Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Solution for Homework 2:

Value-Based Methods

Designed By:

Reza GhaderiZadeh
r.ghaderi2001@gmail.com

Dariush Jamshidian
drjm313@gmail.com

Spring 2025

1 Epsilon Greedy

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

- 0.1: The regret is lower because the agent is less inclined to choose any action other than the best one. This causes the agent to greedily choose its actions, which, in this environment, might be beneficial.
- 0.5: The regret is higher than for epsilon = 0.1 because the agent is more inclined to explore new actions. The regret follows a more linear trend compared to epsilon = 0.1, with a few noticeable jumps, especially toward the end.
- 0.9: The regret is completely linear and higher than in the other cases. There are almost no jumps because, even if the agent finds a better policy, it does not remain committed to it.

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

The regret increases as epsilon increases in the cliff-walking environment (this might not be true for other environments or even for the same environment with a different setup). With a high epsilon, the regret increases linearly because the agent does not exploit the knowledge it has gained. When epsilon is lower, the regret tends to slow down, and noticeable jumps may appear when the agent discovers a better policy.

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

The agent doesn't stick to a policy and chooses the actions almost randomly.

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]

The plot for epsilon = 0.9 is, ironically, well-structured because the agent has experienced many different states and has a strong understanding of the environment. This is not the case for epsilon = 0.1, as the agent, after finding a good policy, does not explore further and instead exploits the found policy, even if it is suboptimal.

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]

With faster epsilon decay:

- The agent quickly reduces its random exploration (epsilon decreases rapidly)

- It finds a "good enough" policy early and exploits it
- This policy is likely the safer path further away from the cliff
- Once committed to this path, it rarely explores alternatives

With lower (slower) epsilon decay:

- The agent maintains higher exploration rates for longer
- It continues to try different paths, including riskier ones near the cliff
- This extended exploration allows it to discover that paths closer to the cliff, while riskier, offer shorter routes to the goal
- The agent can learn the optimal path that balances risk and efficiency

The plots show that with slower decay, the agent is more likely to use the row adjacent to the cliff because:

Through continued exploration, it gains more opportunities to learn the true value of different states, allowing it to better determine which states near the cliff are actually safe to use. As a result, it develops a more refined policy that maximizes reward by taking the shortest safe path.

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 and SARSA are both reinforcement learning algorithms used to solve Markov Decision Processes (MDPs), but they differ in their update mechanisms and exploration-exploitation trade-offs.

Q-Learning (Off-Policy): Q-Learning is an off-policy algorithm, meaning it updates its Q-values using the maximum possible future reward, regardless of the action the agent actually takes. This makes it more exploratory and aggressive, as it always assumes the best possible outcome.

SARSA (On-Policy): SARSA, on the other hand, is an on-policy algorithm, meaning it updates its Q-values based on the action that the agent actually follows under its current policy. This makes SARSA more conservative and safer in environments with risky actions (such as the cliff-walking scenario).

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

For Q-Learning:

- A low n (e.g., 1-step Q-learning) updates the Q-values frequently but with more variance, leading to potentially unstable learning.
- A high n allows for better credit assignment across multiple steps but may slow down convergence due to increased computation and delayed updates.

For SARSA:

- A low n results in faster updates but might cause the agent to act too cautiously, as it focuses more on immediate rewards.
- A high n allows the agent to plan further ahead, leading to a more refined and stable policy, but it can also make the agent less adaptive to environmental changes.

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 optimal n value, it depends on the environment and the learning objectives.
Advantages of Low n:

- Faster updates and quicker adaptation to changes.
- Works well in highly dynamic environments where conditions change frequently.
- Less computationally expensive.

Disadvantages of Low n:

- High variance in updates, which can lead to instability in learning.
- The agent may focus too much on short-term rewards and miss better long-term strategies.

Advantages of High n:

- Smoother and more stable learning due to better credit assignment.
- Helps the agent develop a more strategic and long-term policy.
- Reduces variance in updates, improving convergence.

Disadvantages of High n:

- Slower learning, as updates require more computational steps.
- The agent might overfit to specific trajectories and become less adaptable.

3 DQN vs. DDQN

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

In the CartPole environment, due to its simplicity, it is not easy to determine which algorithm will perform significantly better. Some students may conclude that DQN[1] performs better, while others may find that DDQN is superior.

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

Since DDQN tries to prevent Q overestimation, it should generally have tighter upper and lower bounds in most cases. This was somewhat evident in my training process. However, in a simple environment like CartPole, this can lead to slower convergence compared to DQN. Below, you can see two examples of training graphs:

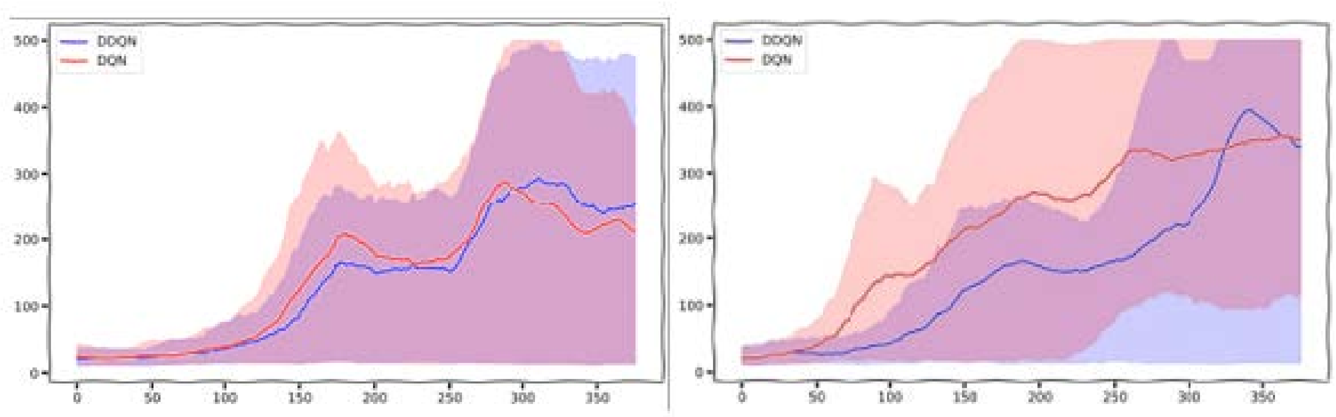


Figure 1: In each graph, DDQN has tighter upper and lower bounds up to episode 250.

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

Yes, this can be interpreted as DDQN being more stable in learning. Due to the use of a target network and its ability to prevent Q-value overestimation, DDQN is more likely to achieve a stable learning process in different environments. However, the simplicity of the CartPole environment may not clearly demonstrate this advantage.

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

Q-value overestimation: The DQN algorithm, introduced in the 2013 paper, does not support a target network. However, even if it did, since $Q(s', a')$ is entirely estimated by the target network, it would still suffer from overestimation.

Instability in learning: In the original DQN, the absence of a target network leads to instability in the learning process.

Exploration challenges: Every algorithm requires exploration. DQN uses the epsilon-greedy policy, but in some cases, this policy can cause instability or even prevent the algorithm from learning.

Sample inefficiency: DQN is highly sample inefficient, making it struggle in complex environments like Atari. This inefficiency can lead to learning a suboptimal policy while also requiring a long training time.

Hyperparameter sensitivity: Due to the issues mentioned, DQN is highly sensitive to hyperparameters, making tuning crucial for its performance.

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]

DDQN addresses several issues that DQN faces. One major issue with DQN is the overestimation of Q-values, which can cause instability in learning. DDQN helps resolve this by using a target network, which is a common technique to stabilize learning in reinforcement learning algorithms. This addition improves learning stability in DDQN, especially in environments with more complex state-action spaces. While DQN could also benefit from a target network, DDQN explicitly separates the action selection and action evaluation processes, reducing the overestimation of Q-values.

DDQN works by using a policy network for selecting actions and a separate target network for evaluating them. In contrast, DQN uses the same network for both, which leads to Q-value overestimation due to maximization bias. By decoupling action selection and evaluation, DDQN reduces overestimation, allowing for more stable and accurate learning, especially when actions have similar values.[2]

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

Due to the simplicity of the environment, the issue is less likely to be observed, but it is still somewhat noticeable. The main point is that since both algorithms can solve the CartPole environment, the Q-values produced by DQN may not necessarily be considered overestimated. This issue should be examined in more complex problems with higher-dimensional state spaces.

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]

- Atari environments are a primary challenge for DDQN as they use images as states, meaning they have high-dimensional state spaces, which is a major issue in reinforcement learning.
- These environments typically have a large action space, making action selection more complex.
- Game progression is structured in levels, where each level becomes more difficult. This requires the algorithm to continuously learn and explore to adapt to new challenges.
- High dynamism: Even within a single level, the environment changes frequently, forcing the algorithm to constantly adapt.
- Many Atari games have sparse rewards, making exploration crucial for learning an optimal policy.
- Additionally, delayed rewards in some games make it harder to assign credit to specific actions, requiring more advanced credit assignment strategies.

In contrast, the CartPole environment:

- Has a low-dimensional state space and action space.
- Does not require continuous exploration after a certain point.
- Does not increase in complexity over time; the difficulty remains the same.
- Has a simple, deterministic dynamic, making it much easier to solve compared to Atari games.

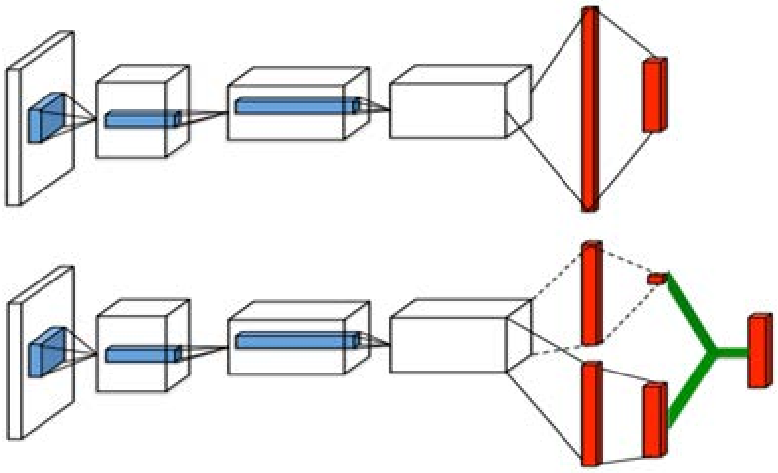
3.8 How do you think DQN can be further improved? (This question is for your own analysis, but you may refer to external sources such as research papers and blog posts be sure to cite them properly.) [2-points]

- Use of Target Network.
- Use of soft replacement.
- Delayed update of the target network relative to the main network (all three of these are implemented in the DDQN notebook) [3]].

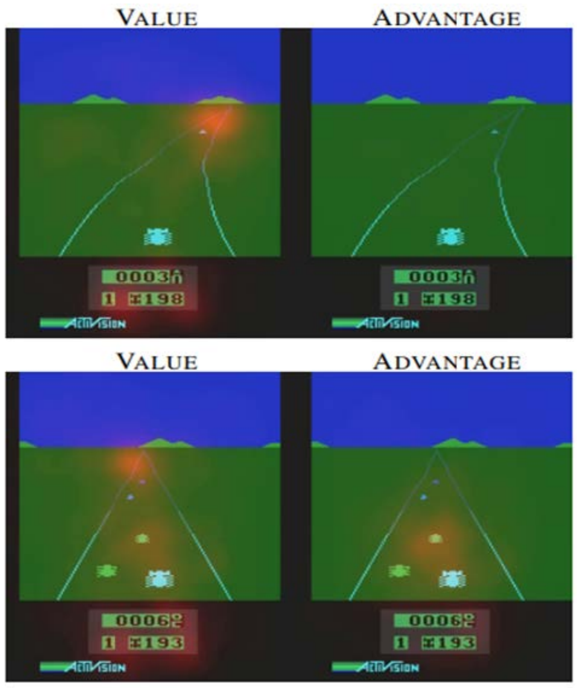
Optimal Use of Data Stored in the Replay Buffer

- Use of prioritized experience replay buffer: This type of buffer assigns a sampling probability to each data point based on its Temporal Difference (TD) error (higher TD error leads to a higher sampling probability) and samples based on these probabilities. This ensures that data the network has not been effectively trained on is selected for training with higher probability, ultimately leading to improved sample efficiency in the algorithm [4].

Use of Dual Q Networks



In the figure above, at the top, you can see a regular Q Network. The input, which comes from a convolutional network, is fed into it, and $Q(s, a)$ is output. However, at the bottom, there is a Dual Q Network. This network has two branches: one estimates $V(s)$ (the state value) and the other estimates $A(s, a)$ (the advantage of action a in state s). Then, these two outputs are combined by the green branch to form $Q(s, a)$. This approach helps to understand whether a state is valuable or not, without the need to perform or analyze the effect of the action in that state.



The figure shows the saliency maps for value and advantage at two different time steps. In one time step (the leftmost pair of images), we see that the value network stream focuses on the road and, in particular, on the horizon where new cars appear. It also pays attention to the score. On the other hand, the advantage stream does not focus much on the visual input because when no cars are in front, the action choice is practically irrelevant. However, in the second time step (the rightmost pair of images), the advantage stream pays attention as there is a car immediately in front, making its action choice highly relevant.

In reinforcement learning scenarios where multiple actions have similar or redundant effects; the dueling network architecture improves policy evaluation by more efficiently identifying the optimal action. This efficiency comes from its dual design, where the state value function and the action advantage function are estimated separately. With this approach, the architecture can generalize learning across actions and lead to faster identification of the correct action, even when redundant or similar actions are added to the learning problem.

References

[1] V. Mnih et al., "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.

[2] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in Proceedings of the AAAI conference on artificial intelligence, 2016, vol. 30, no. 1

[3] M. Morales, Grokking deep reinforcement learning. Manning, 2020. [4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," arXiv preprint arXiv:1511.05952, 2015.

[4] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in International conference on machine learning, 2016: PMLR, pp. 1995-2003.

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

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

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

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

[9] Cover image designed by freepik