به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

گزارش تمرین شماره ۴

شیما سادات ناصری

۴۰۱۱۲۸۱۴

دکتر مهدی علیاری شوره دلی

تیر ۱۴۰۳

فهرست مطالب

سوال ۱

۱

پیاده‌سازی Q-learning:

Q-learning یک الگوریتم یادگیری تقویتی بدون مدل است که در آن عامل از طریق آزمون و خطا ارزش یک عمل در یک حالت خاص را یاد می‌گیرد. ارزش Q برای هر جفت حالت-عمل با استفاده از معادله بلمن به‌روزرسانی می‌شود:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a} Q\,(s', a') - Q(s,a) \right]$$

که در آن:

s حالت فعلی است،

a عمل فعلی است،

r پاداش دریافت شده پس از انجام عمل a است،

s' حالت بعدی است،

α نرخ یادگیری است،

γ ضریب تخفیف است.

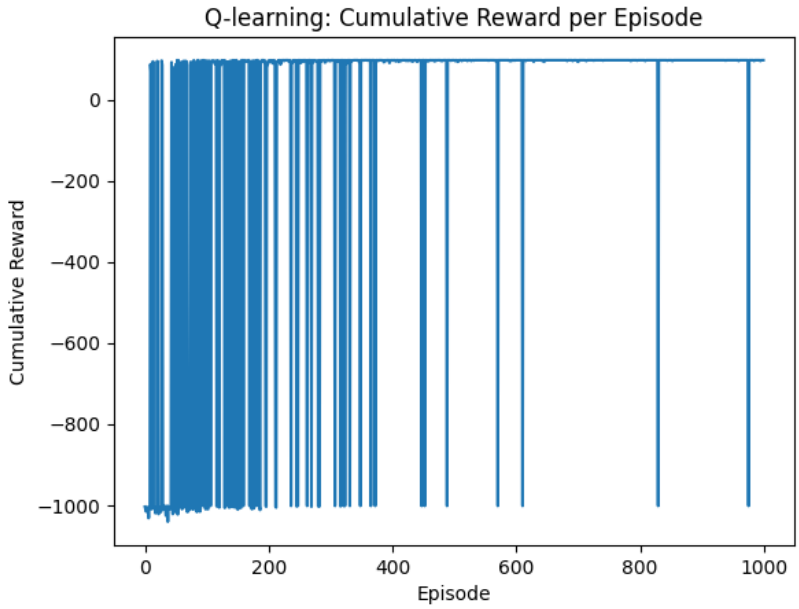
بر اساس دیتای مسئله، کلاس زیر تعریف شد:

```python
class WumpusWorld:
    def __init__(self):
        self.grid_size = 4
        self.reset()

    def reset(self):
        self.agent_position = [0, 0]
        self.wumpus_position = [1, 2]
        self.gold_position = [2, 2]
        self.pit_positions = [[3, 0], [1, 3]]
        self.state = (self.agent_position[0], self.agent_position[1])
        return self.state

    def step(self, action):
        if action == 0:  # up
            self.agent_position[0] = max(0, self.agent_position[0] - 1)
        elif action == 1:  # down
            self.agent_position[0] = min(self.grid_size - 1, self.agent_position[0] + 1)
        elif action == 2:  # left
            self.agent_position[1] = max(0, self.agent_position[1] - 1)
        elif action == 3:  # right
            self.agent_position[1] = min(self.grid_size - 1, self.agent_position[1] + 1)

        self.state = (self.agent_position[0], self.agent_position[1])

        reward = -1  # movement penalty
        done = False

        if self.agent_position == self.gold_position:
            reward = 100
            done = True
        elif self.agent_position == self.wumpus_position:
            reward = -1000
            done = True
        elif self.agent_position in self.pit_positions:
            reward = -1000
            done = True

        return self.state, reward, done
```

سپس کد Q-learning برای آن به صورت زیر پیاده شد:

```python
for episode in range(num_episodes):
    state = env.reset()
    total_reward = 0
    done = False

    while not done:
        if random.uniform(0, 1) < epsilon:
            action = random.choice([0, 1, 2, 3])
        else:
            action = np.argmax(Q_table[state[0], state[1], :])

        next_state, reward, done = env.step(action)
        total_reward += reward

        best_next_action = np.argmax(Q_table[next_state[0], next_state[1], :])
        td_target = reward + discount_factor * Q_table[next_state[0], next_state[1], best_next_action]
        Q_table[state[0], state[1], action] = (1 - learning_rate) * Q_table[state[0], state[1], action] + learning_rate * td_target

        state = next_state

    rewards_per_episode.append(total_reward)
    epsilon = max(epsilon_min, epsilon * epsilon_decay)
```

نتیجه این کد به صورت زیر است:



پیاده‌سازی Deep Q-learning (DQN):

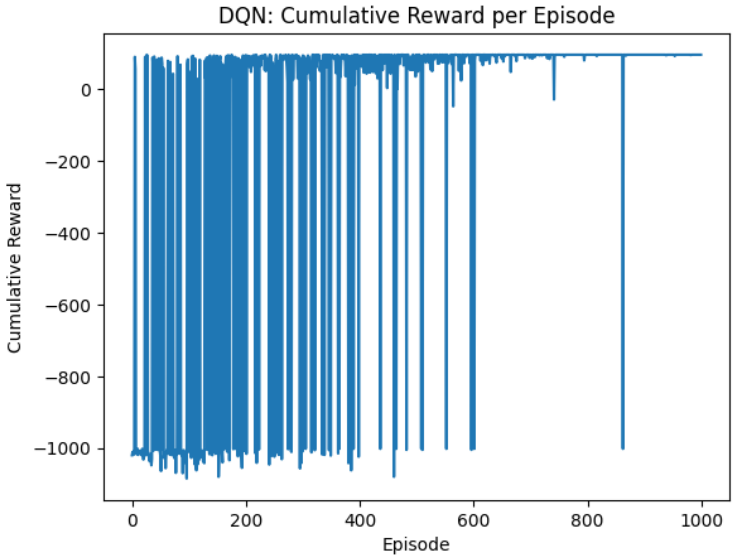
DQN از یک شبکه عصبی برای تقریب تابع Q-value استفاده می‌کند. شبکه عصبی حالت را به عنوان ورودی می‌گیرد و Q-values برای تمام اعمال ممکن را خروجی می‌دهد. کد زیر را برای این کار پیاده سازی می‌کنیم:

```python
class DQN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, output_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

سپس به صورت زیر آموزش انجام می‌شود:

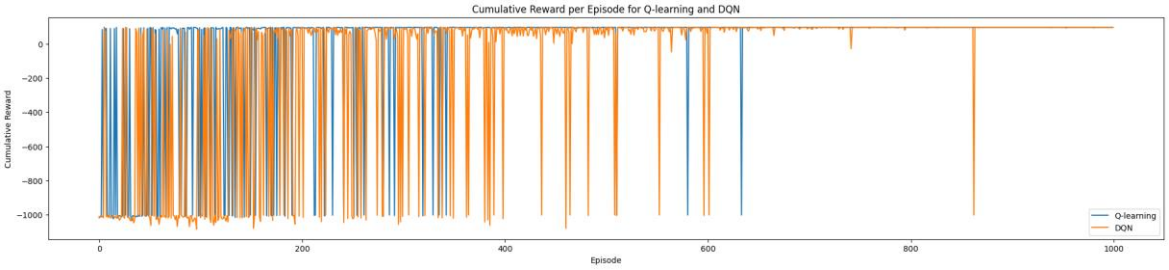
```python
for episode in range(num_episodes):
    state = env.reset()
    state = torch.tensor(state, dtype=torch.float32).unsqueeze(0)
    total_reward = 0
    done = False

    while not done:
        if random.uniform(0, 1) < epsilon:
            action = random.choice([0, 1, 2, 3])
        else:
            q_values = dqn(state)
            action = torch.argmax(q_values).item()

        next_state, reward, done = env.step(action)
        total_reward += reward
        next_state = torch.tensor(next_state, dtype=torch.float32).unsqueeze(0)

        memory.append((state, action, reward, next_state, done))
        if len(memory) > 10000:
            memory.pop(0)

        if len(memory) >= batch_size:
            batch = random.sample(memory, batch_size)
            batch_states, batch_actions, batch_rewards, batch_next_states, batch_dones = zip(*batch)
            batch_states = torch.cat(batch_states)
            batch_actions = torch.tensor(batch_actions)
            batch_rewards = torch.tensor(batch_rewards)
            batch_next_states = torch.cat(batch_next_states)
            batch_dones = torch.tensor(batch_dones, dtype=torch.float32)

            current_q_values = dqn(batch_states).gather(1, batch_actions.unsqueeze(1)).squeeze(1)
            next_q_values = dqn(batch_next_states).max(1)[0]
            expected_q_values = batch_rewards + (1 - batch_dones) * discount_factor * next_q_values

            loss = criterion(current_q_values, expected_q_values)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        state = next_state

    rewards_per_episode_dqn.append(total_reward)
    epsilon = max(epsilon_min, epsilon * epsilon_decay)
```

نتیجه به صورت زیر است:



۲

نمودار های بدست آمده در کنار هم به صورت زیر است:



هم یادگیری Q و هم DQN نوسانات قابل توجهی را در پاداش‌های تجمعی نشان می‌دهند. مواردی وجود دارد که پاداش به شدت کاهش می یابد و نشان می دهد که عامل با یک موقعیت چالش برانگیز مواجه شده یا تحت تأثیر یک رویداد نامطلوب قرار گرفته است. با پیشرفت آموزش، به نظر می‌رسد فراوانی چنین پاداش‌های منفی کاهش می‌یابد، که نشان می‌دهد هر دو عامل در طول زمان یاد می‌گیرند که از خطرات اجتناب کنند. در کل، به نظر می‌رسد که هر دو عامل از نظر عملکرد تثبیت می‌شوند، با کاهش شدید پاداش‌ها. در پایان ۱۰۰۰ قسمت، به نظر می‌رسد که پاداش‌های انباشته کاهش یافته است، که نشان می‌دهد عوامل سیاست‌های نسبتاً پایداری را آموخته‌اند.

در کنار آن داریم:

```
Average reward per episode for Q-learning: 96.98
Average reward per episode for DQN: 96.93
```

هر دو الگوریتم از نظر میانگین پاداش تقریباً یکسان عمل کردند. تفاوت جزئی (۰.۰۵) در میانگین پاداش‌ها در هر قسمت ناچیز است، که نشان می‌دهد هر دو روش یاد گرفته‌اند تا در محیط Wumpus World با کارایی یکسان حرکت کنند.

میانگین پاداش های قابل مقایسه نشان می دهد که هیچ یک از روش ها مزیت قابل توجهی نسبت به دیگری در این محیط خاص نداشتند.

۳

نوسانات قابل توجهی در پاداش های تجمعی در ابتدای آموزش برای هر دو عامل مشاهده می شود. این مرحله مربوط به یک مقدار اپسیلون بالا است، جایی که عوامل در درجه اول با انجام اقدامات تصادفی در حال کاوش در محیط هستند. قسمت‌هایی با کاهش شدید پاداش‌ها نشان می‌دهند که عوامل اغلب در چاله‌ها می‌افتند یا توسط Wumpus خورده می‌شوند، که در طول این مرحله اکتشاف انتظار می‌رود.

تأثیر اپسیلون:

- اپسیلون بالا ($\varepsilon \approx 1$):

اقدامات تصادفی توسط ماموران انجام می شود که منجر به طیف گسترده ای از تجربیات، هم خوب (یافتن طلا) و هم بد (افتادن در چاله ها) می شود و درک جامعی از محیط در این مرحله با جمع آوری تجربیات متنوع ایجاد می شود و کاهش و افزایش شدید پاداش ها در طرح منعکس می شود و تصادفی بودن اقدامات را نشان می دهد.

- اپسیلون کم ($\varepsilon \approx 0.01$):

اتکا به خط مشی های آموخته شده اتفاق می افتد، با اقداماتی که بر اساس بالاترین مقادیر Q انتخاب می شوند و پاداش های تجمعی پایدارتر نشان دهنده تصمیم گیری آگاهانه بر اساس تجربیات آموخته شده است و اگرچه به طور متوسط عملکرد بهتری مشاهده می شود، اما به دلیل اکتشاف محدود، خطر از دست دادن کشف سیاست های بالقوه بهتر وجود دارد.

مشاهدات زیر در دو روش دیده شد:

- یادگیری Q:

در ابتدا، به دلیل اکتشاف، جوایز منفی بالا اغلب نشان داده می شود. با پیشرفت آموزش، پاداش‌ها تثبیت می‌شوند که نشان‌دهنده یادگیری مؤثر سیاست است. افزایش پاداش منفی کمتر در قسمت‌های پایانی نشان می‌دهد که یک خط‌مشی به خوبی آموخته شده وجود دارد.

- DQN:

تنوع بالایی در پاداش ها در قسمت های اولیه به نمایش گذاشته می شود. با کاهش اپسیلون، با کاهش شدید در قسمت‌های بعدی، انتقال به الگوی پاداش پایدارتر اتفاق می‌افتد. تثبیت نشان می دهد که یادگیری موثر برای جلوگیری از خطرات و جمع آوری طلا به طور موثر صورت گرفته است.

۴

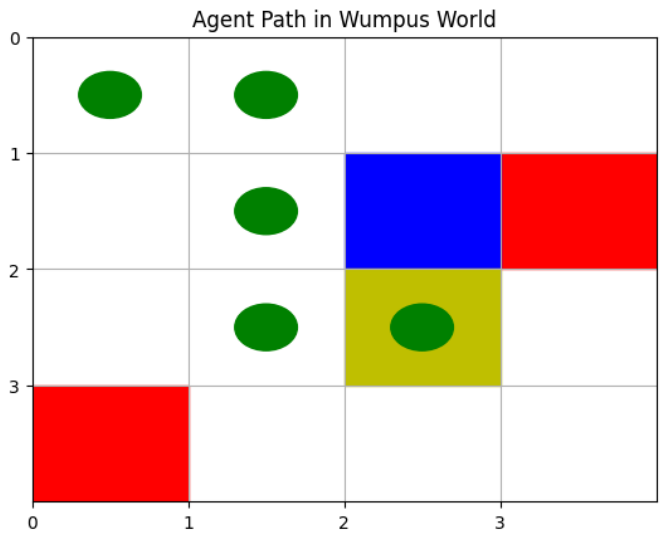
کد زیر پیاده سازی شد:

Count episodes required for Q-learning agent to consistently find gold

```
[ ] consistent_episodes = 0
    for i in range(100, num_episodes):
        if rewards_per_episode[i] > 0:
            consistent_episodes += 1

    # Compare learning efficiency
    print(f'Episodes required for Q-learning agent to consistently find gold: {consistent_episodes}')
```

```
Episodes required for Q-learning agent to consistently find gold: 856
```

همانطور که مشاهده می‌شود الگوریتم اول در ۸۵۶ اپیزود توانسته به جواب درست برسد. مسیر بدست آمده توسط این روش به صورت زیر است:

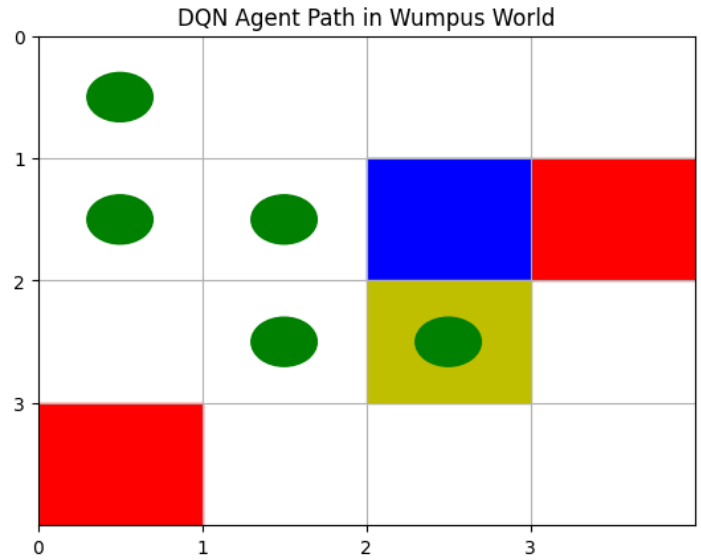


Count episodes required for DQN agent to consistently find gold

```
[ ]  consistent_episodes_dqn = 0
     for i in range(100, num_episodes):
         if rewards_per_episode_dqn[i] == max(rewards_per_episode_dqn):  # Episode where the agent finds gold
             consistent_episodes_dqn += 1

     # Print the number of consistent episodes
     print(f'Number of episodes where DQN agent consistently found gold: {consistent_episodes_dqn}')
```

```
Number of episodes where DQN agent consistently found gold: 451
```

مشاهده می‌شود الگوریتم دوم در ۴۵۱ اپیزود به همان نتیجه رسیده است. مسیر بدست آمده توسط این روش به صورت زیر است:



تفاوت این دو در ظاهر فقط در اکشن اول است اما هر دو در نهایت در بهترین مسیر به جواب رسیده‌اند.

برای مقایسه دقیق‌تر کد زیر پیاده سازی شد:

Comparing

```
import numpy as np

# Calculate the moving average of rewards to smooth out the performance
def moving_average(data, window_size):
    return np.convolve(data, np.ones(window_size), 'valid') / window_size

# Smooth the reward data
window_size = 10
q_learning_smoothed_rewards = moving_average(rewards_per_episode, window_size)
dqn_smoothed_rewards = moving_average(rewards_per_episode_dqn, window_size)

# Determine the episode where each agent consistently gets high rewards
q_learning_threshold_episode = np.argmax(q_learning_smoothed_rewards >= 100) + window_size
dqn_threshold_episode = np.argmax(dqn_smoothed_rewards >= 100) + window_size

# Print the comparison
print(f'Q-learning agent consistently found gold starting from episode: {q_learning_threshold_episode}')
print(f'DQN agent consistently found gold starting from episode: {dqn_threshold_episode}')

# Determine which one learned the optimal policy faster
if q_learning_threshold_episode < dqn_threshold_episode:
    print('Q-learning learned the optimal policy faster.')
else:
    print('DQN learned the optimal policy faster.')
```

```
Q-learning agent consistently found gold starting from episode: 10
DQN agent consistently found gold starting from episode: 10
DQN learned the optimal policy faster.
```

هر دو عامل Q-learning و DQN موفقیت ثابتی را در یافتن طلا از ابتدای آزمایش نشان دادند و از قسمت ۱۰ به بعد به این امر دست یافتند.

علیرغم تجزیه و تحلیل قبلی که نشان می‌دهد DQN هنگام در نظر گرفتن پاداش‌های هموار، عملکرد ثابتی را با سرعت بیشتری به دست می‌آورد (۴۵۱ قسمت تا ۸۵۶ قسمت برای یادگیری Q)، به نظر می‌رسد هر دو عامل در یک نقطه عملکرد قابل اعتمادی را آغاز کرده‌اند.

۵

مدل به صورت زیر است:

```
class DQN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, output_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

معماری از سه لایه کاملاً متصل تشکیل شده است. این یک معماری نسبتا ساده اما موثر برای بسیاری از وظایف یادگیری تقویتی، از جمله محیط Wumpus World است.

استفاده از ۱۲۸ نورون در لایه های پنهان تعادلی بین پیچیدگی مدل و کارایی محاسباتی ایجاد می کند.

تابع فعال سازی ReLU برای لایه های مخفی استفاده می شود. ReLU معمولاً در شبکه های عصبی استفاده می شود زیرا غیرخطی بودن را معرفی می کند، به شبکه اجازه می دهد تا الگوهای پیچیده را یاد بگیرد و از نظر محاسباتی کارآمد است.

پارامترهای زیر نیز در فرایند استفاده شده است:

```
learning_rate = 0.001
discount_factor = 0.9
epsilon = 1.0
epsilon_decay = 0.995
epsilon_min = 0.01
num_episodes = 1000
batch_size = 32
```

این پارامترها برای اطمینان از یادگیری موثر و کارآمد شبکه انتخاب می شوند. نرخ یادگیری به اندازه کافی کوچک است تا یادگیری پایدار را تضمین کند و عامل تخفیف بر پاداش های آینده تأکید دارد. نرخ فروپاشی اپسیلون برای متعادل کردن اکتشاف و بهره برداری در طول زمان تنظیم شده است.

مراجع

https://github.com/cchristoffer/wumpus-world-q-learning/blob/main/Wumpus_World_VG_Christoffer_Helgemo.ipynb