دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

پروژه:

Maze، پیاده سازی توسط الگوریتم QLearning

اعضای گروه:

حمید مهران‌فر(۴۰۰۳۶۱۳۰۵۸)

رادمهر آقاخانی (۴۰۰۳۶۶۳۰۰۲)

علی کثیری (۴۰۰۳۶۱۳۰۵۱)

استاد:

دکتر کارشناس

گروه ۱۰

توضیح کلی پروژه:

هدف این پروژه پیاده سازی الگوریتم QLearning برای یادگیری عامل هوشمندی در که در یک محیط غیر قطعی می‌باشد.

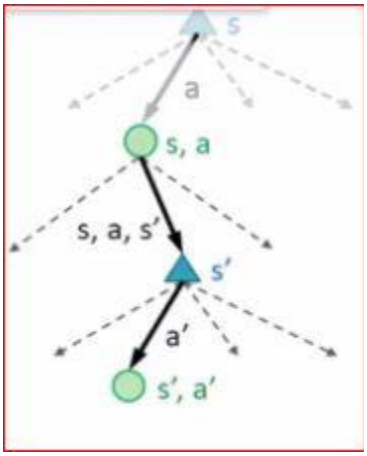
توضیح الگوریتم QLearning:

الگوریتم QLearning یک الگوریتم یادگیری تقویتی فعال می باشد که هدف آن یادگیری سیاست بهینه براساس توازن میان میزان کاوش (exploration) و میزان بهره‌برداری (exploitation) است. این یادگیری آفلاین نیست و عامل واقعا در محیط عملیاتی را انجام می دهد و نتیجه آن را مشاهده می نماید.

توضیح بکارگیری QLearning :

از آنجایی که توابع تغییر حالت (T(s,a,s')) و تابع پاداش (R(s,a,s')) ناشناخته هستند در نتیجه به جای استفاده از مقادیر V(S) برای تعیین سیسات از مقادیر Q که مفید تر هستند استفاده می کنیم. یعنی شروع به یادگیری مقادیر Q(s,a) می کنیم .

که به صورت زیر آمده است :



- دریافت نمونه (s,a,s’,r)
- در نظر گرفتن تخمین قبلی : Q(s,a)
- در نظر گرفتن تخمین مربوط به نمونه جدید:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

- به روز رسانی تخمین(می توان گفت یک میانگین گیری انجام می شود):

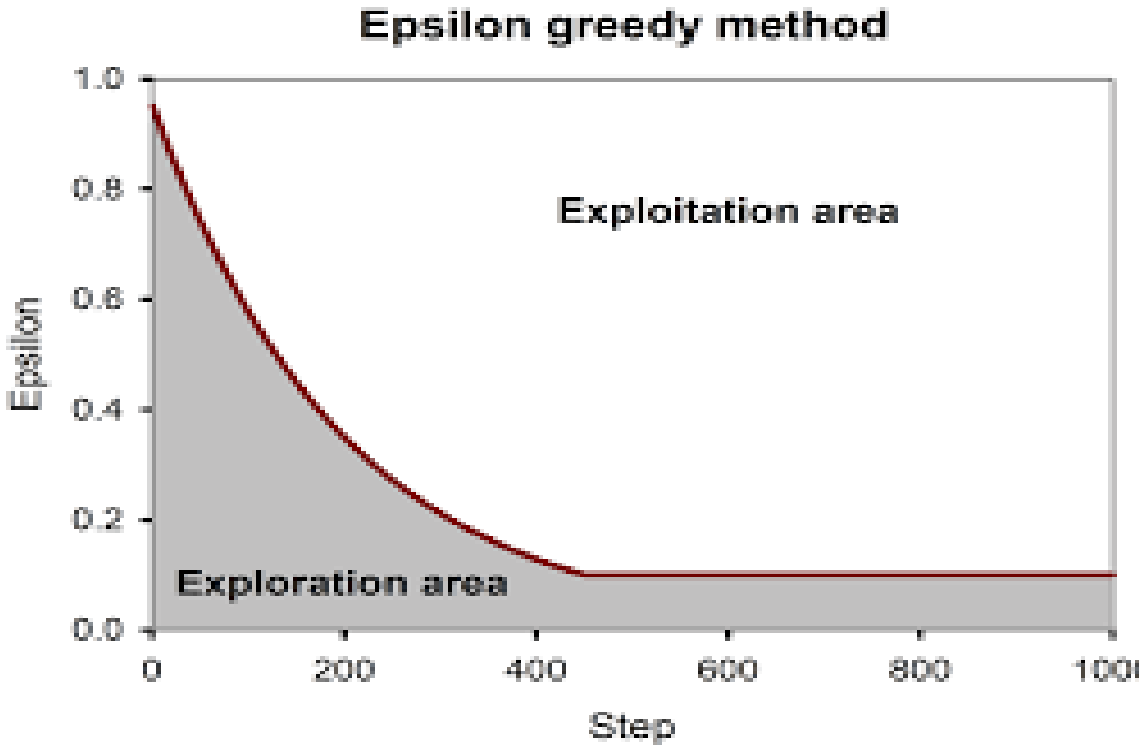
$$Q(s,a) = (1-\alpha)Q(s,a) + (\alpha)[sample]$$

یعنی می توان گفت که ما از TD(temporal difference) یا همان تفاضل زمانی نیز در QLearning استفاده کرده‌ایم. زیرا بعد از هر تجربه ، تخمین Q(s,a) را به روز رسانی کرده‌ایم.

استفاده از قاعده به روز رسانی بر اساس فرمول داده شده نشان می دهد که نمونه های جدید تر از اهمیت بیشتری برخوردارند و این یعنی فراموش کردن نمونه های بسیار قدیمی تر. حال هرچه نخ یادگیری (آلفا) را کاهش بدهیم باعث پایدار شدن الگوریتم یادگیری می شود.

استفاده ازاستراتژی حریصانه اپسیلون (epsilon greedy strategy):

این استراتژی مشخص می کند عامل چه زمانی باید عمل اکتشاف و یا بهره‌برداری را انجام دهد. در واقع کاوش پراکنده شده از دانش یادگرفته شده استفاده نمی کند و بهره‌برداری برخلاف آن از دانش یادگرفته شده برای افزایش پاداش دریافتی استفاده می کند. یعنی ابتدا یک نرخ جستجو اپسیلون تعیین می شود که در آغاز برابر با 0.9 قرداده می شود. سپس یک عدد تصادفی تولید می شود. اگر این عدد بزرگتر از اپسیلون بود، بهره‌برداری یا همان استخراج انجام می‌شود و در غیر این صورت، جست‌وجو یا همان کاوش صورت می‌پذیرد. پس در واقع اپسیلون به تدریج و پس از آنکه اطمینان در تخمین ارزش زوج حالت-کنش ها افزایش یافت، مقدار آن کاهش می‌یابد.



توضیح ماژول main_maze_dual_policy :

در این فایل ما با دو رویکرد به پیدا کردن سیاست بهینه پرداخته‌ایم:

- تکرار ۱۰۰۰۰۰ مرتبه در محیط ماز برای به دست آوردن سیاست بهینه
- اجرا ۱۰۰۰ بار قسمت و در هر قسمت هنگامی که done مقدار True را گرفت، محیط reset شده و عامل درepisode بعدی شروع به یادگیری مجدد برای بهبود بخشیدن به ارزش زوج حالت-کنش ها (Q-Values) می‌پردازد.

تعریف ثابت ها

در هر دو رویکرد ثابت هایی همچون ضریب کاهش(discount_factor همان گامای موجود در فرمول گفته شده بالا)، نرخ یادگیری (آلفا) سطر و ستون های محیط، بیشترین و کمترین مقادیر اپسیلون و همچنین نرخ کاهش نمایی برای اپسیلون که منجر به کاهش اکتشاف می شود، تعریف و مقدار دهی شده اند.

تعریف attributes ها برای رویکرد اول:

- q_table که یک آرایه سه بعدی برای ذخیره مقادیر ارزش های Q در سیاست اول،
- policy که برای ذخیره سیاست بهینه در رویکرد اول استفاده می شود، یک آرایه دو بعدی ۱۰ * ۱۰ می باشد.

تعریف attributes ها برای رویکرد دوم:

- q_values که یک آرایه سه بعدی برای ذخیره زوج مرتب حالت-کنش در سیاست اول به ابعاد ۱۰*۱۰*۴ می باشد (۴، تعداد کنش مجاز در هر حالت می باشد: بالا، پایین، چپ، راست).
- second_policy که برای ذخیره سیاست بهینه (کنش مناسب) استفاده می شود. این آرایه مانند سیاست اول، دو بعدی ۱۰*۱۰ می‌باشد.

تعریف دو لیست برای نمایش متوسط پاداش مجموع دریافتی با افزایش اپیزودها:

- لیست episode_rewards در هر اپیزود مقدار پاداش دریافتی را ذخیره می کند

```
learning_rate = 0.1
discount_factor = 0.95
# epsilon_configuration
max_epsilon = 0.9
min_epsilon = 0.01
decline_rate_for_decrease_probability_of_exploration = 0.01
# First Policy Attribution
q_table = np.zeros(shape=(10, 10, 4))
policy = np.zeros(shape=(10, 10))
# Second Policy attribution
second_policy = np.zeros(shape=(10, 10))
environment_rows = 10
environment_columns = 10
q_values = np.zeros((environment_rows, environment_columns, 4))
episode_rewards = []
```

کد تعریف ثابت ها وattributes ها برای دو رویکرد در ماز

تعریف توابع استفاده شده در رویکرد دوم:

تابع def reduce_epsilon(episode):

در این تابع همانطور که در بخش استراتژی حریصانه اپسیلون توضیح داده شد، پس از هر اپیزود مقدار اپسیلون به صورت نمایی بر اساس نرخ کاهش(که در کد بالا آورده شده است) کاهش می یابد. در واقع این تابع الگوریتم را به سمت استخراج و بهره‌براری سوق می دهد.

```
def reduce_epsilon(episode):
    epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp(
        -decline_rate_for_decrease_probability_of_exploration * episode)
    return epsilon
```

کد تابع کاهش اپسیلون با گذشت زمان (اپیزود)

تابع def get_next_action():

آرگومان های این تابع به ترتیب عبارتند از:

- current_row_index،
- current_column_index،
- epsilon

در این تابع در واقع ما سیاست در یادگیری تقویتی را ترکیبی از از سیاست تصادفی و حریصانه قرار می‌دهیم. به طوریکه اگر rt >= epsilon (rt تقریبا برابربا random()) آنگاه به صورت حریصانه کنش را براساس ارزش های Q به دست آمده انتخاب می کنیم و اگر کوچکتر از آن باشد به صورت تصادفی کنش را در آن حالت برمی‌گردانیم.

```
def get_next_action(current_row_index, current_column_index, epsilon):
    if np.random.random() > epsilon:
        return np.argmax(q_values[current_row_index, current_column_index])
    else:
        return np.random.randint(4)
```

کد تابع اتخاب سیاست مربوط به هر حالت با استفاده از ترکیب تصادفی و حریصانه

تابع get_reward(state):

این تابع حالت را گرفته و پاداش متناظر با آن را بر می گرداند. به طوریکه اگر عامل به حالت پایانی، که همان خانه (۹و۹) است، رسیده باشد پاداش ۱۰۰ و در غیر اینصورت پاداش منفی یک را به عنوان پاداش محیط برای حالت غیر پایانی انتخاب می کند.

```
def getReward(state):
    if state[0] == 9 and state[1] == 9:
        return 100
    return -1
```

کد تابع پاداش برای حالت پایانی و حالت های غیرپایانی

تابع def calculatePolicy_second (env):

عامل با استفاده از این تابع یادگیری در محیط را انجام می دهد. به طوریکه عامل در ۱۰۰۰ اپیزود سعی می کند محیط را را به طور قابل قبولی شناسایی کرده و سیاست بهینه را به دست آورد. پس در هر اپیزود عامل کنش هایی را انجام می دهد تا به حالت پایانی برسد. بدیهی است که در اپیزود های ابتدایی عامل سریع نتواند به حالت پایانی برسد و در نتیجه حلقه while not done زمان زیادتری می برد تا تمام شود(هرچه به اپیزود های پایانی نزدیک بشویم تعداد کنش های انجام شده در حلقه while not done کاهش می یابد). بعد از حلقه while، مقدار اپسیلون توسط تابع کاهش اپسیلون (reduce_epsilon)، کاهش می‌یابد.

محاسبات انجام شده در حلقه while

- برای به روز رسانی جدول q_values با توجه به بخش توضیح الگوریتم QLearning که در ابتدای مستند توضیح داده شد، ابتدا تفاضل زمانی (Temporal Difference = TD

) محاسبه می شود. سپس نرخ یادگیری در آن ضرب شده و با مقدار قبلی زوج مرتب کنش-حالت جمع می‌شود و مقدار QValue مربوطه را می توان به روز رسانی کرد.

- برای اینکه در نمودار میانگین پاداش ها را نشان دهیم، در هر بار اجرا حلقه در یک اپیزود، total_reward برای آن اپیزود را محاسبه می کنیم.

پس از اتمام حلقه while not done، پاداش به دست آمده (total_reward) را به لیست episode_reward و اپیزود مربوطه را نیز به لیست episode_steps اضافه می‌کنیم. در انتها هر اپیزود همانطور که گفته شد مقدار اپسیلون کاهش می‌یابد.

در پایان تابع آرایه دو بعدی second_policy، مقادیر خود را با استفاده از ماکسیمم کردن زوج مرتب کنش-حالت برای هر یک از ۱۰۰ حالت موجود، مقدار دهی می کنیم.

```
while not done:
    action_index = get_next_action(int(row_index), int(column_index), initialized_epsilon)
    # TODO: Implement the agent policy here
    next_state, reward, done, truncated = env.step(action_index)
    reward = getReward(next_state)
    total_reward += reward
    if truncated:
        print("truncated")
        print(f"truncated next state{next_state}")
        break
    row_index_old, column_index_old = row_index, column_index
    old_q_value = q_values[int(row_index_old), int(column_index_old), action_index]
    row_index = next_state[0]
    column_index = next_state[1]
    temporal_difference = reward + (
            discount_factor * np.max(q_values[int(row_index), int(column_index)])) - old_q_value
    new_q_value = old_q_value + (learning_rate * temporal_difference)
    q_values[int(row_index_old), int(column_index_old), action_index] = new_q_value
episode_rewards.append(total_reward)
```

کد به‌دست آوردن سیاست بهینه

تعریف توابع استفاده شده در رویکرد اول:

در این بخش از آوردن توابعی و توضیح محاسبتی که به صورت مشترک در دو رویکرد استفاده شده‌اند، خودداری شده است.

تابع def getAction (action, epsilon):

این تابع عملکردی مشابه با تابع get_next_action دارد، بطوریکه اگر rt) rt >= epsilon تقریبا برابربا ()random) آنگاه به صورت حریصانه کنش را براساس ارزش های Q به دست آمده انتخاب می کنیم و اگر کوچکتر از آن باشد به صورت تصادفی کنش را در آن حالت برمی‌گردانیم.

```
def getReward(state):
    if state[0] == 9 and state[1] == 9:
        return 100
    return -1
```

کد تابع انتخاب کنش

تابع def calculatePolicy(env):

در این تابع عامل سعی می کند در صد هزار تکرار موفق به یادگیری محیط ماز شود. به این صورت که در هر تکرار یک کنش با استفاده از ترکیب سیاست تصادفی و حریصانه به‌دست می آید.سپس براساس آن، عامل کنش را انجام داده و پس از آن شروع به بروز رسانی مقادیر QValue ها در q_table می کند(انجام محاسبات مانند رویکرد دوم است که در تابع calculatePolicy_second بطور مفصل به آن پرداختیم). سپس هنگامی که عامل به حالت

پایانی می رسد مقدار done ارزش درست می گیرد و عامل به خانه اول برگشته و در تکرار های بعدی سعی در بهبود بخشیدن سیاست خود می کند.

```
for __ in range(100000):
    action = getAction(q_table[state[0]][state[1]], initialized_epsilon)
    old_state = state.copy()
    state, _, done, _ = env.step(action)
    reward = getReward(state)
    old_q_value = q_table[old_state[0]][old_state[1]][action]
    TD = reward + (discount_factor * np.max(q_table[state[0]][state[1]]) - old_q_value)
    q_table[old_state[0]][old_state[1]][action] += learning_rate * TD
    if done:
        env.reset()
        state = [0, 0]
    episode += 1
    initialized_epsilon = reduce_epsilon(episode)
```

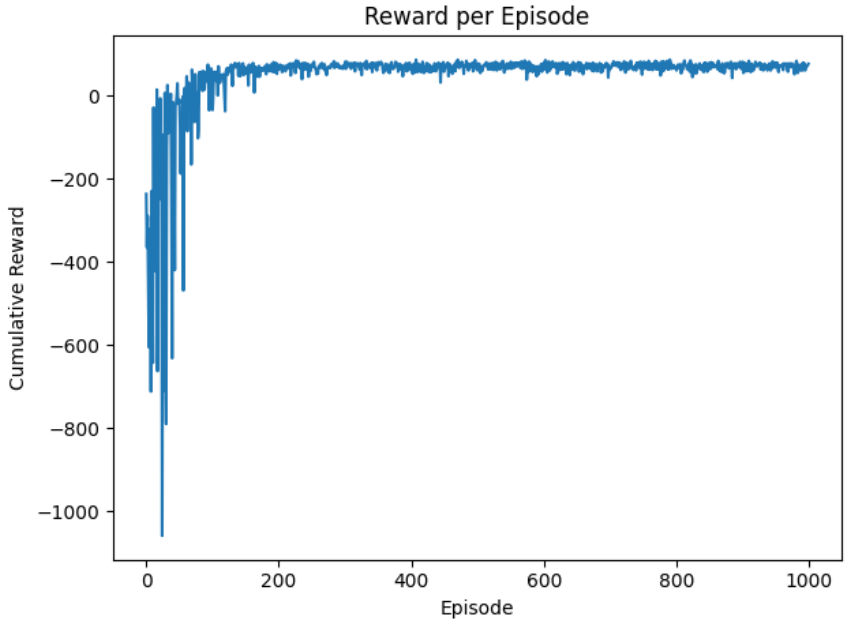
کد به‌دست آوردن سیاست بهینه رویکرد اول

نمایش لیست episode_rewards به طول ۱۰۰۰

لیست پاداش ها را پس از آنکه تابع calculatePolicy_second در اسکریپت main صدا زده شد، با استفاده از کتابخانه matplotlib به صورت نمودار نشان داده می‌شود. لازم به یادآوری است که محور x همان تعداد اپیزود (برابر با ۱۰۰۰ است) و محور y مجموع پاداش ها در هر اپیزود می باشد.

```
plt.plot(episode_rewards)
plt.xlabel('Episode')
plt.ylabel('Cumulative Reward')
plt.title('Reward per Episode')
average_reward = sum(episode_rewards) / len(episode_rewards)
print(f"The average reward is: {average_reward}")
plt.tight_layout()
plt.show()
```

کد مربوط به نمایش لیست episode_reward



نمودار متوسط پاداش مجموع دریافتی با افزایش اپیزودها در یک بار محیط

```
The average reward is: 47.411
```

میانگین پاداش به دست آمده در ۱۰۰۰ اپیزود

منابع :

https://www.researchgate.net/figure/Epsilon-greedy-method-At-each-step-a-random-number-is-generated-by-the-model-If-the_fig2_334741451

https://blog.faradars.org/reinforcement-learning-and-q-learning/

https://youtu.be/QUNM-QyM5PA?si=jvuFwNq60DcuLhxQ

https://youtu.be/kaDEw5qMTLs?si=_CnoCt7uqAzSdw46

https://www.geeksforgeeks.org/q-learning-in-python/

پایان