Reinforcement Learning with Frozen Lake

José Antonio Sánchez González

Gymnasium

Problem definition

In this Reinforcement Learning Framework, the algorithm involves crossing a frozen lake from Start(s) to Goal(G) without falling into any Holes(H) by walking over the Frozen(F) lake. The agent may not always move in the intended direction due to slippery nature of the frozen lake.

The agent takes a 1-element vector for actions. The observation is a value representing the agent's current position as current_row * nrows + current_col (it starts at 0). For example, the goal position in the 4x4 map can be calculated as follows: 3 * 4 + 3 = 15.

The number of possible observations is dependent on the size of the map. For us is going to be an 8x8 map.

Algorithms we use:

Epsilon-Greedy Algorithm

```
eps=1
if random() < eps:
  select random action
else:
  select best action

#At the end of each episode:
eps = eps - decay rate
```

Q-Table

Q-Learning Formula

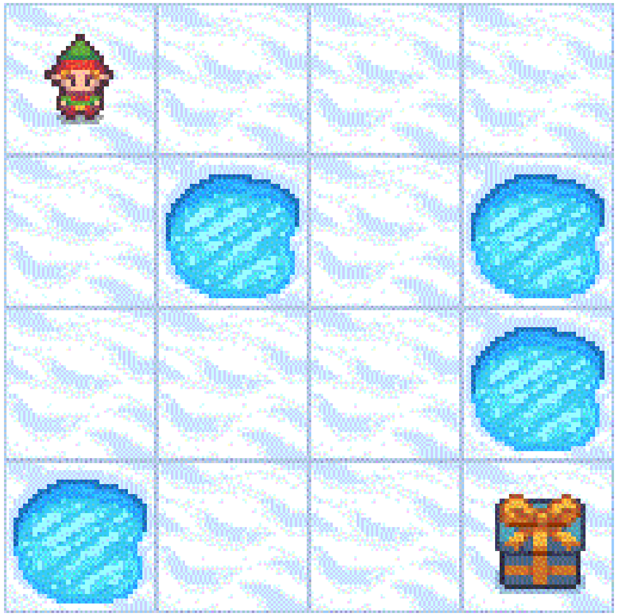
```
q[state,action]=q[state,action]+learning_rate*
(reward+discount_factor *max(q[new_state,:]) -
q[state,action])
```

We are gonna work with **discrete data**, that is going to be stored in a file that is actually gonna be the q-table, to train the model.

Frozen Lake

Frozen Lake is part of the Toy Text environments.

```python
import gymnasium as gym
gym.make("FrozenLake-v1")
```



Frozen lake involves crossing a frozen lake from Start(S) to Goal(G) without falling into any Holes(H) by walking over the Frozen(F) lake. The agent may not always move in the intended direction due to the slippery nature of the frozen lake.

Action space:

- 0: LEFT
- 1: DOWN
- 2: RIGHT
- 3: UP

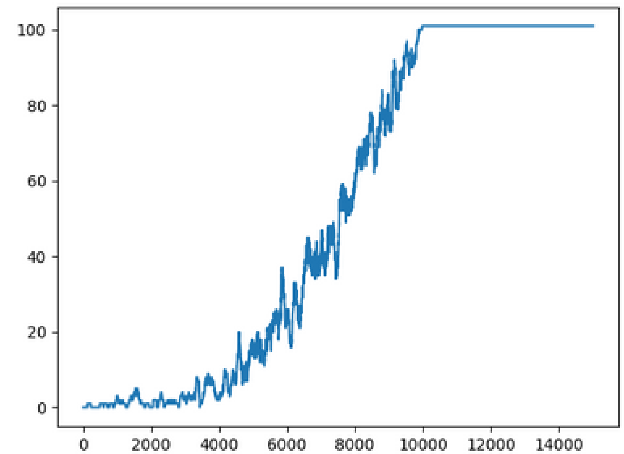
for more information read [gymlilbrary] documentation.

FrozenLake: 15,000 episodes

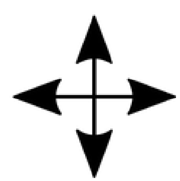
Y-axis for reward and X-axis for # of episodes.

we can see that at 100 episodes is not nearly enough to learn. As the epsilon decreases, at 5000 episodes it starts to use the Q-table to learn.

At 10,000 episodes, epsilon is 0 this means it is only gonna follow the Q-table and it's not gonna learn anymore.

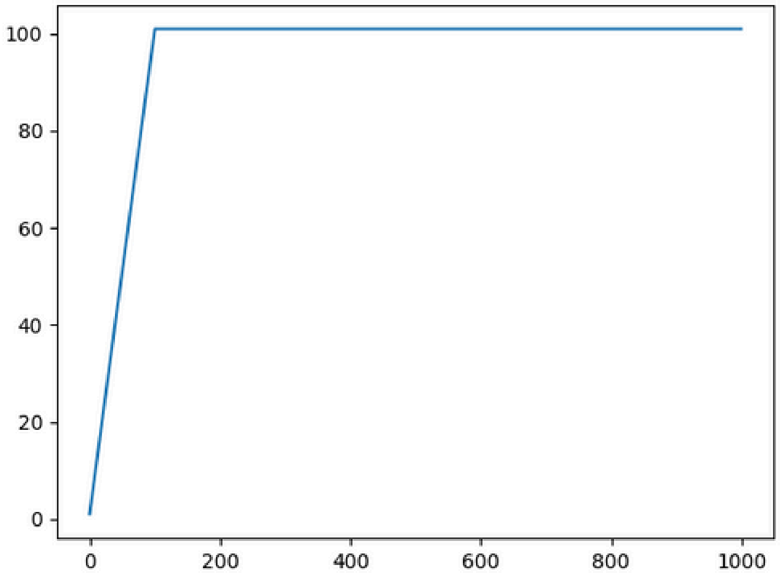


after 10,000 episodes we get to the goal a 100% of the time!



from 0 to 100 it’s going to look like an increasing line.

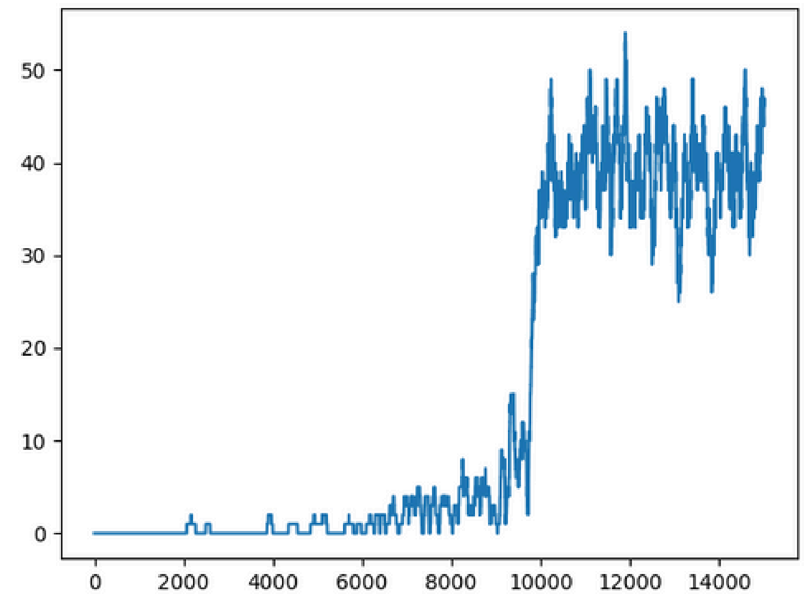
Then it gets 100% reward.



running for 1000 episodes

If we run 15,000 episodes and with slippery factor we get this result.

barely after 8,000 episodes it starts getting rewards.



the slippery factor makes training a lot more difficult!

Here is the full code if you want to peek at it.

```python
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt
import pickle

def run(episodes, is_training=True, render=False):

    env = gym.make("FrozenLake-v1", map_name= "8x8", is_slippery=False, render_mode="human" if render else None)

    if(is_training):
        q = np.zeros((env.observation_space.n, env.action_space.n)) #init a 64x4 array
    else:
        f = open('frozen_lake8x8.pkl', 'rb')
        q = pickle.load(f)
        f.close()


    learning_rate_a = 0.9 #alpha or learnig rate
    discount_factor_g = 0.9 #gamma or discount factor

    epsilon = 1 # 1=100% random actions
    epsilon_decay_rate = 0.0001 # epsilon decay rate
    rng = np.random.default_rng() # random number generator

    rewards_per_episode = np.zeros(episodes)

    for i in range(episodes):
        state = env.reset()[0]  #states: 0 to 63, 0=top left corner, 63=bottom right corner
        terminated = False      #True hwne fall in hole or reached goal
        truncated = False       #True when actions > 200

        while(not terminated and not truncated):
            if is_training and rng.random() < epsilon:
                action = env.action_space.sample() # actions: 0=left, 1=down, 2=right, 3=up
            else:
                action = np.argmax(q[state,:])

            new_state, reward, terminated, truncated, _ = env.step(action)

            if is_training:
                q[state,action] = q[state,action] + learning_rate_a * (
                    reward + discount_factor_g * np.max(q[new_state,:]) - q[state,action]
                )

            state= new_state

        epsilon = max(epsilon - epsilon_decay_rate, 0)

        if(epsilon==0):
            learning_rate_a = 0.0001

        if reward == 1:
            rewards_per_episode[i] = 1

    env.close()

    sum_rewards = np.zeros(episodes)
    for t in range(episodes):
        sum_rewards[t] = np.sum(rewards_per_episode[max(0, t-100):(t+1)])
    plt.plot(sum_rewards)
    plt.savefig('frozen_lake8x8.png')

    if is_training:
        f = open("frozen_lake8x8.pkl", "wb")
        pickle.dump(q, f)
        f.close()

if __name__ == '__main__':
    #run(15000)

    run(1000, is_training=True, render=False)
```

Conclusions

- After we trained and tested the data there are instances where the goal is not met, it is highly dependant of the # of episodes and if you activate the slippery factor.

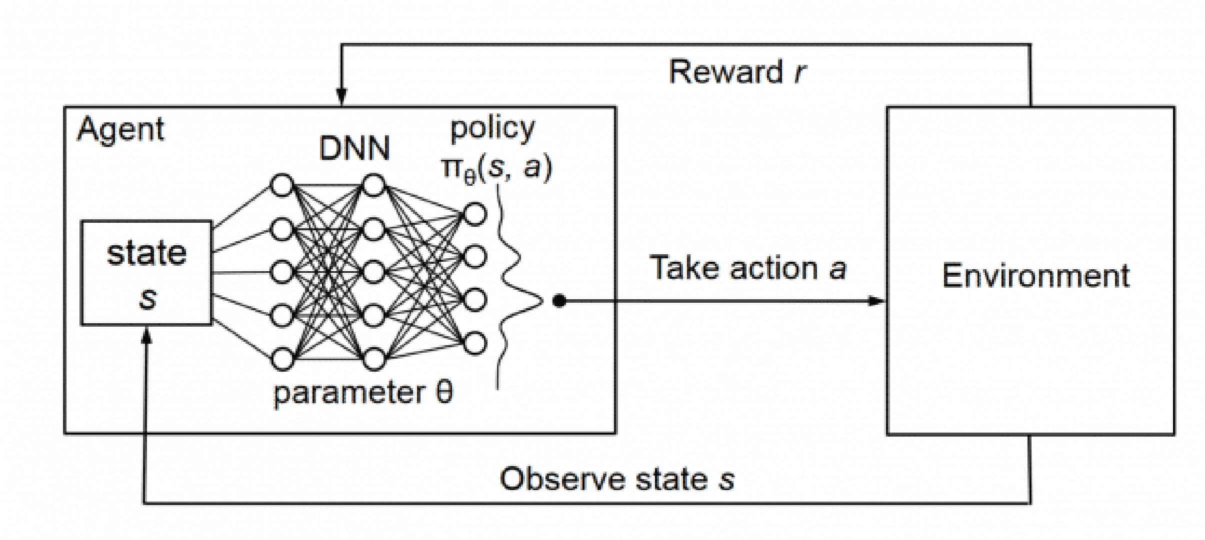
- The model can improve by adjusting the # of episodes and the epsilon factor

- There is an option to create a custom environment, therefore creating a custom environment might be a better option

- we can also use a **Deep Q-Learning** Formula which is the most promising as a custom environment.

DQL Formula

q[state,action] = reward if new_state is terminal else reward + discount_factor*max(q[new_state,:])



A DQL model would help us fine-tune the model and train it on different map sizes and worry less about the slippery factor, so it is truly advised to work on this further.