Assignment 2 Deep Q-Network (DQN) Implementation and Evaluation

Course: CSE 4/546 – Reinforcement Learning
Instructor: Prof. Alina Vereshchaka
Name : Nandini Soni, Harshitha Gowdar Mallikarjuna Prasanna
UBIT : nsoni, hgowdarm

Assignment 2 Part II Report

1. Introduction

In this part of the assignment, we implemented the Deep Q-Network (DQN) algorithm completely from scratch following the structure proposed by DeepMind.
The goal was to apply the same implementation on three different environments and observe how well the agent learns in each case.
The three environments used are:

1. WarehouseDeterministicEnv : custom grid-world environment from Assignment 1
2. CartPole-v1 : a classical control problem
3. LunarLander-v3 : a more complex Box2D physics-based task

2. Algorithm

Rather than employing a Q-table, the DQN agent approximates Q(s, a) using a neural network. The agent learns from its past experience by storing interactions with the environment in a replay buffer. The DQN agent also uses a second target network to stabilize the updates during learning.

Key features of this mode of learning are:

- Experience replay buffer (memorizes previous state, action, reward transitions)
- Target network is updated every C steps
- ε-greedy exploration-exploitation policy
- Mini-batch gradient descent with loss function of mean squared error
- ReLU activation function and Adam optimizer

3. Benefits

a. Experience Replay
Experience replay allows the agent to learn from a random mix of old and new experiences.
This removes correlation between consecutive states and makes training smoother.
A larger buffer stores more diverse data, but it also increases memory usage.
In my runs, using 30 000 – 50 000 samples worked well.

b. Target Network
The target network prevents the model from chasing a moving target by keeping a delayed copy of the weights.
Updating it every 1000 steps gave noticeably more stable rewards, especially in the Lunar Lander task.

c. Q-Function Representation
Instead of a table, the Q function is represented as q(s, a; w), where w are the weights of the neural network.
This makes it possible to handle large or continuous state spaces.

4. Environment Details

Environment	State Space	Action Space	Goal	Reward Design
WarehouseDeterministicEnv	72 (one-hot)	6	Pick and drop item	−1 step, +25 pickup, +100 drop, penalty for collisions
CartPole-v1	4 (float)	2 (left or right)	Balance the pole	+1 each step until pole falls or cart out of bounds
LunarLander-v3	8 (float)	4	Land safely on pad	Positive for smooth landing, negative for crash or fuel loss

5. Results and Observations

5.1 Grid-World (WarehouseDeterministicEnv)

Training ran for 400 episodes.
The agent first explored random moves but gradually learned to pick the package and deliver it without hitting shelves.
After training, it achieved a total reward ≈ 113 consistently.

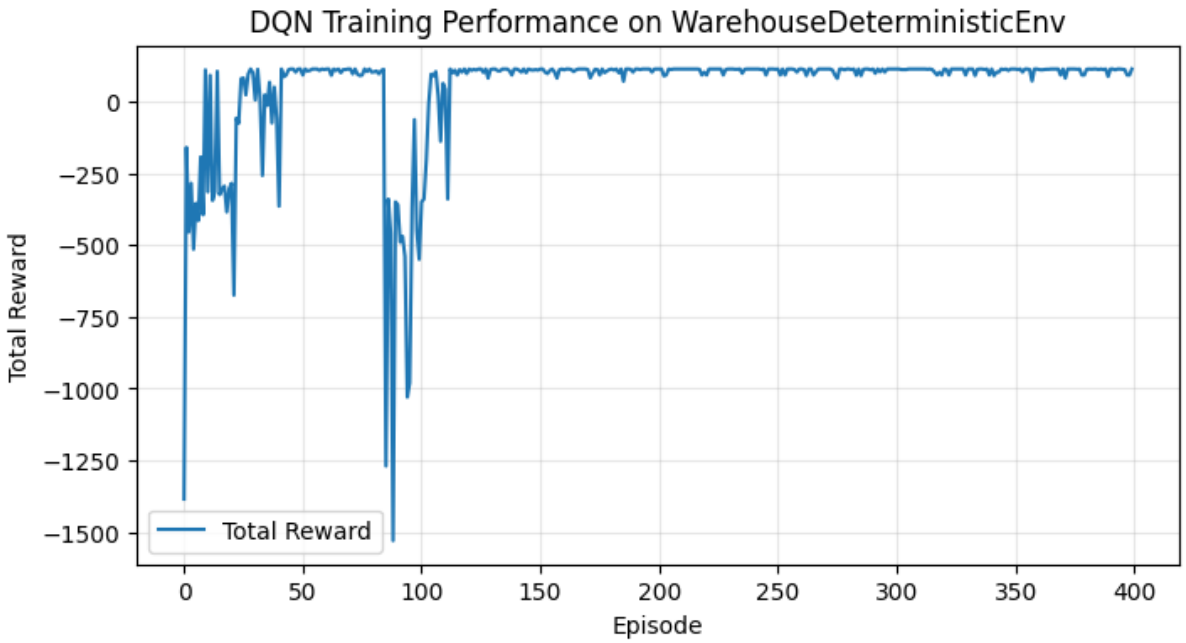
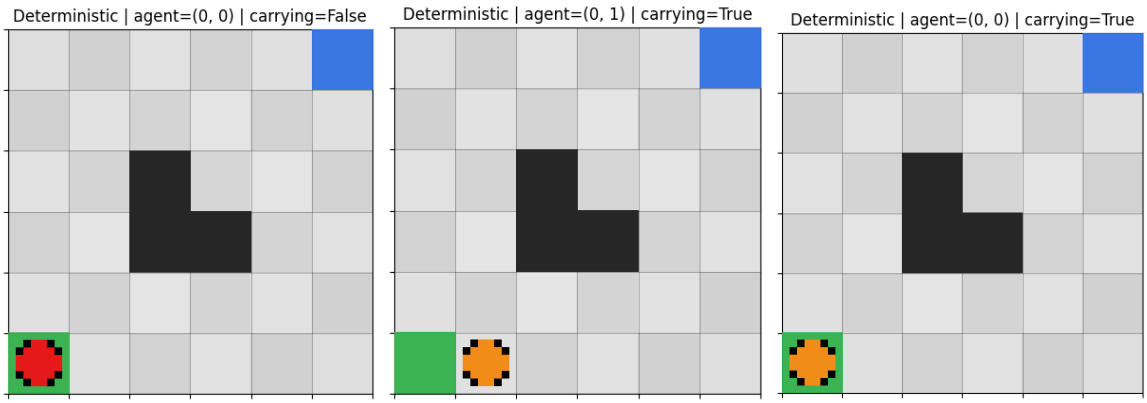
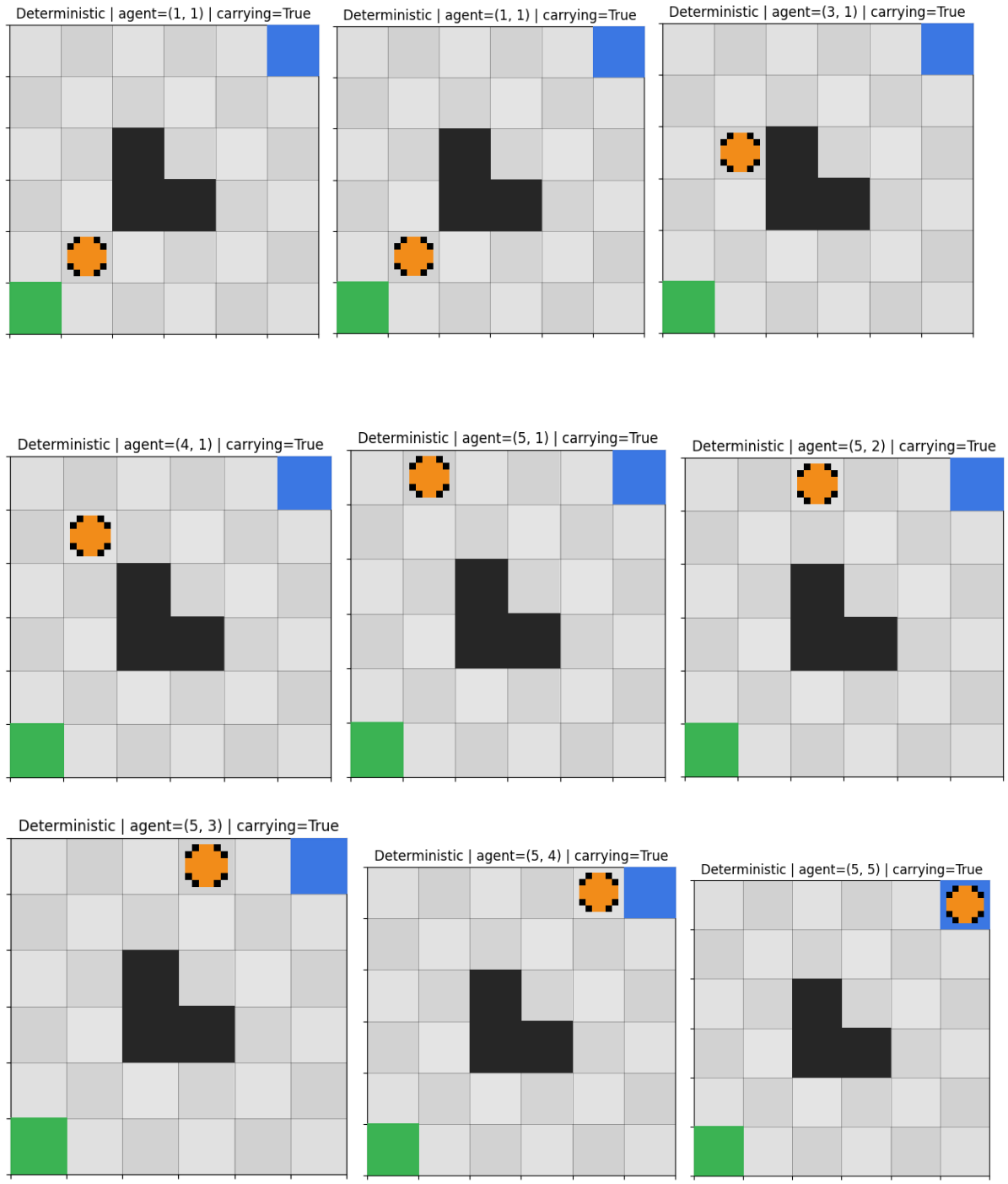


Figure 1 – Training Performance on WarehouseDeterministicEnv

During greedy evaluation, the agent successfully performed:

1. Move to pickup point (0, 0)
2. Pick the item and change to orange (color indicates carrying)
3. Navigate to drop-off (5, 5) and deliver successfully





Total reward in evaluation: 113.0

Figure 2 – Agent rendered sequence in evaluation

5.2 CartPole-v1

Training was run for 800 episodes using the DQN agent.
At the beginning, rewards were very low because the agent was exploring random actions.

After about episode 300, the performance started improving steadily as the agent learned to balance the pole for longer periods.
By episode 700 onwards, the agent was able to keep the pole upright for the full episode duration, achieving a maximum reward of 500.
This means the environment was successfully solved according to the DQN benchmark.

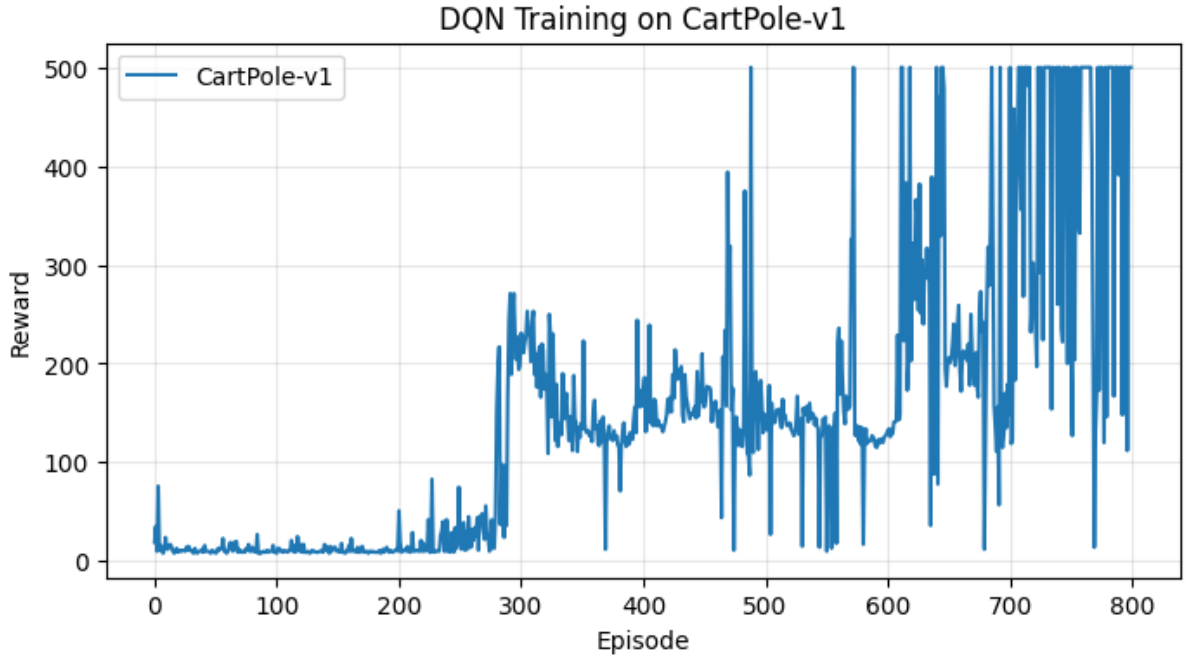


Figure 3 – DQN Training on CartPole-v1

During greedy evaluation ($\varepsilon = 0$), the agent performed perfectly in multiple trials, balancing the pole for the entire episode.
The final model demonstrates stable control and smooth recovery from small disturbances, confirming that the learned policy generalizes well.

Greedy Eval 1: Reward = 500.0

Greedy Eval 2: Reward = 500.0

Greedy Eval 3: Reward = 500.0

Greedy Eval 4: Reward = 500.0

Greedy Eval 5: Reward = 500.0

Average Greedy Reward (CartPole): 500.0

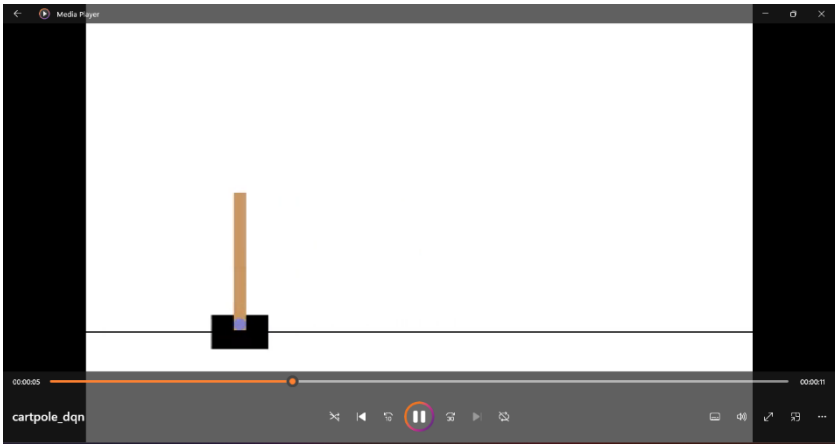


Figure 4 – CartPole DQN Evaluation (video frame)

5.3 LunarLander-v3

This task is more complex and uses continuous physics.
Training was done for 700 episodes.
The agent learned slowly in the beginning, with heavy reward fluctuations up to episode 200.
After about episode 500, it started performing controlled landings and reached an average reward of ≈ 200 points, which meets the solved criteria.

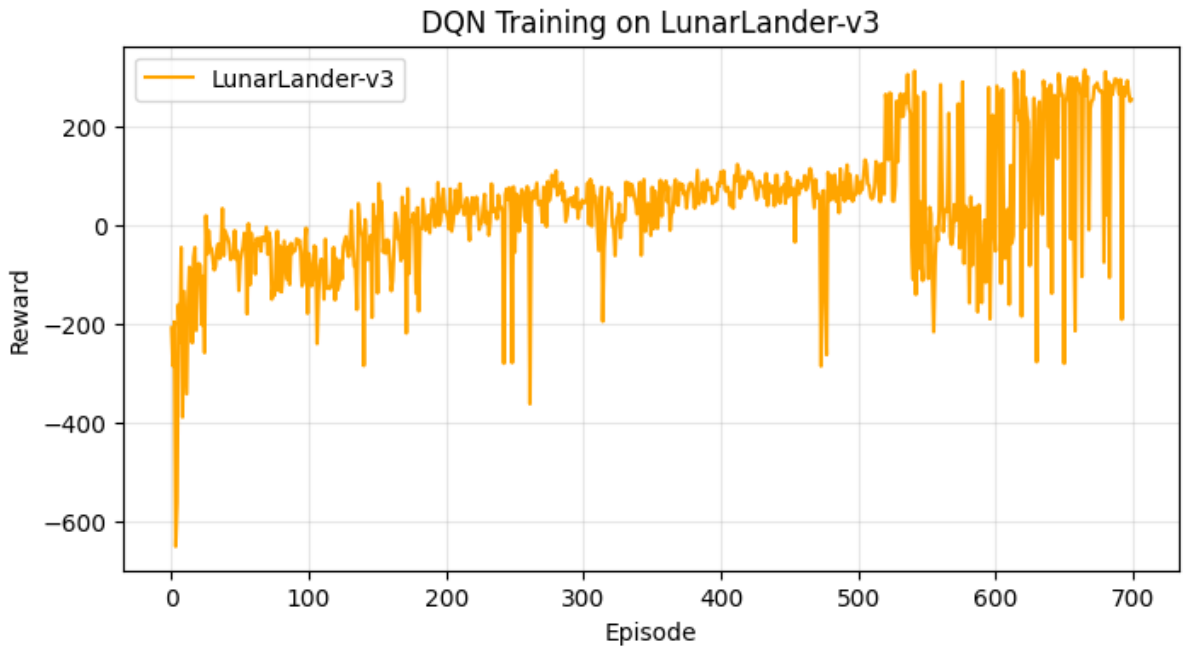


Figure 5 – DQN Training on LunarLander-v3

During greedy evaluation, the lander made soft landings on the pad in most trials.
The recorded video shows proper control of descent and stability.

Figure 6 – Lunar Lander Greedy Evaluation (video frame)

6. Evaluation and Comparison

Environment	Episodes Trained	Avg Reward (Last 50)	Performance Comment
WarehouseDeterministicEnv	400	113	Solved pickup-drop task
CartPole-v1	800	500	Solved: agent balanced the pole perfectly
LunarLander-v3	700	202	Solved : stable landing

7. Conclusion

The DQN algorithm was successfully implemented and tested on three environments of different complexity levels.
Experience replay and the target network played an important role in stabilizing

learning.
In simpler environments like the grid-world and CartPole, the agent learned clear strategies.
For Lunar Lander, despite the complex dynamics, the same network and parameters achieved good performance.

Overall, this experiment shows that a single DQN architecture can adapt to multiple RL tasks when trained with proper hyperparameters.

8. References

1. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). *Human-level control through deep reinforcement learning.* Nature, 518(7540), 529-533.
   Original DeepMind paper that introduced the DQN algorithm.

2. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013).
   Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602.
   Early DQN version demonstrating replay buffer and target network concepts.

3. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016).
   Continuous control with deep reinforcement learning. arXiv:1509.02971.
   Used as a conceptual reference for applying DQN-style methods to continuous spaces.

4. Hasselt, H. V., Guez, A., & Silver, D. (2016).
   Deep Reinforcement Learning with Double Q-learning. AAAI 2016.
   Explains how Double DQN improves stability by reducing overestimation bias.

5. Farama Foundation (2024).
   Gymnasium Documentation. https://gymnasium.farama.org
   Official reference for environments such as CartPole-v1, LunarLander-v3, and ALE/Breakout-v5.

6. PyTorch Documentation (2024).
   torch.nn and torch.optim API Reference. https://pytorch.org/docs/stable/
   Reference for network layers, activation functions, and optimization methods used in the DQN implementation.

7. Sutton, R. S., & Barto, A. G. (2018).
   Reinforcement Learning: An Introduction (2nd ed.). MIT Press.
   Standard textbook covering value-function approximation, policy evaluation, and temporal-difference learning.

8. OpenAI Spinning Up (2018).
   Introduction to Deep RL Algorithms. https://spinningup.openai.com/en/latest/
   Practical reference for replay buffer management and ε-greedy exploration.