MSDS Reinforcement Learning

Final Project Summary Report

Training a Reinforcement Learning agent to play Frozen Lake game from OpenAI gym

Katja Wittfoth

Content:

1. Research Question

In this project, I trained an agent to play and win the Frozen Lake game from OpenAI gym. I used Q-learning and Deep Q-learning to train the agent.

The game is a grid world, where a player or an agent starts in the upper left tile at the start position and needs to navigate through the frozen lake to the goal, which is the lower right tile of the square matrix. The goal is to find a frisbee, so if you reach the goal tile, you will get your frisbee back. The other tiles are either frozen tiles, they are safe to step on, or holes, they are dangerous. If an agent steps on the hole, it will lose the game.

Additionally, the game is not deterministic, the agent does not always land where it intends to go. The reason is that the lake is extremely slippery, which makes the agent slide to the different tiles.

The state space is a square metrics of 4x4. Given the rewards in the states, the agent learns which action to take to best navigate to the goal.

2. Reinforcement Learning problem formation

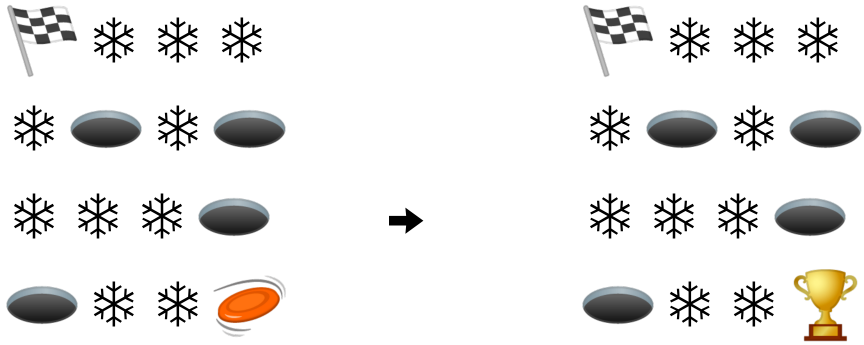
Environment: Gridworld with F, G, H, S, where S is a start, G is a goal, F is frozen lake and H is a hole. There is 4x4 and 8x8 Frozen Lake environment

Agent: Player / Machine

States: Up, Down, Left, Wright

Rewards: In state F, the agent gets 0 rewards, in state H -1 and in G it gets +1 reward.

I have modified the Frozen Lake environment to render the grid world as follows:



3. Q-learning

Q-learning is an off-policy algorithm which attempts to learn the value of being in a given state from a specific action.

Simple Q-learning uses a table of values for every state and action in the environment. In each state, the agent learns a value from taking an action which led to this state.

Frozen Lake environment has 16 possible states and 4 different actions which correspond to the directions: up, down, left, right. This gives us a 16x4 matrix, Q-table with values.

I initialize the Q-table with zeros, and as we observe the rewards from various actions, I update the Q-table accordingly.

We use the Bellman equation to update Q-tables, which calculates the expected long-term reward for the given action.

$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \times \left[\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1}, a_t)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \right]$$

Source: https://commons.wikimedia.org/wiki/File:Q-l%C3%A6ring_formel_1.png

Pseudocode of Q-learning:

Q-learning: Learn function $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$

Require:
Sates $\mathcal{X} = \{1, \ldots, n_x\}$
Actions $\mathcal{A} = \{1, \ldots, n_a\}$, $A : \mathcal{X} \Rightarrow \mathcal{A}$
Reward function $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Black-box (probabilistic) transition function $T : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$
Learning rate $\alpha \in [0, 1]$, typically $\alpha = 0.1$
Discounting factor $\gamma \in [0, 1]$
procedure QLEARNING($\mathcal{X}$, A, R, T, α, γ)
  Initialize $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ arbitrarily
  while Q is not converged **do**
    Start in state $s \in \mathcal{X}$
    while s is not terminal **do**
      Calculate π according to Q and exploration strategy (e.g. $\pi(x) \leftarrow \arg\max_a Q(x, a)$)
      $a \leftarrow \pi(s)$
      $r \leftarrow R(s, a)$ ▷ Receive the reward
      $s' \leftarrow T(s, a)$ ▷ Receive the new state
      $Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'))$
      $s \leftarrow s'$
  return Q

Source:
https://towardsdatascience.com/intro-to-various-reinforcement-learning-algorithms

My implementation:

For my solution, I added some improvements:

- **Decaying epsilon:** I implemented the decaying probability for exploration. Since, in the beginning, I want my agent to explore more, so I can have well updated Q-table. The epsilon decays gradually over the episodes until it gets to the minimum exploration of 0.1 and 0.9 exploitation.
- **Stopping condition:** the OpenAI gym suggests that the game is “solved” when the total average rewards are above 0.8 in 100 consequences episodes. Thus I implemented the stopping condition for learning. I check every 5,000 episodes if the condition is met, then I stop the learning. I chose 5,000 episodes to make sure that the agent has enough time to learn.

- **Tuned Hyperparameters:** I played with hyperparameters and tuned learning rate to be 0.01 and max step in each episode as 200. This allows the agent to learn well. I played with different decay rate for exploration as well and decided to use 0.01.

One of the results of Q-table for 4x4 Frozen Lake:

```
array([[0.06254502, 0.06147818, 0.0614298 , 0.05775705],
       [0.04216826, 0.04252887, 0.0424989 , 0.05890609],
       [0.07927089, 0.06478184, 0.06909751, 0.05579205],
       [0.03792381, 0.03912212, 0.0327212 , 0.05446649],
       [0.08354597, 0.07081063, 0.0617107 , 0.04969037],
       [0.        , 0.        , 0.        , 0.        ],
       [0.09355594, 0.0909519 , 0.11643995, 0.02154405],
       [0.        , 0.        , 0.        , 0.        ],
       [0.07201452, 0.10874869, 0.10085845, 0.13836245],
       [0.16740141, 0.2487421 , 0.21479903, 0.12380664],
       [0.30388773, 0.22850673, 0.19706028, 0.10965101],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.19755163, 0.3067553 , 0.39031158, 0.24443158],
       [0.39139558, 0.62409048, 0.56917854, 0.59403982],
       [0.        , 0.        , 0.        , 0.        ]])
```

If we analyze the table and the moves, it seems that the algorithm learns to take the least risky action. Especially it learns that the first safest move is to go to the left. Also, the agent learns that being in the second row on a frozen tile, the less riskiest command is left and most dangerous is to go right. Which we can eyeball as human very easily.

The 4x4 grid usually takes about 50,000 to 200,000 episodes to have a total reward of above 0.8 in 100 consecutive games.

4. DQN

Tables are great for the environment with finite states, but tables don't scale well. We need a function which would approximate rewards for all actions. In Deep Q-learning (DQN) we replace Q-table with Neural Networks, so we can learn more complex environment.

For Frozen Lake problem, I built a one-layer neural network using Tensorflow. The network takes in a state, which is encoded as a 1x16 vector with zeros and ones. The network gives a prediction as a vector with Q-values for each of the 4 actions.

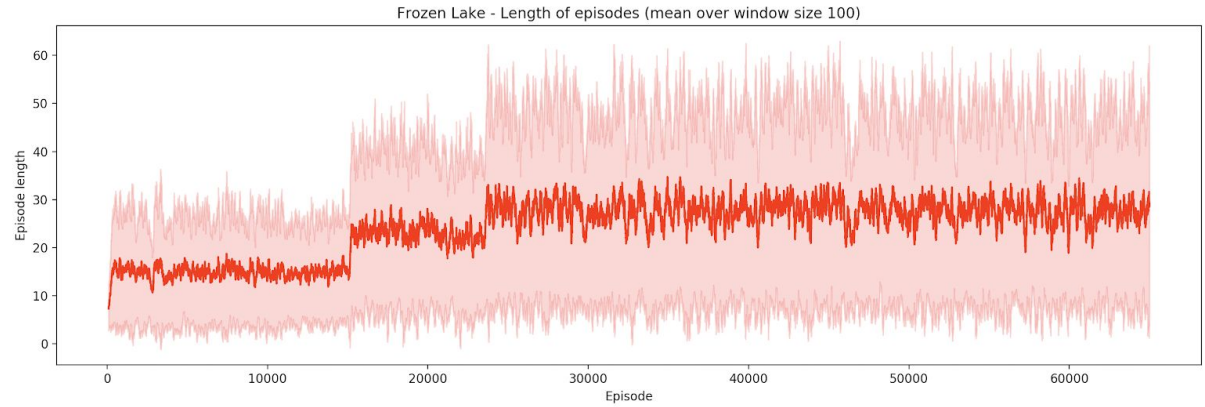
I use loss function as a sum of squares of the difference between the predicted values and the actual value.

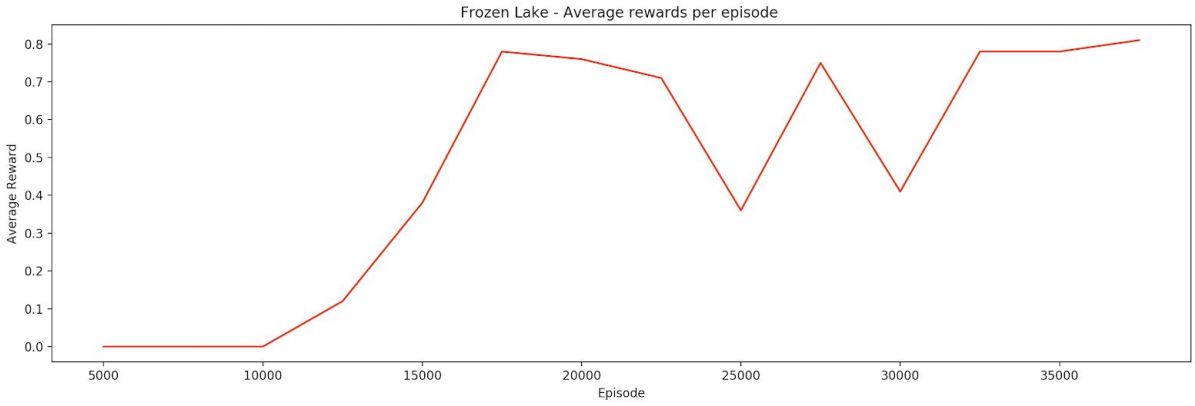
Similarly to the Q-learning, I am adding decayed probability for exploration as well as stopping condition. Although, the DQL takes longer to train, therefore I decreased the stopping condition to 0.4.

5. Results

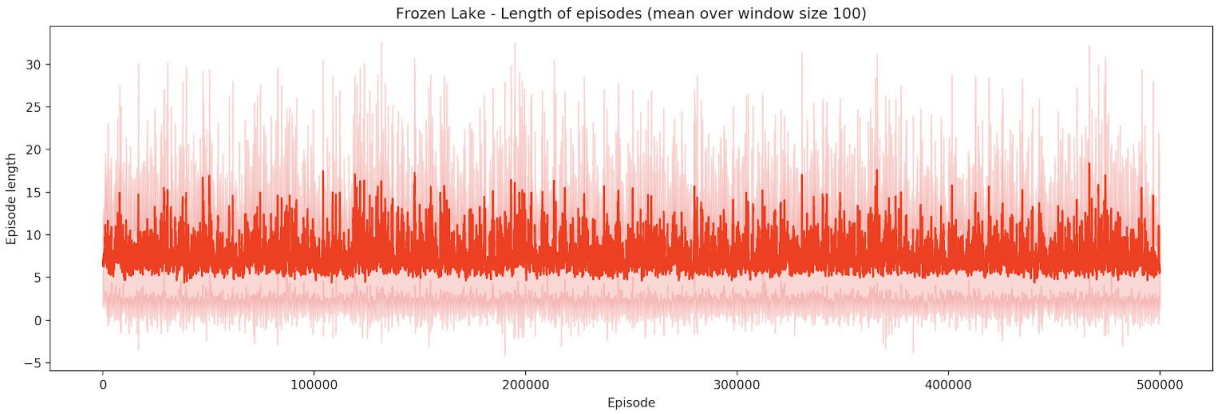
Q-learning seems to be better for small grids and DQN for the complex environment but takes long to learn as it trains and fits in each episode.

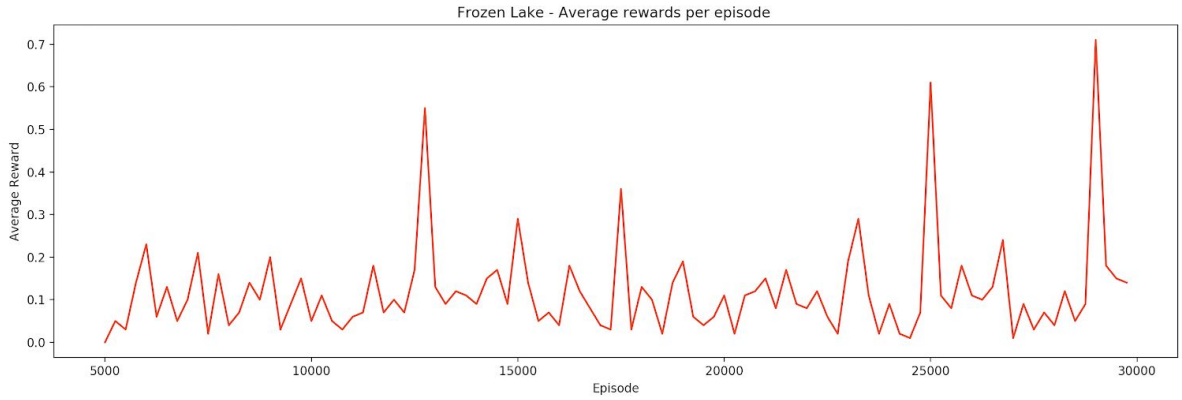
Q-learning:





DQN:





Seems that DQN doesn't converge very well and tends to take much more time to train. More layers in DQN and experience replay might yield better results, would also try on 8x8 grid. Additionally, was experimenting with higher slippery level, though the algorithm took very long to converge.

6. Next steps

In this project, I used Q-learning and DQN. For the next steps, I would like to try Deep Q-learning with more layers and using experience replay. Moreover, I would build a more complex environment, for instance, 8x8 or 12x12 and evaluate, which algorithm would perform well.

References:

1. Spiering, B. Reinforcement Learning course USF https://github.com/brianspiering/rl-course
2. https://gym.openai.com/envs/FrozenLake-v0/
3. https://www.kaggle.com/charel/learn-by-example-reinforcement-learning-with-gym