Teaching an Agent to Navigate a Risky World: A Q-Learning Adventure

Jeevan Hebbal Manjunath

October 3, 2025

Abstract

In this report, we explore how a simple agent can learn complex, intelligent behavior through pure trial and error. We apply the Q-learning algorithm to a classic gridworld problem, a world filled with rewards, penalties, and unpredictable movement. Through a comprehensive analysis of the agent's learning process, hyperparameter tuning, and final strategies in both standard and high-risk environments, we demonstrate its remarkable ability to adapt its behavior in response to potential danger.

---

1 The Challenge: A Risky Gridworld

Imagine a robot trying to find its way through a maze. The floor is slippery, so moving forward might cause it to slide sideways. The maze has a goal with a prize, but also a dangerous trap. This is exactly the challenge we've set for our learning agent.

The world is a 3×4 grid. The agent's mission is to get from the start at (1,1) to the prize at (4,3), which gives a '+1' reward. To make things harder, there's a wall at (2,2), a living penalty of '-0.04' for every move, and the agent's movement is stochastic—it only follows its intended direction 80% of the time.

2 Our Approach: Learning from Experience

How can an agent learn with no initial instructions? We used Q-learning, a powerful reinforcement learning technique.

2.1 The "Cheat Sheet" Analogy

At its heart, Q-learning is like creating a "cheat sheet" (called a **Q-table**) for the agent. This cheat sheet has a score for every possible action in every possible square. A high score says "This is a great move!" while a low score says "Avoid this!"

Initially, the agent knows nothing, so all the scores are zero. But as it explores the world, it constantly updates the cheat sheet based on the rewards and penalties it finds. The formula it uses to update the scores is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{1}$$

This looks complex, but it simply means: "The new score for this move is a blend of the old score and any new information we just learned."

2.2 The Learning Algorithm

The agent's training process is a continuous loop of exploring and updating its cheat sheet. To make sure it doesn't just stick to the first path it finds, we use an ϵ-greedy strategy: most of the time it follows its cheat sheet, but sometimes it tries a random move, just to see what happens. This randomness (exploration) fades over time as the agent becomes more confident.

```
Initialize the Q-table with all zeros for every state-action pair.
Set the learning parameters (alpha, gamma, epsilon).

For a large number of episodes:
    1. Place the agent at the START state.

    2. While the agent has not reached a terminal state (prize or trap):
        a. Decide whether to explore or exploit:
           --- With probability epsilon, choose a random action.
           --- Otherwise, choose the best action from the Q-table for the current state.

        b. Perform the chosen action.

        c. Observe the reward and the new state.

        d. Update the Q-table score for the action just taken using the Q-learning formula.

        e. Move to the new state.

    3. Slightly decrease epsilon to encourage less exploration over time.
```

3 Discussion of Findings

The Q-learning agent successfully learned to navigate the gridworld. Analyzing its learning process and final strategies reveals key insights into how the algorithm operates and adapts.

3.1 Comparing Standard vs. High-Risk Scenarios

To see how the agent adapts to risk, we compared the learned policies from two different experiments: a standard run with a small penalty of -1, and a high-risk run with a massive penalty of -200.

3.1.1 Case 1: Standard Penalty (-1)

In the standard case, the agent learns an efficient and safe policy, shown in Figure 1. The optimal path is to move up the left side and across the top, safely avoiding the column with the penalty. This is a logical and effective strategy.

→	→	→	**+1.0**
↑	WALL	↑	**-1.0**
↑	←	↑	←

Figure 1: The agent's learned strategy for the standard problem (penalty = -1).

3.1.2 Case 2: High-Risk Penalty (-200)

When the penalty is increased to -200, the agent becomes extremely **risk-averse**, as seen in Figure 2. The new policy shows two key changes:

1. **Strong Aversion:** Any state near the '-200' trap now has a policy that points sharply away from it. The policy in state (3,2), for example, changes from '↑' to '←'.

2. **"Hunker Down" Strategy:** In the bottom-right corner (4,1), the agent's policy is to move '↓', causing it to stay put. It has learned that doing nothing is safer than risking a move that might stochastically slip toward danger.

This comparison makes it clear that the agent doesn't just learn a single path; it learns the true value of its actions and adapts its strategy to become more cautious as the environment becomes more dangerous.

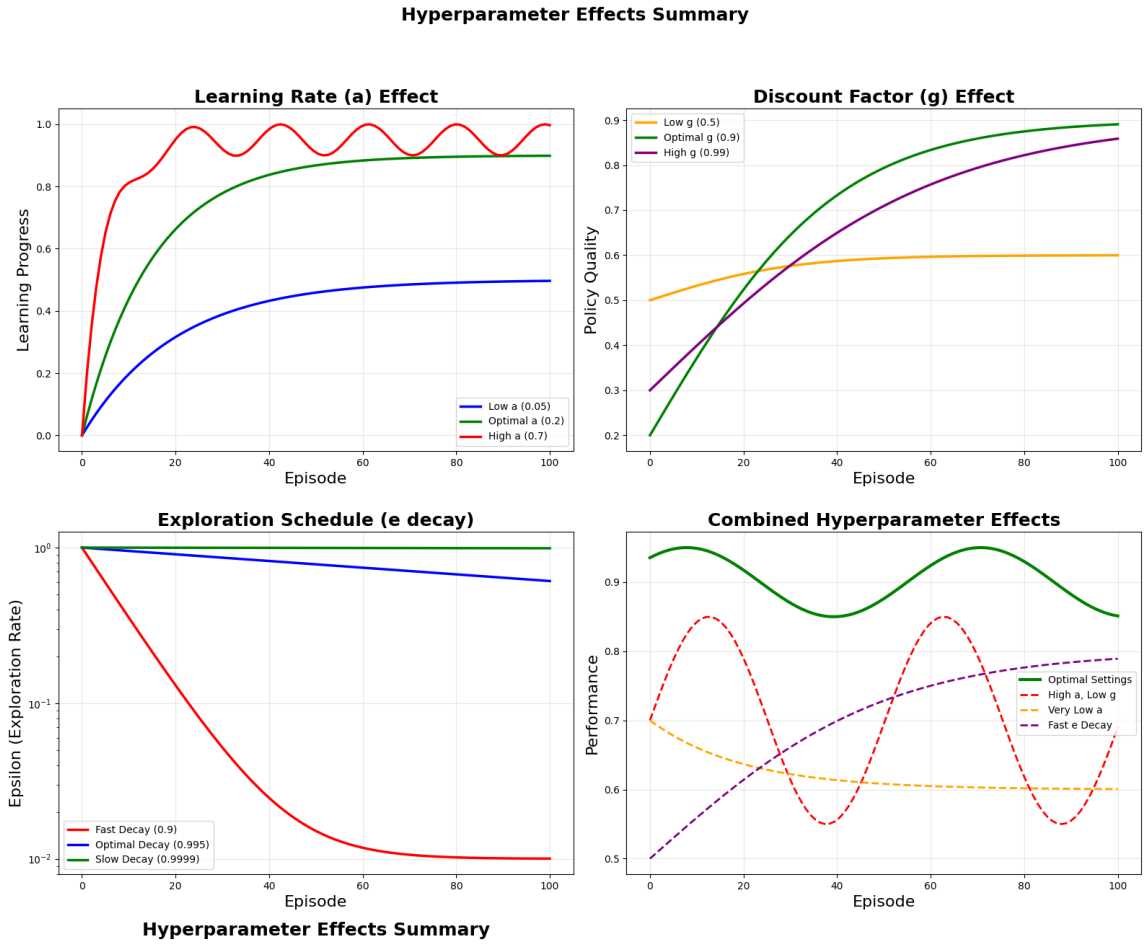
→	→	→	**+1.0**
↑	WALL	←	**-200.0**
↑	←	←	↓

Figure 2: The agent's highly cautious strategy with a -200 penalty.

3.2 The Crucial Role of Hyperparameters

The agent's learning effectiveness is highly dependent on tuning its core hyperparameters $(\alpha, \gamma, \epsilon)$. Figure 3 provides a high-level conceptual overview of their effects.

- **Learning Rate (α):** Controls how **impressionable** the agent is. A high α leads to faster learning but can be unstable, while a low α is stable but slow.
- **Discount Factor (γ):** Controls how **farsighted** the agent is. A high γ is essential for prioritizing long-term rewards like the '+1' prize over short-term penalties.
- **Exploration Schedule (ϵ decay):** A **decaying schedule** is best. The agent starts with high randomness to explore, then gradually reduces it to exploit the optimal path it has found.



Parameter	Too Low	Optimal Range	Too High
Learning Rate (a)	Slow convergence	0.1 - 0.3	Oscillations
Discount Factor (g)	Myopic behavior	0.8 - 0.95	Slow convergence
Epsilon Decay	Insufficient exploration	0.995 - 0.999	Excessive exploration

KEY FINDINGS:
- a (Learning Rate): Controls update step size.
- g (Discount Factor): Balances immediate vs. future rewards.
- e-decay: Manages exploration vs. exploitation trade-off.

RECOMMENDATIONS:
- Start with a=0.1, g=0.9, e-decay=0.995.
- Monitor convergence and performance.
- Adjust based on environment complexity.

Figure 3: A conceptual summary of how learning rate (α), discount factor (γ), and the exploration schedule (ϵ) impact Q-learning performance, with a table summarizing optimal ranges.

A more detailed, data-driven analysis is presented in the following figures. This comprehensive plot reveals several key trade-offs:

- Figure 4 shows that an intermediate learning rate ($\alpha \approx 0.1 - 0.3$) provides the best balance between convergence speed and stability. Very high values lead to erratic performance.
- Figure 5 confirms that a high discount factor ($\gamma \geq 0.9$) is necessary for achieving a good policy, as it encourages long-term planning.
- Figure 6 illustrates different exploration schedules. A moderate decay rate (e.g., 0.999) allows for sufficient exploration without wasting too much time on random actions.

- Figure 7 visually summarizes the performance for different α-γ pairs, highlighting a "sweet spot" of high performance in the upper-right quadrant (high γ, moderate α).
- Figure 8 shows how the episode length changes as the agent learns.

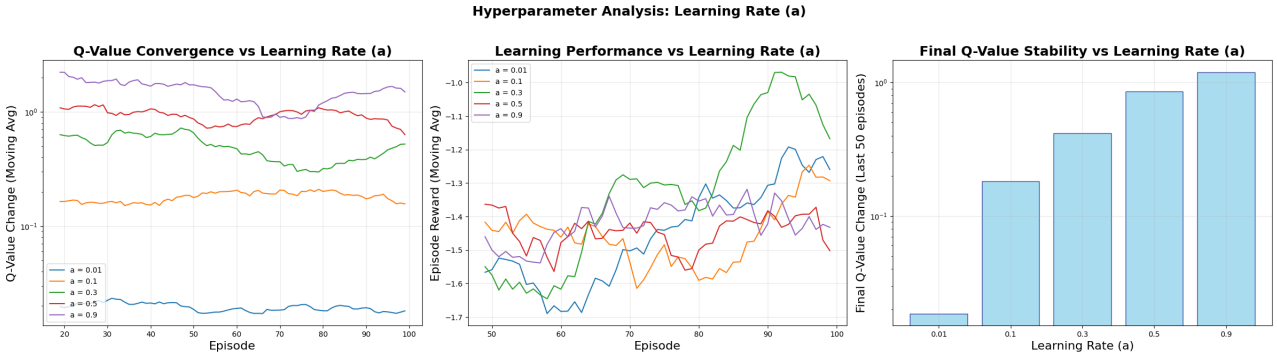


Figure 4: Analysis of the Learning Rate (alpha) on Q-value convergence and learning performance.

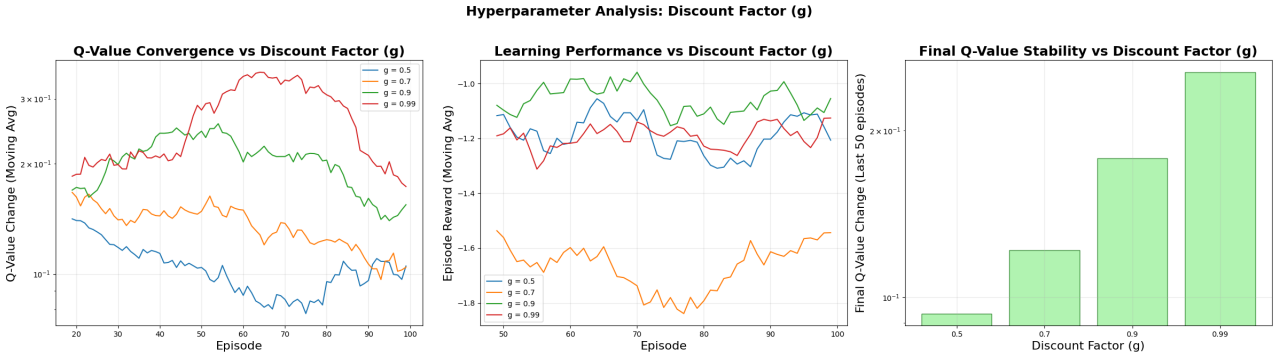


Figure 5: Analysis of the Discount Factor (gamma) on Q-value convergence and learning performance.

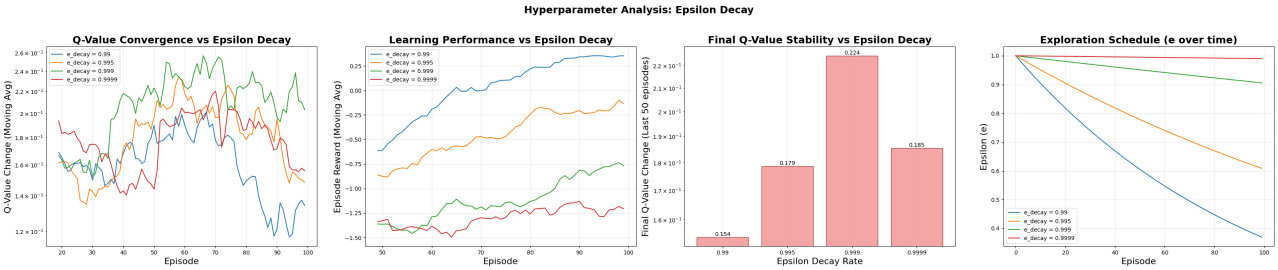


Figure 6: Analysis of the Epsilon Decay on Q-value convergence and learning performance.

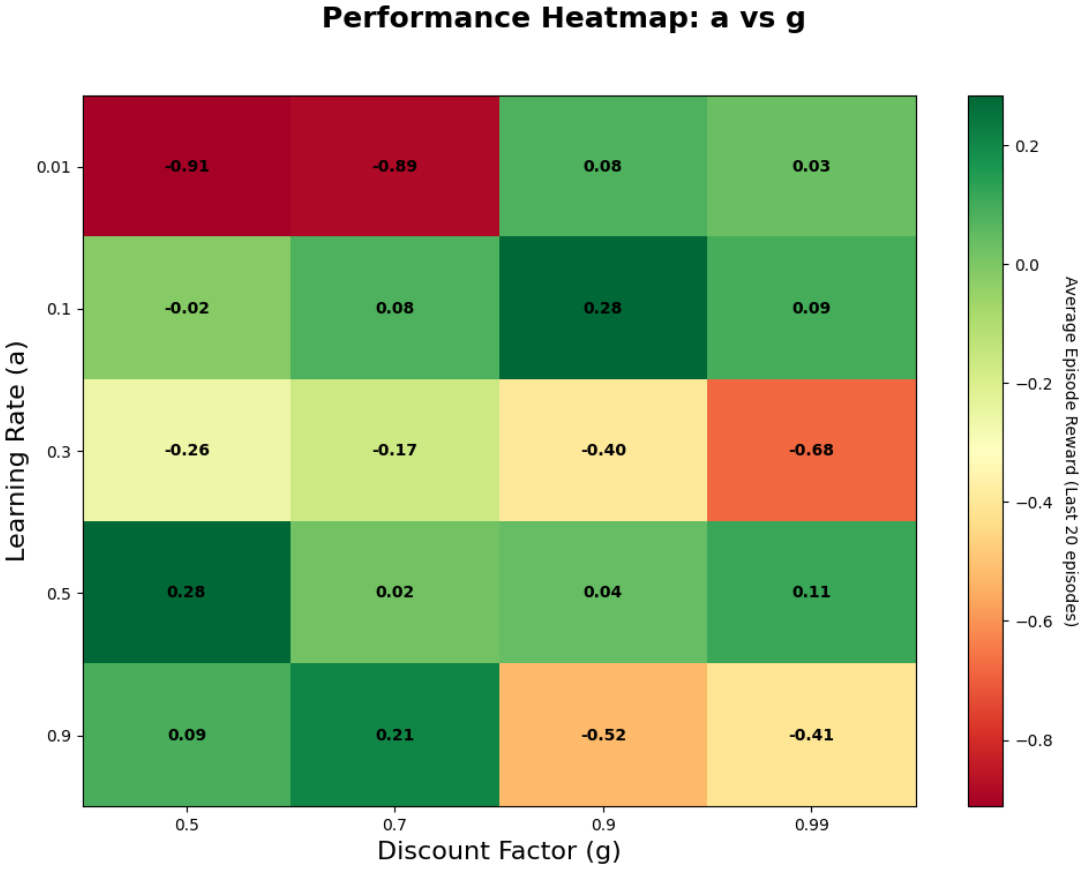


Figure 7: A heatmap showing the performance of different combinations of learning rate (α) and discount factor (γ).

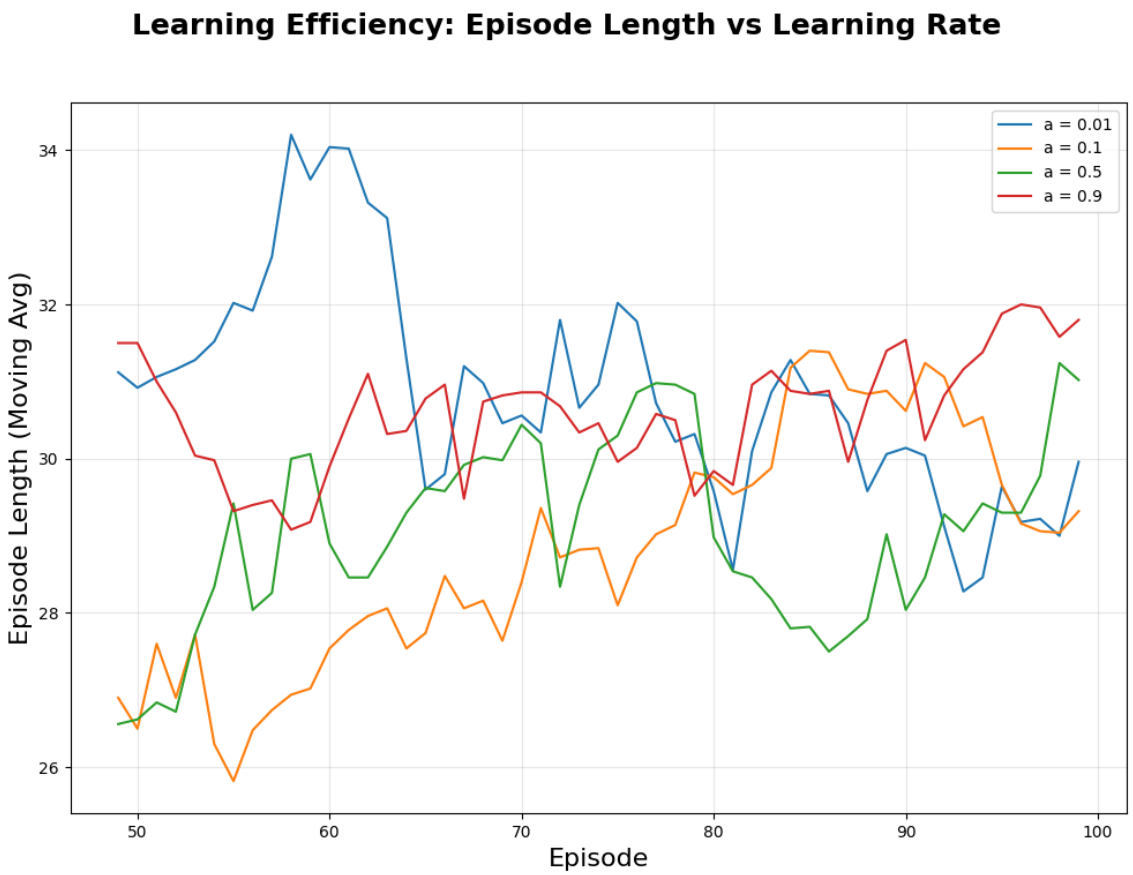


Figure 8: Learning efficiency for different learning rates.

3.3 Understanding Convergence: Policy vs. Q-Values

An important observation is that the agent's **policy converges much earlier than its Q-values**. The policy—what the agent decides to do—only depends on which action has the highest Q-value in a state. This relative ordering can stabilize long before the Q-values themselves stop changing numerically.

As shown in Figure 9, the number of policy changes (bottom plot) drops to zero after about 40 episodes, meaning the agent has settled on a final strategy. However, the total change in Q-values (top plot) continues to fluctuate for the entire training run, as the agent fine-tunes its "cheat sheet" scores. This demonstrates that a stable, optimal policy can be achieved well before the underlying value function has fully converged.

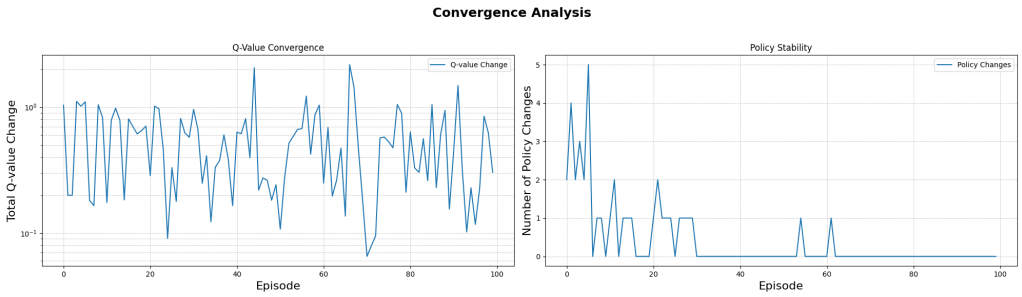


Figure 9: Convergence of Q-values (top) and the policy (bottom) over 100 episodes. The policy stabilizes quickly, while the Q-values continue to be refined.

4 Final Thoughts

This experiment was a wonderful success. Our agent started with zero knowledge of its world and, through nothing but trial, error, and a simple reward mechanism, it developed sophisticated strategies. It proved that a simple learning algorithm can lead to surprisingly intelligent behavior, allowing an agent to adapt its level of risk aversion in response to the dangers of its environment. The complete code for this simulation is available on my GitHub.