MACHINE LEARNING LAB

ASSIGNMENT 5

SK NAID AHMED

302311001005

A2

Repo- Github

TITLE: Reinforcement Learning & Deep Reinforcement Learning Based Solutions Using OpenAI Gym

1. Introduction

Reinforcement Learning (RL) is a learning paradigm where an autonomous agent interacts with an environment and learns an optimal policy to maximize cumulative reward.
Deep Reinforcement Learning (DRL) extends RL by using neural networks as function approximators to handle high-dimensional state spaces.

In this assignment, we apply both RL and DRL to solve classical control problems from the OpenAI Gym package and compare their performance with a shortest-path problem in a user-defined graph.

---

2. Objectives of the Assignment

1. Implement **any two** RL tasks using classical Reinforcement Learning:
    - MountainCar
    - CarRacing
    - Roulette
2. Apply **Deep Reinforcement Learning (DQN)** to solve the same problems.
3. Implement **both RL (tabular)** and **DRL (DQN)** for the shortest-path problem in a user-defined graph and compare their performance.

---

3. Problems Selected

For this assignment, the following two problems were selected:

1. **MountainCar-v0** – Continuous state environment where the agent must learn to drive a car uphill.
2. **Roulette (Multi-Armed Bandit)** – Stochastic decision-making problem where the agent learns the best arm to maximize reward.

Additionally, both **RL and DRL** were applied on a **custom shortest-path graph**.

4. Methodology

4.1 Reinforcement Learning (RL)

Classical RL uses **Tabular Q-learning**, where Q-values for each state–action pair are computed using:

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

Key components:

- **Learning rate (α)**
- **Discount factor (γ)**
- **Exploration probability (ε)** – ε-greedy strategy
- **Q-table** storing values for (state, action)

4.2 Deep Reinforcement Learning (DRL)

Deep Q-Network (DQN) replaces the Q-table with a **neural network**:

$$Q(s,a) = \text{NN}(s)$$

Key concepts used:

- Neural network with dense layers
- Experience replay buffer
- Target network
- ε-greedy exploration
- MSE loss + Adam optimizer

DQN is suitable for environments where states are continuous or very large.

---

5. Implementation Overview

5.1 MountainCar (RL)

- The environment has **continuous** state space: position & velocity.
- Discretized into bins to make Q-learning applicable.
- Agent receives **-1 reward per step**, goal reward when reaching the top.
- After sufficient episodes, the agent learns momentum-based strategy.

5.2 MountainCar (DQN)

- No discretization needed.
- Neural network approximates Q-values.
- Training done using Replay Memory + Target Network.
- Agent learns similar strategy but more consistently than tabular RL.

5.3 Roulette (RL)

- Implemented as **multi-armed bandit**.
- ε-greedy strategy to explore different arms.
- Agent gradually identifies the arm with highest reward probability.
- Simple but effective example of RL.

(optional note: DRL is not required for bandit since state is 1-dimensional)

---

6. Shortest Path Problem

The user provides:

- Nodes
- Edges
- Goal state

6.1 RL (Tabular Q-Learning)

- Reward matrix R is created from the graph:
  - `100` reward for reaching goal
  - `0` for valid transitions
  - `-1` or negative for invalid transitions
- Q-table is updated until convergence
- Path extracted using greedy policy:
- `route = [start, argmax(Q[s]), ... , goal]`

6.2 DRL (DQN for Graph)

- Each state represented as **one-hot vector**.
- Neural network predicts Q-values for every action.
- Invalid actions masked manually.
- DQN successfully learns shortest path after training.

---

7. Results & Observations

7.1 MountainCar

Method	Performance	Observation
Q-learning	Slow convergence, sensitive to discretization	Needs many episodes due to continuous state space
DQN	Faster & more stable	Learns smoother trajectory

7.2 Roulette

Method	Performance	Observation
Epsilon-greedy Bandit	Correctly identifies best arm	Simple and effective for stateless problems

---

7.3 Shortest Path: RL vs DRL Comparison

Metric	RL (Q-table)	DRL (DQN)
Avg steps	Lower for small graphs	Better for large graphs
Training time	Very low	Higher
Convergence	Fast for small state space	More stable for complex graphs
Scalability	Poor	Excellent
Memory usage	Minimal	High

Interpretation:

- For **small graphs**, tabular RL is enough.
- For **large or complex graphs**, DRL handles generalization better.

---

8. Advantages & Limitations

RL

✓ Simple
✓ Fast for small state spaces
✗ Cannot handle continuous states
✗ Weak scalability

DRL

✓ Works for high-dimensional states
✓ More accurate
✓ Better generalization
✗ Requires GPU/time
✗ Hyperparameter sensitive

---

9. Conclusion

In this assignment, both classical Reinforcement Learning and Deep Reinforcement Learning methods were successfully implemented on different environments using OpenAI Gym.

Key conclusions:

1. **DQN performs better** than tabular Q-learning in continuous/large state spaces.
2. For simple tasks like **Roulette**, classical RL is sufficient.
3. For **shortest-path problems**, RL works well for small graphs, but DRL scales better.
4. MountainCar demonstrates the importance of reward shaping, exploration, and function approximation.

Overall, DRL proved to be more powerful but computationally heavier, whereas classical RL remained simple and effective for discrete problems.

---

10. References

1. Sutton & Barto, *Reinforcement Learning: An Introduction*
2. OpenAI Gym Documentation