**Reinforcement Learning**

Jean P. Melendez Villanueva

University of Maryland

Data 660 Advanced Topics in Data Science

Professor Ed Herranz

**Reinforcement Learning**

**Introduction**

This assignment introduced the concept of Q-learning, a form of reinforcement learning that empowers agents to make decisions by interacting with an environment. The task simulated a warehouse-like environment represented by 9 labeled locations (L1 through L9). The agent was required to find optimal paths between these locations based on a reward structure that encoded allowable transitions. The learning process involved balancing immediate rewards and long-term gains, which was governed by hyperparameters such as the learning rate (alpha) and the discount factor (gamma).

Through this hands-on implementation in NumPy, we applied core reinforcement learning principles by training the agent via thousands of simulated episodes. Each episode allowed the agent to update its knowledge (Q-values) about which actions yielded the highest cumulative reward. We then used this trained knowledge to extract optimal paths between different points in the environment.

Throughout the assignment, we modified the code to track the number of steps taken by the agent, explored the effect of different hyperparameters, and addressed potential pitfalls such as unreachable states or infinite loops. We also extended the state space by introducing a 10th location (L10), increasing the complexity of the environment and testing the scalability of our solution. These changes helped us understand not just how the Q-learning algorithm works, but also how subtle changes to environment design and reward structure can dramatically influence the learning behavior of an agent.

In doing so, we not only deepened our understanding of how Q-learning operates but also laid the foundation for applying reinforcement learning to more complex real-world challenges such as route optimization, resource allocation, and adaptive decision-making in uncertain environments.

**Q-Learning Code Overview**

The Q-learning implementation used in this assignment simulates a simplified environment consisting of labeled locations (L1 to L9), each mapped to a numerical state. These locations and their connections are defined using a reward matrix, where a value of 1 indicates a valid transition and 0 indicates no direct path. A separate Q-table, initialized with zeros, is used to store the agent’s learned estimates of cumulative reward values for state-action pairs. Over time, this table is updated through repeated training episodes, allowing the agent to determine which actions lead to long-term rewards.

Training is handled by a custom function that performs Q-learning iterations using the Bellman equation. For each episode, the agent starts at a random state, selects a valid action, and updates the Q-value for that action using the observed reward and the estimated future reward from the next state. The algorithm’s behavior is shaped by two key hyperparameters: the learning rate (alpha), which determines how quickly the agent learns from new experiences, and the discount factor (gamma), which controls how much the agent values future rewards. Additionally, to guide the agent toward a goal, the code temporarily increases the reward of the target location, ensuring it is prioritized during training.

After training, the agent uses the learned Q-table to compute the optimal path from a starting location to a goal using a function that selects the highest-valued action at each step. The code was enhanced to track the number of steps taken and later extended to accommodate changes in the environment, such as fixing unreachable paths and adding a new state (L10). These enhancements allowed for a deeper understanding of how environment structure, hyperparameter tuning, and reward design influence the behavior and learning efficiency of a reinforcement learning agent.

**Answers to questions 3, 5, 6, 7, 8 and 9**

**Answer to Question 3:** To evaluate how many steps the agent takes to reach the goal, I enhanced the get\_optimal\_route() function by adding a step counter. This modification allowed the function to return both the computed route and the number of transitions taken. After training the Q-table using the default hyperparameters (gamma = 0.75, alpha = 0.9) over 1000 iterations, I executed the route calculation from L9 to L1. The result was a five-location path: ['L9', 'L8', 'L5', 'L2', 'L1'], completed in four steps. This route is optimal based on the environment’s reward structure and the direct connections between locations.

The agent successfully learned this efficient path through repeated exposure to state transitions and consistent Q-value updates. Since Q-learning prioritizes actions that maximize expected cumulative rewards, the agent converged on the shortest path through high-reward transitions. The inclusion of a large reward at the goal state during training helped reinforce the value of reaching the destination. This outcome demonstrates that the learning process worked as intended and that the chosen hyperparameters and iteration count were appropriate for allowing the agent to generalize the optimal route from L9 to L1.

**Answer to Question 5:** To explore how hyperparameters influence learning behavior, I conducted an experiment using significantly lower values for both the learning rate and discount factor. I set gamma to 0.05 to reduce the agent’s focus on future rewards and alpha to 0.05 to slow its ability to learn from new experiences. After training the Q-table using these values for 1000 iterations, I evaluated the route from L9 to L1. The result was a one-step path: ['L9', 'L1'], which is incorrect based on the environment’s structure, as L9 and L1 are not directly connected.

This outcome highlights the importance of tuning hyperparameters appropriately in reinforcement learning. With such low values, the agent failed to explore the environment adequately and did not sufficiently update its Q-values. As a result, it prioritized an invalid path that appeared to yield a high reward, likely due to uninitialized or misleading values in the early training stages. This experiment demonstrates that when the learning rate is too small and future rewards are heavily discounted, the Q-learning algorithm may converge on suboptimal or even invalid policies, particularly in environments where exploration and sequential decision-making are required.

**Answer to Question 6:** To evaluate whether 1000 training iterations are necessary for the agent to learn the optimal policy, I conducted additional experiments using fewer iterations—specifically 50 and 200. In both cases, I kept the learning rate (alpha = 0.9) and discount factor (gamma = 0.75) unchanged. After training the Q-table with these lower iteration counts, I tested the route from L9 to L1. Surprisingly, the results remained consistent with previous tests, and the agent still returned the optimal route: ['L9', 'L8', 'L5', 'L2', 'L1'], with a total of four steps.

These findings suggest that for small, deterministic environments, fewer training iterations can still yield accurate and efficient learning outcomes. Because the agent is able to repeatedly visit all important state transitions, even with 50 or 200 iterations, it manages to converge on the correct path quickly. This efficiency is further supported by the structure of the environment, which contains relatively few states and predictable connections. While 1000 iterations provide robustness in more complex settings, this test shows that simpler environments allow Q-learning to converge in a fraction of the time.

**Answer to Question 7:** When I first attempted to compute the route from L1 to L9 using the original reward matrix, the function failed to return a result and appeared to enter an infinite loop. Upon investigation, I discovered that although the agent could move from L1 to L2, it was unable to proceed further because the path from L2 to L5 was blocked due to a missing reward entry. Specifically, the reward for the transition from L2 to L5 was set to 0, making it invisible to the agent during training. As a result, the agent had no knowledge of a valid continuation beyond L2 and could not complete the route to L9.

To correct this issue, I modified the reward matrix to enable movement between L2 and L5 by setting rewards[1,4] = 1 and rewards[4,1] = 1, making the connection bidirectional. After retraining the Q-table with these updates, I re-ran the route from L1 to L9. This time, the agent returned a correct and efficient path: ['L1', 'L2', 'L5', 'L8', 'L9'], completed in four steps. This fix demonstrated how critical proper reward structures are to the success of Q-learning. Even if the algorithm itself is sound, incorrect or incomplete reward configurations can severely limit the agent’s ability to learn or complete a task.

**Answer to Question 8:** To test the flexibility and scalability of the Q-learning implementation, I expanded the environment by adding a tenth location, L10. This new state was only connected to L9, both ways, meaning it could only be accessed through or lead to L9. To accommodate this change, I updated the reward matrix from 9×9 to 10×10 and modified the location\_to\_state and state\_to\_location mappings to include L10. These adjustments required no changes to the core Q-learning logic, which confirmed that the model was generalizable enough to handle more complex environments.

After retraining the Q-table, I tested the route from L10 to L1. The agent returned the path ['L10', 'L9', 'L8', 'L5', 'L2', 'L1'], which was completed in five steps. This route is correct, as it moves through L9, the only gateway from L10 to the rest of the environment and then follows the optimal path to L1. The successful outcome demonstrated that the agent was able to incorporate the new state into its learning process without issue. It also showed that the reward structure and routing logic could easily support larger environments as long as transitions were clearly defined.

**Answer to Question 9:** Following the successful addition of L10 to the environment, I further tested the agent’s ability to find optimal paths by computing a route from L10 to L4. Since L10 is only connected to L9, the agent would first need to travel through L9 and then continue toward L4 using the most efficient route. This test was essential in verifying whether the agent could generalize from the new state to destinations that required more complex navigation beyond direct neighbors.

After training the Q-table, the agent returned the path ['L10', 'L9', 'L8', 'L7', 'L4'], which required four steps. This path is both valid and efficient, as it takes advantage of the direct connections from L9 to L8 and then through L7 to reach L4. The agent successfully integrated the new state into its decision-making process, treating L10 as a viable starting point and choosing actions that led to maximum long-term reward. This result confirmed that the Q-learning algorithm can scale and adapt to changes in the environment when rewards and transitions are properly defined.

**Conclusion and Takeaways**

This assignment provided a practical and insightful exploration into how Q-learning functions in a controlled environment. By simulating a navigation task within a simple state-space and reward structure, I was able to observe firsthand how reinforcement learning enables agents to make optimal decisions through trial, error, and reward-based updates. Implementing the algorithm in NumPy from scratch helped reinforce my understanding of core concepts like the Q-table, temporal difference updates, and the critical role of hyperparameters such as learning rate and discount factor.

Through various experiments, ranging from hyperparameter tuning to iteration count adjustments, I observed the agent's behavior change drastically depending on how it was trained. Poor parameter settings led to incomplete or incorrect routes, while well-tuned configurations consistently produced optimal paths. Additionally, the environment’s reward structure proved to be just as important as the algorithm itself. When the reward matrix was incomplete (as seen with the missing L2 to L5 transition), the agent was effectively blind to viable paths. Fixing those issues highlighted the importance of a well-defined environment when applying reinforcement learning.

Finally, expanding the state space by introducing L10 demonstrated the scalability of Q-learning. Without changing the algorithm, the agent was able to integrate a new state into its policy and still produce efficient routes to existing locations. This reinforces the value of Q-learning for real-world applications such as autonomous navigation, logistics optimization, and dynamic decision-making systems. The assignment ultimately provided a hands-on foundation for understanding how reinforcement learning agents learn, adapt, and generalize in evolving environments.

**References**

Mwiti, D. (2022, March 31). *9 Reinforcement Learning Real-Life Applications*. V7 Labs. <https://www.v7labs.com/blog/reinforcement-learning-applications>

Mwiti, D. (2022, March 31). *10 Real-Life Applications of Reinforcement Learning*. Neptune.ai. <https://neptune.ai/blog/reinforcement-learning-applications>

**Appendix**

**Appendix A –** Refer to attached .ipynb code

**Appendix B -** Screenshots

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 1 - Optimal route and step count from L9 to L1

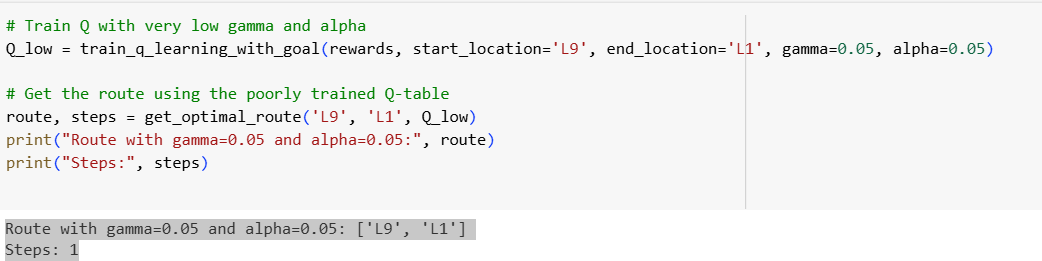


Figure 2 - Route result with gamma=0.05 and alpha=0.05

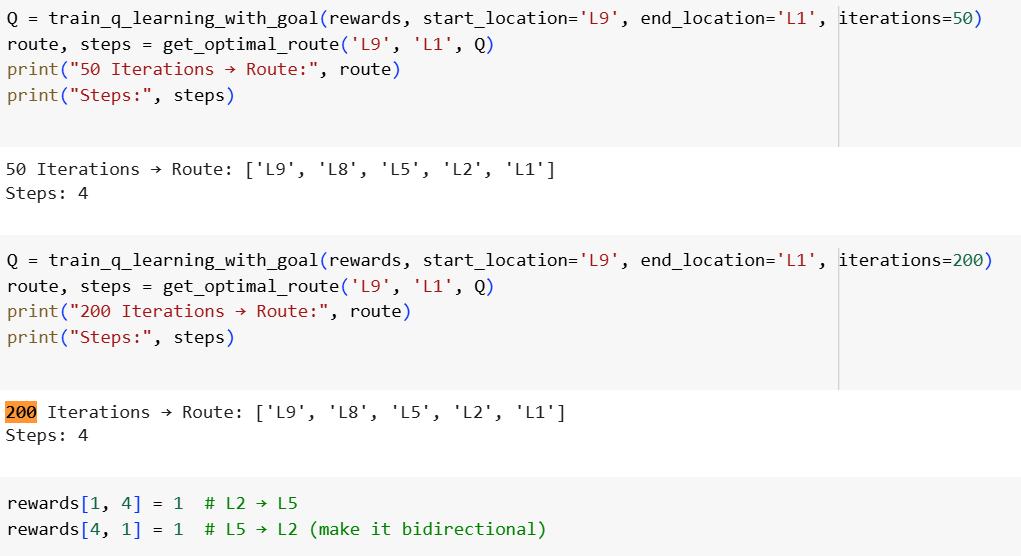


Figure 3 - Outputs of 50 and 200 iteration tests