Comparative Study of Reinforcement Learning using Pacman Game

Santhiya Theanraj
Department of Computer and Information Science, Temple University

Abstract

This project investigates various Q-learning based reinforcement learning algorithms applied to the classic Pac-Man game environment. The primary objective is to train an intelligent Pac-Man agent capable of maximizing rewards by consuming food pellets while strategically avoiding ghosts. Three reinforcement learning approaches are implemented and evaluated: Q-Learning, Approximate Q-Learning with hand-crafted features, and Deep Q-Learning (DQN) with convolutional neural networks. The performance of these algorithms is analyzed across four different grid layouts of varying complexity: tinyMaze, smallClassic, mediumGrid, and mediumClassic. Results demonstrate that while basic Q-Learning performs adequately on small grids, it becomes computationally inefficient as state space complexity increases. Approximate Q-Learning shows significant improvements through intelligent feature engineering, achieving high win rates on medium-sized grids. Deep Q-Learning exhibits the most promising results on complex environments by automatically learning feature representations, though requiring substantial computational resources. With sufficient training iterations, the trained agents demonstrate gameplay skills that can surpass human performance on standardized grid layouts.

1 Introduction

Reinforcement learning (RL) has emerged as a powerful paradigm for training autonomous agents to make sequential decisions in complex environments. This project applies reinforcement learning techniques to the classical Pac-Man game, providing a challenging testbed for evaluating different algorithmic approaches.

The Pac-Man environment presents several key challenges characteristic of real-world RL problems: large state spaces, delayed rewards, partial observability (ghost behavior), and the need for both short-term tactical decisions and long-term strategic planning. The agent must learn to navigate mazes efficiently, collect food pellets, utilize power pellets strategically to consume frightened ghosts, and avoid capture by enemy ghosts.

This project implements and compares three fundamental reinforcement learning approaches across multiple grid layouts:

- **Q-Learning:** A tabular method storing Q-values for state-action pairs
- **Approximate Q-Learning:** Function approximation using hand-crafted features
- **Deep Q-Learning (DQN):** Neural network-based approach with automatic feature learning

The game environment is built upon the Berkeley AI Pac-Man framework, providing standardized grid layouts for consistent performance evaluation. Four different grid configurations are utilized, ranging from simple 6×6 grids (tinyMaze) to complex 16×20 layouts (mediumClassic), allowing for systematic analysis of algorithm scalability and generalization capabilities.

Layout	Dimensions	Ghosts	State Space
tinyMaze	6×6	1	Small
smallClassic	16×6	1	Medium
mediumGrid	10×10	1	Large
mediumClassic	16×20	2	Very Large

Table 1: Grid layout configurations and complexity

2 Related Work

The application of deep learning to reinforcement learning has seen significant advancement in recent years, particularly through the development of Deep Q-Networks (DQN). Mnih et al. [2] introduced the first deep learning model capable of learning control policies directly from high-dimensional sensory input using reinforcement learning. Their approach successfully addressed several key challenges that had previously hindered the combination of deep neural networks with RL.

The DQN architecture employs a convolutional neural network trained with a variant of Q-learning, where the input consists of raw pixel data and the output is a value function estimating future rewards. Two critical innovations enabled stable training: **experience replay**, which stores agent experiences in a replay buffer and randomly samples transitions to break correlations in the training data, and a **target network**, which is updated periodically to provide stable Q-value targets during training [2].

When applied to Atari 2600 games, DQN demonstrated remarkable performance, outperforming previous approaches on six out of seven games tested and surpassing human expert performance on three games, all without any game-specific architectural modifications or handcrafted features [2]. This work established that deep reinforcement learning could successfully scale to complex, high-dimensional state spaces.

Building upon this foundation, our implementation adapts the DQN approach to the Pac-Man domain, using a similar convolutional architecture but with modifications suited to the grid-based nature of the game environment.

3 Environment Design

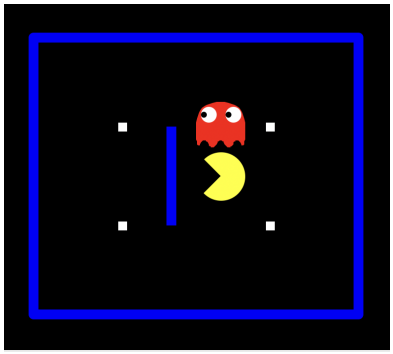
The Pac-Man environment consists of a grid-based maze where the agent (Pac-Man) must collect food pellets while avoiding ghosts. A key challenge in creating an effective learning environment is balancing difficulty: ghosts that are too predictable provide insufficient challenge, while perfectly optimal ghosts make learning impossible.

3.1 Ghost Movement Algorithm

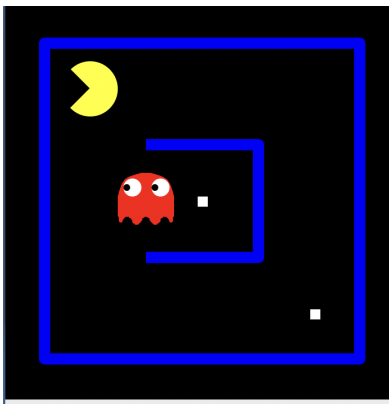
We implemented a weighted probabilistic chase algorithm that creates intelligent yet beatable ghost behavior.This approach ensures ghosts actively chase Pac-Man but occasionally make suboptimal moves, creating a balanced learning environment where the agent can develop both offensive (food collection) and defensive (ghost avoidance) strategies.

Algorithm 1 Action Execution and State Transition

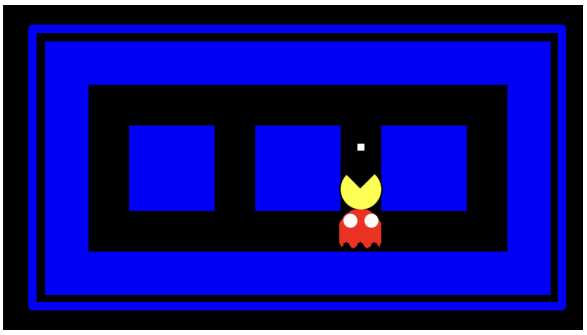
function ExecuteAction($state, action$)
    $target \leftarrow$ ApplyAction($state.pacman, action$)
    if $target$ is wall **then**
        $new\_pacman \leftarrow state.pacman$
    else
        $new\_pacman \leftarrow target$
    end if
    $new\_dots \leftarrow state.dots$
    if $new\_pacman \in state.dots$ **then**
        $new\_dots \leftarrow state.dots \setminus \{new\_pacman\}$
    end if
    $ghost\_moves \leftarrow$ GetLegalMoves($state.ghost$)
    $ghost\_moves \leftarrow ghost\_moves \setminus \{OppositeDir(state.ghost\_dir)\}$
    $sorted\_moves \leftarrow$ SortByDistance($ghost\_moves, new\_pacman$)
    $weights \leftarrow [n, n-1, \ldots, 1]$ where $n = |sorted\_moves|$
    $new\_ghost\_dir \leftarrow$ WeightedRandomChoice($sorted\_moves, weights$)
    $new\_ghost \leftarrow$ ApplyAction($state.ghost, new\_ghost\_dir$)
    $new\_state \leftarrow$ State($new\_pacman, new\_ghost, new\_dots, state.walls, new\_ghost\_dir$)
    return new_state
end function
function GetLegalMoves(position)
    $moves \leftarrow \emptyset$
    for $direction \in \{North, South, East, West\}$ **do**
        $next \leftarrow$ ApplyAction($position, direction$)
        if $next$ is not wall **then**
            $moves \leftarrow moves \cup \{direction\}$
        end if
    end for
    return $moves$
end function



(a) Tiny Grid



(b) Medium Grid



(c) Medium Classic Grid

Figure 1: Pacman grid environments of increasing complexity used in this study

3.2 Distance Metrics and Features

For feature extraction in Approximate Q-Learning, we use Manhattan distance to measure distances between grid positions:

$$d_{\text{Manhattan}} = |x_1 - x_2| + |y_1 - y_2| \tag{1}$$

Manhattan distance is ideal for grid-based movement where diagonal moves are not allowed, providing accurate path length estimates.

3.3 Reward Function

The reward structure encourages efficient food collection while penalizing dangerous behavior.The small negative reward per time step prevents the agent from wandering aimlessly and encourages goal-directed behavior.

4 Experimental Setup

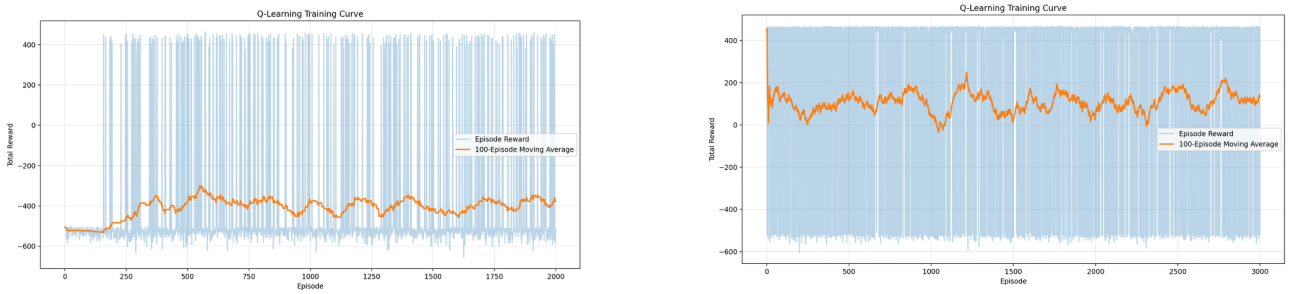
4.1 Q-Learning Implementation and Parameter Tuning

For the Q-Learning implementation, I began with a tiny 6×6 grid environment to evaluate the agent's learning capabilities. The initial training configuration consisted of 2,000 episodes with a learning rate (α) of 0.5, an exploration factor (ϵ) of 0.3, and a discount factor (γ) of 0.9. As illustrated in Figures 2a and 2b, the agent's performance demonstrated significant improvement after the second episode, indicating effective learning and adaptation to the environment. However, the high exploration factor resulted in excessive random actions even after

Algorithm 2 Reward Calculation

```
function CalculateReward(old_state, action, new_state)
    reward ← 0
    dots_eaten ← |old_state.dots| − |new_state.dots|
    reward ← reward + (dots_eaten × 10)
    reward ← reward − 1
    if new_state.pacman = new_state.ghost then
        reward ← reward − 500
    else if |new_state.dots| = 0 then
        reward ← reward + 500
    end if
    return reward
end function
```

the agent had acquired substantial knowledge about optimal policies. To refine the agent's behavior and reduce unnecessary exploration, I subsequently decreased the epsilon value from 0.3 to 0.1. This adjustment allowed the agent to exploit its learned policy more frequently while still maintaining sufficient exploration to avoid local optima, resulting in more consistent and improved performance across subsequent training episodes.



(a) Performance with $\epsilon = 0.3$ (initial episodes)

(b) Performance with $\epsilon = 0.3$ (later episodes)

Figure 2: Q-Learning agent performance showing improvement after Episode 2 with $\epsilon = 0.3$

During the play phase with the tiny maze, the Q-Learning agent demonstrated strong performance, winning 75% of the games, which validated the effectiveness of the learned policy in a limited state space. Encouraged by these results, I proceeded to evaluate the agent on a medium grid of size 10×6, where the state space increased significantly. However, the Q-Learning agent with the same hyperparameters ($\alpha = 0.5$, $\epsilon = 0.3$, $\gamma = 0.9$, 2,000 episodes) that performed well in the tiny maze suffered considerably in this expanded environment. The agent failed to win any games, highlighting a critical limitation of tabular Q-Learning: as the state space grows exponentially with grid size, the same number of training episodes becomes insufficient for adequate exploration and convergence. This performance degradation underscores the curse of dimensionality in reinforcement learning, where the agent requires significantly more training episodes to visit and learn optimal actions for the vastly increased number of state-action pairs in the larger environment.

Grid Size	State Space	Win Rate	Training Episodes
Tiny (6×6)	Small	75%	3,000
Medium (10×6)	Large	0%	3,000

Table 2: Q-Learning performance comparison across different grid sizes

4.2 Approximate Q-Learning Configuration and Hyperparameters

For the Approximate Q-Learning implementation on the medium grid, I configured the agent with carefully tuned hyperparameters to balance exploration, learning stability, and long-term planning. The training was conducted over 3,000 episodes using the simple feature extractor with a discount factor (γ) of 0.95, emphasizing the importance of future rewards and encouraging strategic planning over immediate gains. The learning rate (α) was set to 0.3, providing moderate update steps to ensure stable convergence of feature weights without overshooting optimal values. To leverage the generalization capabilities of function approximation, the exploration rate (ϵ) was reduced to 0.1, significantly lower than the 0.3 used in tabular Q-Learning, as the agent's ability to generalize across similar states reduces the need for extensive random exploration. After training, the agent's learned policy was evaluated over 5 play episodes. This configuration—combining increased training episodes (3,000 vs 2,000), higher discount factor (0.95 vs 0.9), and reduced exploration (0.1 vs 0.3)—was specifically designed to exploit the feature-based representation's efficiency in handling the expanded state space of the medium grid.

Method	**Grid Size**	**Episodes**	α	γ	ϵ
Approx Q-Learning	10×6 (Medium)	3,000	0.3	0.95	0.1

Table 3: Comparison of hyperparameters across different Q-Learning approaches

Figure 3 illustrates the training progress of the Approximate Q-Learning agent on the classic 7×10 grid over 10,000 episodes. The learning curve demonstrates the agent's gradual improvement in cumulative rewards as the feature weights converge toward optimal values. The initial episodes show high variance due to exploration and random weight initialization, while later episodes exhibit more stable performance as the agent exploits its learned policy. The feature-based approach enables effective generalization across similar states, resulting in consistent improvement even in the larger state space where tabular Q-Learning failed.

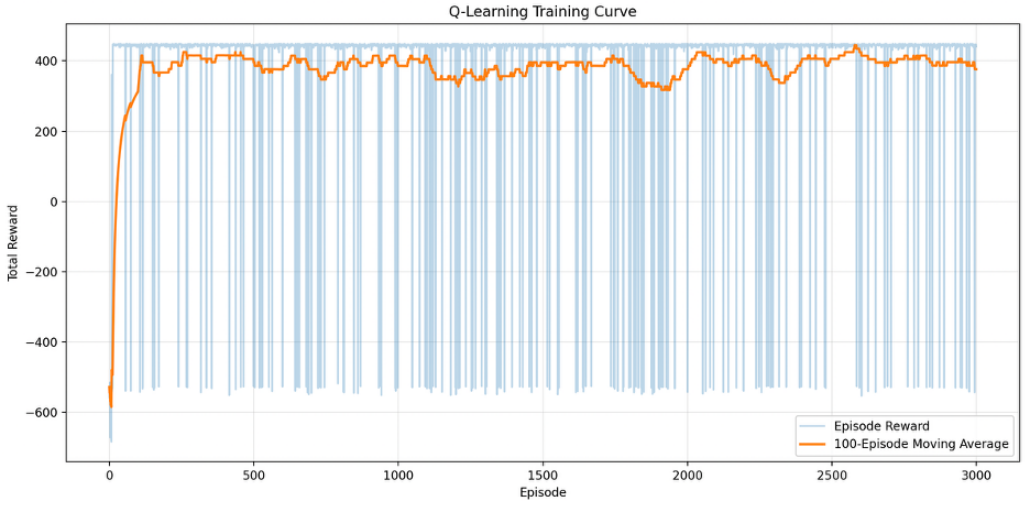


Figure 3: Approximate Q-Learning training performance on classic grid (7×10) over 10,000 episodes showing cumulative rewards per episode

5 Deep Q-Network (DQN) Implementation

To overcome the limitation of manual feature engineering in Approximate Q-Learning, I implemented a Deep Q-Network(DQN) that automatically learns state representations through

a neural network.The DQN agent was implemented using a CNN designed to efficiently process a structured, grid-based representation of the Pac-Man environment. Instead of raw pixel input, the state was encoded as a five-channel equivalent image, representing the positions of Pac-Man, ghosts, food pellets, capsules, and walls. This representation significantly reduced computational overhead while preserving essential spatial information.The network architecture consists of three convolutional layers with increasing channel depths, followed by two fully connected layers. ReLU activation functions were applied throughout the network to introduce non-linearity. The output layer produces Q-values corresponding to the four possible movement actions.

Training was performed using experience replay with a memory capacity of 100,000 transitions and a batch size of 64. A target network was updated every 50 training steps to stabilize learning. The learning rate was set to 0.00025, and an epsilon-greedy exploration strategy was employed, with epsilon decaying from 1.0 to 0.01 over 1,000 episodes.

Experimental results indicate that the DQN agent learns effective policies on small grid layouts within approximately two hours of training Training was conducted on the smallClassic grid for 5,000 episodes with learning rate $\alpha = 0.001$ and discount factor $\gamma = 0.95$:

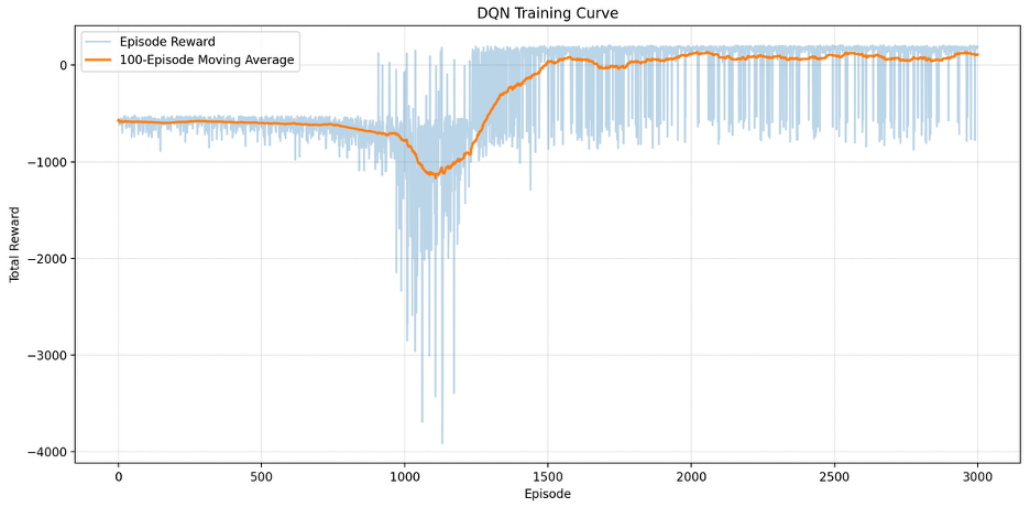


Figure 4: DQN training curve with epsilon decay overlay

Table 4: Pacman Q-Learning Training Performance

Metric	**Value**
Total Training Episodes	3,000
Wins	1,506
Losses	1,494
Overall Win Rate	50.20%
Final Win Rate (Last 100 Episodes)	93.00%
Overall Average Reward	-265.81
Final Average Reward (Last 100 Episodes)	111.51
Best Episode Reward	205.00
Worst Episode Reward	-3912.00

The DQN agent achieved superior generalization through automatic feature learning, though requiring longer training compared to approximate Q-Learning (5,000 vs 3,000 episodes for comparable grids).

6 Other Attempts

Several challenges were encountered during the initial implementation of the Deep Q-Network (DQN). The choice of the number of network layers, kernel size, stride, and activation functions plays a critical role in the performance of convolutional neural networks (CNNs). Most of the architectures explored in this work were inspired by the classical Deep Q-learning framework proposed by Mnih et al. However, the inclusion of max-pooling layers after every convolutional layer resulted in degraded performance.

An attempt was made to use raw pixel data from game frames captured during gameplay as input to the network. This approach proved to be computationally expensive and challenging due to the high dimensionality of the input space.

State-space information was stored in the replay memory at every step of the Pacman game and accessed during each training step. Therefore, selecting an efficient data structure for adding, removing, and sampling experiences was essential. Initially, NumPy arrays were used to implement the replay memory with the expectation of faster access compared to a queue-based approach. However, this implementation was inefficient, and the design was subsequently changed to use a `deque`, which significantly improved performance.

Training on the medium classic grid remained a major challenge due to the large model size, consisting of approximately 1.8 million parameters. Even after more than 16 hours of training with extensive hyperparameter tuning, the agent failed to win any games and exhibited near-random behavior. Multiple modifications to the neural network architecture and batch size were attempted to improve training performance. However, the large state space caused the agent to require significantly longer training times than expected even after using colab GPU.

7 Conclusion

This project helped in developing a strong understanding of the fundamentals of reinforcement learning and how agents learn using a Q-table. It highlighted the importance of feature extraction in approximate Q-learning, which enables agents to learn efficiently in larger state spaces. Similarly, the project demonstrated how Deep Q-learning automatically learns important feature representations through exploration and experience replay, even in smaller classic grid environments.

8 Acknowledgement

I would like to sincerely thank Prof. Pei Wang for his continuous support throughout the course. His assignments on diverse topics and the accompanying discussions were quite different from conventional coursework and provided a unique learning experience. Through this course, I was introduced to the foundational work of artificial intelligence from 1960s to present day. Overall, this course enabled me to learn a wide range of new theories and research concepts related to artificial intelligence.

References

[1] Berkeley Pacman Code. *http://ai.berkeley.edu/projectoverview.html.*

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, 2013.

[3] David Silver et al., "Human-level control through deep reinforcement learning," *Nature*, doi:10.1038/nature14236, 2015.

[4] Matthew Hausknecht and Peter Stone, "Deep Recurrent Q-Learning for Partially Observable MDPs," *arXiv preprint arXiv:1507.06527*, 2015.

[5] Hado van Hasselt, Arthur Guez, and David Silver, "Deep Reinforcement Learning with Double Q-Learning," *arXiv preprint arXiv:1509.06461*, 2015.

[6] Volodymyr Mnih et al., "Reinforcement Learning with Unsupervised Auxiliary Tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[7] Mehdi Mirza et al., "Asynchronous Methods for Deep Reinforcement Learning," *arXiv preprint arXiv:1602.01783*, 2016.

[8] Ioannis Antonoglou et al., "Prioritized Experience Replay," *arXiv preprint arXiv:1511.05952*, 2016.

[9] Python Documentation. *https://docs.python.org/3/*.