WiDS'25 Assignment 3

Balaaditya Matcha

January 8, 2026

Contents

1 Introduction

The MountainCar-v0 environment presents a significant challenge for traditional Tabular Q-Learning due to its continuous state space and sparse reward structure(-1 per step). While discretization allows Tabular methods to function, they lack the ability to generalize across similar states.

To address this, I implemented a Deep Q-Network(DQN) as suggested, which approximates the Q-value function using a neural network.

2 Methodology Implementation

2.1 Network Architecture

To handle the continuous input state of size 2, I implemented a multi-layer network with two hidden layers of 128 neurons each. I utilized ReLU activation functions for non-linearity as it is standard.

```
1
2  class DNN(nn.Module):
3      def __init__(self):
4          super().__init__()
5          self.s = nn.Linear(2, 128)
6          self.l1 = nn.Linear(128, 128)
7          self.r = nn.Linear(128, 3)
8
9      def forward(self, x):
10         x = F.relu(self.s(x))
11         x = F.relu(self.l1(x))
12         x = self.r(x)
13         return x
```

Listing 1: DQN Architecture

2.2 Stability Mechanisms

Deep Reinforcement Learning is notoriously unstable due to correlations in sequential data and non-stationary targets. We addressed this using two standard DQN techniques:

- **Experience Replay:** We utilized a `ReplayBuffer` to store transitions $(s_t, a_t, r_{t+1}, s_{t+1}, done/terminated)$. By sampling random batches during training, we break the temporal correlation between consecutive steps, stabilizing the gradient updates.
- **Target Network:** We maintained a separate `target_net` that is a lagged copy of the policy network. The target weights are updated only periodically (every episodes) to prevent the moving target problem where the network chases its own tail.

```
1  # Inside the training loop:
2  model.update_target_net()
```

Listing 2: Target Network Update Logic

2.3 Reward Shaping

This is the most important part as it is indirectly the strategy itself.

The standard MountainCar environment returns a sparse reward of -1 for every step until success. For an epsilon-greedy agent, this provides no gradient information during early exploration. To guide the agent, I implemented a reward shaping mechanism based on Potential Energy(roughly).

I modified the reward to make the agent to move away from the valley floor (-0.5). I utilized the absolute distance from the center, ensuring the agent is rewarded for building momentum in any direction, to temporarily worsening the state in order to achieve long-term success as stated.

```
1  # Reward modification for the "Optimal Strategy"
2  pos = next_state[0]
3          modified_reward = reward + 0.5 * abs(pos - (-0.5)) # Encourage moving away from
       bottom
4
```

```
        if pos >= 0.5: # Goal
            modified_reward += 100
```

Listing 3: Reward Shaping Implementation

3 Performance Analysis

3.1 Tabular Q-Learning vs. DQN

I compared the performance of a discretized Tabular Q-Learner against the DQN agent.

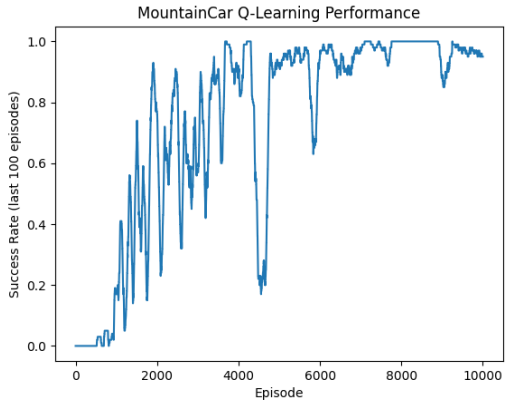

Figure 1: Tabular Q-Learning

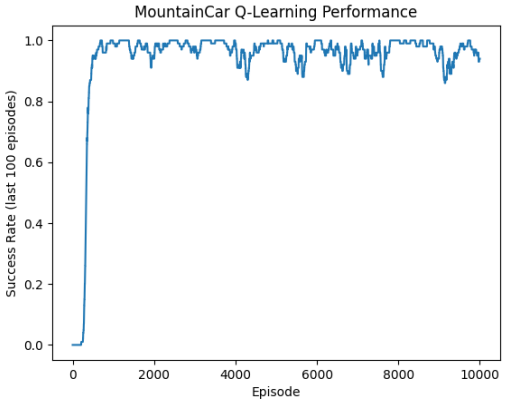

Figure 2: Deep Q-Network (DQN)

The Tabular approach (Figure 1) shows high variance and instability even near the convergence. In contrast, the DQN (Figure 2) exhibits a characteristic "phase shift" around episode 300, where the Success Rate shoots vertically to 100% and remains stable. This demonstrates the superior generalization capability of the neural network; once it learns the concept of "momentum" for one state, it applies it effectively to neighboring states in the continuous space(It really amazed me!).

4 Discussion on Failure Modes

1. First I forgot to use the optimal strategy which lead to 0 success rate for all episodes.
2. Then I added the modified_reward part a slightly different from the one in **Section 2.3** where I used $+ 10 * abs(...)$ and the success rates were too bad, very unstable because I gave too much points for the incentivization which lead the model to cheat and go in the wrong direction. Then I reduced it to 0.5 so that the modified reward still is < 0 but > -1 to let it become stable.
3. And then I reduced the hidden layers in the neural network to 2 after some instability in my 4 layer version.

5 Conclusion

The implementation of Deep Q-Learning successfully solved the MountainCar-v0 environment with a success rate consistently nearing 100%. By replacing the tabular lookup with a neural network and implementing crucial stability features like Experience Replay and Target Networks, the agent was able to learn a robust policy that generalizes well across the continuous state space.