q_learning_vs_sarsa

March 3, 2023

1 Monte Carlo Simulation

1.1 TODO

[32]:
```python
print("todo")
```

```
todo
```

2 Q learning versus SARSA

2.1 Helpers

[22]:
```python
import random
import numpy as np
import matplotlib.pyplot as plt

ROWS=4
COLUMNS=21

START = "S"
GOAL = "G"
CLIFF = "C"
SNAKE_PIT = "P"

UP = "^"
DOWN = "v"
RIGHT = ">"
LEFT = "<"
ACTIONS = [UP, DOWN, RIGHT, LEFT]
ACTION_POSITION_MAPPER = {
    UP: (-1, 0),
    RIGHT: (0, 1),
    DOWN: (1, 0),
    LEFT: (0, -1)
}

START_POSITION = (3, 0)
GOAL_POSITION = (3, 20)
```

```
SNAKE_PIT_POSITION = (0, 11)
CLIFF_POSITIONS = [(3, cliff_column_index) for cliff_column_index in range(1, COLUMNS - 1)]

def get_world(with_snake_pit):
    world = [[random.choice(ACTIONS) for _ in range(COLUMNS)] for _ in range(ROWS)]

    world[START_POSITION[0]][START_POSITION[1]] = START
    world[GOAL_POSITION[0]][GOAL_POSITION[1]] = GOAL

    if with_snake_pit:
        world[SNAKE_PIT_POSITION[0]][SNAKE_PIT_POSITION[1]] = SNAKE_PIT

    for cliff_position in CLIFF_POSITIONS:
       world[cliff_position[0]][cliff_position[1]] = CLIFF

    return world

def print_world(world):
    for row in world:
        print(" ".join(row))

def choose_action(Q, new_state, epsilon):
    if np.random.uniform(0, 1) < epsilon:
        return random.choice(ACTIONS)
    else:
        actions_utilities = Q[new_state]
        return ACTIONS[np.argmax(actions_utilities)]

def get_start_state():
    return START_POSITION

def get_new_state(current_state, action):
  position_updater = ACTION_POSITION_MAPPER[action]
  new_state = (current_state[0] + position_updater[0], current_state[1] + position_updater[1])

  # row invalidation
  if new_state[0] < 0 or new_state[0] > ROWS - 1:
     return current_state

  # column invalidation
  if new_state[1] < 0 or new_state[1] > COLUMNS - 1:
    return current_state

  return new_state
```

```python
def final_state(state):
    return state == GOAL_POSITION or state in CLIFF_POSITIONS

def get_reward(position, with_snake_pit):
    if position == GOAL_POSITION:
        return 20

    if position in CLIFF_POSITIONS:
        return -100

    if with_snake_pit and position in SNAKE_PIT_POSITION:
        return -100

    return -1
```

2.2 Sarsa implementation

[23]:
```python
def run_sarsa(epochs = 2000, epsilon=0, with_snake_pit=False, alpha = 0.1,␣
 ↪gamma = 0.9):
  Q = {(i,j):[0, 0 ,0 , 0] for i in range(ROWS) for j in range(COLUMNS)}

  rewards_history = []
  for _ in range(epochs):
    current_state = get_start_state()
    current_action = choose_action(Q, current_state, epsilon=epsilon)

    reward_per_epoch = 0
    while not final_state(current_state):
      new_state = get_new_state(current_state, current_action)
      new_action = choose_action(Q, new_state, epsilon=epsilon)

      reward = get_reward(new_state, with_snake_pit)
      reward_per_epoch += reward

      current_action_index_in_Q = ACTIONS.index(current_action)
      Q_current = Q[current_state][current_action_index_in_Q]

      new_action_index_in_Q = ACTIONS.index(current_action)
      Q_next = Q[new_state][new_action_index_in_Q]

      Q[current_state][current_action_index_in_Q] = Q_current + (alpha *␣
 ↪(reward + gamma * Q_next - Q_current))

      current_state = new_state
      current_action = new_action
```

```
    rewards_history.append(reward_per_epoch)

  return Q, rewards_history
```

2.3 Q Learning + replay buffer implementation

[24]:
```
REPLAY_BUFFER_SIZE = 2048
REPLAY_BUFFER_BATCH_SIZE = 1024

def run_q_learning(epochs = 2000, epsilon=0, replay_buffer_enabled= False,␣
 ↪with_snake_pit=False, alpha = 0.1, gamma = 0.9):
  Q = {(i,j):[0, 0 ,0 , 0] for i in range(ROWS) for j in range(COLUMNS)}
  rewards_history = []
  replay_buffer = []

  for _ in range(epochs):
    current_state = get_start_state()

    reward_per_epoch = 0
    while not final_state(current_state):
      current_action = choose_action(Q, current_state, epsilon=epsilon)
      new_state = get_new_state(current_state, current_action)

      reward = get_reward(new_state, with_snake_pit)
      replay_buffer.append((current_state, current_action, reward, new_state))

      if len(replay_buffer) > REPLAY_BUFFER_SIZE:
        replay_buffer.pop(0)

      if replay_buffer_enabled:
        if len(replay_buffer) > REPLAY_BUFFER_BATCH_SIZE:
          buffer_memory = random.sample(replay_buffer, REPLAY_BUFFER_BATCH_SIZE)
          for experience in buffer_memory:
            current_state_history, current_action_history, reward_history,␣
 ↪new_state_history = experience

            current_action_index_in_Q = ACTIONS.index(current_action_history)
            Q_current = Q[current_state_history][current_action_index_in_Q]
            max_Q_next = max(Q[new_state_history])

            Q[current_state_history][current_action_index_in_Q] = Q_current +␣
 ↪(alpha * (reward_history + gamma * max_Q_next - Q_current))
      else:
        current_action_index_in_Q = ACTIONS.index(current_action)
        Q_current = Q[current_state][current_action_index_in_Q]
```

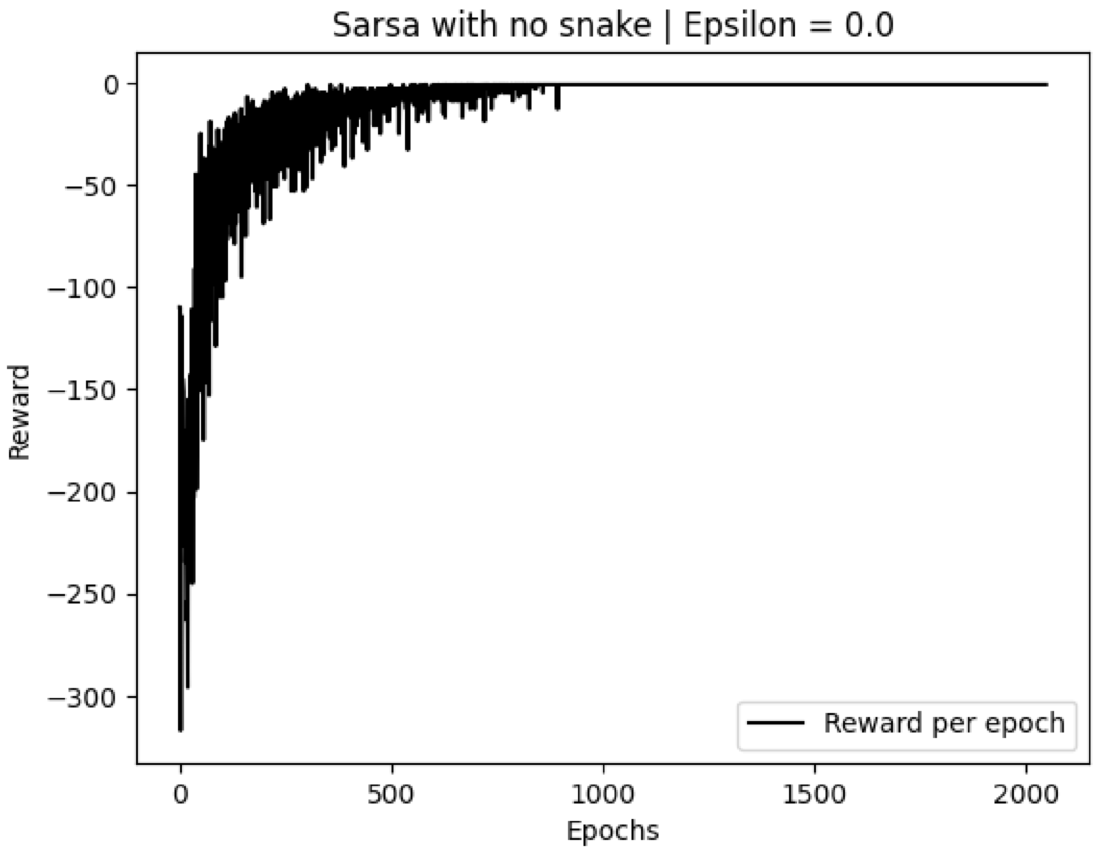
```
        max_Q_next = max(Q[new_state])

        Q[current_state][current_action_index_in_Q] = Q_current + (alpha *␣
↪(reward + gamma * max_Q_next - Q_current))

      current_state = new_state
      reward_per_epoch += reward

    rewards_history.append(reward_per_epoch)

  return Q, rewards_history
```

2.4 Results

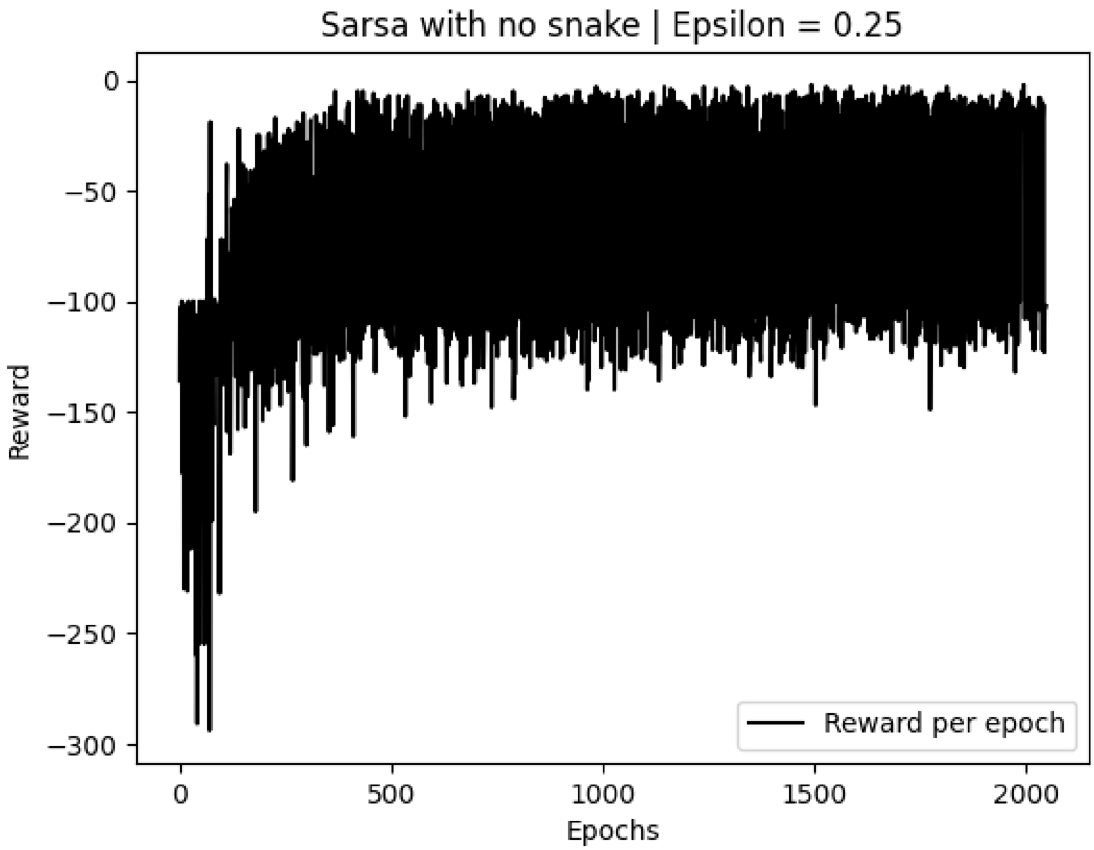
2.4.1 Sarsa with no snake

- We notice that in this case for smaller epsilon values, the agent manages to find the path to the goal, walking `safe path`.
- It manages to construct better global policy for smaller epsilon values.
- As epsilon increases, his exploration increases, and arguably, by change, he just falls down the cliff and ends the episode. That's the reason for some experiments, he didn't manage to construct a good policy to the goal position.

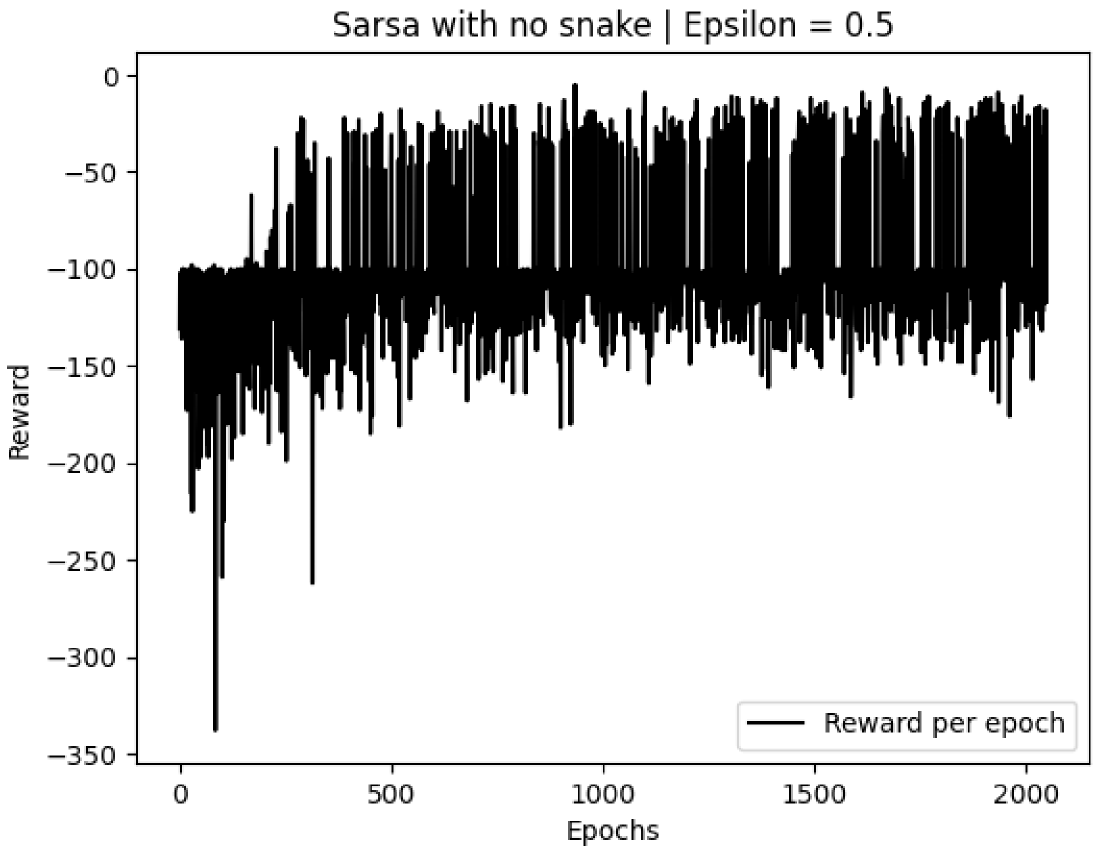
[25]:
```
epochs = 2048
x = np.linspace(1, epochs, epochs)

for epsilon in [0.0, 0.25, 0.5, 0.75, 1.0]:
    Q, rewards_history = run_sarsa(epochs=epochs, epsilon=epsilon)
    # Plot rewards
    plt.plot(x, rewards_history, color='black', label=f"Reward per epoch")
    plt.xlabel('Epochs')
    plt.ylabel('Reward')
    plt.legend()
    plt.title(f"Sarsa with no snake | Epsilon = {epsilon}")
    plt.show()

    world = get_world(with_snake_pit=False)
    for position in Q:
        actions_utilities = Q[position]
        if world[position[0]][position[1]] not in [GOAL, START, CLIFF]:
            world[position[0]][position[1]] = ACTIONS[np.
↪argmax(actions_utilities)]

    # Printing world
    print_world(world)

    print("-" * 80)
```



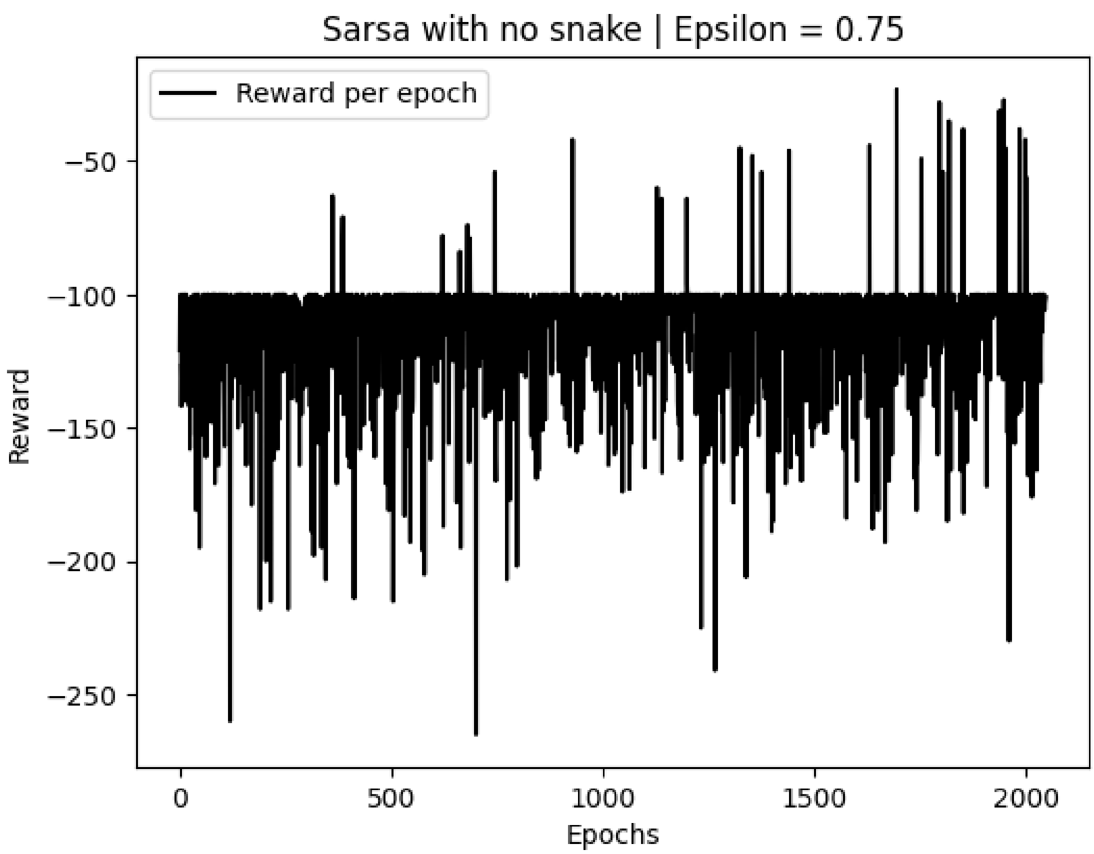

```
v > > > > > > > > > > > > > > > > > > > v
> > > > > > > > > > > > > > > > > > > > v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```



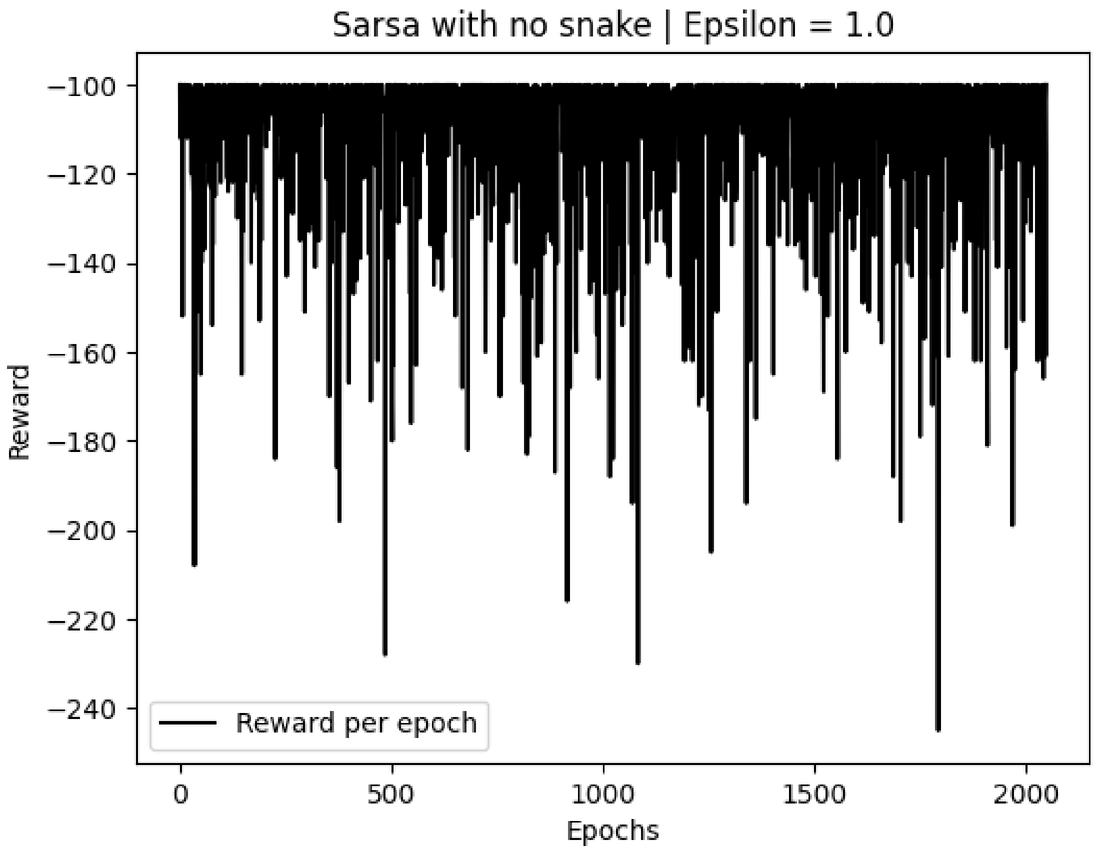

```
> > > > > > > > > > > > > > > > > > > > v
> > > > > > > ^ > > > > > > > > > > < > v
> > > > > > > > > > > > > > > > > > > ^ v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
< < > > > > > > > > > > > > > > > > > > v
> ^ > > > > > > > > > > > > > > > ^ > ^ v
> > > > > > > > > > > > < > > > < > > ^ v
S C C C C C C C C C C C C C C C C C C C G
----------------------------------------
```




```
> > > > > > > > > > > > > > > > ^ > > > v
> > > > > > > > > > > > > > > > > > > > v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```



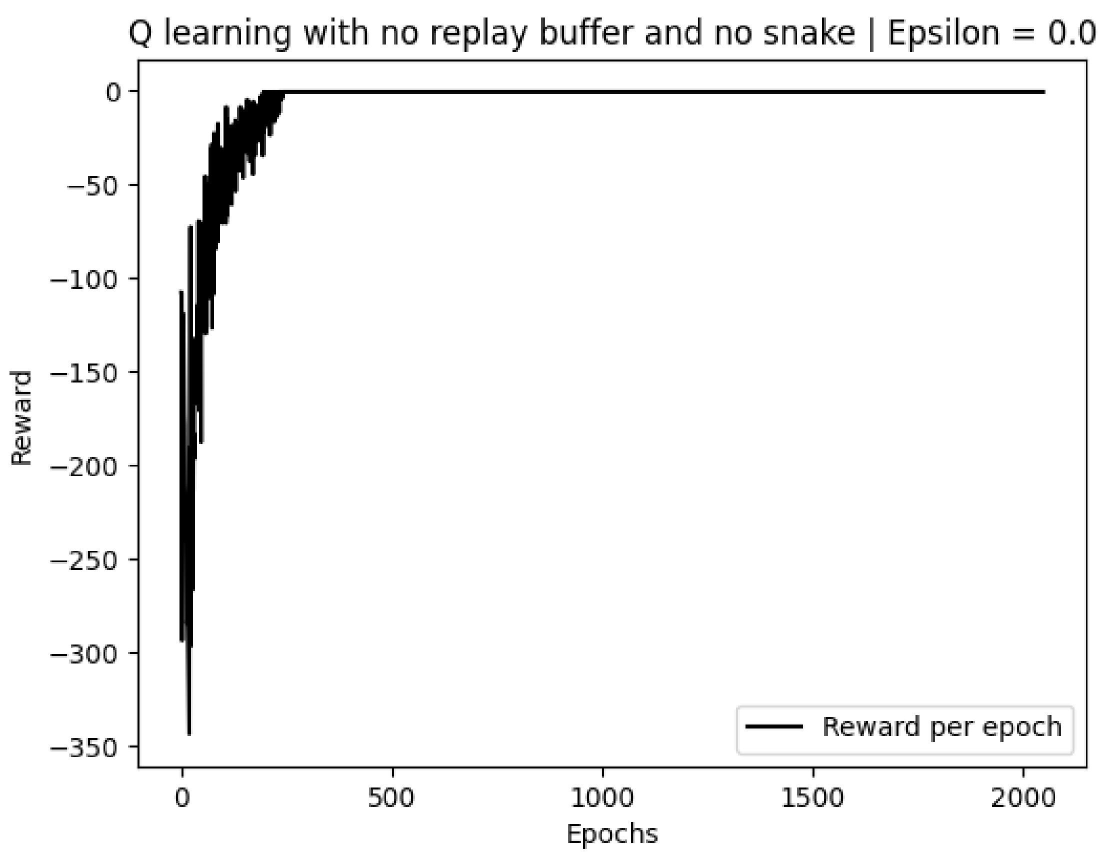

```
> > > > > > > > > > v ^ < v v ^ v < v ^ ^
> > > > > > > > > < ^ < v ^ < > ^ ^ ^ ^ ^
> > > > > > > > < ^ ^ < v ^ v < ^ ^ ^ ^ ^
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```

2.4.2 Q learning with no replay buffer and no snake

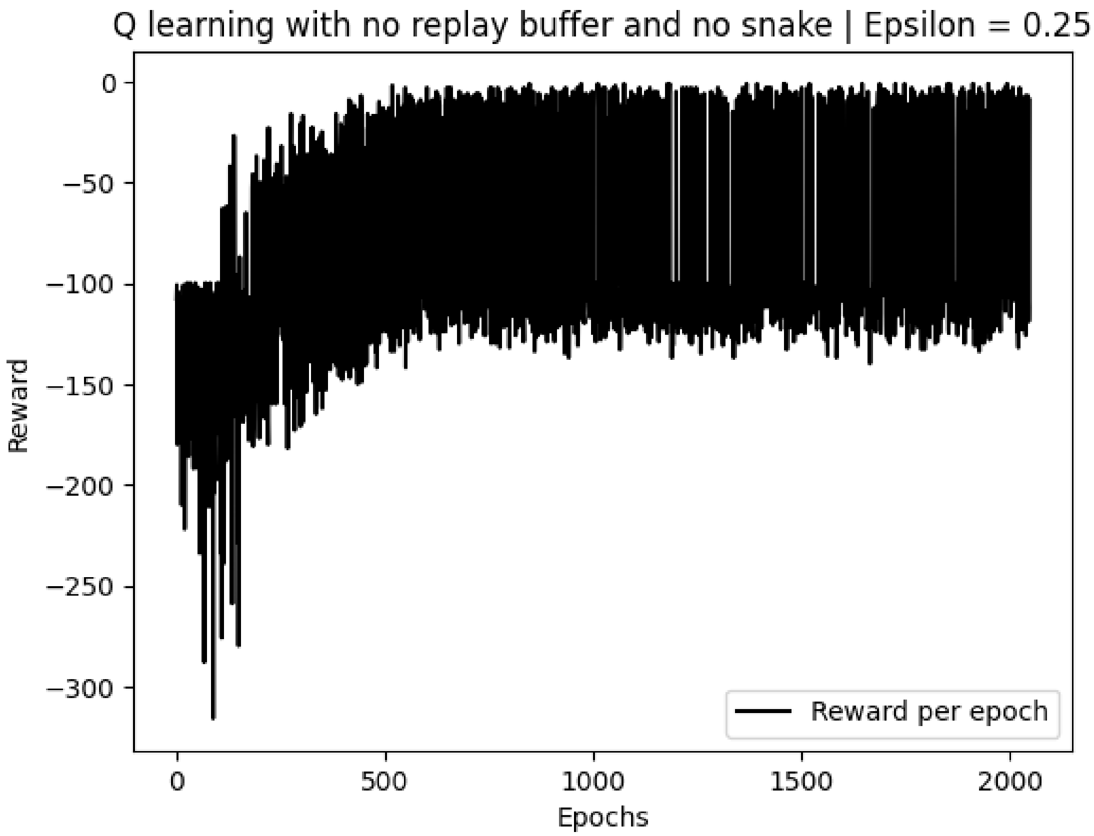
- We notice that in this case for smaller epsilon values, the agent manages to find the `optimal path` to the goal.
- We observe that for any values of epsilon, the agent is always trying to construct the most `optimal path`.
- For higher values of epsilon, the agent fails to construct a viable policy because is `falling down` the cliff most of the times. That thing will change when we add replay buffer.

[27]:
```
epochs = 2048
x = np.linspace(1, epochs, epochs)

for epsilon in [0.0, 0.25, 0.5, 0.75, 1.0]:
```

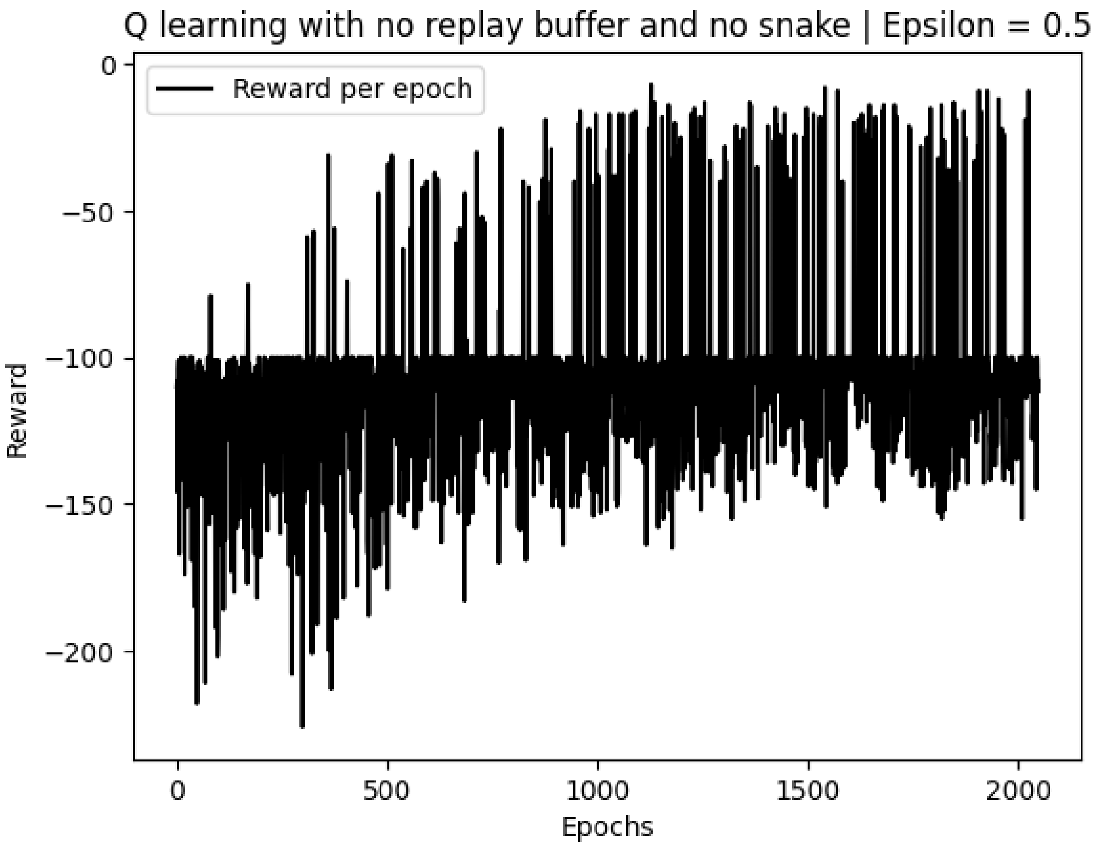
```
    Q, rewards_history = run_q_learning(epochs=epochs, epsilon=epsilon,␣
↪replay_buffer_enabled=False, with_snake_pit=False)
    # Plot rewards
    plt.plot(x, rewards_history, color='black', label=f"Reward per epoch")
    plt.xlabel('Epochs')
    plt.ylabel('Reward')
    plt.legend()
    plt.title(f"Q learning with no replay buffer and no snake | Epsilon =␣
↪{epsilon}")
    plt.show()

    world = get_world(with_snake_pit=False)
    for position in Q:
        actions_utilities = Q[position]
        if world[position[0]][position[1]] not in [GOAL, START, CLIFF]:
            world[position[0]][position[1]] = ACTIONS[np.
↪argmax(actions_utilities)]

    # Printing world
    print_world(world)

    print("-" * 80)
```



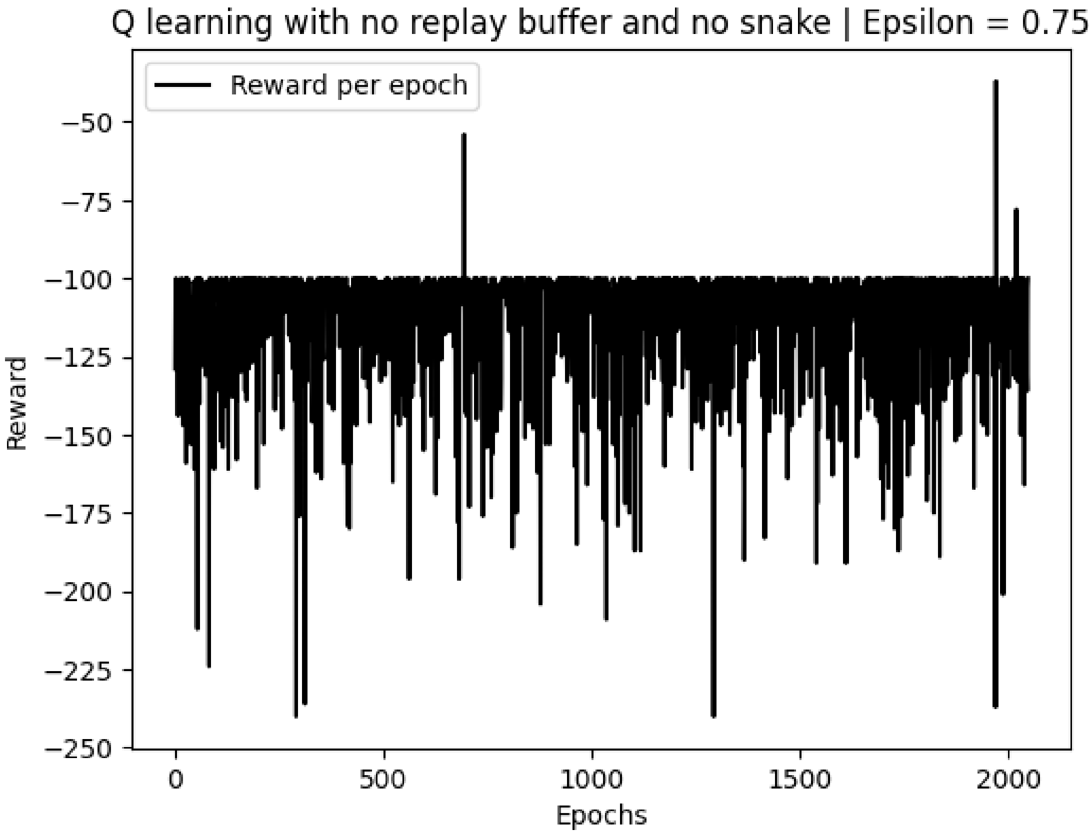
```
^ ^ < > > > v v ^ v > v > > v > v v > > v
> > > > v v > v > v > v v > v > v v v > v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
----------------------------------------
```



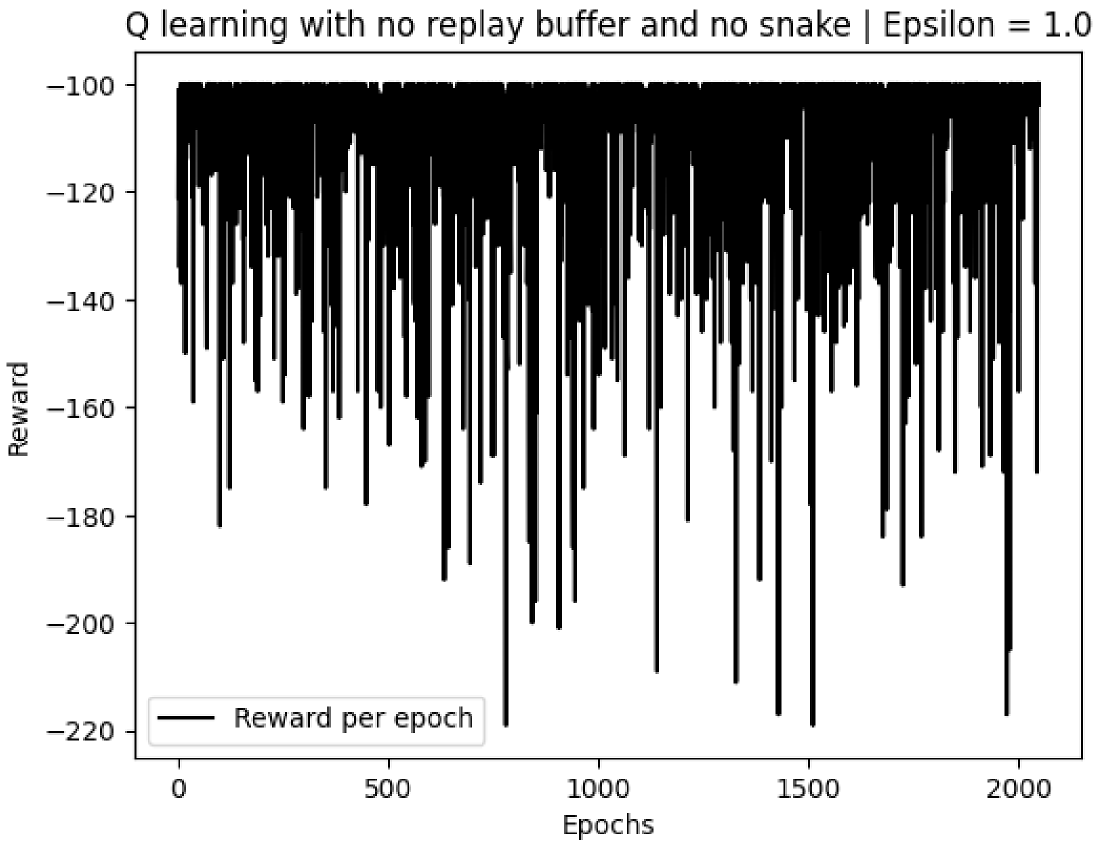

```
v > v > v v v > v v v v v > v v v v > v v
v v v > > > > > > > > > v > > v > v > > v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
----------------------------------------
```




```
v > > > v > > > > > v v > v v > v v v v v
> > > > > > > > > > > > > > > > > v v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
> > > > v > > > > > > > > > ^ > > v > > ^
> > > > > > > > > > > > > > v ^ > > v v v
> > > > > > > > > > > > > > > > ^ < > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```



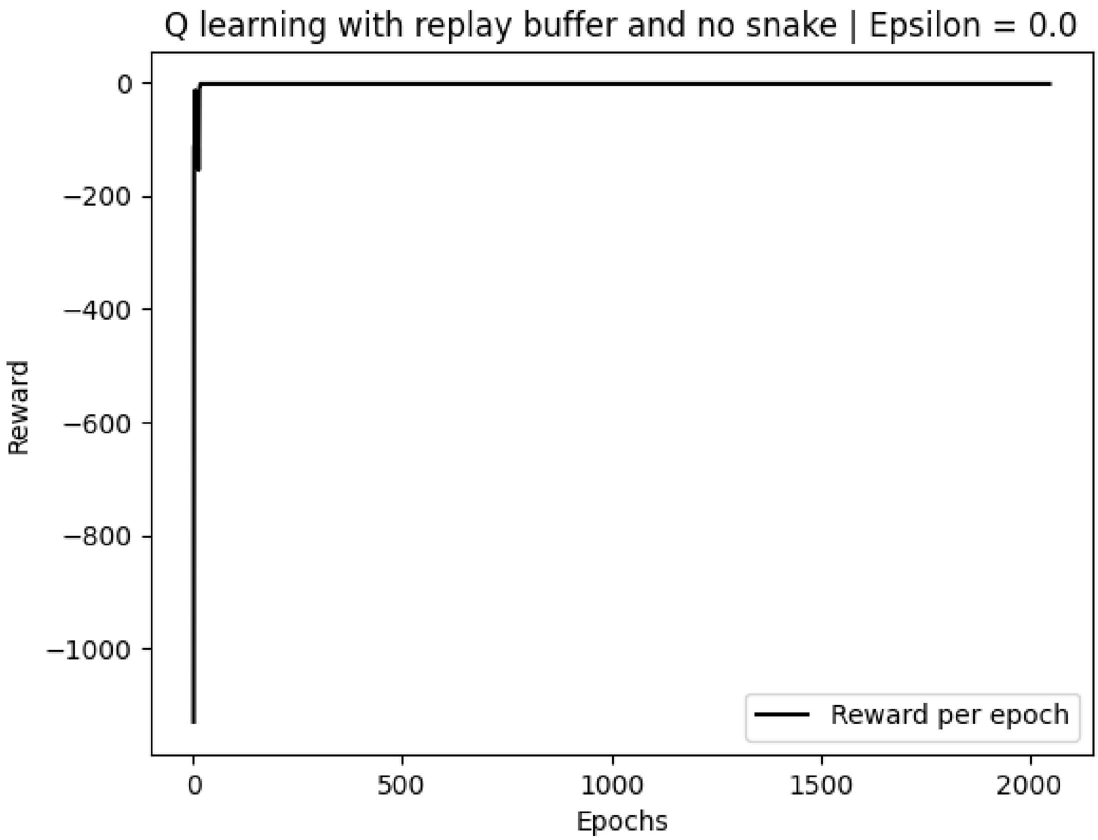

```
> > > > > > > > > > < ^ < > v ^ ^ ^ ^ ^ ^
> > > v v v > v > v > ^ < v > ^ ^ ^ ^ ^ ^
> > > > > > > > > > < ^ > ^ ^ ^ ^ ^ ^ ^ ^
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```

2.4.3 Q learning with replay buffer and no snake

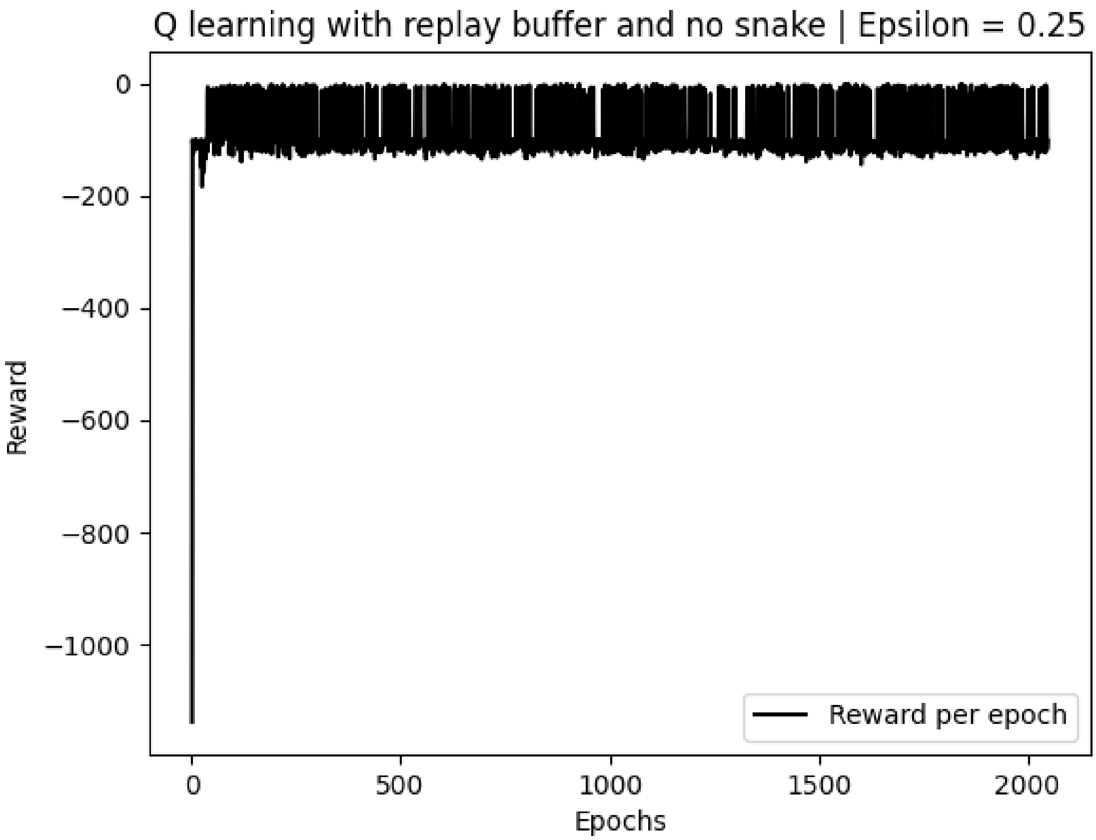
- Replay buffer is an element of stability to Q-learning execution.
- We observe that for a high value of epsilon = 0.75 the agent managed to find a viable (in particular, optimal) policy to the goal state.
- Clearly, replay buffer is an improvement to the normal version, but for epsilon = 1, it didn't manage to construct the proper policy. The reasons for that are:
  - `Chance`, because due to a completely random exploration, it might just happen (Given 2048 epochs) for the agent to never find the path to the goal state.
  - `Replay buffer hyperparameters` (REPLAY_BUFFER_SIZE = 2048, REPLAY_BUFFER_BATCH_SIZE = 1024). I argue that by fine tuning replay buffer's hyperparameters and by experimenting some epochs in which the agent manages to find a way to the goal under the complete randomness of epsilon=1, then a proper policy might be derived.

[28]:

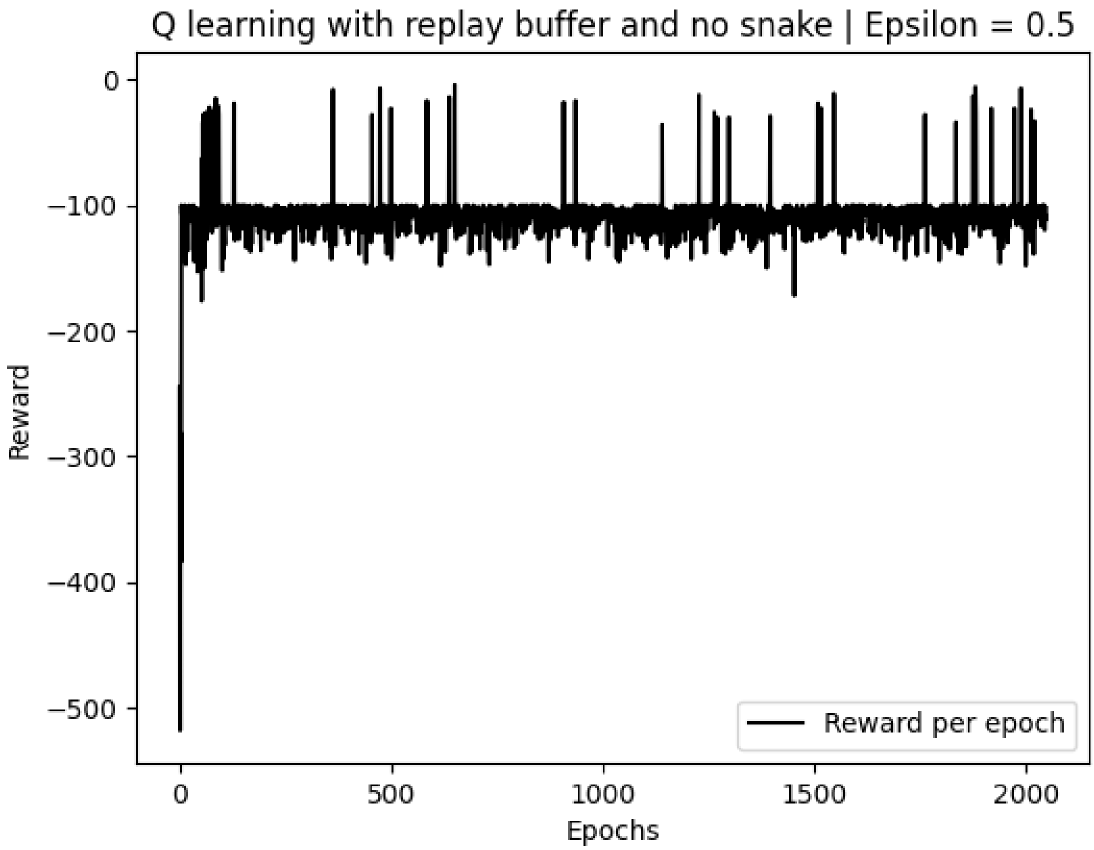
```
epochs = 2048
x = np.linspace(1, epochs, epochs)

for epsilon in [0.0, 0.25, 0.5, 0.75, 1.0]:
    Q, rewards_history = run_q_learning(epochs=epochs, epsilon=epsilon,␣
 ↪replay_buffer_enabled=True, with_snake_pit=False)
    # Plot rewards
    plt.plot(x, rewards_history, color='black', label=f"Reward per epoch")
    plt.xlabel('Epochs')
    plt.ylabel('Reward')
    plt.legend()
    plt.title(f"Q learning with replay buffer and no snake | Epsilon =␣
 ↪{epsilon}")
    plt.show()

    world = get_world(with_snake_pit=False)
    for position in Q:
        actions_utilities = Q[position]
        if world[position[0]][position[1]] not in [GOAL, START, CLIFF]:
            world[position[0]][position[1]] = ACTIONS[np.
 ↪argmax(actions_utilities)]

    # Printing world
    print_world(world)

    print("-" * 80)
```



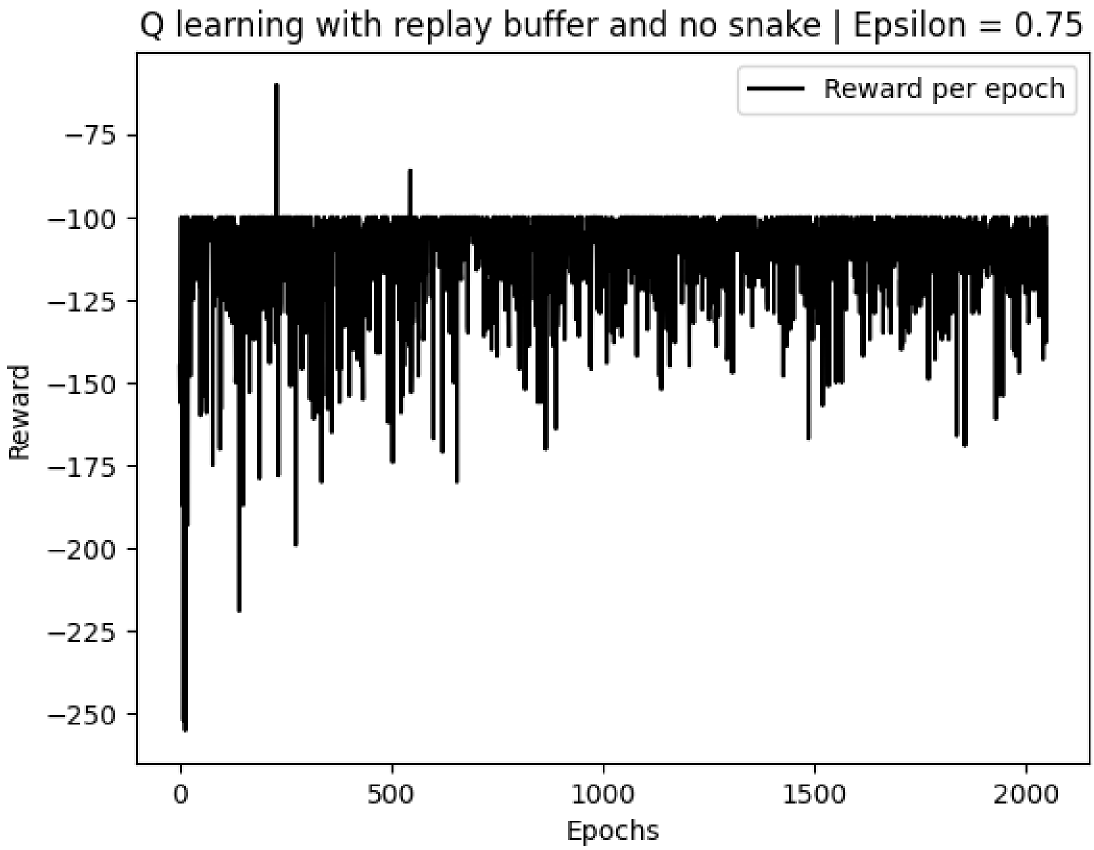

```
> > > > > v v v v v v v v v v v v v v v v
> > > > > > > > > > > > > > > > > > > > v
> > > > > > > > > > > > ^ ^ ^ ^ ^ ^ ^ ^ v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```



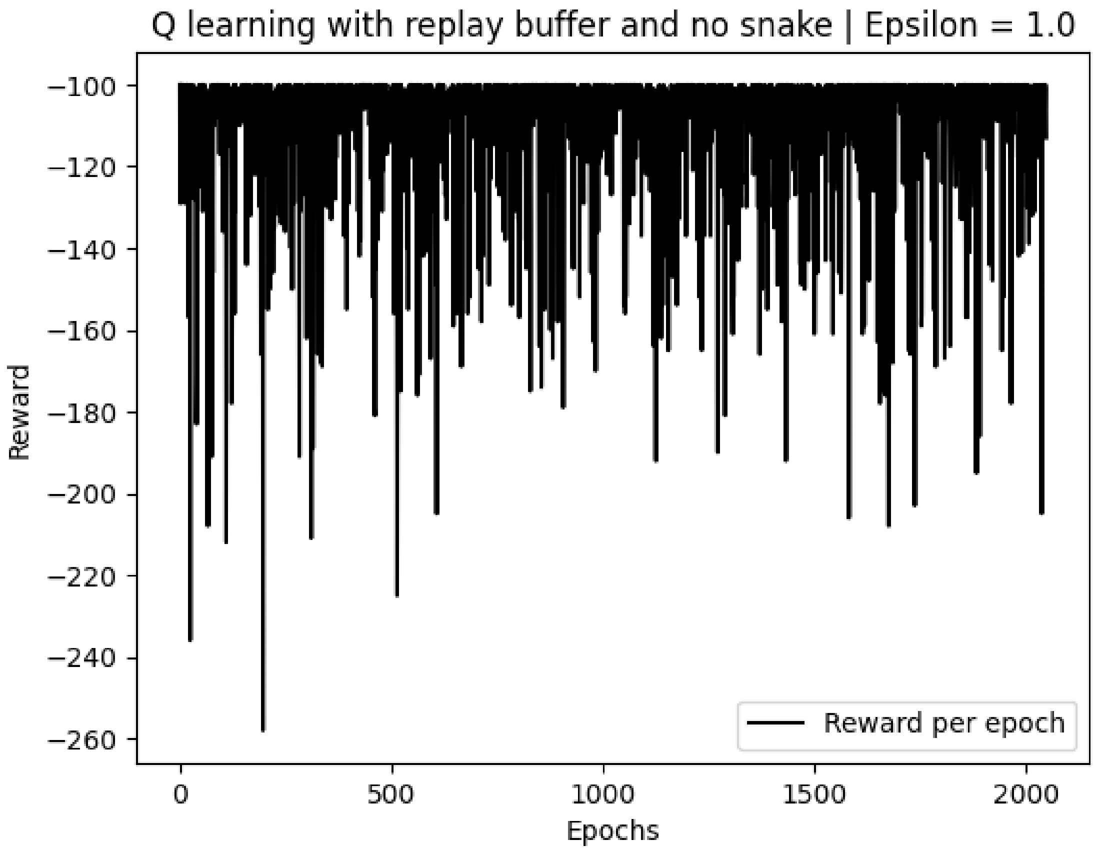

```
v v v v v v v v v v v v v v v v v v v v v
v v v v v v v v v v v v v v v v v v v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
v v v v v v v v v v v v v v v v v v v v v
v v v v v v v v v v v v v v v v v v v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
v v v v v v v v v v v v v v v v v > v v v
v v v v v v v v v v v v v v v v v v v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```



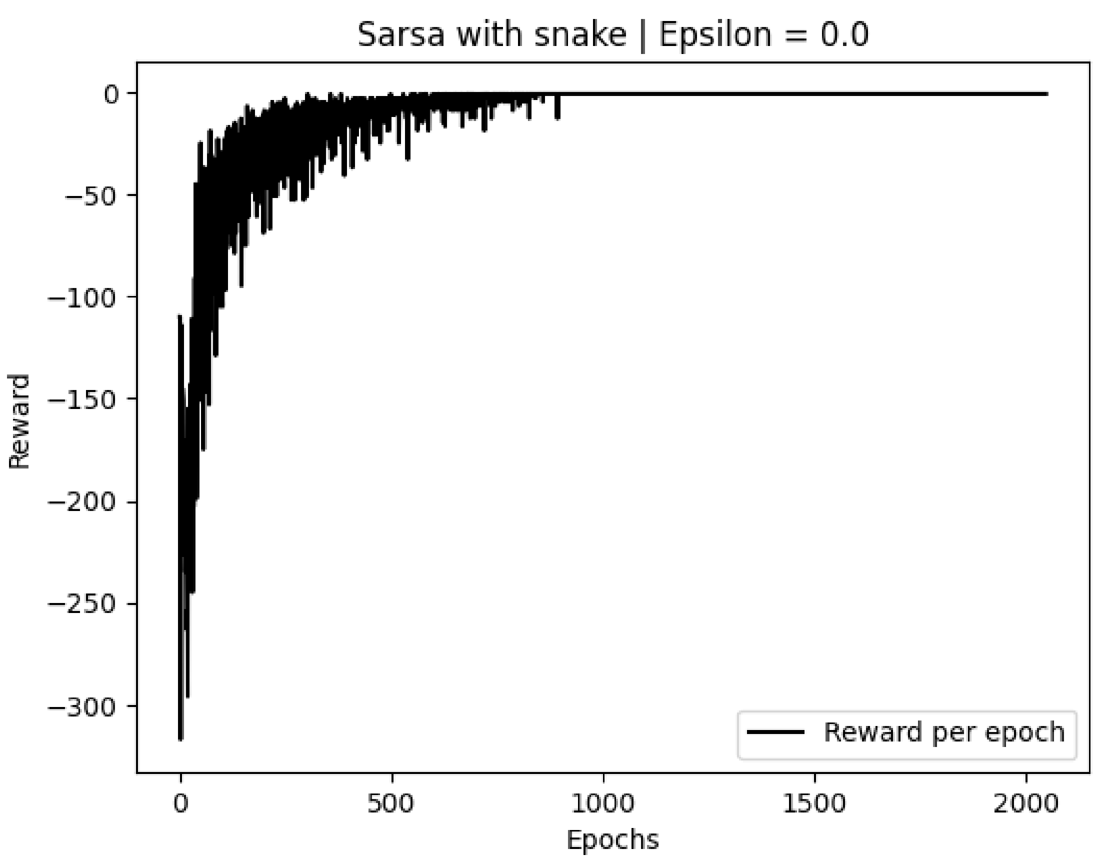

```
> > > > > > > > > > > > ^ > v ^ ^ ^ ^ ^ ^
v v v v v v v v v v v v ^ ^ v ^ ^ ^ ^ ^ ^
> > > > > > > > > > > ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```

2.4.4 Sarsa with snake

- Adding the snake created some disturbances around its position, but the agent is still managing to derive the right `safe policy` for small values of epsilon.
- For high values of epsilon we experience the same situation as in with no snake situation.

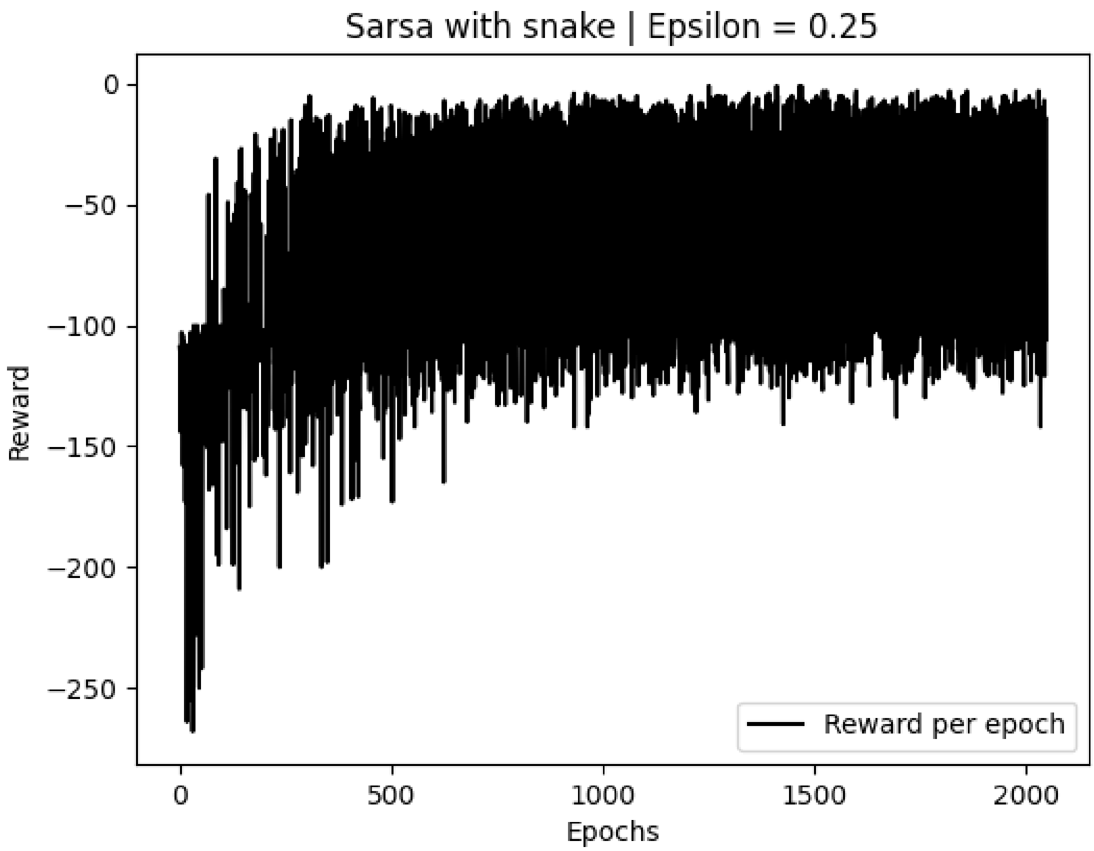
[29]:
```
epochs = 2048
x = np.linspace(1, epochs, epochs)

for epsilon in [0.0, 0.25, 0.5, 0.75, 1.0]:
    Q, rewards_history = run_sarsa(epochs=epochs, epsilon=epsilon,␣
 ↪with_snake_pit=True)
    # Plot rewards
    plt.plot(x, rewards_history, color='black', label=f"Reward per epoch")
```

```
    plt.xlabel('Epochs')
    plt.ylabel('Reward')
    plt.legend()
    plt.title(f"Sarsa with snake | Epsilon = {epsilon}")
    plt.show()

    world = get_world(with_snake_pit=True)
    for position in Q:
        actions_utilities = Q[position]
        if world[position[0]][position[1]] not in [GOAL, START, CLIFF,␣
↪SNAKE_PIT]:
            world[position[0]][position[1]] = ACTIONS[np.
↪argmax(actions_utilities)]

    # Printing world
    print_world(world)

    print("-" * 80)
```

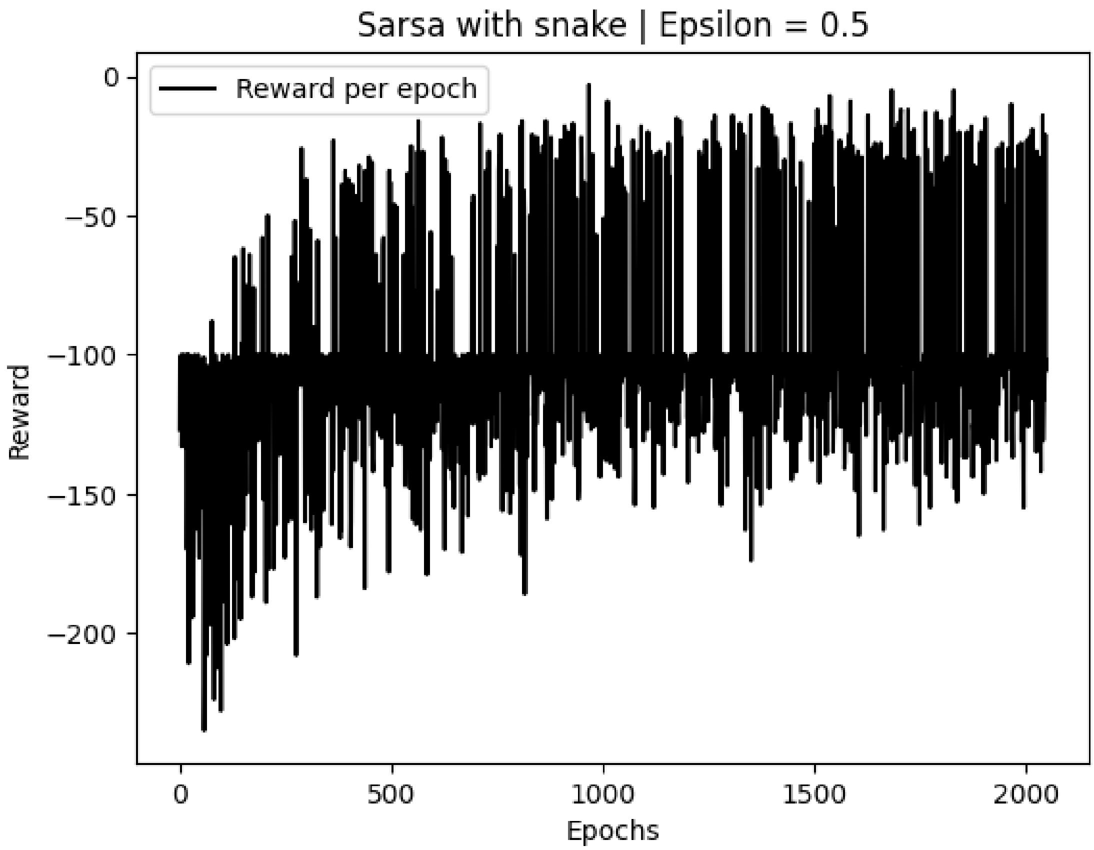



```
v > > > > > > > > > > P > > > > > > > > v
```

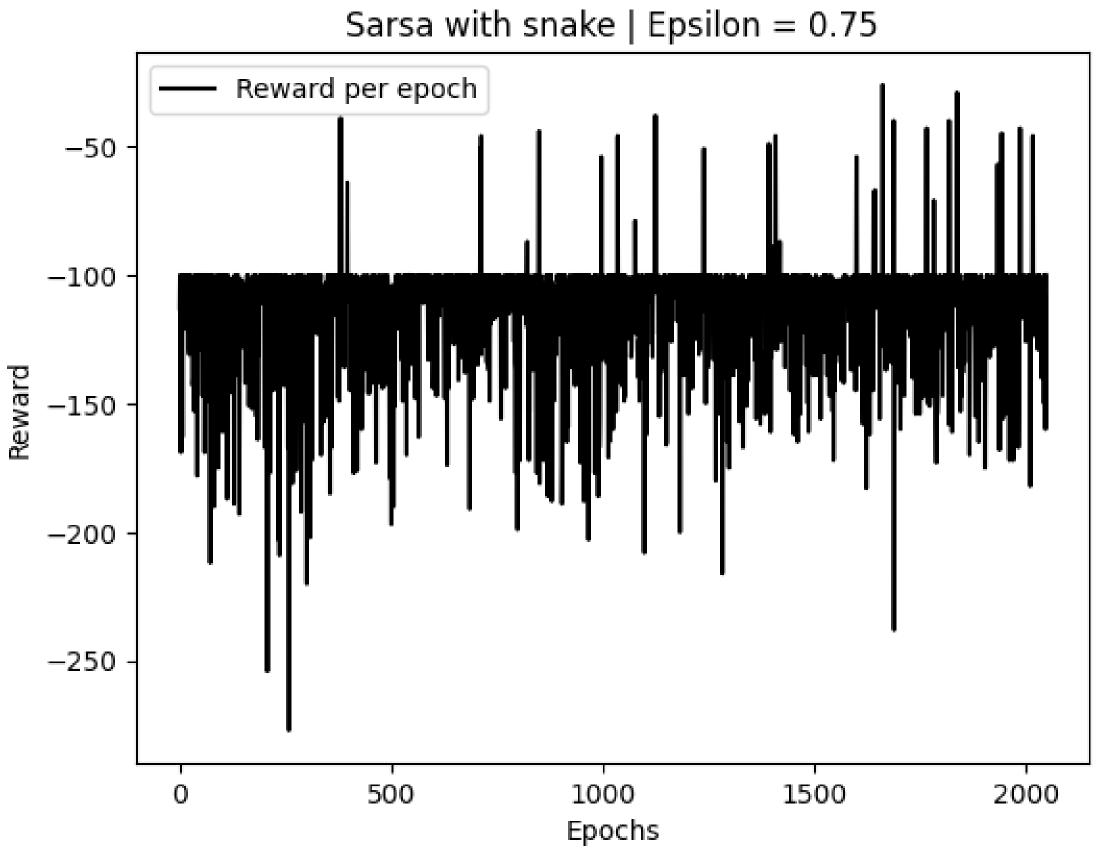
```
> > > > > > > > > > > > > > > > > > > > v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
```



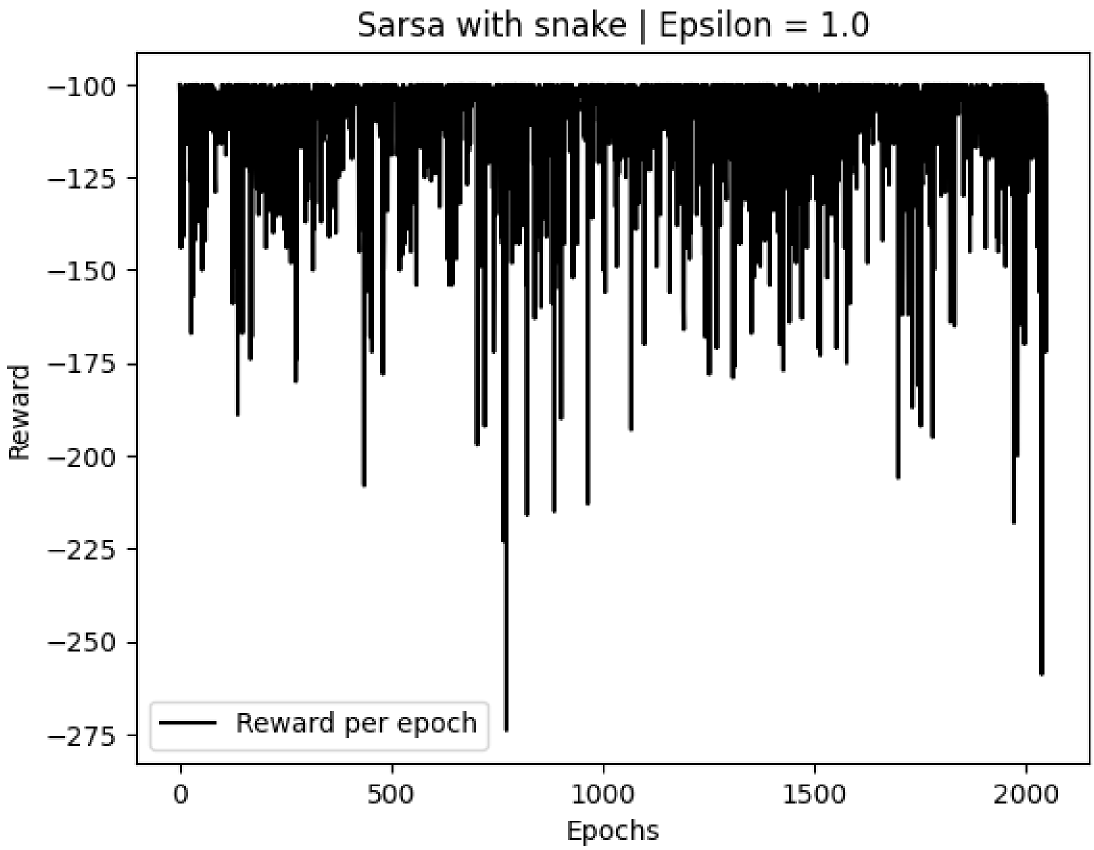

```
v > > > > > > > > > > P > > > > > > > > v
> ^ > > > > > > > > > ^ > > > > > > ^ ^ v
> > > > > > > > > > > > > > > > > > > < v
S C C C C C C C C C C C C C C C C C C C G
```




```
> > ^ > > > > > > > > P > > ^ > > > > > v
^ > > ^ > > > > > > > ^ ^ > > > ^ > ^ ^ v
> > > > > > > > > > > > > > < > > > > ^ v
S C C C C C C C C C C C C C C C C C C C G
-------------------------------------------------------------------------------
```




```
> > > > > > > > > > > P > > > > > ^ > > v
> > > > > > > > > > > > > > > > > > > < v
> > > > > > > > > > > > > > > > > > < < v
S C C C C C C C C C C C C C C C C C C C G
-------------------------------------------------------------------------------
```



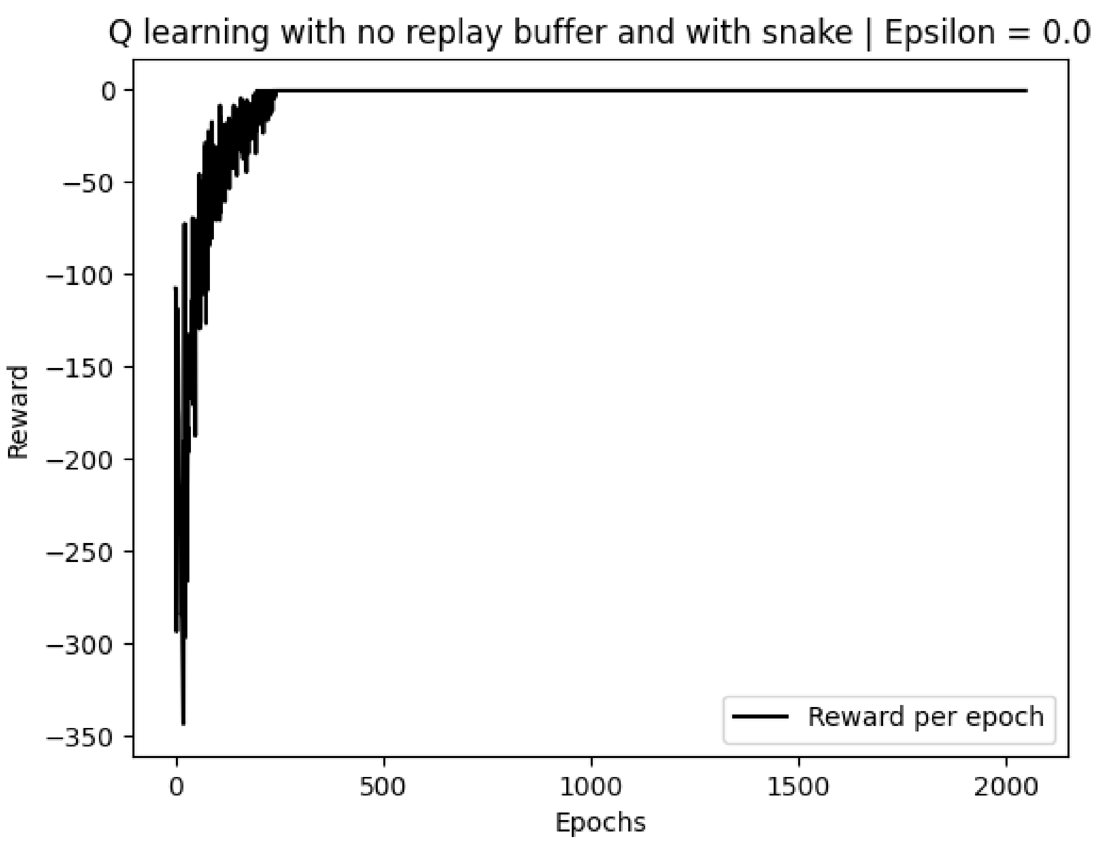

```
> > > > > > > > > > > P < ^ ^ ^ ^ ^ ^ ^ ^
> > > > > > > > > > ^ < ^ v ^ v ^ ^ ^ ^ ^
> > > > > > > > > ^ < < > ^ ^ ^ ^ ^ ^ ^ ^
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```

2.4.5 Q learning with no replay buffer and with snake

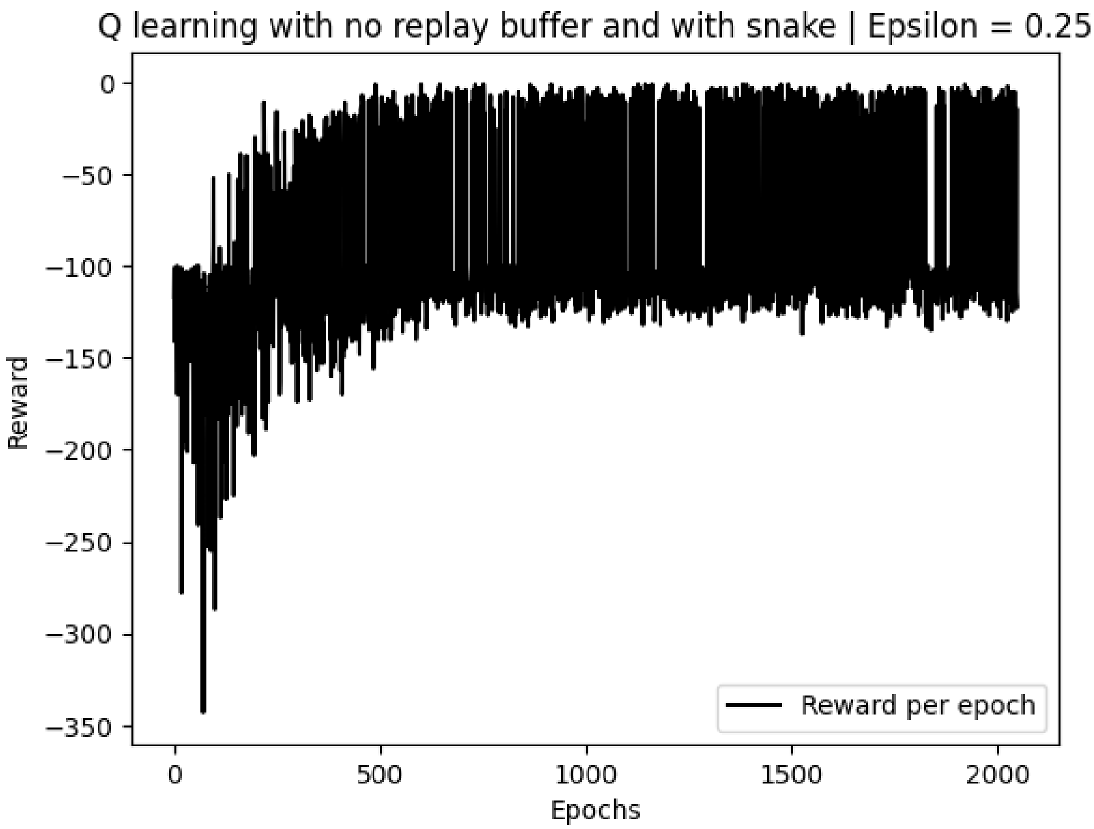
- We observe that the snake doesn't affect the agent's ability to derive the `optimal policy` to the goal state.
- For higher values of epsilon (same case as before), the agent (arguably, falls down the cliff) fails to construct an appropriate policy to the goal state.

[30]:
```
epochs = 2048
x = np.linspace(1, epochs, epochs)

for epsilon in [0.0, 0.25, 0.5, 0.75, 1.0]:
    Q, rewards_history = run_q_learning(epochs=epochs, epsilon=epsilon,␣
 ↪replay_buffer_enabled=False, with_snake_pit=True)
    # Plot rewards
```

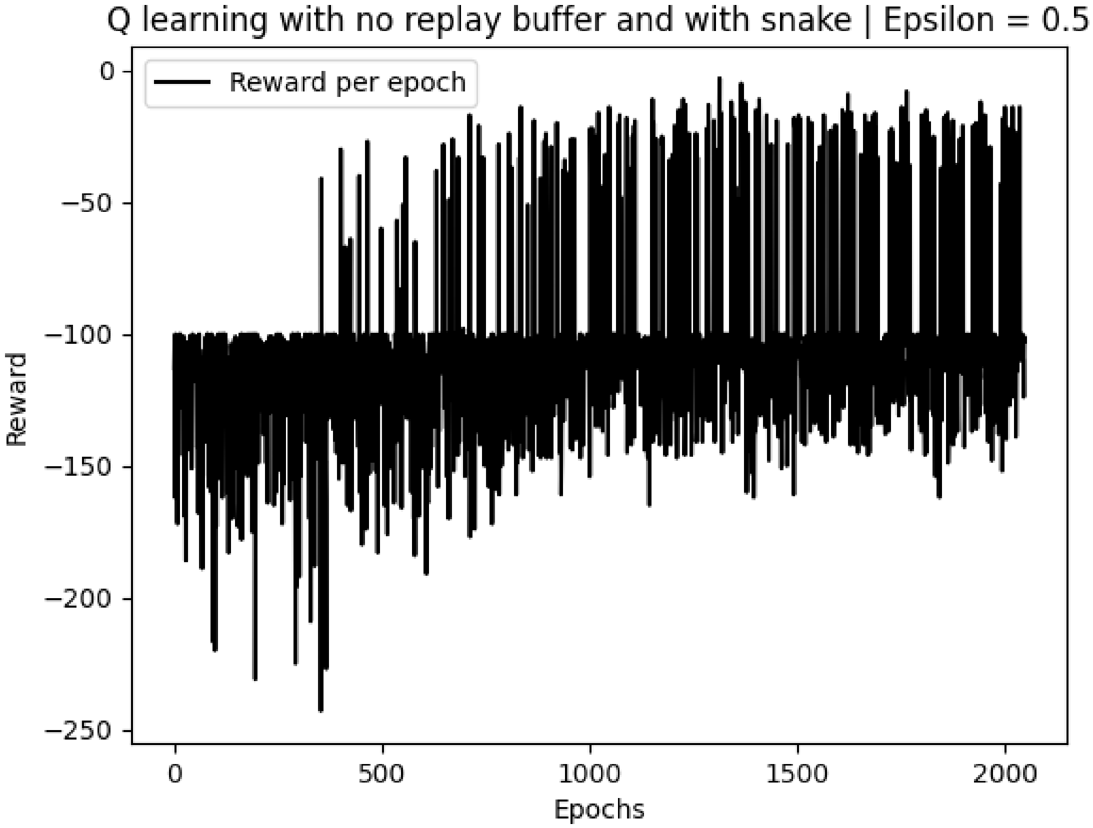
```
    plt.plot(x, rewards_history, color='black', label=f"Reward per epoch")
    plt.xlabel('Epochs')
    plt.ylabel('Reward')
    plt.legend()
    plt.title(f"Q learning with no replay buffer and with snake | Epsilon =␣
↪{epsilon}")
    plt.show()

    world = get_world(with_snake_pit=True)
    for position in Q:
        actions_utilities = Q[position]
        if world[position[0]][position[1]] not in [GOAL, START, CLIFF,␣
↪SNAKE_PIT]:
            world[position[0]][position[1]] = ACTIONS[np.
↪argmax(actions_utilities)]

    # Printing world
    print_world(world)

    print("-" * 80)
```



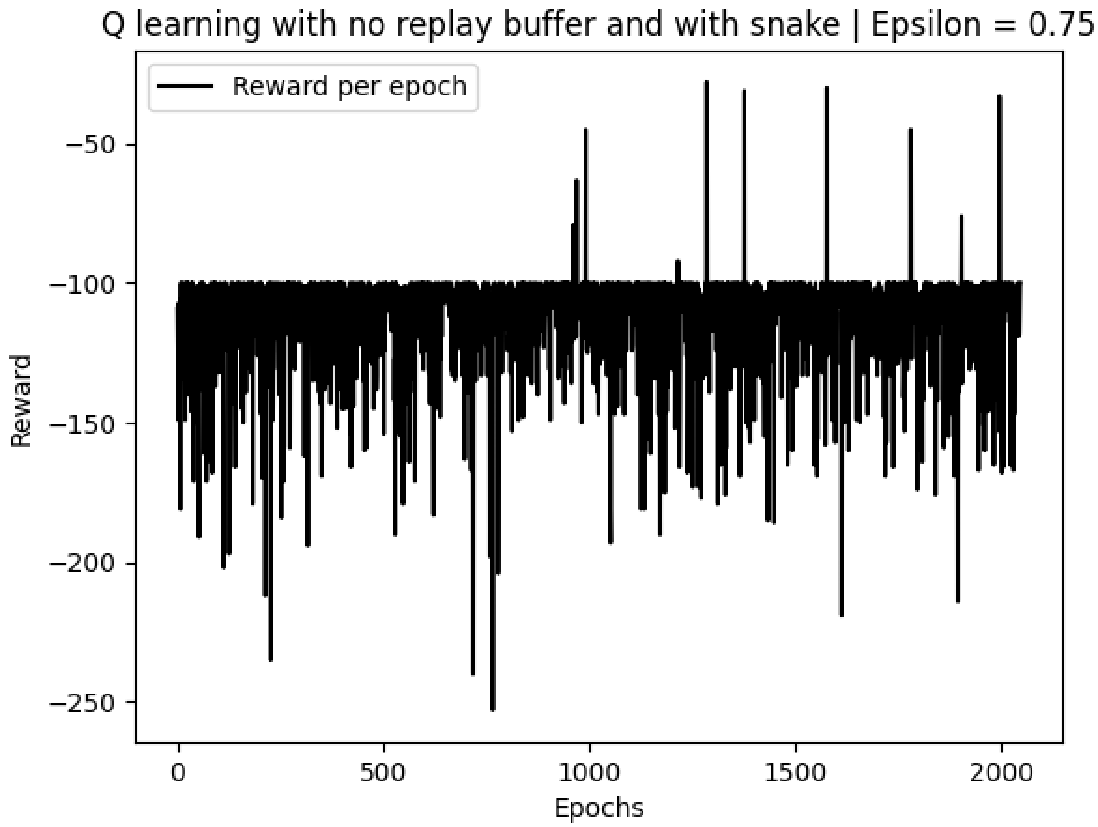
```
^ ^ < > > > v v ^ v > P > > v > v v > > v
> > > > v v > v > v > v v > v > v v v > v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```



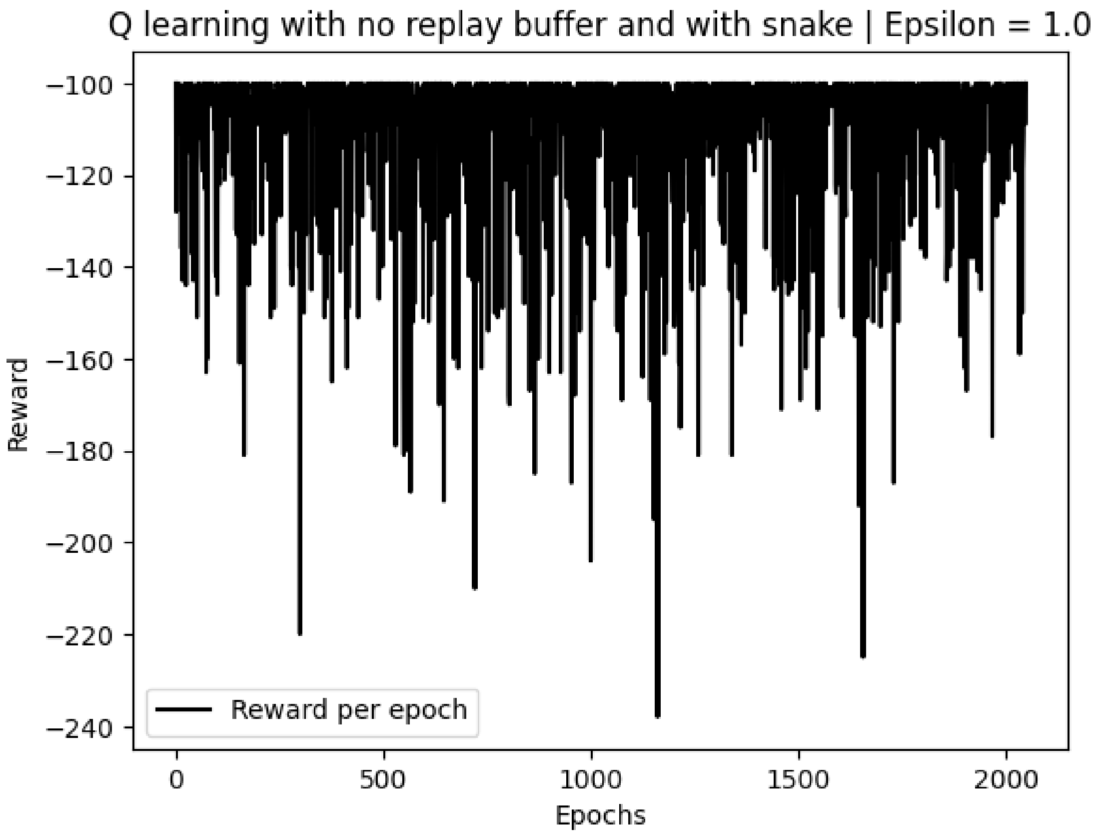

```
v v v > v v v > > > > P v > v v v v v v v
v v v v > > > > > > > > v v v v v > v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
v v > > > > > > > > > > P > v > v v v v v v
> > > > > > > > > > > > > > > > > > v > v
> > > > > > ^ > > ^ > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
v > > > > > > > > v > P v > v < > > ^ v v
> > > > > > > v > v > > > > v v > ^ v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```



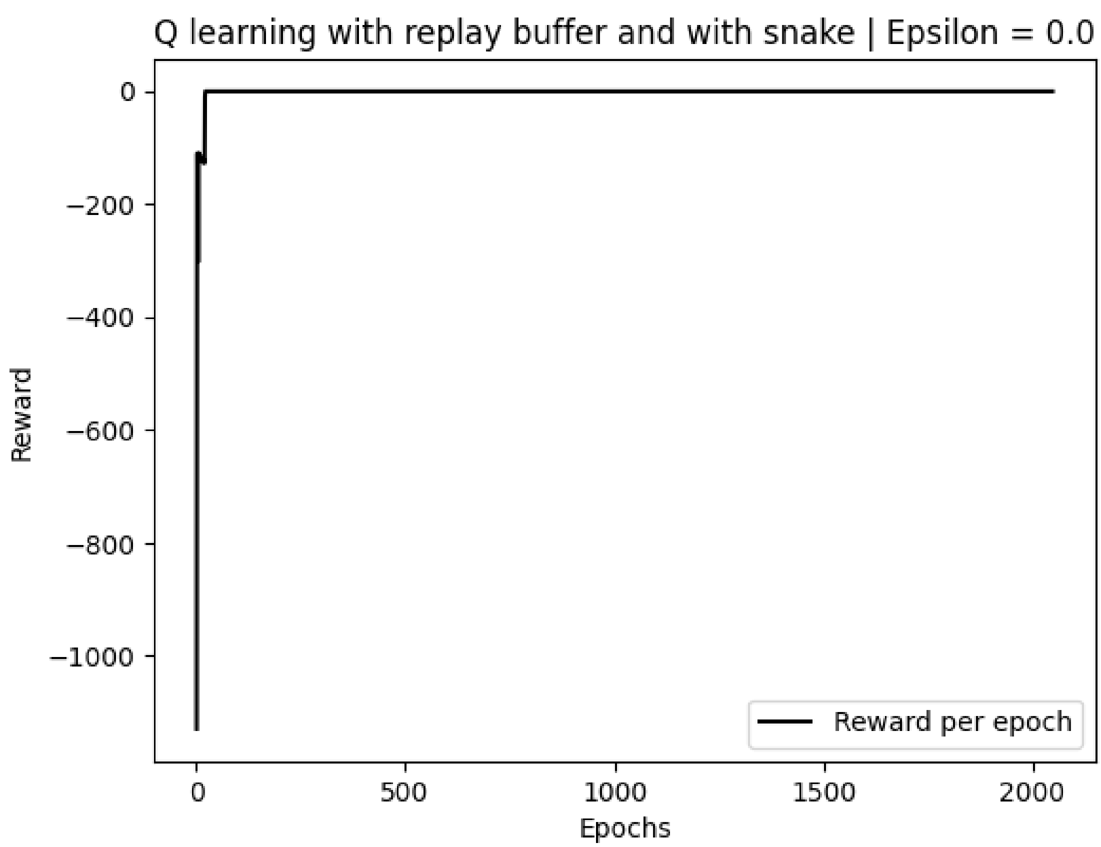

```
> > > > > > > > > > > P v v ^ > ^ ^ ^ ^ ^
> > > > v v v ^ > > v > ^ ^ v ^ ^ ^ ^ ^ ^
> > > > > > > > > > > ^ ^ > ^ ^ ^ ^ ^ ^ ^
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```

2.4.6 Q learning with replay buffer and with snake

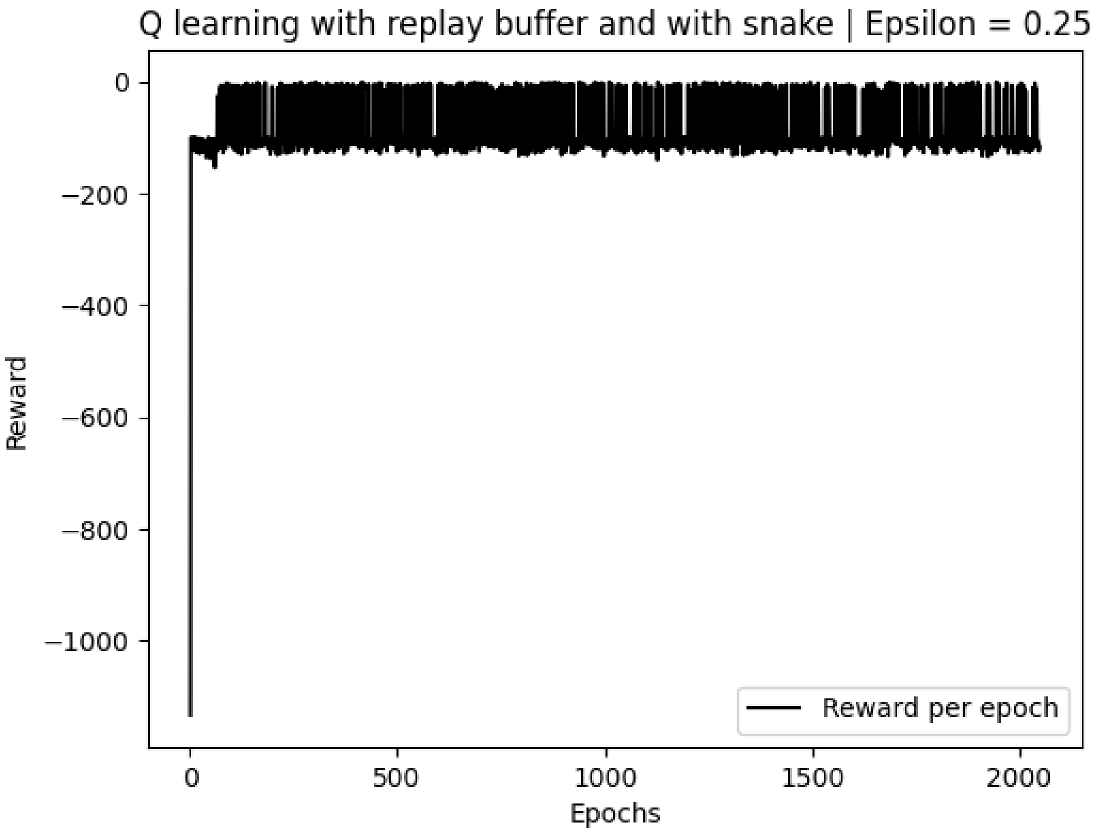
- Same situation as before, the agent manages to construct the `optimal policy` under low values of epsilon.
- The snake pit seems to not have any quantifiable effect on the policy derivation process.
- Under bigger values of epsilon, the same problems(`Chance` and `Hyperparameters`) prevent the agent to derive a `valid policy`.

[31]:
```python
epochs = 2048
x = np.linspace(1, epochs, epochs)

for epsilon in [0.0, 0.25, 0.5, 0.75, 1.0]:
    Q, rewards_history = run_q_learning(epochs=epochs, epsilon=epsilon,␣
 ↪replay_buffer_enabled=True, with_snake_pit=True)
    # Plot rewards
```

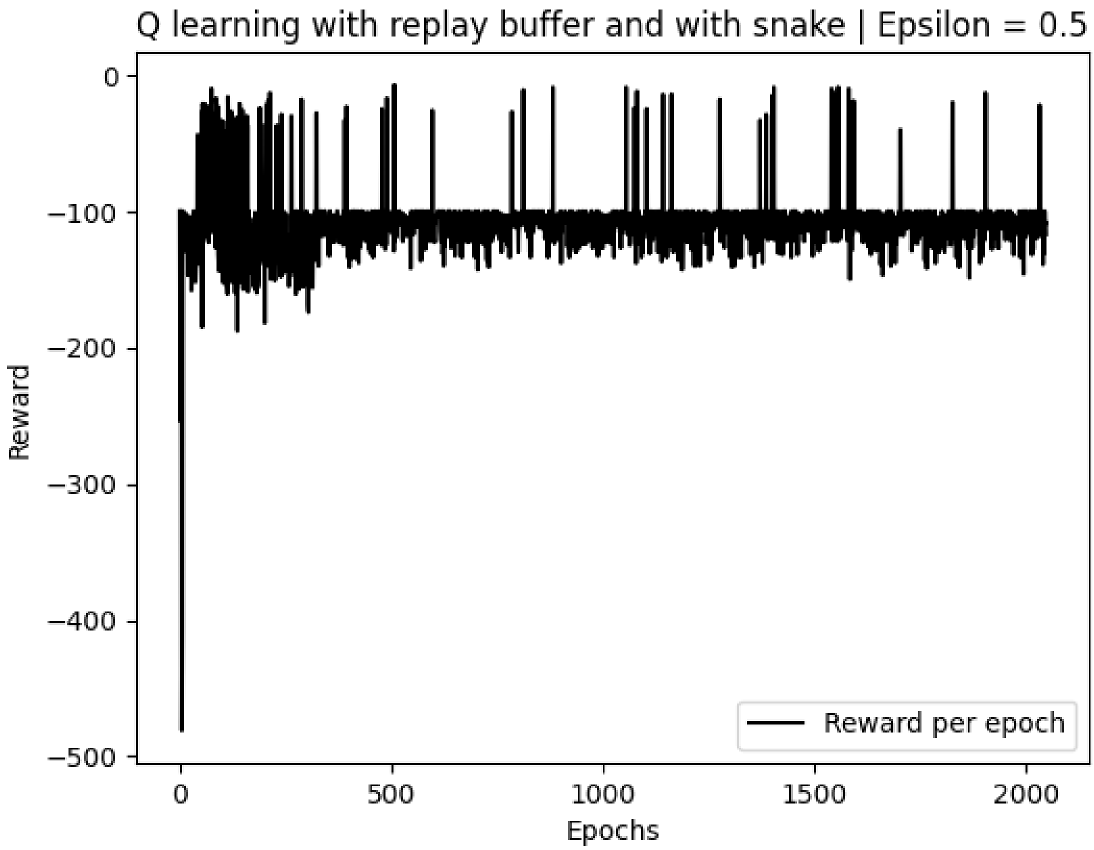
```
    plt.plot(x, rewards_history, color='black', label=f"Reward per epoch")
    plt.xlabel('Epochs')
    plt.ylabel('Reward')
    plt.legend()
    plt.title(f"Q learning with replay buffer and with snake | Epsilon =␣
↪{epsilon}")
    plt.show()

    world = get_world(with_snake_pit=True)
    for position in Q:
        actions_utilities = Q[position]
        if world[position[0]][position[1]] not in [GOAL, START, CLIFF,␣
↪SNAKE_PIT]:
            world[position[0]][position[1]] = ACTIONS[np.
↪argmax(actions_utilities)]

    # Printing world
    print_world(world)

    print("-" * 80)
```



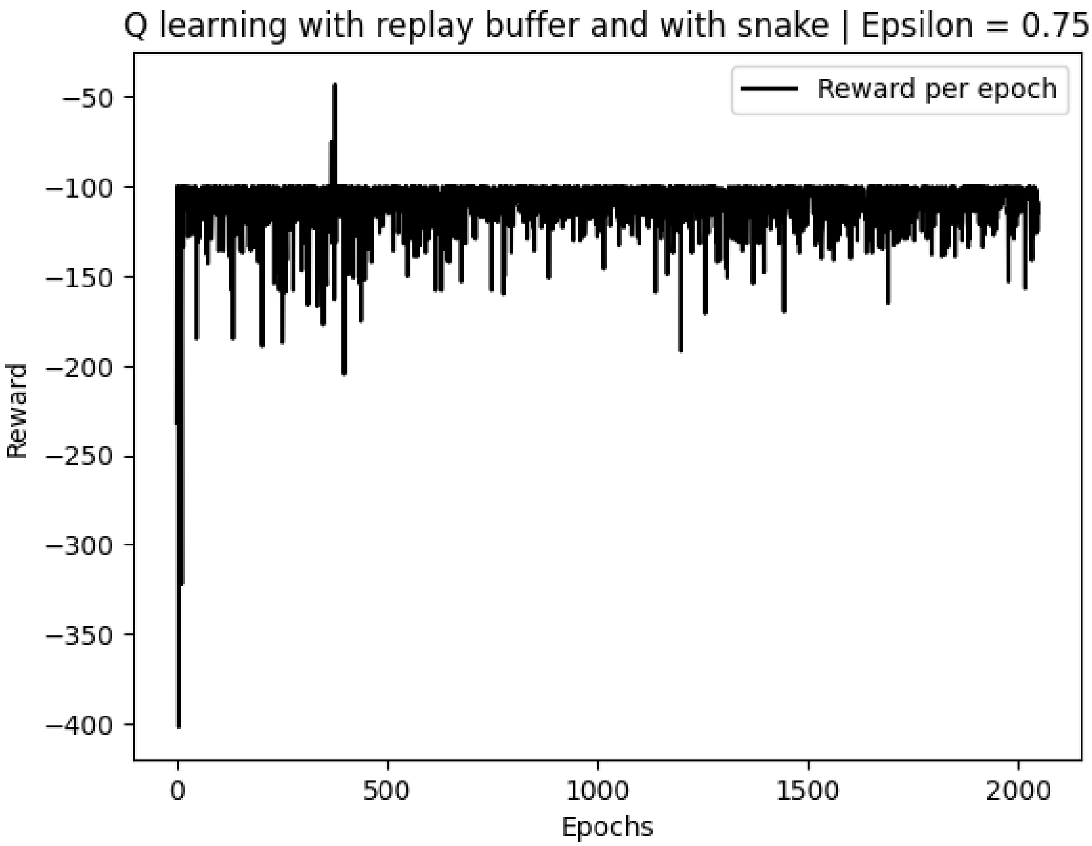
```
v v v v v v v v v v v P v v v v v v v v v
v v v v v v v v v v v v v v v v v v v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```



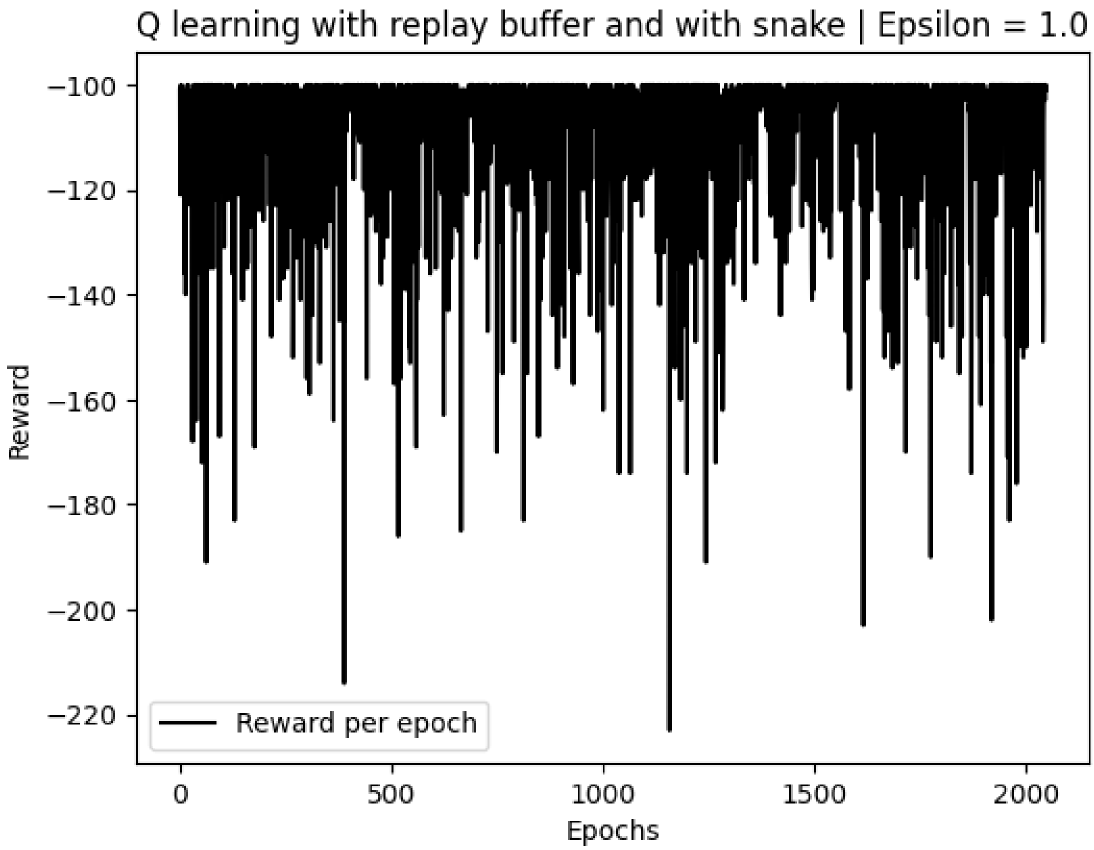

```
v v v v v v v v v v v P v v v v v v v v v
v v v v v v v v v v v v v v v v v v v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
v v v v v v v v v v v P v v v v v v v v v
v v v v v v v v v v v v v v v v v v v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
v v v v v v v v v v v P v v v v v v v v v
v v v v v v v v v v v v v v v v v v v v v
> > > > > > > > > > > > > > > > > > > > v
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```




```
> > > > > > > > ^ v > P > ^ ^ ^ ^ ^ ^ ^ ^
> > > > > > > > > ^ > ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
> > > > > > > > ^ ^ ^ ^ ^ v v ^ ^ ^ ^ ^ ^
S C C C C C C C C C C C C C C C C C C C G
--------------------------------------------------------------------------------
```