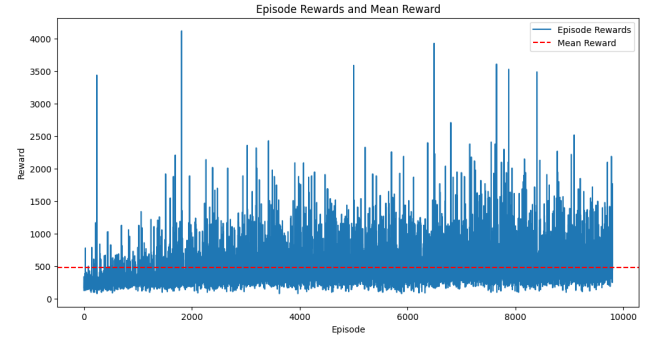


Figure 6: Episode rewards and mean rewards over 10,000 episodes

From Figure 6, it is evident that while the rewards varied significantly across episodes, the mean reward (indicated by the red dashed line) stabilized around 500 points. The agent occasionally achieved very high rewards, with a maximum score of 4,120 points recorded during the training period.

C. Convergence and Stability

The stability and convergence of the DQN algorithm were assessed by analyzing the variation in scores across different training episodes. The training curves indicate a general upward trend with occasional fluctuations, reflective of exploration phases and learning updates. The agent's performance stabilized towards the latter stages of training, suggesting successful convergence of the *learning algorithm*.

V. CONCLUSION

This research demonstrates the successful implementation of a Deep Q-Learning Network (DQN) for playing the PAC-MAN game, illustrating the potential of reinforcement learning in complex gaming environments. The DQN agent showed a marked improvement in performance over a random policy, effectively learning to navigate the game environment and achieve high scores.

The training process, which spanned 10,000 episodes, revealed key insights into the agent's learning dynamics. The mean reward per episode and for every 100 episodes showed significant improvements, indicating effective learning and adaptation by the DQN agent. Despite fluctuations in episode rewards, the mean reward stabilized around 500 points, and the agent occasionally achieved very high scores, with a maximum recorded score of 4,120 points. This variability in rewards underscores the inherent exploratory nature of the reinforcement learning process and highlights the agent's capacity to identify and exploit favorable game scenarios.

The study also underscores the importance of architectural and hyperparameter choices in shaping the agent's learning trajectory and performance. The CNN architecture employed in this research, coupled with strategies like experience replay and target networks, proved effective in

addressing the challenges of high-dimensional state spaces and unstable learning targets.

In conclusion, this work not only showcases the efficacy of DQN in the PAC-MAN game but also provides a foundation for future research in reinforcement learning. Future work could explore enhancements such as Double DQN, Prioritized Experience Replay, and more sophisticated neural network architectures to further improve the agent's performance. Additionally, investigating the application of these methods to other games and real-world problems could yield valuable insights and advancements in the field of reinforcement learning.

VI. REFERENCES

- [1] "Wikipedia," [Online]. Available: <https://en.wikipedia.org/wiki/Pac-Man>.
- [2] "Gymnasium," OpenAI, [Online]. Available: https://gymnasium.farama.org/environments/atari/ms_pacman/.
- [3] A. Choudhary, "A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python," analyticsvidhya, [Online]. Available: <https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>.
- [4] "Epsilon-Greedy Algorithm in Reinforcement Learning," geeksforgeeks, [Online]. Available: <https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/>.