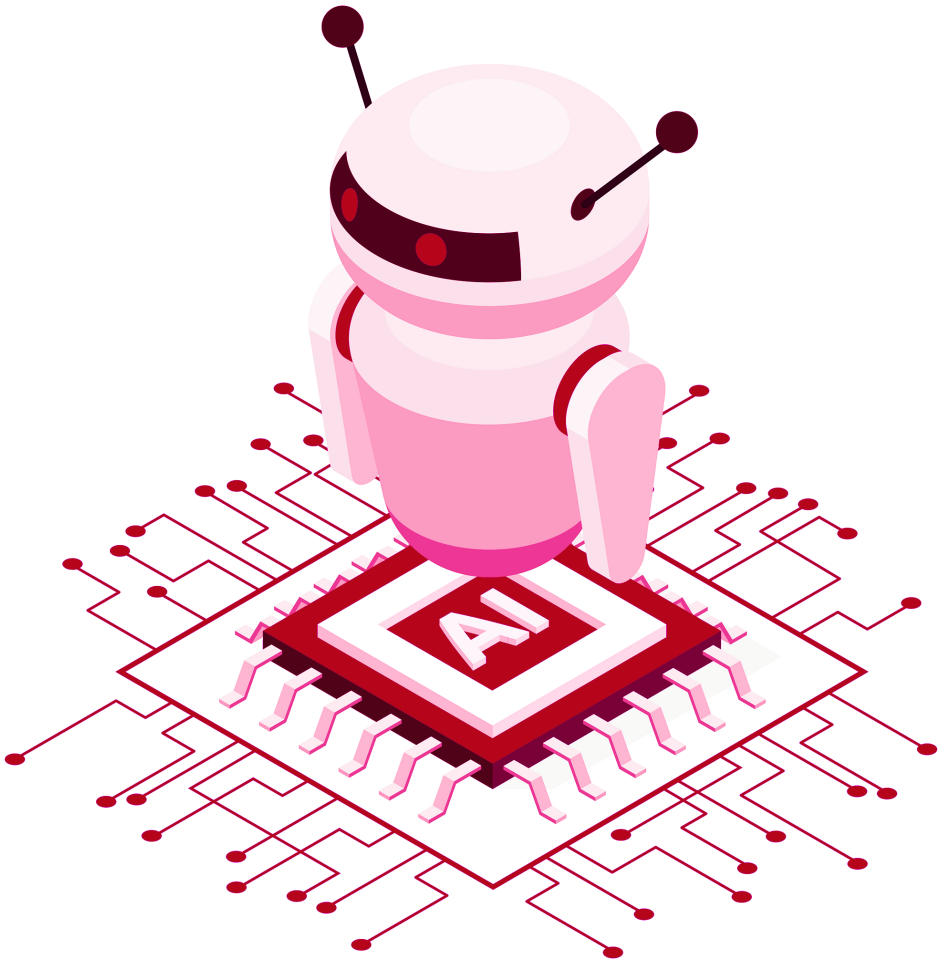
„lipsum



Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 2:

Value-Based Methods

By:

Amir Kooshan Fattah Hesari

401102191

RIML

Deep RL Course

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]

The high regret at the beginning is mainly due to the exploration phase. With an ϵ of 0.1, the algorithm randomly selects non-optimal actions 10% of the time. Early on, when it hasnt yet accurately estimated which action is best, these random choices incur higher regret compared to consistently choosing the optimal action. However, as the algorithm gathers more data, its estimates improve, and it increasingly exploits the best-known option (90% of the time). This shift rapidly reduces the regret rate since most actions chosen are near-optimal

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

the jumps in the regret rate for both $\epsilon = 0.1$ and 0.5 in this environment are primarily due to the interplay between the stochastic nature of exploratory actions and the sensitive dynamics of the environment. Early in training, exploratory moves can cause sudden, large drops in performance (high regret), but as the agent learns the task, these events become less common, leading to a decline in regret over time

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

With such a high exploration rate, the agents performance tends to be uniformly poor. It rarely follows any learned, optimal strategy, so on average, the reward per time step remains roughly constant and far below the optimal reward. Because the gap between the optimal performance and the agent's performance remains nearly the same at each step, the cumulative regretthe sum of these differencestends to increase in a nearly linear fashion over time.

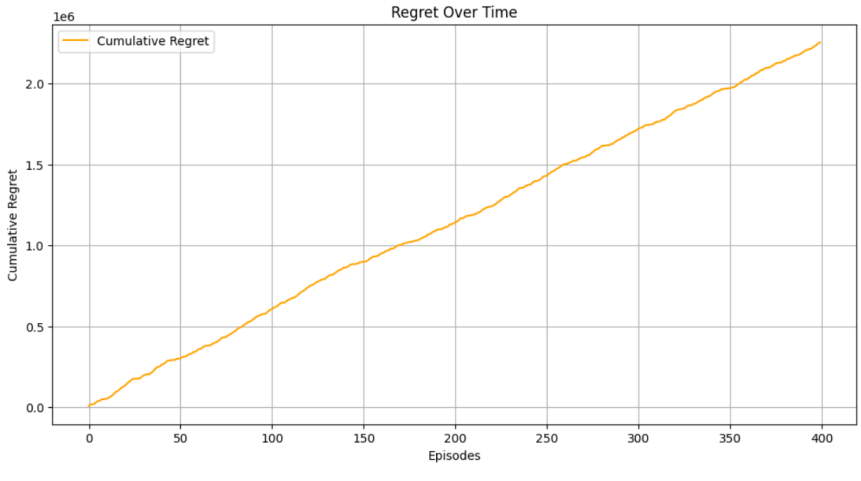


Figure 1: The linear nature of the Regret over time for $\epsilon = 0.9$

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]

- with $\epsilon = 0.1$:
  - High Exploitation: The agent exploits its current best estimate 90% of the time. This means that once it starts to recognize which actions yield high rewards, it mostly sticks to those actions.
  - Rapid Convergence: Because the agent rarely deviates from what it believes is the best action, it quickly locks in on a near-optimal policy. As a result, the cumulative regret tends to be lower since most actions are close to optimal after initial exploration.
  - Low Variability: The policy appears smoother in the plots because there are fewer random actions causing sudden drops in performance.
- with $\epsilon = 0.9$:
  - High Exploration: The agent chooses random actions 90% of the time. This heavy exploration means that even if it identifies a good action, it rarely exploits that knowledge.
  - Stochastic Behavior: The policy appears to "jump up and down" because the majority of the actions are random. This causes significant fluctuations in performance from one step to the next.
  - Linear Regret: Since the agent consistently takes suboptimal actions due to random exploration, the shortfall (regret) accumulates roughly at a constant rate per time step. This constant loss adds up linearly over time.

In summary, the lower ϵ value leads to quicker exploitation of the best actions and hence smoother, near-optimal performance with lower regret. The higher ϵ value, while useful for thorough exploration, introduces significant stochasticity, causing the policy to fluctuate and regret to grow linearly as the agent frequently takes random actions.

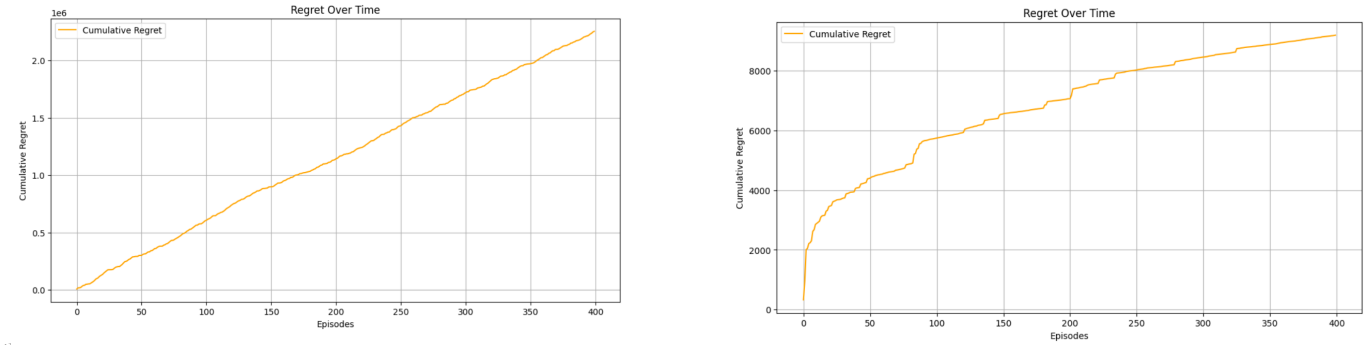


Figure 2: Comparison of the cumulative Regret of $\epsilon = 0.1$ and $\epsilon = 0.9$, it can be seen that the regret of $\epsilon = 0.9$ algorithm (on the left) is much more (order of 10^6) than the regret of $\epsilon = 0.1$ algorithm (which is of the order 10^3)

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]

Because it is the row near the cliff , we shouldn't be careless and try to choose our actions mindfully. So using a **fast-decay** epsilon policy is the best choice here to **avoid useless epxloring and falling of the cliff !!**. In the **medium-decay** it takes longer to stabilize, but still ends up with decent returns once exploration decreases. In the **slow-decay**,exploration remains high for many episodes and because there is a lot of exploration , random actions lead to more falls off the cliff, lowering average returns and causing volatility and it settles on a safer but longer route or never fully stops exploring, resulting in lower final rewards

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

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

First of all, both of these algorithms are designed to estimate the optimal policy in an **MDP** and are both value-based methods,but they differ in how they update their Q-values and the way they handle exploration.

- on-policy vs off-policy : Q-learning is an off-policy algorithm, which means that it learns the optimal policy independently of the agents actions. In other words, Q-learning updates its Q-values based on the maximum possible reward from the next state, regardless of the action taken by the agent. Whereas SARSA is an on-policy algorithm, which means it updates its Q-values based on the actual actions taken by the agent.
- Update rule : Q-learning follows this update rule :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

  But the update rule for SARSA is a little bit different :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

- Exploration vs. Exploitation : Q-Learnin emphasizes exploration by learning based on the maximum possible future reward, which may result in an overestimation of Q-values as it assumes optimal actions will always be taken.
  On the other hand, SARSA is more conservative and aligns its learning with the agents current policy.

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

In both cases, meaning both alogirthms, using 1-step Q-learning, reaching the reward only informs the state from which it is reached in the first episode; whereas for 5-step Q-learning, it informs the previous five steps. Then, in the 2nd episode, if any action reaches a state that has been visited, it can access the TD-estimate for that state.

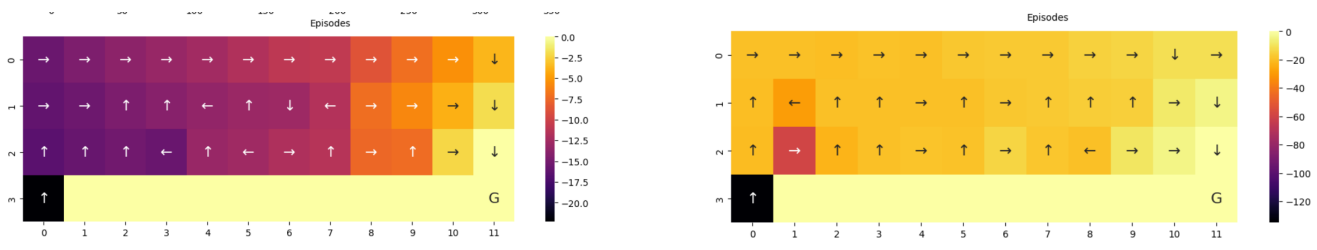


Figure 3: In the 1-step SARSA (on the left) the spread of Q-values takes much more than in the 5-step case (on the right) , just as it was said in the above paragraph.

There are five such states in 5-step Q-learning; and just one in 1-step Q-learning. On all subsequent iterations, there is more chance of encountering a state with a TD estimate and those estimates are better informed. The end result is that the estimates spread throughout the Q-table more quickly.

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]

First of all , if we increase n to be inifinte is the same as Monte-Carlo reinforcement learning, as we would no longer use TD estimates in the update rule. As we have seen, this leads to more variance in the learning.
But when we want to set n there is no theoretically optimal value for this parameter, as it depends on the specific application and the reward function being trained. In practice, values between 4 and 8 tend to produce effective updates because they allow for clear credit assignment to the corresponding actions. This means we can determine whether the 4-8 actions in the lookahead contributed to the final score, as the updates rely on TD estimates.

3 DQN vs. DDQN

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

The DDQN algorithm performs better because it builds upon the DQN architecture by introducing the double Q-learning approach, using two Q-networks to provide more accurate Q-value estimates and address the overestimation bias present in standard DQN. This modification enhances the stability and convergence of the learning process in reinforcement learning scenarios.

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

Both have the somewhat similair upper and lower bound for the average returne , but it looks like that **DDQN** has a tighter bound.

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 **DQN** algorithm has an oversestimation bias that leads to instability but in DDQN the use of second network is for this exact reason, to be able to handle the overestimation problem.

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

Just like said in the last, there are many problems with DQN, some of which are :

- Overestimation bias
- Sample inefficieny where DQN requires millions of environment interactions to learn a good policy where by using a Prioritized Experience Replay (PER) and Distributional RL we can improve sample efficiency.
- DQN training is often unstable due to correlations in training data and the non-stationary nature of the environment.

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]

In Q-learning, there exists what is known as **maximization bias**, which arises due to the update rule:

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

If the Q-values are slightly overestimated, this error can compound over time. As illustrated in the *Sutton and Barto* book, overestimation can lead to instability in learning.

The idea behind **tabular Double Q-learning** is to mitigate this bias by maintaining two Q-value functions, Q_1 and Q_2. Instead of using a single Q-function to select and evaluate actions, the update process follows these steps:

1. An action a' is selected using both Q-networks, e.g., from $Q_1 + Q_2$. 2. A coin flip determines which Q-function to update. 3. If updating Q_1, the update target is computed as:

$$r + \gamma Q_2(s', \arg\max_{a'} Q_1(s', a')).$$

This approach helps reduce overestimation because the second Q-network acts as a correction mechanism when selecting the maximum Q-value.

Deep Double Q-learning extends this idea by leveraging the *target network* in DQN instead of maintaining two separate Q-networks. The update target is given by:

$$r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-),$$

where $Q(s, a; \theta^-)$ represents the **target network**, which is periodically updated to match the current network every C time steps.

This modification ensures that if the main Q-network overestimates the value of state s' while selecting the greedy action, the target network provides a more stable estimate, reducing maximization bias.
references
Reinforcement Learning , Sutton and Barto.
This answer on AI stack exchange
This paper On the Reduction of Variance and Overestimation of Deep Q-Learning

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

Yes it can be seen that the overestimation problem has been addressed in the DDQN implementation, but due to the high sensitivity to the hyperparameters , the results are not optimal.

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]

While DDQN shines in complex, high-dimensional, and dynamic environmentswhere overestimation bias can severely hamper learningits advantages are less critical in simpler environments like CartPole. In CartPole, a standard DQN often performs well because the risk of large overestimations is lower. Thus, although DDQN generally improves stability and performance in challenging settings (as demonstrated in the Atari experiments), CartPole does not exhibit the same characteristics that make DDQNs corrections particularly necessary. This comparison highlights that the choice of algorithm can be highly dependent on the environment's complexity and dynamics.
Picture is from Deep Reinforcement Learning with Double Q-learning paper (2015)

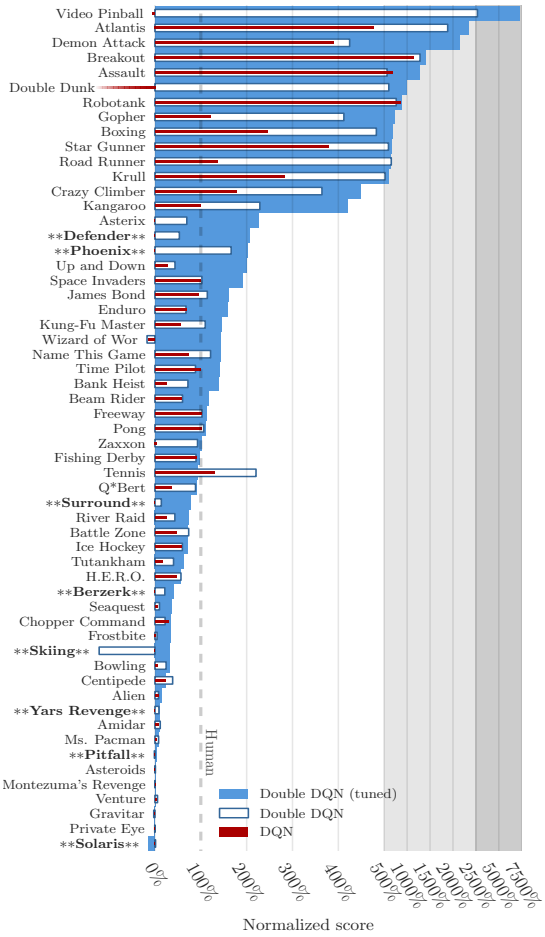


Figure 4: Normalized scores on 57 Atari games, tested for 100 episodes per game with human starts

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]

By using techniques such as **experience replay** which we have implemented in our own code.It solves the autocorrelation problem. We can also implement a target network which turns the DQN algorithm into Double Q-network algorithm. Another way that we can improve the DQN algorithm is by using **Dueling Network Architectures**. The first part of a normal DQN, learning the features with a (convolutional) neural network, is the same as before. But now, instead of calculating the Q-values right away, the dueling network has two separate streams of fully connected layers. One stream estimates the state-value, and the other stream the advantages for each action. Because the dueling architecture learns the values of states, it can determine which states are valuable. This is an advantage because there is no need to learn the effect of each action for each state.
Reference : This blog on tehcniques to improve the performance of a DQN agent

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