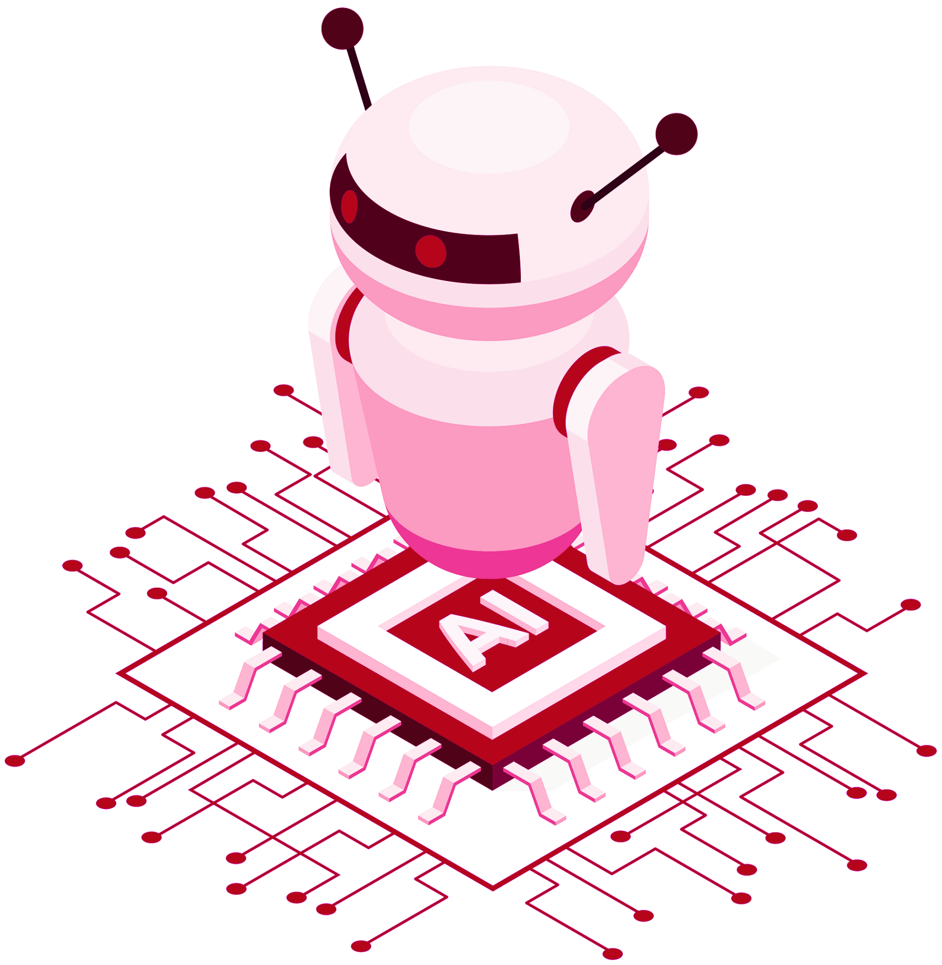


Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 2:

Value-Based Methods

By:

[Full Name]

[Student Number]

Spring 2025 w

Contents

1 Epsilon Greedy 1

1.1 Epsilon 0.1 initially has a high regret rate but decreases quickly. Why is that? [2.5-points] 1

1.2 Both epsilon 0.1 and 0.5 show jumps. What is the reason for this? [2.5-points] 1

1.3 Epsilon 0.9 changes linearly. Why? [2.5-points] 1

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] 1

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] 2

2 N-step Sarsa and N-step Q-learning 3

2.1 What is the difference between Q-learning and sarsa? [2.5-points] 3

2.2 Compare how different values of n affect each algorithm's performance separately. [2.5-points] 4

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] 4

3 DQN vs. DDQN 6

3.1 Which algorithm performs better and why? [3-points] 6

3.2 Which algorithm has a tighter upper and lower bound for rewards? [2-points] 6

3.3 Based on your previous answer, can we conclude that this algorithm exhibits greater stability in learning? Explain your reasoning. [2-points] 6

3.4 What are the general issues with DQN? [2-points] 6

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] 6

3.6 Based on the plotted values in the notebook, can the main purpose of DDQN be observed in the results? [2-points] 7

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] 7

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] 7

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 start is because $\epsilon = 0.1$ and agent may get stuck in sob-optimal choices. But since 90% of the time it's exploiting learned values, it quickly improves and starts making better choices. As algorithm updates the policy based on actual actions taken, the agent learns fast because of its low ϵ, reducing regret quickly.

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

The jumps in regret happen because epsilon-greedy randomly explores, meaning sometimes it chooses bad actions (regret spikes) and other times finds better ones (regret drops).

- $\epsilon = 0.5$ (more exploration) $\rightarrow$ More frequent jumps since it's trying different actions a lot.
- $\epsilon = 0.1$ (less exploration) $\rightarrow$ Fewer jumps, but they still happen when it stumbles on a better or worse action.
- Noise in rewards can also make regret fluctuate.
- Updates in action values can suddenly change what the agent thinks is best, causing jumps.
- Basically, the randomness in exploration + learning makes regret go up and down sometimes.

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

For $\epsilon = 0.9$, the agent explores 90% of the time, meaning it mostly selects random actions. Since exploration is dominant, the regret decreases at a steady rate as the agent gradually finds better actions. This leads to a near-linear decline in regret because the learning process is more uniform, with fewer sudden jumps compared to lower ϵ values.

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]

For $\epsilon = 0.1$, the agent mostly exploits the best-known action (90% of the time), leading to a stable policy that quickly converges but may get stuck in local optima.

For $\epsilon = 0.9$, the agent explores heavily (90% of the time), meaning its policy is more random and takes longer to converge. It avoids getting stuck early but learns more slowly.

The key difference is that $\epsilon = 0.1$ favors exploitation (leading to faster stability), while $\epsilon = 0.9$ favors exploration (leading to more randomness and a slower but thorough search for the best action).

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, the optimal policy for the row adjacent to the cliff (the lowest row) prioritizes safe moves over risky shortcuts. Initially, high ϵ values encourage exploration, leading to frequent falls and lower rewards. As ϵ decays, the policy stabilizes, avoiding the cliff while efficiently reaching the goal.

Comparing learned policies:

- High ϵ (constant or slow decay): More exploratory, often taking risky moves, leading to lower average rewards.

- Low ϵ (fast decay): Quickly settles on a near-optimal path, reducing falls and achieving higher rewards.

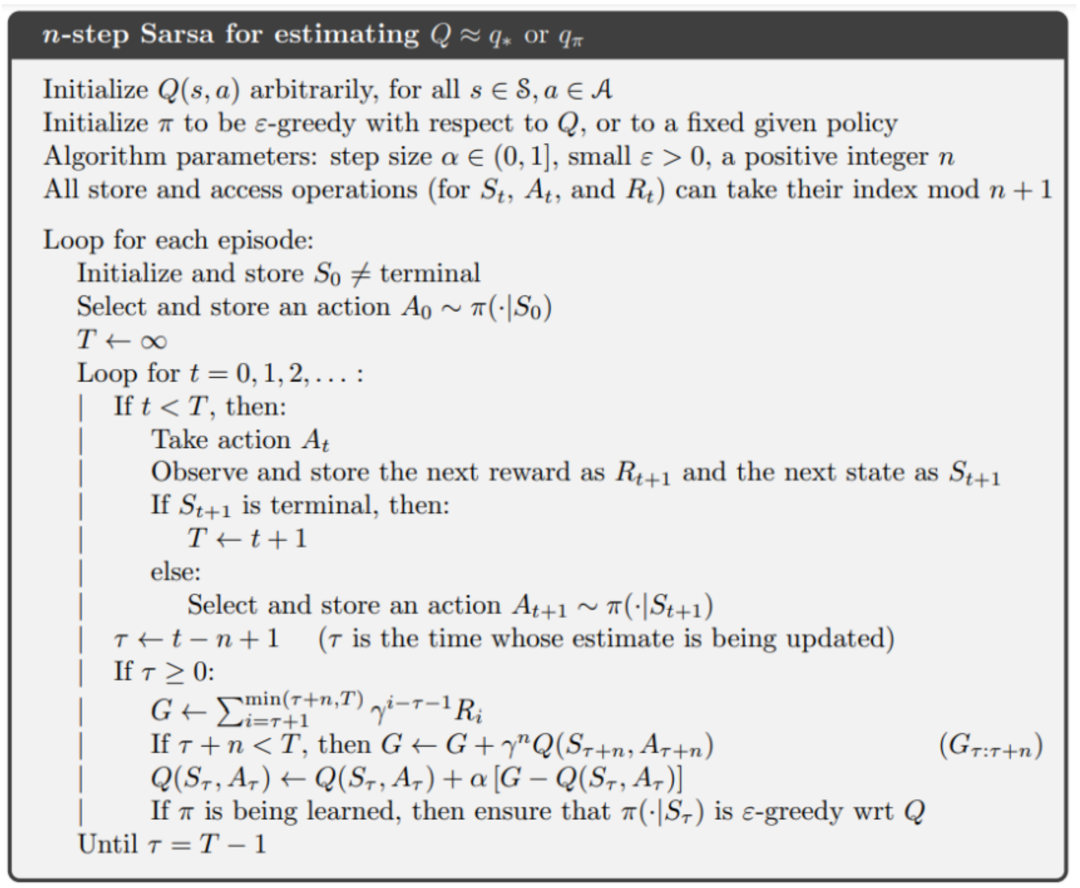
- Moderate decay: Balances exploration and exploitation, resulting in a gradual increase in reward over time.

The key difference is how each policy **trades off exploration and risk**, impacting both stability and reward efficiency.

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

In this section, there were many problems in the main structure! action[τ + n] was sampled from policy in sarsa function, but it was also sampled in the learning function, which makes the algorithm different from the main one! Additionally, epsilon was not passed to the algorithms, which makes the exploration rate of evaluation part different from our main policy and become like an off-policy algorithm.

Because of these problems, I had to change the structure based on the following algorithm from Prof. sutton book (Figure 1).



```
$n$-step Sarsa for estimating $Q \approx q_*$ or $q_\pi$

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be $\varepsilon$-greedy with respect to $Q$, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer $n$
All store and access operations (for $S_t$, $A_t$, and $R_t$) can take their index mod $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \ldots$ :
    |   If $t < T$, then:
    |       Take action $A_t$
    |       Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
    |       If $S_{t+1}$ is terminal, then:
    |           $T \leftarrow t+1$
    |       else:
    |           Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    |   $\tau \leftarrow t - n + 1$   ($\tau$ is the time whose estimate is being updated)
    |   If $\tau \geq 0$:
    |       $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    |       If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$        $(G_{\tau:\tau+n})$
    |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
    |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
    Until $\tau = T - 1$
```

Figure 1: n-step on-policy SARSA algorithm.

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

Q-learning and SARSA are both reinforcement learning algorithms for estimating action-value functions, but they differ in how they update Q-values:

Q-learning (Off-policy): Uses the maximum possible future reward regardless of the actual next action taken. Updates using the greedy action (max future Q-value), meaning it learns the best policy regardless of the agent's actual behavior. More aggressive and converges faster but can be unstable.

Update rule:

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

SARSA (On-policy): Uses the next action chosen by the current policy to update Q-values. Updates using the agent's actual next action, meaning it follows its current policy. More cautious and stable in risky environments.

Update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

Key Difference: Q-learning is more exploratory and optimal, while SARSA is safer and adapts better to stochastic environments.

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

In both n-step SARSA and n-step Q-learning, the value of n controls how far into the future the updates consider, affecting learning speed and stability.

n-step SARSA: Small n (e.g., $n = 1$) behaves like standard SARSA, updating cautiously and adapting well to stochastic environments. Large n smooths updates over multiple steps, improving stability but slowing reaction to sudden changes.

n-step Q-learning: Small n (e.g., $n = 1$) makes updates aggressively based on the max future Q-value, leading to fast learning but higher variance. Large n averages over multiple steps, reducing variance and improving learning in complex environments.

Generally, small n values lead to faster but noisier learning, while large n values provide smoother updates and better long-term stability. 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]

There is no universally "better" n; the choice depends on the task and environment.

Low n (e.g., $n = 1$):

Advantages: Faster updates, allowing quick adaptation to changes.Less memory usage and lower computational cost. More suitable for highly dynamic or stochastic environments.

Disadvantages: More variance in updates, leading to unstable learning. Short-sighted decisions may slow convergence to optimal policies.

High n (e.g., large n):

Advantages: Smoother updates, reducing variance and improving stability. Captures long-term rewards better, leading to improved policy quality.

Disadvantages: Slower adaptation to sudden changes. Higher memory and computation requirements. More sensitive to reward sparsity, requiring careful tuning.

For **n-step SARSA**, a lower n ensures safer learning in stochastic environments, while a higher n smooths updates. For **n-step Q-learning**, lower n makes it more greedy and aggressive, while higher n helps stabilize learning in complex environments.

3 DQN vs. DDQN

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

DDQN generally performs better than DQN because it reduces overestimation bias in action-value estimates. DQN tends to overestimate Q-values, leading to suboptimal learning, while DDQN mitigates this by using separate networks for action selection and evaluation, leading to more stable and accurate learning.

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

DDQN has a tighter upper and lower bound for rewards compared to DQN. This is due to its reduced overestimation of Q-values, leading to more consistent and reliable learning outcomes.

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

Yes, DDQN exhibits greater stability in learning because its reduced overestimation bias leads to smoother convergence and less variance in learning updates. This results in more predictable improvements in policy compared to DQN.

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

DQN suffers from several issues, including:

- Overestimation bias in Q-values, leading to suboptimal policy updates.
- Instability due to correlated updates when training the Q-network.
- Sample inefficiency, requiring a large number of training steps to converge.

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]

Some methods to mitigate these issues include:

- **Double DQN (DDQN)**: Reduces overestimation by using a separate network for action selection and evaluation [?].
- **Dueling DQN**: Introduces separate streams for state-value and advantage estimation, improving learning efficiency [?].
- **Prioritized Experience Replay (PER)**: Improves sample efficiency by prioritizing more significant experiences during training [?].

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

Yes, the results show that DDQN achieves more stable and consistent learning compared to DQN. The Q-values exhibit less fluctuation, and reward convergence is smoother, indicating that DDQN effectively reduces overestimation bias.

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]

Complex environments with high-dimensional state spaces and stochastic transitions tend to amplify the overestimation bias in DQN, making DDQN more beneficial. DDQN performs well in environments with frequent decision-making requirements, where accurate Q-value estimation is crucial.

The CartPole environment is relatively simple, with a low-dimensional state space and deterministic transitions. As a result, the benefits of DDQN over DQN might be less pronounced compared to more complex environments like Atari games. However, even in CartPole, DDQN can provide more stable learning by mitigating Q-value overestimation.

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 further improved by:

- **Using Distributional RL**: Instead of predicting only the expected Q-value, distributional RL predicts a distribution over possible Q-values, improving robustness [?].
- **Integrating Meta-Learning**: Adaptive learning rate strategies and meta-learning techniques can help adjust exploration-exploitation dynamically [?].
- **Combining with Model-Based RL**: Hybridizing DQN with model-based approaches can improve sample efficiency and decision-making speed [?].

References

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