AI- REINFORCEMENT LEARNING

[**1.** **Introduction** 2](#_Toc156067860)

[**2.** **Experimental Setup** 2](#_Toc156067861)

[ **Grid Environment** 2](#_Toc156067862)

[ CODE 2](#_Toc156067863)

[ **Algorithms and Parameters** 3](#_Toc156067864)

[**3.** **Experimental Setup** 3](#_Toc156067865)

[**4.** **Code Implementation** 4](#_Toc156067866)

[1. Monte Carlo 4](#_Toc156067867)

[ CODE 4](#_Toc156067868)

[ FINDING 5](#_Toc156067869)

[2. SARSA 6](#_Toc156067870)

[ CODE 6](#_Toc156067871)

[ FINDING 7](#_Toc156067872)

[3. Q LEARNING 8](#_Toc156067873)

[ CODE 8](#_Toc156067874)

[ FINDING 9](#_Toc156067875)

[**5.** **Results** 10](#_Toc156067876)

[1. Monte Carlo: 10](#_Toc156067877)

[2. SARSA: 11](#_Toc156067878)

[3. Q-learning: 11](#_Toc156067879)

[**6.** **Discussion** 11](#_Toc156067880)

[ Impact of Slip Probabilities 11](#_Toc156067881)

[ Epsilon-Greedy Exploration 11](#_Toc156067882)

[**7.** **Conclusion** 11](#_Toc156067883)

**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 3x4 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.

def initialize\_environment(grid\_size, start\_state, safe\_end\_states, dangerous\_end\_state, slip\_prob):

### CODE

# Initialize the grid world

grid = np.zeros(grid\_size)

for state in safe\_end\_states:

if 0 <= state[0] < grid\_size[0] and 0 <= state[1] < grid\_size[1]:

grid[state[0], state[1]] = 20

for state in dangerous\_end\_state:

if 0 <= state[0] < grid\_size[0] and 0 <= state[1] < grid\_size[1]:

grid[state[0], state[1]] = -50

specific\_reward\_location = (2, 0)

if 0 <= specific\_reward\_location[0] < grid\_size[0] and 0 <= specific\_reward\_location[1] < grid\_size[1]:

grid[specific\_reward\_location[0], specific\_reward\_location[1]] = 2

return grid

## **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.

# **Experimental Setup**

For Monte carlo number of episodes set to 1000 as it run in short period of time but SARSA and Q Learning taking huge amount of time for 1000 so we set it up to 100 only.

def run\_experiments():

grid\_size = (3, 4)

start\_state = (0, 0)

safe\_end\_states = [(3, 1), (1, 4)]

dangerous\_end\_state = [(0, 2),(1,2)]

num\_episodes = 1000

slip\_probabilities = [0.0, 0.1, 0.2, 0.3]

epsilon\_values = [0.1]

for slip\_prob in slip\_probabilities:

for epsilon in epsilon\_values:

# Initialize the environment

grid = initialize\_environment(grid\_size, start\_state, safe\_end\_states, dangerous\_end\_state, slip\_prob)

Q\_values, avg\_utility = monte\_carlo(grid, slip\_prob, num\_episodes, epsilon)

print(f"Results for Slip Probability {slip\_prob} and Epsilon {epsilon}:")

print("Q-values:")

print(Q\_values)

print(f"Average Utility: {avg\_utility}")

print("\n")

# **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:
* [[ 0. -1569.64668094 -1537.6753507 0. ]
* [ 0. -1588.28365879 -1518.718802 0. ]
* [ 0. -1589.35 -1694.26229508 0. ]]
* Average Utility: -791.4947322924572
* Results for Slip Probability 0.1 and Epsilon 0.1:
* Q-values:
* [[-775.84653465 -764.94678492 -714.60655738 0. ]
* [-777.55263158 -765.10374332 -725.25876461 -67.44186047]
* [-832.19753086 -756.596 -703.19354839 0. ]]
* Average Utility: -573.5619963476214
* Results for Slip Probability 0.2 and Epsilon 0.1:
* Q-values:
* [[-538.58802178 -536.88439955 -496.44583333 0. ]
* [-529.83105023 -543.75593952 -491.3894081 -12.3015873 ]
* [-481.66480447 -538.81 -487.07291667 0. ]]
* Average Utility: -388.06199674611224
* Results for Slip Probability 0.3 and Epsilon 0.1:
* Q-values:
* [[-390.57903494 -405.00235294 -349.02887701 0. ]
* [-395.75862069 -400.13785311 -342.87687688 -53.0617284 ]
* [-413.06944444 -392.188 -331.11428571 0. ]]
* Average Utility: -289.4014228429964

## 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:
* [[[ 10.32668441 5.72549305 16.64659036 12.9212533 ]
* [ 1.6290686 -13.55 3.54522269 13.83139074]
* [ -9.5 0. 0. 0. ]
* [ 0. 0. 0. 0. ]]
* [[ 14.36534223 10.66245633 18.96631098 15.73438785]
* [ 11.26012385 -48.86833379 16.77771751 14.98450085]
* [-13.55 0. 0. 0. ]
* [ 0. 0. 0. -5. ]]
* [[ 17.02681309 16.21961605 18.78792968 18.9672753 ]
* [ 14.77340075 13.13121903 16.74309547 18.79410182]
* [-38.56160377 0. 11.24124724 16.45794691]
* [ 0. 0. 0. 0. ]]]
* Average Utility: 4.176859385709616
* Results for Slip Probability 0.1, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 8.89681676 6.05087807 14.26380643 8.36230994]
* [ -2.46013945 -7.28752499 5.85129469 -3.99680865]
* [ -9.5 0. -5. 0. ]
* [ 0. 0. 0. 0. ]]
* [[ 11.76392551 9.56296069 16.8886093 13.61427879]
* [ 5.28247067 -40.92839487 6.84600504 13.16163536]
* [-17.195 -0.32631365 3.02760789 -3.75636025]
* [ -4.5 -5.45 -5.01753072 -8.57493911]]
* [[ 14.91320416 14.0398193 16.32989945 16.8199079 ]
* [ 10.88045803 9.64356374 12.26866285 14.81400167]
* [-32.66977004 -1.13513584 4.78406386 13.27077168]
* [ 0. 0. 0. 0. ]]]
* Average Utility: 2.157063212802655
* Results for Slip Probability 0.2, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 6.28626386 3.92246096 10.40009183 6.67949898]
* [ -2.20804934 -23.42795 -7.70437312 -7.8667622 ]
* [-13.55 -13.32092459 -9.90276407 -5.0582207 ]
* [ 0. 0. 0. 0. ]]
* [[ 9.62579357 7.7957166 14.13912152 11.90600812]
* [ -2.30042017 -30.46048026 0.37894969 2.87909493]
* [-14.74716164 -7.11136329 -5.32128871 -7.23083725]
* [ 0. 0. 0. -5. ]]
* [[ 13.03651694 12.17308362 14.28090962 15.20186014]
* [ 7.51332788 7.07902442 10.75586404 15.22464873]
* [-23.06312042 -3.57970205 -4.04035398 5.13835113]
* [ 0. 0. 0. 0. ]]]
* Average Utility: -0.23910802505965986
* Results for Slip Probability 0.3, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 1.72243428 1.58025014 6.86468954 3.63150617]
* [-15.14411002 -29.20906502 -10.81143086 -1.48100291]
* [-13.55 -5.19125955 -9.91606656 -11.27433011]
* [ 0. 0. 0. 0. ]]
* [[ 5.64674773 3.05491506 9.08824257 5.2172607 ]
* [ -8.8349295 -24.87564605 -10.72224532 2.62888215]
* [-13.67665686 -3.59873613 -9.70776489 -13.18019938]
* [ -5.78528491 -5.06339406 -9.61484998 -10.02822378]]
* [[ 9.21549377 7.25811955 9.65405893 9.29473499]
* [ 0.59167395 1.94174297 4.25582656 6.99074315]
* [-25.96635919 -7.43661468 -10.84964254 -9.35660743]
* [ 0. 0. 0. 0. ]]]
* Average Utility: -3.679939531361627

## 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:
* [[[ 8.68037268 14.58 12.91626834 10.92111922]
* [ 12.14843125 -49.84783736 16.2 10.93030752]
* [-20.4755 0. -12.31083799 0. ]
* [ 0. 0. 0. 0. ]]
* [[ 13.122 16.2 20. 18. ]
* [ 14.02857238 -36.32852139 18. 17.8479422 ]
* [ -5. 0. 16.05860698 1.62 ]
* [ 0. 0. 0. -3.70153289]]
* [[ 18. 18. 20. 20. ]
* [ 16.2 16.2 18. 20. ]
* [-32.17280508 0. 13.96025205 18. ]
* [ 0. 0. 0. 0. ]]]
* Average Utility: 4.995350789582502
* Results for Slip Probability 0.1, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 2.38996421e+00 3.65258683e-01 1.61375869e+01 4.78837294e+00]
* [-4.99087887e+00 -9.04953581e+00 8.58820376e+00 -1.60305921e+00]
* [-5.00000000e+00 1.52667775e-03 0.00000000e+00 -1.58838860e-01]
* [ 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
* [[ 1.43648254e+01 1.38061290e+01 1.86587898e+01 1.61927774e+01]
* [-2.93435438e+00 -2.50795000e+01 1.58732582e+01 6.56151714e+00]
* [-1.35498473e+01 -6.23295000e-01 5.91664474e+00 5.55308434e-01]
* [-5.00000000e+00 -5.00000000e+00 -2.63063295e+00 -5.00000000e+00]]
* [[ 1.66897336e+01 1.66834662e+01 1.84614167e+01 1.83175684e+01]
* [ 1.37656212e+01 1.33713493e+01 1.63332107e+01 1.78624750e+01]
* [-1.08591037e+01 8.22637302e-01 5.37217401e+00 1.61444311e+01]
* [ 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]]
* Average Utility: 3.8863583450750094
* Results for Slip Probability 0.2, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 7.87451224 3.46960431 13.21080528 8.39087302]
* [ -3.56341977 -20.89453065 -0.22082774 1.50325775]
* [-23.35510446 -4.5 -5. -1.88993249]
* [ 0. 0. 0. 0. ]]
* [[ 11.92931264 6.43382247 15.77676438 12.72066329]
* [ -4.71554767 -33.73411148 -4.93011347 12.51940512]
* [-20.69244699 -8.22850577 -6.46355996 -5.96580832]
* [ -3.645 -8.18030123 -5.30824024 -5. ]]
* [[ 13.95434273 13.83311214 15.87709111 15.53579626]
* [ 9.05675554 8.00319344 12.31665857 14.24477503]
* [-27.10364176 -0.82495728 -3.930551 8.05619226]
* [ 0. 0. 0. 0. ]]]
* Average Utility: 0.13667369413020314
* Results for Slip Probability 0.3, Epsilon 0.1, Alpha 0.1, Gamma 0.9:
* Q-values:
* [[[ 2.51201084 -0.56811531 7.59974959 1.25927329]
* [-15.25009209 -17.76841803 -18.10984965 -0.23367288]
* [-13.55 -10.68520973 -15.24779252 -17.02196703]
* [ 0. 0. 0. 0. ]]
* [[ 6.41459161 2.22395563 11.69672373 6.34744387]
* [-17.25511498 -25.28078723 7.60049856 -12.65206957]
* [-14.16852097 -7.51216889 -11.25782749 -12.52717116]
* [ -6.70269316 -4.45485441 -4.5 -10.07557201]]
* [[ 10.68571672 10.90492116 12.97272071 12.14040267]
* [ 2.37263108 -1.06549415 3.14399681 9.88300666]
* [-12.82576398 -1.12974094 -16.36920624 -11.53953375]
* [ 0. 0. 0. 0. ]]]
* Average Utility: -3.541541525958051

# **Results**

## Monte Carlo:

Monte Carlo exhibited promising results in terms of learning optimal policies. However, it appears to struggle in environments with slip probabilities, especially evident in higher slip probability scenarios (e.g., Slip Probability 0.3). The learned policies may not be as efficient, leading to lower average utility.

## SARSA:

SARSA, being an on-policy algorithm, showed competitive performance. It learned state-action pairs effectively and demonstrated adaptability to different slip probabilities. The results indicate that SARSA can handle slip-prone environments reasonably well, producing policies with decent average utility.

## Q-learning:

Q-learning, an off-policy algorithm, also performed well in various scenarios. It demonstrated the capability to find optimal policies, and the average utility suggests effective learning. Similar to SARSA, Q-learning seems robust to slip probabilities and can adapt to different environments.

# **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 experimental findings highlight distinctive characteristics of Monte Carlo, SARSA, and Q-learning algorithms in grid-based environments. Monte Carlo demonstrates promise, particularly in low-slip scenarios, while SARSA showcases competitive adaptability across slip probabilities. Q-learning consistently exhibits strong performance, suggesting its robustness in finding optimal policies. Understanding algorithmic strengths and weaknesses aids in selecting appropriate configurations for specific scenarios. Future work may involve fine-tuning hyperparameters and testing on more complex environments to enhance algorithmic performance.