CSCI/ROBO 7000/4830: Homework #3
Stabilizing Deep Q-Networks

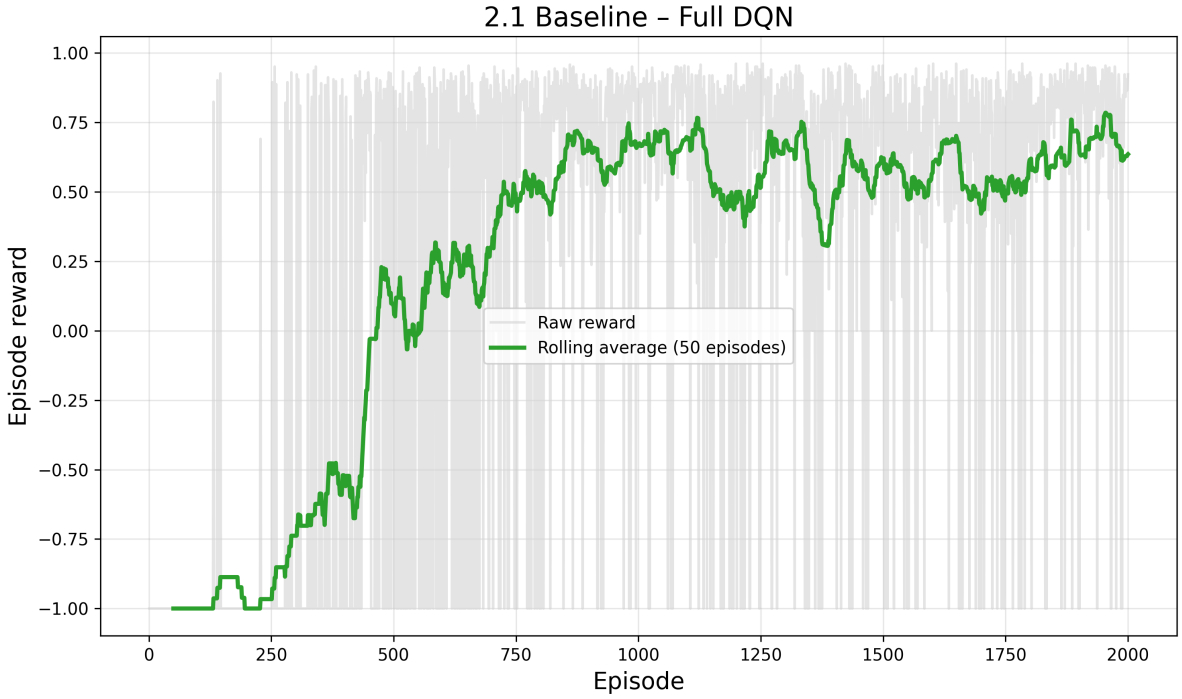
Stephen Gillet

Fall 2025

Part 2: Analysis and Reasoning

2.1 Baseline (Full DQN) [10 Points]

Run your full DQN implementation from Part 1 and submit your plot. Your agent should show a clear learning trend.



The agent successfully learns the task using a DQN network with replay buffer and target network. It shows that upward learning trend and settles around 0.6 reward after about 750 episodes.

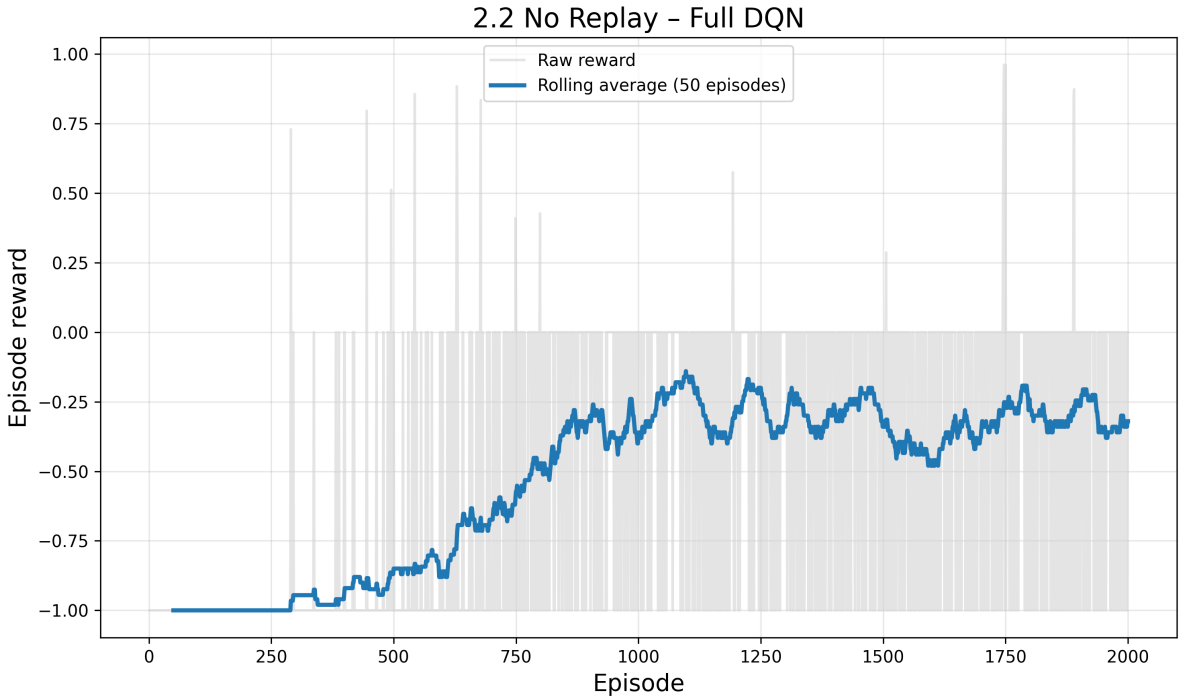
Briefly explain the high-level role of your ReplayBuffer and your target_network in achieving this stable learning.

The replay buffer collects batches of unassociated data to train on so that the model doesn't learn any of those temporal trends or dependencies.

Because the environment is always changing it would be more difficult to try to learn every step since what we are learning is constantly changing, so we pause it with the target network so that we can learn a bit then take in some changes little by little.

2.2 Experiment B – No Experience Replay [10 Points]

Modify your code to remove the replay buffer... train online... Submit your new plot. Compare its stability and performance to the baseline.



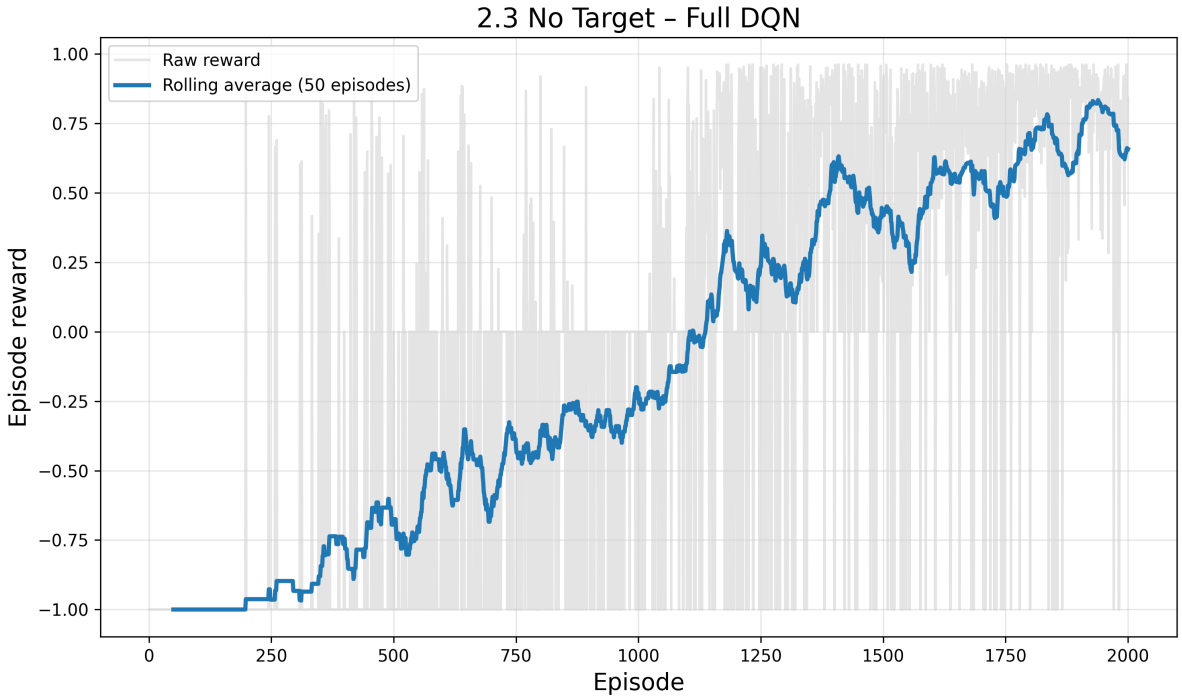
The performance is much worse with the model settling after about 1000 episodes on a policy that yields on average -0.33 rewards.

Using concepts from Lecture 13, explain why training on sequential, correlated samples is a problem...

Neural networks assume training data is i.i.d. Consecutive transitions in an episode are highly correlated (similar states, same policy). This violates the i.i.d. assumption, causing large correlated gradient updates that push the network parameters in harmful directions ("catastrophic steps"). Experience replay decorrelates samples and enables reuse of past experiences, dramatically improving stability and sample efficiency.

2.3 Experiment C – No Target Network [10 Points]

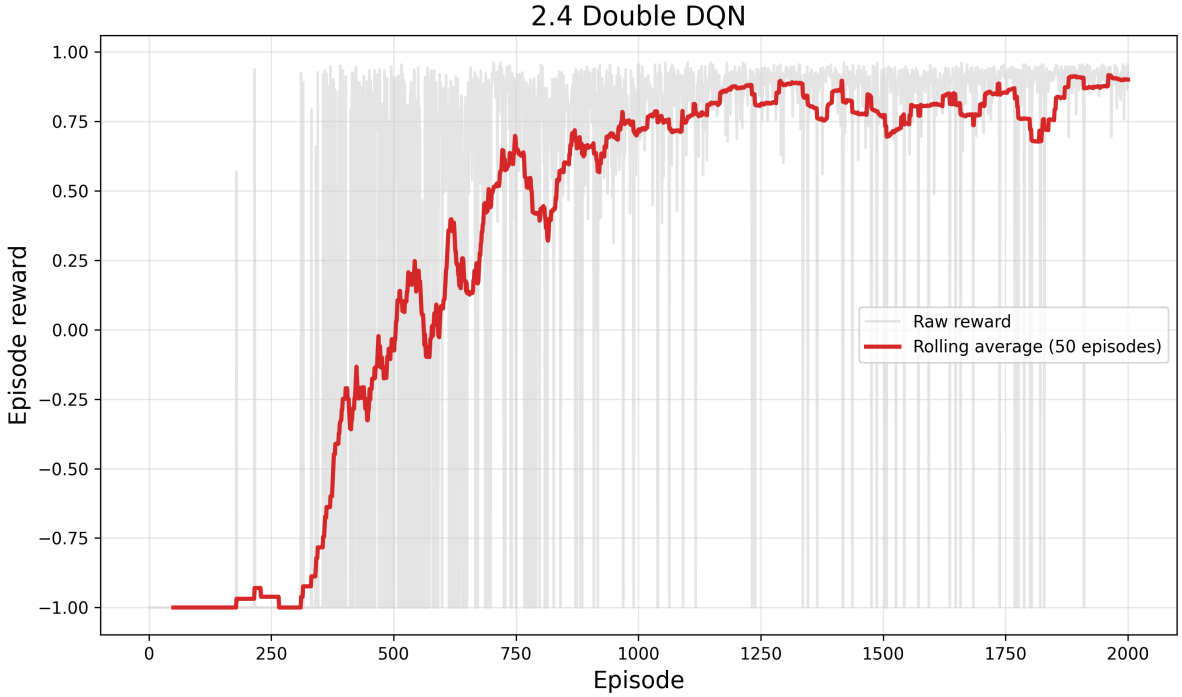
Restore the replay buffer. Now modify your update rule to remove the target network (equivalent to setting C=1)... Submit your third plot... Explain why this instability occurs by referencing the "moving target" problem.



The network does improve and eventually converge on about the same place that regular DQN did in this example. However, the training is noticebly slower and less stable not converging until the very end of training.

2.4 Experiment D – Fixing Overestimation with Double DQN [15 Points]

Implement Double DQN by modifying your TD target calculation... Submit your fourth plot (DDQN). Compare this plot to your baseline DQN... Explain why this new update rule helps prevent the overestimation of Q-values.



Double DQN learns about as fast but reaches a higher reward in the end reaching a policy that yields 0.9 average reward after about 1000 episodes.

Standard DQN suffers from **maximization bias**: the $\max_a Q(s', a)$ operator always uses the maximum action value, even if it comes from noise or overestimation. Over many updates this bias accumulates. Double DQN decouples action selection (done on the online network) from action evaluation (done on the target network), yielding $Y = r + \gamma Q(s', \arg\max_a Q(s', a; w); w^-)$. This dramatically reduces overestimation, leading to more accurate value estimates, faster convergence, and better final policy quality.

2.5 Conceptual Check [5 Points]

In HW2, you used a Q-table for "Cliff Walking." Explain in detail why a Q-table is a completely impractical solution for MiniGrid-Dynamic-Obstacles-v0. How does your QNetwork from Part 1 solve this "scaling" problem?

In 'Cliff Walking' a Q-table was possible because we had a stable, known environment so we only needed to know the Q values for each action in each of the squares. In MiniGrid we would have to calculate the Q-value of each action in each grid square for every possible permutation of the grid which involves the locations of the obstacles and position/orientation of the agent. The observation space after flattening the 7×7×3 partial view is 147-dimensional with integer values 0–10, yielding more than 10^{147} possible distinct states — far larger than the number of atoms in the observable universe. Even ignoring the moving obstacles, a tabular approach would require impossible memory and exploration time.

The **QNetwork** (a two-layer MLP with 64→64 hidden units) is a function approximator $Q(s, a; w)$ that generalizes across unseen states via learned feature representations and weight sharing. It requires only ∼10k parameters regardless of state-space size and can generalize from observed states to similar unseen ones, solving the curse of dimensionality that makes tabular methods completely infeasible.