گزارش فاز دوم پروژه: دریاچه یخی

مبانی و کاربردهای هوش مصنوعی

دکتر حسین کارشناس

آدرینا ابراهیمی ۹۹۳۶۲۳۰۰۲

کیان مجلسی ۹۹۳۶۱۳۰۵۱

اردیبهشت ۱۴۰۲

نام و علت انتخاب الگوریتم

از الگوریتم Q-Learinig که Off-policy (مقادیر جدول q_table را متفاوت از سیاست رفتاری epsilon-gready آپدیت می‌کند.) است استفاده شده زیرا نسبت به الگوریتم SARSA که On-policy (مقادیر جدول q_table را مشابه سیاست رفتاری epsilon-gready آپدیت می‌کند.) است تخمین بهتری از سیاست بهینه به دست می‌آورده، مستقیما از سیاست بهینه یاد می‌گیرد و تصمیم‌گیری بهینه‌تری نسبت به الگوریتم SARSA دارد؛ این در حالی است که SARSA از سیاستی نزدیک به سیاست بهینه یاد می‌گیرد و برای محاسبه پاداش حالت‌های بعدی، نیازی به داشتن تمام q_table نیست

گزارش کار الگوریتم

تابع Q_Learning_Algorithm:

در این تابع عملیات اصلی الگوریتم Q Learning انجام می‌شود. این تابع جدول Q table و مشاهدات از محیط را در دریافت می‌کند. سپس، در یک حلقه به طول حداکثر تعداد دفعاتی که می‌خواهیم عمل یادگیری الگوریتم انجام شود، یک حلقه دیگر به طول حداکثر تعداد حرکات در محیط می‌زنیم. در حلقه دوم با توجه به مشاهداتی که از محیط داریم با استفاده از سیاست epsilon-greedy یک کنش انتخاب می‌کنیم. پس از آن، با استفاده از تابع step کنش انتخاب شده را در محیط اعمال کرده و این تابع مشاهده محیط، پاداش، خاتمه یافتن یا نیافتن، متوقف شدن یا نشدن و اطلاعات محیط را برمی‌گرداند.

مقدار مورد انتظار سودمندی در صورت افتادن آدمک در چاله ۱- و در صورت رسیدن آدمک به مقصد ۱ در نظر گرفته شده است. در صورتی که آدمک روی خانه‌های یخ‌زده باشد مقدار مورد انتظار سودمندی برابر فاصله اقلیدسی مختصات آدمک تا مختصات هدف است.

سپس، مقدار Q table را برای آن حالت و کنش در محیط با توجه به فرمول آپدیت مقدار Q table در الگوریتم Q Learning آپدیت می‌کنیم. در صورت پایان یا متوقف شده حرکت آدمک در محیط از حلقه دوم خارج شده و محیط را ریست می‌کنیم. در نهایت شرط همگرایی را بررسی که و در صورتی که مقادیر جدول Q Learning تغییر چندانی نداشته باشند از حلقه اول نیز for نیز خارج می‌شویم.

```python
def Q_Learning_Algorithm(observation, q_table):
    # train the agent for max_iter_number number of episodes
    for i in tqdm(range(0, max_iter_number)):

        # reset the environment
        observation, info = env.reset()
        terminated, truncated = False, False

        # for each episode, run the algorithm for MAX_STEP number of steps
        for __ in range(MAX_STEP):
            # choose the action using epsilon greedy policy
            action = epsilon_greedy_policy(q_table, observation)

            next_observation, reward, terminated, truncated, info = env.step(action)

            # if the agent reaches the goal, then reward = 1, else if it falls to a hole, reward = -1
            if (reward == 0) and terminated:
                reward = -1
            elif (reward == 1) and terminated:
                reward = 1
            else:
                # Calculate the distance to the goal for agent and reward it based on the distance
                x, y = next_observation // map_size, next_observation % map_size
                dist_to_goal = np.sqrt(np.power(x - (map_size - 1), 2) + np.power(y - (map_size - 1), 2))
                reward = -0.1 / (1 + np.exp(-dist_to_goal))

            # choose the next action
            next_action = np.argmax(q_table[next_observation])

            # update the q_table
            q_table[observation][action] = q_table[observation][action] + LEARNING_RATE *\
                (reward + DISCOUNT_FACTOR * q_table[next_observation][next_action] - q_table[observation][action])

            # update the observation
            observation = next_observation

            # if the agent reaches the goal or falls into a hole, then reset the environment
            if terminated or truncated:
                observation, info = env.reset()
                break

        convergenceTrack.append(np.linalg.norm(q_table.flatten().tolist()))
        if (i >= 1000) and np.isclose(convergenceTrack[-1], convergenceTrack[-2]):
            print('\nValues Converged')
            return
    print("Training Completed")
    sleep(2)
```

تابع epsilon_greedy_policy:

در این تابع یک عدد رندوم تولید شده و در صورتی که آن عدد کوچکتر از اپسیلون باشد، به صورت رندوم یک کنش انتخاب کرده و در صورتی که بزرگتر از اپسیلون باشد بهترین کنش ممکن را از جدول Q table انتخاب می‌کنیم.

```python
def epsilon_greedy_policy(q_table, observation):
    if np.random.uniform(0, 1) < EPSILON:
        action = env.action_space.sample()
    else:
        action = np.argmax(q_table[observation])
    return action
```

تابع save_q_table:

در این تابع ابتدا حالت‌های پایانی محیط را به دست آورده سپس، با استفاده از جدول Q table و با یک حلقه روی تمام حالت‌های محیط نقشه محیط به همراه policy به دست آورده شده را رسم می‌کنیم.

```python
def save_q_table(q_table):
    terminal_states = set()
    goal_state = (map_size - 1) * map_size + (map_size - 1)

    for state in env.P:
        if state == goal_state:
            continue
        for act in env.P[state]:
            for probability, nextState, reward, isTerminalState in env.P[state][act]:
                if (reward == 0) and isTerminalState:
                    terminal_states.add(nextState)

    with open('q_table_frozen.txt', 'w', encoding="utf-8") as inp:
        for state in range(map_size**2):
            if state in terminal_states:
                inp.write(u'☠\t')
            elif state == goal_state:
                inp.write(u'🪙\t')

            else:
                if np.all(q_table[state] == 0):
                    inp.write(u'□\t')
                else:
                    argm = np.argmax(q_table[state])
                    if argm == 0:
                        inp.write(u'←\t')
                    elif argm == 1:
                        inp.write(u'↓\t')
                    elif argm == 2:
                        inp.write(u'→\t')
                    elif argm == 3:
                        inp.write(u'↑\t')
            if (state + 1) % map_size == 0:
                inp.write('\n')
```

قسمت main برنامه:

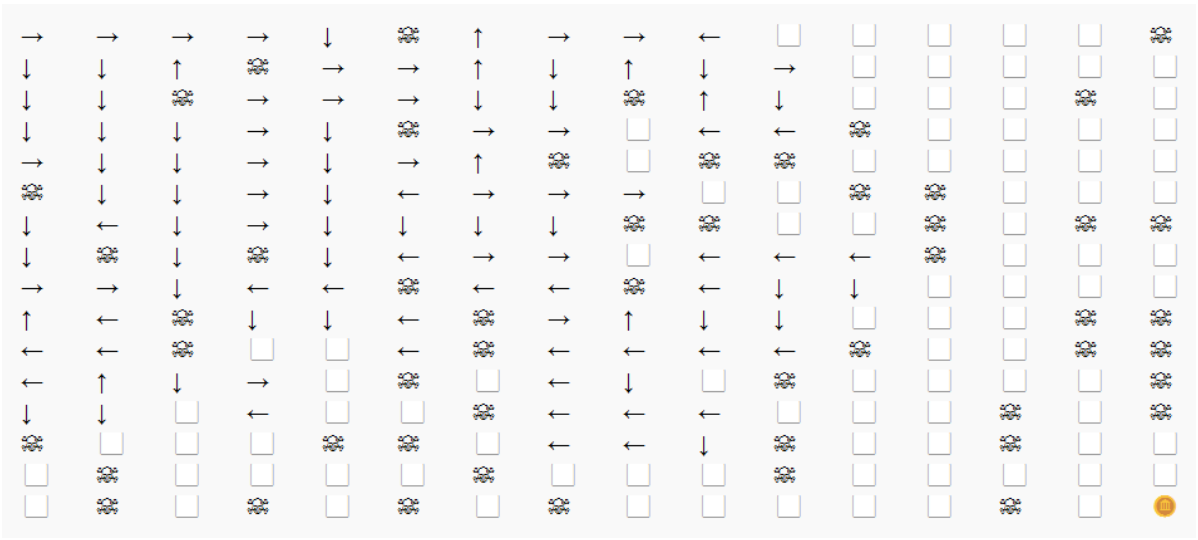
در ابتدا قسمت train را با استفاده الگوریتم Q Learning انجام شده و سیاست به دست آمده را ذخیره می‌کنیم.

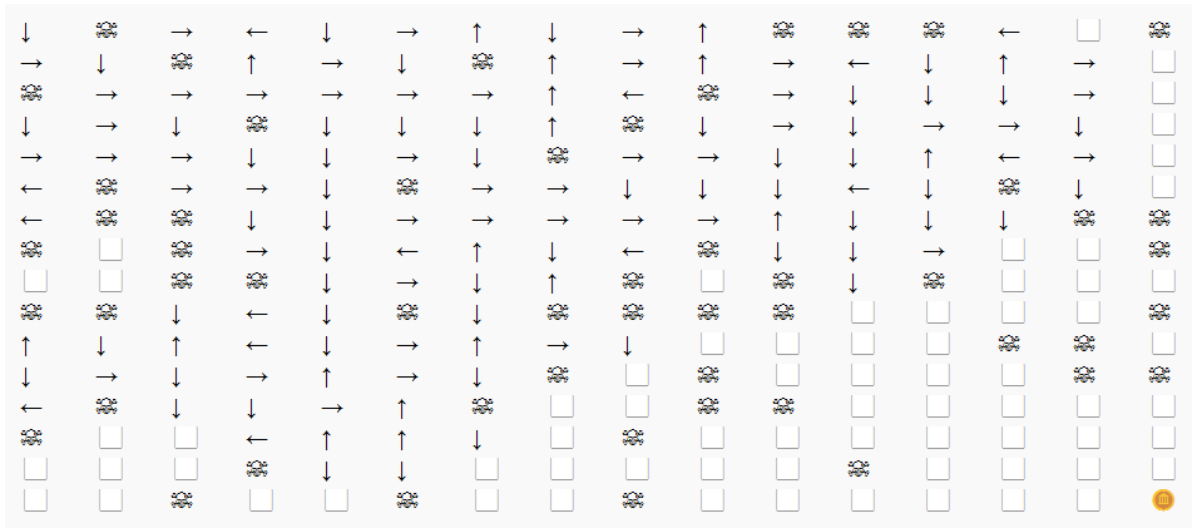
پس از آن محیط را ریست کرده و در یک اپیزود سیاست به دست آمده را تست می‌کنیم. و در نهایت پس از پایان اپیزود نمودار همگرایی را رسم می‌کنیم.

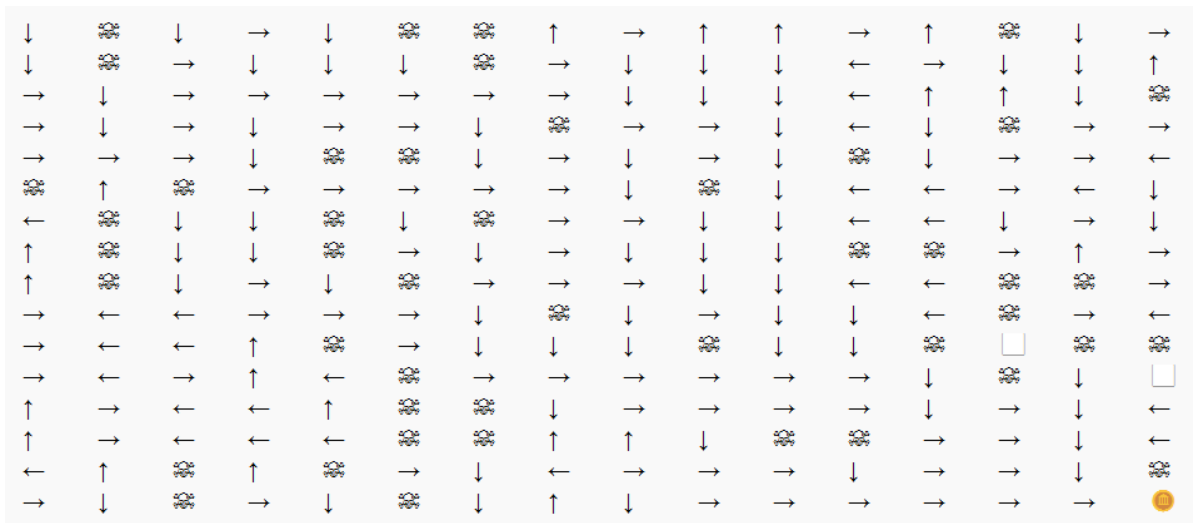
```python
if __name__ == "__main__":

    # train the agent with no graphical output
    Q_Learning_Algorithm(observation, q_table)

    save_q_table(q_table)

    env.close()

    # create Enviroment
    env = gym.make("FrozenLake-v1", desc=map_shape, render_mode="human", is_slippery=False)

    # reset enviroment
    observation, info = env.reset()

    # test the agent for max_iter_number number of episodes
    for i in range(1):

        terminated, truncated = False, False

        for _ in range(MAX_STEP):
            # select the action
            action = np.argmax(q_table[observation])

            next_observation, reward, terminated, truncated, info = env.step(action)

            env.render()

            # check if the agent reaches the goal or falls into a hole
            if terminated or truncated:
                observation, info = env.reset()
                break

            observation = next_observation


    env.close()
    plot_convergence(convergenceTrack)
```

تحلیل نتایج

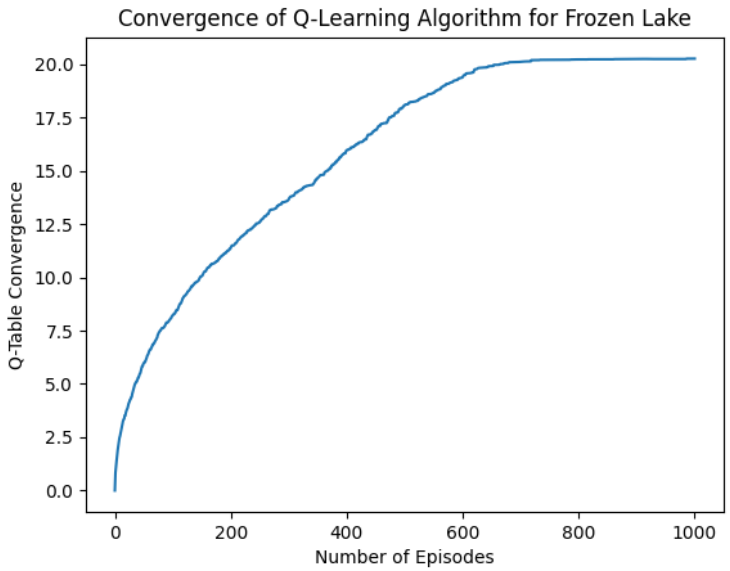
با تغییر مقدار اپسیلون از ۰/۹ به سمت ۰/۲ مشاهده می‌شود سیاست به دست آمده توسط الگوریتم بهبود پیدا کرده و به سمت سیاست بهینه تغییر پیدا می‌کند. شکل اول مقدار اپسیلون برابر ۰/۹ و شکل دوم مقدار اپسیلون برابر ۰/۵ و شکل سوم مقدار اپسیلون برابر ۰/۲ است. خانه‌های سفید نشان‌دهنده این است که عامل تا به حال در این خانه‌ها نرفته و خانه‌هایی که اسکلت دارند نشان‌دهنده چاله و خانه سکه نشان‌دهنده خانه هدف هستند. همانطور که مشاهده می‌شود با کاهش مقدار اپسیلون خانه‌های بیشتری کاوش شده و سیاست تکمیل می‌شود.







نمودار همگرایی که روی مقادیر Q table رسم شده است به شکل زیر می‌باشد. مشاهده می‌شود مقادیر جدول از حدود اپیزود ۸۰۰ به همگرایی می‌رسند.



منابع ایده

از منابع زیر برای آشنایی با نحوه پیاده‌سازی و الگو گرفتن اولیه استفاده شده است.

Stuart J. Russell, Peter Norvig - Artificial Intelligence_ A Modern Approach, Global Edition-Pearson (2021)

Frozenlake benchmark - Gymnasium Documentation (farama.org)