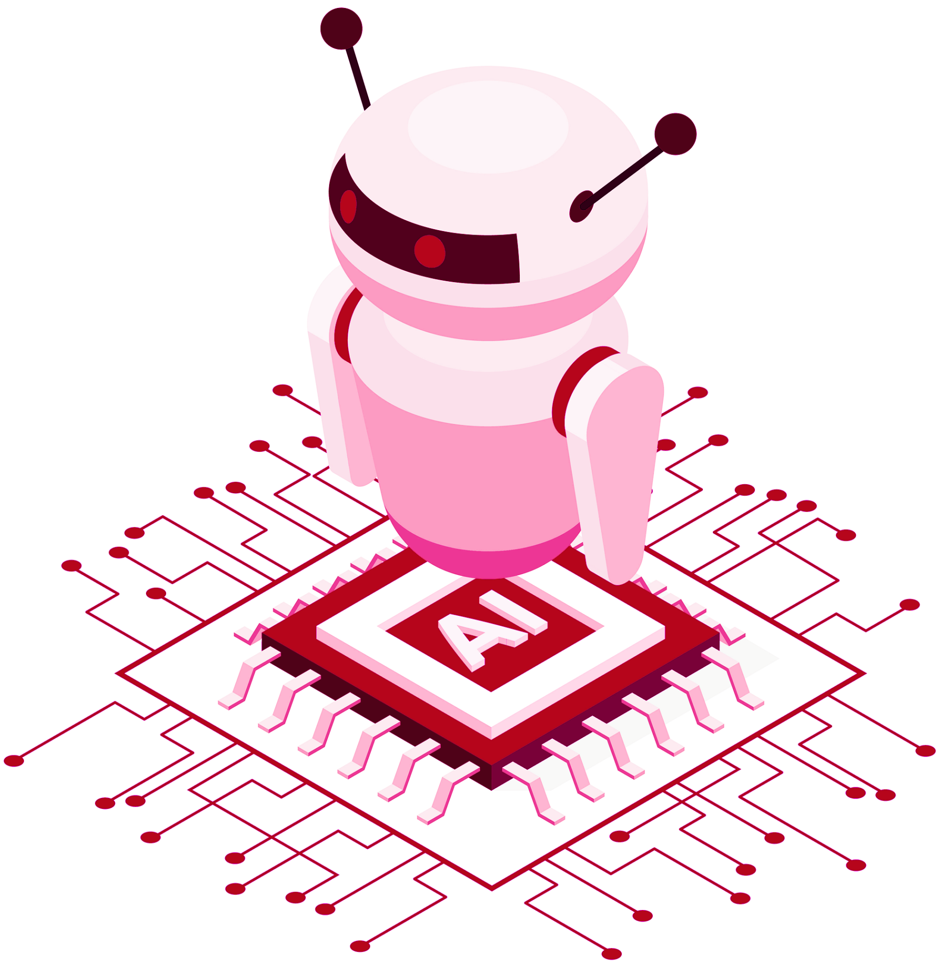

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 2:

Value-Based Methods

By:

[Asemaneh Nafe]

[400105285]

Spring 2025 w

Contents

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	Points
Task 1: Epsilon Greedy & N-step Sarsa/Q-learning	40
Jupyter Notebook	25
Analysis and Deduction	15
Task 2: DQN vs. DDQN	50
Jupyter Notebook	30
Analysis and Deduction	20
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing your report in Latex	10

Notes:

- Include well-commented code and relevant plots in your notebook.
- Clearly present all comparisons and analyses in your report.
- Ensure reproducibility by specifying all dependencies and configurations.

1 Epsilon Greedy

1.1 Epsilon 0.1 initially has a high regret rate but decreases quickly. Why is that? [2.5-points]

High regret at the beginning: Since $\epsilon = 0.1$ means only 10% of the time an action is chosen randomly, the policy initially lacks sufficient exploration and may follow suboptimal choices. Regret decreases quickly: Once the agent starts learning the optimal actions, it mostly exploits them (since 90% of the time, it picks the best-known action). Hence, regret rapidly drops as the policy stabilizes.

1.2 Both epsilon 0.1 and 0.5 show jumps. What is the reason for this? [2.5-points]

These jumps in performance happen when the agent discovers a significantly better path in the environment. At $\epsilon = 0.5$, exploration is high, leading to frequent changes in the learned policy, causing fluctuations (jumps). At $\epsilon = 0.1$, early exploration leads to minor jumps, but the policy stabilizes faster.

1.3 Epsilon 0.9 changes linearly. Why? [2.5-points]

With $\epsilon = 0.9$, the agent explores 90% of the time, meaning it mostly selects random actions. As a result, learning is slow and steady, causing a linear trend rather than sharp jumps (since no single discovery has a big impact).

1.4 Compare the policy for epsilon values 0.1 and 0.9. How do they differ, and why do they look different? [2.5-points]

$\epsilon = 0.1$: Mostly exploits the best-known actions. More structured policy (smoother path avoiding the cliff). Converges faster to an optimal solution. $\epsilon = 0.9$: Mostly explores, leading to an unstructured or highly varied policy. Takes much longer to converge since it prioritizes random actions. Might not settle on the best policy within a reasonable number of episodes.

1.5 In the epsilon decay section, analyze the optimal policy for the row adjacent to the cliff (the lowest row). Then, compare the different learned policies and their corresponding rewards. [2.5-points]

In the epsilon decay setting, fast decay helps the agent learn to avoid the cliff earlier, leading to a safer policy in the lowest row. Since it exploits safer paths sooner, it falls into the cliff less often and gets higher rewards.

On the other hand, slow decay keeps the agent exploring longer, which means more risky moves near the cliff. This results in more falls, a worse policy in the lowest row, and overall lower rewards.

2 N-step Sarsa and N-step Q-learning

2.1 What is the difference between Q-learning and sarsa? [2.5-points]

Q-learning is off-policy, meaning it updates Q-values using the greedy action (max future reward), regardless of the agent's actual behavior. This makes it more aggressive and prone to risk but often leads to faster learning. SARSA is on-policy, meaning it updates Q-values based on the action the agent actually takes, considering its exploration strategy (epsilon-greedy). This results in safer and more stable learning, especially in environments like CliffWalking, where risky moves can be highly punishing.

2.2 Compare how different values of n affect each algorithm's performance separately. [2.5-points]

For both N-step SARSA and N-step Q-learning, the value of n affects how far into the future the agent looks when updating Q-values.

When $n = 1$, both algorithms behave like their standard forms (1-step SARSA and 1-step Q-learning), leading to more frequent updates but with high variance. As n increases, the updates consider longer reward sequences, leading to smoother learning but slower adaptation. In CliffWalking, smaller n values tend to be safer, while larger n values can make the agent more optimistic, sometimes stepping into the cliff due to delayed updates.

2.3 Is a Higher or Lower n Always Better? Explain the advantages and disadvantages of both low and high n values. [2.5-points]

Neither is always better—it depends on the trade-offs:

Low n: Faster updates, adapts quickly to changes. More stable in high-risk environments like CliffWalking. Higher variance, leading to noisier learning. High n: Uses more information for each update, leading to smoother learning. Can converge faster in simple environments but may be too optimistic in risky environments. Slower to react to immediate changes and more prone to suboptimal moves near hazards (like the cliff).

3 DQN vs. DDQN

3.1 Which algorithm performs better and why? [3-points]

DDQN generally outperforms DQN because it reduces the overestimation bias in Q-values, leading to more reliable decision-making. DQN tends to overestimate action values, causing instability, especially in environments with high variance. DDQN addresses this by using a separate target network to select actions, improving learning efficiency. This results in better long-term performance and more stable training.

3.2 Which algorithm has a tighter upper and lower bound for rewards. [2-points]

DDQN has tighter reward bounds because it reduces Q-value overestimation, leading to more controlled value updates. DQN's overestimation can cause reward spikes and instability, making its upper and lower reward bounds more spread out. Since DDQN refines action selection using a separate target network, its learned policy is more stable. This makes its reward distribution more consistent across episodes.

3.3 Based on your previous answer, can we conclude that this algorithm exhibits greater stability in learning? Explain your reasoning. [2-points]

Yes, because stability comes from reducing Q-value overestimation, which prevents large fluctuations in learning. A more stable learning process means the agent can generalize better without drastic performance drops. Since DDQN refines its value estimates, it converges more smoothly and avoids oscillations seen in DQN. The controlled reward fluctuations suggest more reliable decision-making.

3.4 What are the general issues with DQN? [2-points]

DQN suffers from overestimation bias, sample inefficiency, and unstable learning due to correlated updates. It uses experience replay, but without proper handling, it still struggles to generalize well. It also has difficulty with catastrophic forgetting, where learned policies can deteriorate over time. Moreover, DQN lacks a proper mechanism for uncertainty estimation, making it vulnerable to deceptive reward structures.

3.5 How can some of these issues be mitigated? (You may refer to external sources such as research papers and blog posts be sure to cite them properly.) [3-points]

Double DQN (DDQN) reduces Q-value overestimation by decoupling action selection and evaluation, improving learning stability (Van Hasselt et al., 2016)[6]. Dueling DQN separates state and action values, allowing the agent to prioritize valuable states more effectively (Wang et al., 2016)[7]. Prioritized Experience Replay (PER) enhances sample efficiency by giving more weight to important transitions, leading to better learning (Schaul et al., 2016)[8]. Distributional RL models reward distributions instead of mean values, which improves decision-making under uncertainty (Bellemare et al., 2017)[9]. These methods collectively enhance learning efficiency and stability.

3.6 Based on the plotted values in the notebook, can the main purpose of DDQN be observed in the results? [2-points]

Yes, DDQN shows more stable Q-values, its advantage is evident in smoother learning curves and reduced reward fluctuations. Compared to DQN, it exhibit fewer spikes in rewards, indicating better policy learning. If implemented correctly, DDQN should show improved convergence and more consistent performance over time.

3.7 The DDQN paper states that different environments influence the algorithm in various ways. Explain these characteristics (e.g., complexity, dynamics of the environment) and their impact on DDQN's performance. Then, compare them to the CartPole environment. Does CartPole exhibit these characteristics or not? [4-points]

DDQN performs better in environments with high variance, delayed rewards, or stochastic transitions because it stabilizes Q-value estimation. In more complex tasks like Atari games, DDQN significantly improves performance. However, in simpler environments like CartPole, where reward structures are less deceptive, DDQN's impact is less pronounced. CartPole does not suffer from severe overestimation issues, making DQN's weaknesses less visible.

3.8 How do you think DQN can be further improved? (This question is for your own analysis, but you may refer to external sources such as research papers and blog posts be sure to cite them properly.) [2-points]

DQN can be improved by incorporating multi-step returns to capture long-term dependencies better. Hybrid model-based RL can integrate learned dynamics to reduce sample inefficiency. Transformer-based RL has potential for handling longer-horizon dependencies. Exploration strategies like intrinsic motivation can help escape local optima. Research in offline RL may also improve DQN's ability to learn from past experiences more efficiently.

References

[1] R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2nd Edition, 2020. Available: http://incompleteideas.net/book/the-book-2nd.html.

[2] Gymnasium Documentation. Available: https://gymnasium.farama.org/

[3] Grokking Deep Reinforcement Learning. Available: https://www.manning.com/books/grokking-deep-reinforcement-learning

[4] Deep Reinforcement Learning with Double Q-learning. Available: https://arxiv.org/abs/1509.06461

[5] Cover image designed by freepik

[6] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," Available: https://arxiv.org/abs/1509.06461.

[7] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," Available: https://arxiv.org/abs/1511.06581.

[8] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," Available: https://arxiv.org/abs/1511.05952.

[9] M. G. Bellemare, W. Dabney, and R. Munos, "A Distributional Perspective on Reinforcement Learning," Available: https://arxiv.org/abs/1707.06887.