Contents

[**1.** **Introduction** 1](#_Toc155990700)

[**2.** **Experimental Setup** 1](#_Toc155990701)

[ **Grid Environment** 1](#_Toc155990702)

[ **Algorithms and Parameters** 1](#_Toc155990703)

[**3.** **Code Implementation** 1](#_Toc155990704)

[1. Monte Carlo 1](#_Toc155990705)

[ CODE 1](#_Toc155990706)

[ FINDING 3](#_Toc155990707)

[2. SARSA 3](#_Toc155990708)

[ CODE 3](#_Toc155990709)

[ FINDING 4](#_Toc155990710)

[3. Q LEARNING 6](#_Toc155990711)

[ CODE 6](#_Toc155990712)

[ FINDING 7](#_Toc155990713)

[**4.** **Results** 9](#_Toc155990714)

[1. Monte Carlo 9](#_Toc155990715)

[2. SARSA 9](#_Toc155990716)

[3. Q-learning 9](#_Toc155990717)

[**4.** **Discussion** 9](#_Toc155990718)

[ Impact of Slip Probabilities 9](#_Toc155990719)

[ Epsilon-Greedy Exploration 9](#_Toc155990720)

[**5.** **Conclusion** 10](#_Toc155990721)

**Report on Model-Free Reinforcement Learning Experiments**

# **Introduction**

Reinforcement learning is a powerful paradigm in machine learning that allows an agent to learn optimal behaviour through interactions with an environment. In this report, we explore the performance of three model-free reinforcement learning algorithms - Monte Carlo, SARSA, and Q-learning - on a grid-based environment with varying slip probabilities, epsilon values, alpha, and gamma.

# **Experimental Setup**

## **Grid Environment**

We defined a 4x4 grid environment with specific start and end states. The environment included safe and dangerous end states, along with slip probabilities that simulated uncertainty in the agent's actions.

## **Algorithms and Parameters**

We implemented three model-free reinforcement learning algorithms:

* Monte Carlo
* SARSA
* Q-learning

Each algorithm was tested with different slip probabilities, epsilon values (exploration-exploitation trade-off), alpha (learning rate), and gamma (discount factor). The goal was to observe how these parameters impact the learning process and the resulting policies.

# **Code Implementation**

## Monte Carlo

### CODE

def monte\_carlo(grid, slip\_prob, num\_episodes, epsilon):

# Initialize Q-values

Q = np.zeros\_like(grid)

returns = np.zeros\_like(grid)

visit\_count = np.zeros\_like(grid)

for episode in range(num\_episodes):

episode\_states = []

episode\_actions = []

episode\_rewards = []

current\_state = (2, 1)

while True:

num\_actions = len(Q) # Total number of possible actions (assuming each state has the same number of actions)

action = epsilon\_greedy(Q, current\_state, epsilon, num\_actions)

episode\_states.append(current\_state)

episode\_actions.append(action)

next\_state = get\_next\_state(current\_state, action, slip\_prob, grid)

reward = get\_reward(next\_state, grid)

episode\_rewards.append(reward)

current\_state = next\_state

if next\_state in [(0, 3), (2, 3)]:

break

total\_return = 0

for t in range(len(episode\_states) - 1, -1, -1):

total\_return += episode\_rewards[t]

if episode\_states[t] not in episode\_states[:t]:

state = episode\_states[t]

action = episode\_actions[t]

visit\_count[state] += 1

Q[state] += (total\_return - Q[state]) / visit\_count[state]

average\_utility = np.mean(Q)

return Q, average\_utility

### FINDING

* Results for Slip Probability 0.0 and Epsilon 0.1:
* Q-values:
* [[20. 20. 20. 0. ]
* [20. 20. 20. 20. ]
* [20. 19.86 17.77777778 0. ]
* [20. 20. 20. 0. ]]
* Average Utility: 16.10236111111111
* Results for Slip Probability 0.1 and Epsilon 0.1:
* Q-values:
* [[19.05970149 19.26238145 19.29149798 0. ]
* [18.97810219 19.26624738 18.81556684 13.28767123]
* [19.16167665 19.23 17.00854701 0. ]
* [20. 16.05633803 20. 0. ]]
* Average Utility: 14.963608140238424
* Results for Slip Probability 0.2 and Epsilon 0.1:
* Q-values:
* [[ 18.50381679 18.60927152 18.83333333 0. ]
* [ 18.75886525 18.40736728 17.68138801 14.06779661]
* [ 18.68421053 17.34 9.42708333 0. ]
* [ 18.67924528 16.41025641 -5.2 -50. ]]
* Average Utility: 9.38766464718444
* Results for Slip Probability 0.3 and Epsilon 0.1:
* Q-values:
* [[ 16.752 16.6589057 17.10869565 0. ]
* [ 16.97297297 15.97065463 14.91097923 8.01369863]
* [ 16.81818182 14.4 6.35782748 0. ]
* [ 15.51282051 13.16091954 1.19402985 -36. ]]
* Average Utility: 8.614480375852136

## SARSA

### CODE

def sarsa(grid, slip\_prob, num\_episodes, epsilon, alpha, gamma):

Q = np.zeros(grid.shape + (4,)) # Separate Q array for each state-action pair

for episode in range(num\_episodes):

current\_state = (2, 1)

num\_actions = Q.shape[2]

current\_action = epsilon\_greedy(Q, current\_state, epsilon, num\_actions)

episode\_complete = False

while not episode\_complete:

next\_state = get\_next\_state(current\_state, current\_action, slip\_prob, grid)

reward = get\_reward(next\_state, grid)

next\_action = epsilon\_greedy(Q, next\_state, epsilon, num\_actions)

Q[current\_state + (current\_action,)] += alpha \* (reward + gamma \* Q[next\_state + (next\_action,)] - Q[current\_state + (current\_action,)])

current\_state = next\_state

current\_action = next\_action

episode\_complete = next\_state in [(0, 3), (2, 3)]

average\_utility = np.mean(Q)

return Q, average\_utility

### FINDING

* Results for Slip Probability 0.0, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 0. 15.3952904 0.8649457 1.16898189]
* [14.10473839 17.66840659 11.08395535 11.48915468]
* [16.65557727 20. 13.29291439 14.28995138]
* [ 0. 0. 0. 0. ]]
* [[10.37943526 0. 0.95779768 0.52032741]
* [15.07069906 9.60641951 10.89873532 6.00772044]
* [17.12346342 0.504 2.33968775 1.38009835]
* [ 5.42 0. 0. 0. ]]
* [[ 1.91938916 7.82819427 0. 0. ]
* [13.56648855 8.58297164 7.22784837 2.92279226]
* [12.20811227 0. 0. 0. ]
* [ 0. 0. 0. 0. ]]
* [[ 0. 1.44875385 0. 0. ]
* [10.62560327 0.02139837 0. 0.06112863]
* [ 0.5414789 0. 0. 0. ]
* [ 0. 0. 0. 0. ]]]
* Average Utility: 4.424632184263154
* Results for Slip Probability 0.1, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 4.5809479 14.27368523 2.93850582 5.38161845]
* [13.83991962 16.69805241 11.20203939 9.33371237]
* [15.27051744 19.7990226 13.27866424 13.12913689]
* [ 0. 0. 0. 0. ]]
* [[11.79510554 7.2205028 1.80937601 0.76135302]
* [14.65984781 13.33670715 10.66629535 8.26949188]
* [16.69592806 1.0054062 1.60157955 5.05023532]
* [13.72378808 0. 0. 0. ]]
* [[10.06327904 3.07313719 0.127064 0.60350017]
* [12.79754007 9.13812489 6.78177679 7.85665425]
* [13.31904828 -5. 0.2551843 -6.92694492]
* [ 0. 0. 0. 0. ]]
* [[ 1.97930073 0.56776119 0. 0.11550118]
* [10.1078541 0.09897068 1.27641198 0.03840283]
* [ 5.77225664 0. 0. 0. ]
* [-5. 0. 0. 0. ]]]
* Average Utility: 4.896347851811532
* Results for Slip Probability 0.2, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 7.19215817 13.71801848 3.87434245 5.83244687]
* [ 13.76749736 16.68492342 11.96431251 10.31074056]
* [ 15.63675562 19.56015693 12.31895473 13.1335529 ]
* [ 0. 0. 0. 0. ]]
* [[ 11.80990914 7.17045419 3.11195391 3.89029958]
* [ 13.71928132 9.74258465 7.45634617 9.2440966 ]
* [ 14.29109391 0.53335824 2.12511107 5.85262916]
* [ 3.93155891 -3.30533573 -4.46800653 -2.57151353]]
* [[ 9.39319259 2.19841078 0.03274759 1.60262567]
* [ 10.98262364 2.40529007 5.63487881 6.09299425]
* [ -5.72846752 -20.4755 3.95665299 -4.82100177]
* [ 0. 0. 0. 0. ]]
* [[ 6.15494554 0. 0.06660467 0. ]
* [ 8.081651 0.68627339 1.8452932 2.43993823]
* [ 0.204312 -0.43755425 0.94365483 4.04027544]
* [ -9.5 -9.23666067 -9.32268205 -9.50848286]]]
* Average Utility: 3.5040577595385747
* Results for Slip Probability 0.3, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 5.31097071 7.80823384 7.63648537 1.36775377]
* [ 9.22000303 15.04725728 9.92273426 7.65771275]
* [ 14.03443543 19.03444389 12.13241774 12.11610548]
* [ 0. 0. 0. 0. ]]
* [[ 4.91748417 9.9443226 5.10664987 7.19433326]
* [ 11.75221569 9.27114811 7.43941485 8.19943569]
* [ 13.93969407 6.70972091 1.77852414 8.93892496]
* [ 10.08692973 6.23757412 2. -3.34239381]]
* [[ 7.41434448 2.01378657 3.91580347 5.3370688 ]
* [ 9.97015741 5.81229791 4.71686068 6.22536095]
* [ 11.62410801 -18.43660636 -7.88536387 -9.51642076]
* [ 0. 0. 0. 0. ]]
* [[ 5.64122501 2.2012216 2.69795904 1.37503939]
* [ 1.28553344 1.36179105 0.86140837 5.35610217]
* [ -1.78745751 -2.55248398 -1.95439552 2.45830124]
* [ -9.28357572 -9.26797008 -8.65179858 -2.09214436]]]
* Average Utility: 3.754729449680376

## Q LEARNING

### CODE

def q\_learning(grid, slip\_prob, num\_episodes, epsilon, alpha, gamma):

Q = np.zeros(grid.shape + (4,))

for episode in range(num\_episodes):

current\_state = (2, 1)

episode\_complete = False

while not episode\_complete:

num\_actions = Q.shape[2]

current\_action = epsilon\_greedy(Q, current\_state, epsilon, num\_actions)

next\_state = get\_next\_state(current\_state, current\_action, slip\_prob, grid)

reward = get\_reward(next\_state, grid)

Q[current\_state + (current\_action,)] += alpha \* (reward + gamma \* np.max(Q[next\_state]) - Q[current\_state + (current\_action,)])

current\_state = next\_state

episode\_complete = next\_state in [(0, 3), (2, 3)]

average\_utility = np.mean(Q)

return Q, average\_utility

### FINDING

* Results for Slip Probability 0.0, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 1.43554257 16.16966334 1.42935663 2.17522747]
* [15.35178506 18. 13.51443224 12.21597405]
* [16.22741087 20. 14.0269638 14.23028215]
* [ 0. 0. 0. 0. ]]
* [[13.94372498 0. 0. 0. ]
* [16.2 13.45566717 11.94633964 10.72090542]
* [17.98256613 0.18 0.95297645 0. ]
* [ 3.8 0. 0. 0. ]]
* [[ 2.08463962 10.12010729 0. 0.62320112]
* [14.58 9.23638977 7.34422099 4.99661602]
* [14.3451499 0. 0. 0. ]
* [ 0. 0. 0. 0. ]]
* [[ 0.08343709 0. 0. 0. ]
* [11.52994563 0. 0.9625912 0. ]
* [ 0. 0. 0. 0. ]
* [ 0. 0. 0. 0. ]]]
* Average Utility: 4.841642446774151
* Results for Slip Probability 0.1, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 5.63346036 15.09986407 4.03937695 2.59952222]
* [15.0314259 16.88990367 13.47346202 13.18148732]
* [16.82708321 19.57741746 14.82082103 15.04793737]
* [ 0. 0. 0. 0. ]]
* [[13.66594288 4.4213522 2.03412419 5.73722096]
* [15.16542656 13.06519873 11.02890456 7.94950784]
* [16.81693084 2.86581595 4.29311273 4.18779217]
* [11.3906558 0. 0. 0. ]]
* [[12.1229339 2.03318155 0.04640202 0. ]
* [13.43074025 11.22355195 8.53327392 8.68751184]
* [15.40960059 -5. 0.13315411 3.14756454]
* [ 0. 0. 0. 0. ]]
* [[ 3.27427844 0. 0. 0. ]
* [10.81063943 0.20590622 1.00993577 0.23606367]
* [ 5.32127452 0. 0. 0. ]
* [ 0. 0. 0. 0. ]]]
* Average Utility: 5.55421499552328
* Results for Slip Probability 0.2, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 6.0485554 14.38971816 7.19838134 7.0487923 ]
* [ 14.23342281 15.82717694 12.30738283 12.78290036]
* [ 16.34841072 19.07838502 14.46459943 15.22690407]
* [ 0. 0. 0. 0. ]]
* [[ 12.7929857 3.2865367 3.68918947 5.23501717]
* [ 14.2856316 12.71567166 9.4498936 10.76432231]
* [ 15.76292763 5.89208723 6.78960743 6.35181947]
* [ 7.56586521 -3.60936919 0. 0. ]]
* [[ 2.45007903 10.5166649 2.90488999 3.16111594]
* [ 11.72137669 6.52507652 8.14380314 9.16168486]
* [ 4.7491419 -20.4755 -0.34231995 1.95132618]
* [ 0. 0. 0. 0. ]]
* [[ 7.40596253 0.67177289 0. 0.60178346]
* [ 9.71368969 0. 0.91721034 1.37874712]
* [ 4.87314106 0. 0.41317933 0. ]
* [ -5. -5. -5. 0. ]]]
* Average Utility: 4.9589006406926694
* Results for Slip Probability 0.3, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 8.57288909 12.60834026 8.10943135 8.17621439]
* [ 13.18378068 15.70159396 12.21026362 11.68543584]
* [ 15.84734906 18.98278278 13.84483107 14.01097165]
* [ 0. 0. 0. 0. ]]
* [[ 11.09037784 8.55509052 6.30735336 7.00509668]
* [ 12.80386561 11.15036286 8.05494033 9.03841283]
* [ 14.93931129 6.24270535 5.91223354 6.28071386]
* [ 17.34867679 0.47264433 -9.5 -3.36012088]]
* [[ 9.60198447 2.98695162 3.15870945 4.10908557]
* [ 9.61567271 4.26597192 6.33255529 5.70879624]
* [ 8.41075772 -22.63540512 -5.38331505 -8.60829079]
* [ 0. 0. 0. 0. ]]
* [[ 7.63596523 1.1567008 0.68839366 0.68608916]
* [ 8.02349698 2.48239098 3.84961698 2.47248245]
* [ 0.87817569 -0.15918533 0.65967819 5.33554946]
* [ -5. 0. 0.45990176 -3.68345429]]]
* Average Utility: 4.817575372220121

# **Results**

## Monte Carlo

The Monte Carlo algorithm provided promising results, demonstrating its ability to learn optimal policies in a variety of scenarios. The Q-values and average utility indicate the effectiveness of the algorithm in navigating the environment.

## SARSA

SARSA, with its on-policy nature, showed competitive performance. The Q-values reveal the learned state-action pairs, and the average utility provides insights into the efficiency of the learned policy.

## Q-learning

Q-learning, an off-policy algorithm, displayed strong performance as well. The Q-values and average utility highlight the algorithm's capability to find optimal policies in different scenarios.

# **Discussion**

## Impact of Slip Probabilities

Slip probabilities introduce uncertainty into the agent's actions. Analyzing the results for different slip probabilities allows us to understand how robust the algorithms are in handling noisy environments.

## Epsilon-Greedy Exploration

Exploration-exploitation is crucial in reinforcement learning. Varying epsilon values allowed us to observe the impact of exploration strategies on the learning process.

# **Conclusion**

In conclusion, the experiments shed light on the strengths and weaknesses of the Monte Carlo, SARSA, and Q-learning algorithms in a grid-based environment. Understanding the impact of different parameters is essential for selecting suitable configurations in real-world applications. Future work may involve further exploration of hyperparameter tuning and testing the algorithms on more complex environments.