Multi-Agent Racing Simulation with Q-Learning and DQN

ZHAO Yuhao

Abstract

This report describes a reinforcement learning game design project that utilises both traditional Q-learning and Deep Q-Network (DQN) methods [7]. The game simulates a multi-agent racing environment where two intelligent racing cars—one controlled by Q-learning agent and the other by a DQN agent—compete on a dynamically generated track. The main goals were to design a game environment with a well-defined state and action space, create a reward function that encourages optimal behaviour, and implement a reinforcement learning algorithm that learns to maximize cumulative rewards. In this report, we provide a detailed description of the game design, environment setup, and the two RL methods employed. The game design is explained, the Q-learning and DQN implementations are detailed, the user interface and simulation environment are described, and performance evaluation is discussed.
Github: https://github.com/Danielyhzhao/Car-Racing-Q-learning-vs-DQN.git
YouTube: https://youtu.be/C96B6mEW-jY

1 Introduction

Reinforcement learning (RL) offers a means for agents to learn complex behaviors by interacting with an environment and receiving rewards based on their actions [3]. In this project, we have designed a two-car racing game in which each agent is trained using a different RL method: one using classical Q-learning and the other using DQN with function approximation via a neural network.
The main goals of this task are:

1. **Game design**: Create a racing environment with a clear goal, defined state space, action space, and reward structure.

2. **Algorithm implementation**: Implement Q-learning using an ε-greedy strategy and build a DQN for function approximation using PyTorch.

3. **Evaluation**: Analyze the learning performance of the two agents based on cumulative rewards, success rates, and convergence behavior.

4. **Innovation Points:** Multi-Agent Competitive RL; Advanced Reward Shaping; Comprehensive Performance Visualization; Using Q-learning and DQN algorithms simultaneously.

2 Environment and Game Mechanics

The game environment is a two-dimensional racing track where two racing cars compete to complete a lap represented by a pixel distance.

1. **Track Generation:** The center of the track is generated by periodic functions (sine and cosine) to simulate curves and undulations. The width of the track can be dynamically adjusted according to the traveling of the cars on the track.
2. **State Representation:** Each state is defined by a three-dimensional vector:
   - **Relative x-offset:** The offset of the car's center from the dynamic track center.
   - **Normalized y-position:** The normalized value of the current Y-coordinate in the total circumferential distance.
   - **Normalized Speed:** Normalized by the maximum permitted speed.
3. **Action Space:** 1. no action 2. Accelerate 3. Decelerate 4. Steer left 5. Steer right
4. **Reward Structure:** Rewards the agent for increasing its vertical distance traveled. Provides a bonus of 50 points for overtaking opponents. Penalizes the agent for going off the track (-100). Rewards are given to agents based on their rankings after completing the race.
5. **Agent Architecture:**
   - **Q-Learning Agent:** leverages a table-based approach for discrete state spaces.
   - **DQN agent:** uses a deep neural network to generalize across continuous and high-dimensional state spaces.

   Both agents run in the same environment at the same time and share the same game dynamics, thus allowing a meaningful comparison between traditional and deep learning approaches.

3 Q-Learning Implementation

Q-Learning is a tabular reinforcement learning algorithm for learning optimal policies by iteratively updating Q values. And also is a model-free RL methods based on learning the optimal Q-value.[6](defined as the expected cumulative reward for different state-action pairs) Key steps include:

1. **Q-Table Initialization:** The Q table is used to store the values of all possible state-action pairs. Initially, these values are usually set to zero (or a very small random value).

2. **Epsilon-Greedy Action Selection:** Q-learning uses an epsilon-greedy strategy. With probability ϵ, a random action is selected, and with probability $1-\epsilon$, the action with the highest Q-value in the current state is chosen.[5]

$$P(a) = \begin{cases} \epsilon, & \text{choose a random action} \\ 1-\epsilon, & \text{choose the best action based on Q-values} \end{cases}$$

3. **State transitions and rewards:** An agent takes an action, receives a reward (rewards based on progress, surpassing, or punishment), and transitions to a new state. In the racing game, the reward may be directly linked to distance traveled, overtaking bonuses, or penalties

4. **Q-Value Update:** Q values are updated based on observed rewards and the maximum expected future reward in the new state.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1}$$

5. **Epsilon Decay:** This means that once the agent starts getting more accurate Q values, it gradually moves from exploration to exploitation.

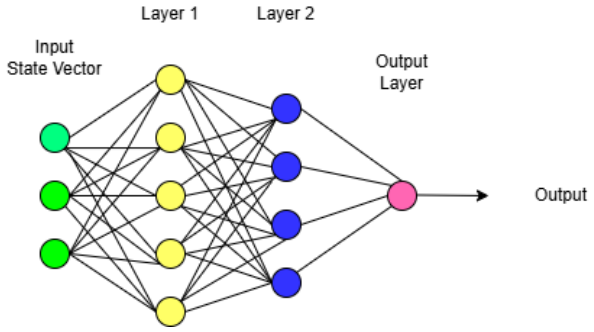
6. **Hyperparameters:**

   - Learning rate (α): 0.1
   - Discount factor (γ): 0.9
   - Initial epsilon (ϵ): 1.0
   - Minimum epsilon: 0.1
   - Epsilon decay rate: 0.995

4 DQN Implementation

Deep Q Networks (DQN) extend Q learning by replacing Q-tables with neural networks that approximate Q-value functions. This approach can handle high-dimensional or continuous state spaces.[7]

1. **Neural Network Architecture:** The network takes the state vector as input and outputs Q-values for each action. For instance, in our racing game, the network takes three features:



- **Input Layer:** Accepts three key state features.
- **Hidden Layers:** Two layers with 128 and 64 neurons, using RELU activations.
- **Output Layer:** Produces Q-values for each of the five actions.

2. **Experience Replay:** DQN use an experience replay memory rather than learning from successive states (which may be highly correlated). Transitions (state, operation, reward, next state, completion) are stored in a buffer. During training, the network randomly samples mini-batches from this buffer.[4] This approach helps to break the correlation between experiences and makes the training process more stable.

$$D = \{(s_t, a_t, r_t, s_{t+1})\}$$

$$\{(s_i, a_i, r_i, s_i') \mid i \in \text{mini-batch}\}$$

3. **Target Network:** The target network in a DQN is an independent neural network that provides a stable target Q value for the loss function by separating the rapidly changing policy network parameters from the estimation of future rewards.

$$\text{Target} = r + \gamma \max_{a'} Q_{\text{target}}(s', a')$$

4. **Loss Function and Optimization:** The loss function in DQN is computed as the Mean Square Error (MSE) between the Q-value predicted by the policy network for the selected operation and the target Q-value. [1]:

$$\text{Loss} = \text{MSE}\Big(Q(s, a), r + \gamma \max_{a'} Q_{\text{target}}(s', a')\Big)$$

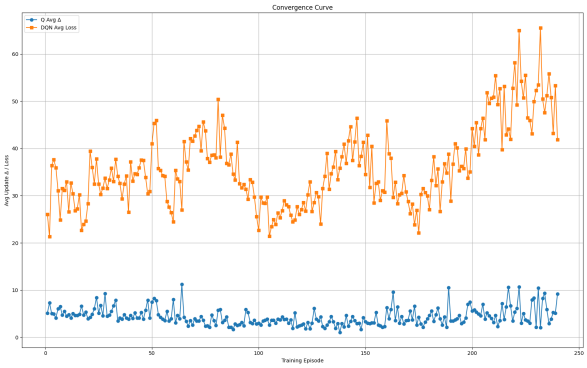
An optimizer (such as Adam) is used to perform backpropagation and update the network weights accordingly.

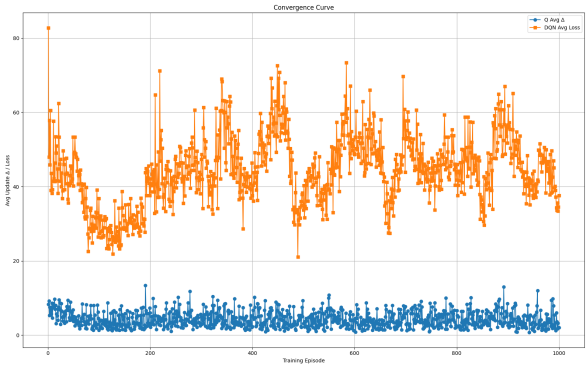
5. **Epsilon-Greedy Exploration:** Just like in Q-learning, DQN employs an epsilon-greedy strategy to balance exploration and exploitation. During training, ϵ gradually decays to encourage more exploitation over time while still maintaining a small chance of exploration by not falling below a defined minimum level.

5 Evaluation by Curve

5.1 Convergence Curve

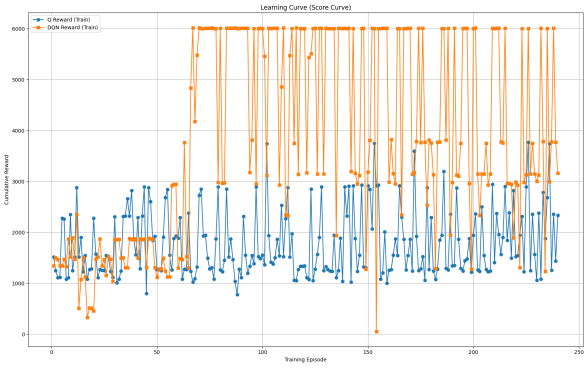
Q-learning successfully converges with minimal fluctuations, demonstrating stable learning and effective policy optimization. DQN struggles with stability, showing high loss variance even after 1000 trials, indicating incomplete convergence. Increasing training episodes to 1000 improves DQN's frequency of high rewards but does not eliminate instability. DQN may suffer from optimization problems such as too fast learning, insufficient empirical replay or improper updating of the target network.

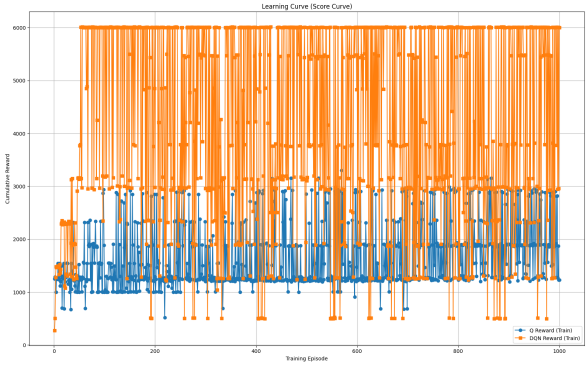




5.2 Learning Curve

Q-learning improves gradually during training. Although it tends to increase, it does not consistently reach optimal performance like DQN. The DQN fluctuates, occasionally alternating between high rewards and very low values. This suggests that the DQN occasionally achieves optimal performance and that stability is lacking. Q-learning maintains a high level of stability, but does not achieve the same peak performance as DQN. Increasing the number of training episodes from 240 to 1000 does not completely stabilize DQN; although DQN obtains high rewards more frequently, the instability may require further tuning.

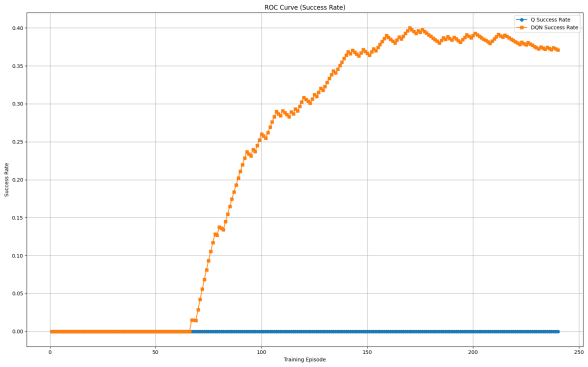


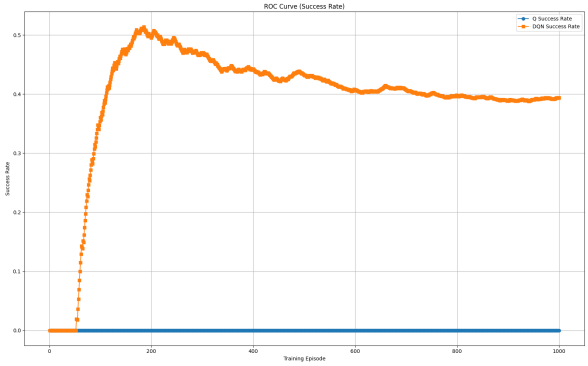


5.3 ROC Curve

Q-learning fails improve success rates, which are close to zero throughout the training process. DQN has made significant progress in learning, around 40 percent success rate in 240 episodes and peaking above 50 percent in 1000 episodes. Longer training times improve the performance of DQN, but also introduce instability, indicating possible overfitting. DQN outperforms Q-learning significantly, but further tuning is needed for stability and sustained peak performance.

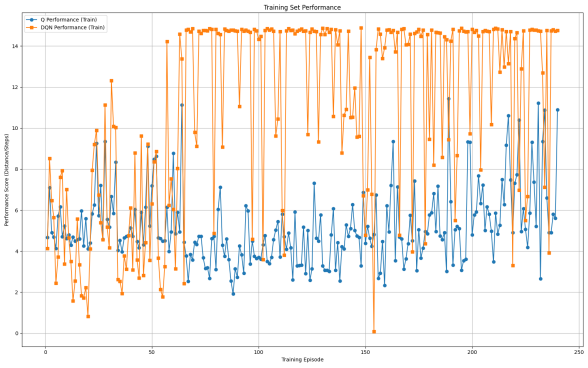
$$TPR = \frac{TP}{TP + FN}$$

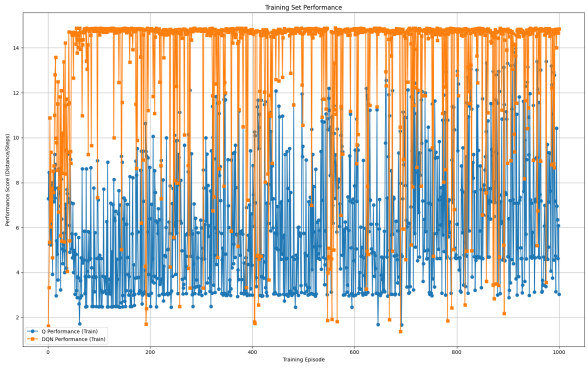




5.4 Training Set Performance

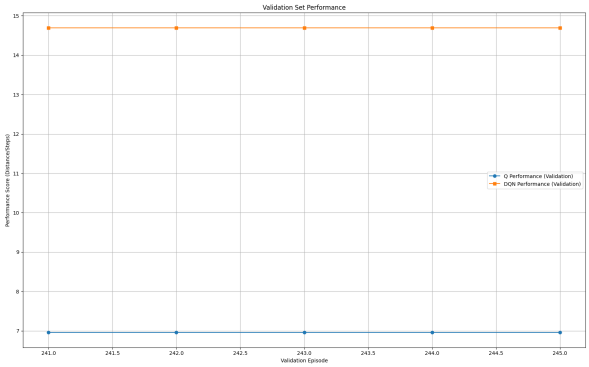
DQN consistently outperforms Q-learning with significantly higher performance scores. The performance of Q-learning gradually improves, but remains lower than that of DQN throughout the training process. Longer training periods stabilise the peak performance of the DQN, but fluctuations remain. DQN learns much faster and outperforms Q-learning, Longer training sets may stabilize DQN's peak performance.

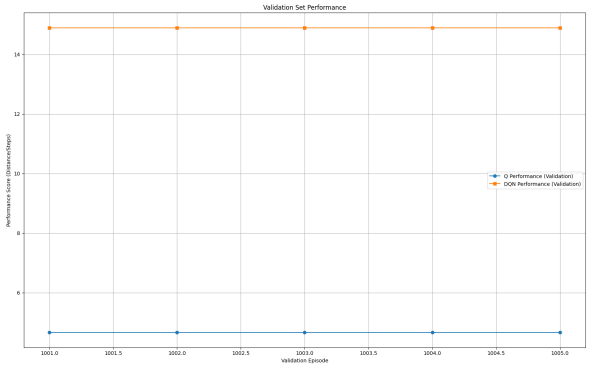




5.5 Validation Set Performance

DQN achieves near-optimal performance scores (15), showing strong generalisation to the validation set. Q-learning achieves a much lower level of performance (7 points), indicating poor learning and generalisation capabilities.





6 Challenges Faced and Solutions Implemented

1. **Concurrent Action Handling:** We ensure that both agents update their states sequentially within each simulation step. This avoids race conditions and keeps state updates in sync during movement and collision detection.

2. **Training Instability:** Oscillating Q-values made it difficult for both Q-learning and DQN to converge. The problem of unstable training is solved by using MSE loss and increasing the training epoch.[2]

3. **Using q-learning and DQN together:** Independent Learning: Q-learning uses a Q-table, while DQN uses a neural network with replay memory, each updating separately.

7 Conclusion

The project successfully implements a two-agent racing game that utilizes both classical Q-learning and a Deep Q-Network. Simulation with dynamically defined road conditions, clear reward structure and visual feedback. Through careful design decisions, iterative testing, and parameter tuning, both agents are able to learn effective driving policies during the training process. DQN outperforms Q-learning, showing faster learning, higher performance, and better generalization. The difficulty with Q-learning is that the learning rate is slow and inconsistent, and even after extended training, no significant progress can be made. Use DQN to get better performance, but optimize the hyperparameters to enforce consistency.

References

[1] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, pages 176–185. PMLR, 2017.

[2] Hanan Ather. Deep reinforcement learning and function optimization. 2023.

[3] Felipe Leno Da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.

[4] Tim De Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Experience selection in deep reinforcement learning for control. *Journal of Machine Learning Research*, 19(9):1–56, 2018.

[5] Richard Dearden, Nir Friedman, Stuart Russell, et al. Bayesian q-learning. *Aaai/iaai*, 1998:761–768, 1998.

[6] Yanwei Jia and Xun Yu Zhou. q-learning in continuous time. *Journal of Machine Learning Research*, 24(161):1–61, 2023.

[7] Yaodong Yang, Jianye Hao, Mingyang Sun, Zan Wang, Changjie Fan, and Goran Strbac. Recurrent deep multiagent q-learning for autonomous brokers in smart grid. In *IJCAI*, volume 18, pages 569–575, 2018.