Frozen Lake Assignment

Setup 1:
FrequentRewards = False
T = 200

- The Q table? why is it full of zeros?

In such setup we define lot of steps in episode and original Frozen Lake Reward. This lead us to the problem that agent almost never gets to the finish point to get Reward (100000 episodes are not enough to reach finish at least once) so never update Q-table (update with zeros)

- Is your agent learning anything?

Agent isn't learning because we are not passing any reward to the start of the chain, updating Q function $Q(s,a)$ for each possible state and action based on formula

$$Q_{s,a} = \alpha * (R_{t+1} + \gamma \hat{Q}(S_2, argmax\ Q(S_{2,}\ a))) - Q_{s,a} \quad (1)$$

Since Reward is always 0 (except last point which is non explored in most cases) and all Q-table initialized with zeros.

- What What happens when is_slippery = False, versus when is_slippery = True

In this setup **slippery** flag doesn't effect on result because even without it agent couldn't find path to finish. Obviously the constraint that there exist uncertanty when performing action should make it harder for agent to get optimal results.

Setup 2:
FrequentRewards = True
Exploration_rate = 0.01
Is_slippery = True

- How has your Q-table changed and why?

Now Q-table actually update because with some values. Agent receives revard all the time. So then we have non-zero values for function (1) to work with.

- Is your agent learning anything?

Agent gets penalty when falls into water, so with this constraint, to maximize reward gain, agent learn not to go there. Also other constraint prevents agent from going around by decreasing reward by a tiny rate each step. Finally agent still get positive reward when gets to finish. This happens more frequent because now agent avoid bumping into water each episode, this increase explored area, and probability of getting to finish is enough to get meaningful probabilities in Q-table.

This setup ensures that agent will learn to go to the finish, minimize number of steps and avoid falling into water.

> Now pick a large value for exploration rate 0.1 or 0.5, how does this impact the total reward curve?

Large exploration rate decrease a success rate of agent significantly. This is becouse we decide to make random decision in 50% steps, so we rather explore all possible options, instead of focusing and optimizing actual best path.

The it possible to solve this by using some epsilon param which dynamically represent exploration_rate. It is initialized as 1 and slovly decresed to 0 during learning. This ensure thath firstly agent explore and for Q-table, and gradually going to master optimal path by following Q-table up to 100% of steps.

> Now set a very short episode length say T= 20

Such hyperparameter is too small for the task. It is almost impossible to find the finish by random, so agent never gets to the finish, without understanding goal(positive reward) agent will never form the path to optimize, even with such path exist.
In our task 20 steps is too small ammount, taking into account that minimal path is 16, but slippery makes probability extremely small $(1 - P(slip))^{16}$

> Reflect on why having frequent rewards has such an impact on learning

The update of the Q-table is more dynamic and consiquent. As the agent interacts with the environment, it collects rewards at many steps, not just at the goal. Each action taken by the agent leads to some sort of immediate feedbcack, which helps in faster learning and adjustment of strategies to maximize rewards. This frequent feedback leads to numerous updates in the Q-table, reflecting learning from various states and actions. Each entry in the Q-table represents the expected future rewards for a given state-action pair, and with more frequent rewards, these values become more refined all the time.

> Note that here each experience is used only once to update the Q-table, what is the disadvantage of this for rare events?

Each experience tuple influences the Q-table only once, at the time it occurs in episode. So for rare events, this can mean too few updates to capture this goal or penalty by agent, potentially leading to underestimation of important but rare strategies.

> Do you have a better solution?

I researched the solution which is commonly used with such problem

Experience Replay is a replay memory technique used in reinforcement learning where we store the agent's experiences at each time-step, in a data-set , pooled over many episodes into a replay memory. We then usually sample the memory randomly for a minibatch of experience, and use this to learn off-policy [link]

At any point in learning, the agent remembered the best episode up to that point. After every four episodes, it replayed this remembered episode, learning from it as if it were a new episode. [Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto]

Such techinque capture the best result and repeat them, so good episodes have more effect on Q-table by repeating them. This solve the issue that rare but significant example have small impact on learning process.

What are other methods to explore other than epsilon-greedy, do you have any proposals?

Classic Exploration Strategies

- **Epsilon-greedy**: The agent does random exploration occasionally with probability and takes the optimal action with probability $1 - \varepsilon$.

- **Upper confidence bounds**: The agent selects the greediest action to maximize the upper confidence bound $\widehat{\boldsymbol{Q}}_{\boldsymbol{t}}(\boldsymbol{a}) + \widehat{\boldsymbol{U}}_{\boldsymbol{t}}(\boldsymbol{a})$, where $\widehat{\boldsymbol{Q}}_{\boldsymbol{t}}(\boldsymbol{a})$ is the average rewards associated with action $\boldsymbol{a}$ up to time $\boldsymbol{t}$ and $\widehat{\boldsymbol{U}}_{\boldsymbol{t}}(\boldsymbol{a})$ is a function reversely proportional to how many times action has been taken.

- **Boltzmann exploration**: The agent draws actions from a boltzmann distribution (softmax) over the learned Q values, regulated by a temperature parameter τ.

- **Thompson sampling**: The agent keeps track of a belief over the probability of optimal actions and samples from this distribution.

The following strategies could be used for better exploration in deep RL training when neural networks are used for function approximation:

- **Entropy loss term**: Add an entropy term $\boldsymbol{H}(\boldsymbol{\pi}(\ \boldsymbol{a} \mid \boldsymbol{s}\ ))$ into the loss function, encouraging the policy to take diverse actions.
- **Noise-based Exploration**: Add noise into the observation, action or even parameter space [link]

In this document I wont provide detailed explanation of this strtegies