گزارش فاز دوم پروژه: تاکسی خودران

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 while تا زمانی که مقدار متغیر done برابر با False باشد می‌زنیم. در این حلقه while 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 و اطلاعات محیط را برمی‌گرداند.

حال مشاهدات محیط پس از اعمال کنش را به تابع decode داده و موقعیت تاکسی، مسافر و هدف را دریافت می‌کنیم. مقدار مورد انتظار سودمندی در صورت رساندن مسافر به مقصد ۲۰ و در صورت اعمال کنش اشتباه سوار یا پیاده کردن ۱۰- در نظر گرفته شده است. در صورتی که مسافر سوار تاکسی باشد مقدار مورد انتظار سودمندی برابر فاصله اقلیدسی مختصات تاکسی تا مختصات هدف بوده و در صورتی که مسافر سوار تاکسی نباشد مقدار مورد انتظار سودمندی برابر فاصله اقلیدسی مختصات تاکسی تا مختصات مسافر در نظر گرفته می‌شود.

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 برقرار شدن شرط حلقه while از آن 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 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 range(0, MAX_ITER_NUMBER):
        # reset the environment
        done = False

        # explore the environment until the agent reaches the goal
        while not done:
            # choose the action using epsilon greedy policy
            action = epsilon_greedy_policy(q_table, observation)

            next_observation, reward, done, info = env.step(action)

            # get the current state of the agent, and the position of the passenger and the goal
            taxi_row, taxi_col, pas_pos, goal_pos = list(env.decode(next_observation))

            # if the agent reaches the goal, then reward = 20, else if drop and pickup the passenger, then reward = -10
            if (reward == 20) and done:
                reward = 20
            elif (reward == -10) and not done:
                reward = -10
            # if the agent picks up the passenger
            elif (pas_pos == 4) and reward == -1:
                # Calculate the distance to the goal
                dist_to_goal = np.linalg.norm(np.array([taxi_row, taxi_col]) - MAP_COLORFUL_PLACES[goal_pos])
                reward = -0.1 / (1 + np.exp(-dist_to_goal))
            # if the agent have not pick up the passenger
            elif (pas_pos != 4) and reward == -1:
                # Calculate the distance to the passenger
                dist_to_goal = np.linalg.norm(np.array([taxi_row, taxi_col]) - MAP_COLORFUL_PLACES[pas_pos])
                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, then reset the environment
        if done:
            observation = env.reset()

        # check convergence
        convergenceTrack.append(np.linalg.norm(q_table.flatten().tolist()))
        if (i > 1000) and np.isclose(convergenceTrack[-1], convergenceTrack[-2]):
            print('Values Converged')
            return

        if (i+1) % 100 == 0:
                print(i+1)

    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 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):
    map_size = 6
    with open('q_table_taxi.txt', 'w', encoding="utf-8") as inp:
        for state in range(map_size**2):
            x, y = state // map_size, state % map_size
            if np.array_equal(MAP_COLORFUL_PLACES[0], np.array([x, y])):
                inp.write(u'🔴\t')
            elif np.array_equal(MAP_COLORFUL_PLACES[1], np.array([x, y])):
                inp.write(u'🟢\t')
            elif np.array_equal(MAP_COLORFUL_PLACES[2], np.array([x, y])):
                inp.write(u'🟡\t')
            elif np.array_equal(MAP_COLORFUL_PLACES[3], np.array([x, y])):
                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')
                    elif argm == 4:
                        inp.write(u'🚖\t')
                    elif argm == 5:
                        inp.write(u'🏠\t')
            if (state + 1) % map_size == 0:
                inp.write('\n')
```

قسمت main برنامه:

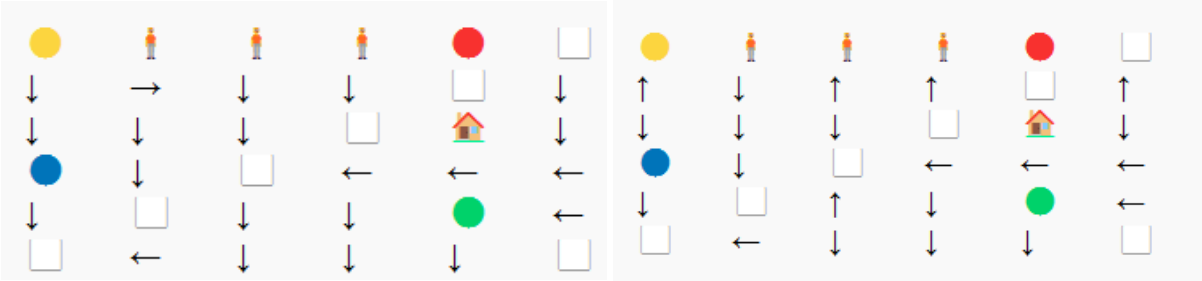
در ابتدا قسمت train را با استفاده الگوریتم Q Learning انجام شده و سیاست به دست آمده را ذخیره می‌کنیم.

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.

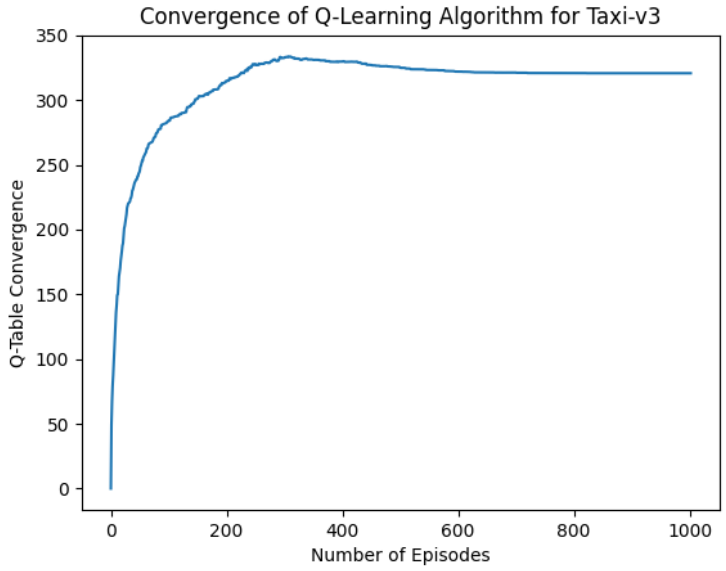
```python
if __name__ == "__main__":

    # train the agent
    Q_Learning_Algorithm(observation, q_table)

    save_q_table(q_table)

    # reset the environment
    observation = env.reset()

    # test the agent for one time
    done = False

    while not done:
        action = np.argmax(q_table[observation])
        observation, reward, done, info = env.step(action)

        print(env.render())
        sleep(1)


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

تحلیل نتایج

با تغییر مقدار اپسیلون از صفر به سمت ۰/۹ مشاهده می‌شود سیاست به دست آمده توسط الگوریتم بهبود پیدا کرده و به سمت سیاست بهینه تغییر پیدا می‌کند. شکل سمت راست مقدار اپسیلون برابر ۰ و شکل سمت چپ مقدار اپسیلون برابر ۰/۹ است. خانه‌های سفید نشان‌دهنده این است که عامل تا به حال در این خانه‌ها نرفته و خانه‌هایی که آدمک دارند نشان‌دهنده انتخاب کنش سوار کردن و خانه ساختمان نشان‌دهنده انتخاب کنش پیاده کردن برای آن حالت از محیط می‌باشد.



نمودار همگرایی که روی مقادیر Q table رسم شده است به شکل زیر می‌باشد. مشاهده می‌شود مقادیر جدول از حدود اپیزود ۶۰۰ 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)

Solving The Taxi Environment With Q-Learning — A Tutorial | by Wouter van Heeswijk, PhD | Mar, 2023 | Towards Data Science