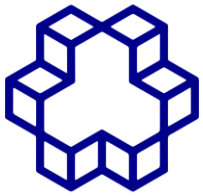
به نام خدا



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

پروژه نهایی

Github link:

https://github.com/FatemehShokrollahiMoghadam

google drive link:

https://drive.google.com/drive/folders/1g3zFZeV9Gx8rP9EjxtkFJR__iZWeMsb_?usp=sharing

نگارنده:

فاطمه شکراللهی مقدم

۴۰۲۰۷۳۶۴

تابستان ۱۴۰۳

فهرست مطالب

چکیده

بهینه سازی مصرف انرژی در سیستم های تهویه مطبوع (HVAC) برای کاهش هزینه و اثرات زیست محیطی بسیار مهم است. این پروژه به بررسی الگوریتم Q-learning، برای کنترل وضعیت شیر سه راهه در یک سیستم HVAC باهدف به حداقل رساندن مصرف انرژی و در حین حال حفظ دمای اتاق می‌پردازد. این پروژه با نمایش مجموعه داده‌ها برای درک روابط بین متغیرها و همبستگی ویژگی‌های مختلف آغاز می‌شود. یک مدل محیط HVAC طراحی شده است که اثر تغییر میزان بازشدگی شیر را بر دمای داخل اتاق شبیه سازی می کند. یک عامل یادگیری Q برای یادگیری موقعیت‌های بهینه از طریق تعامل با این محیط آموزش داده می‌شود و هایپرپارامترها با استفاده از جستجوی شبکه‌ای GridSearch تنظیم می‌شوند. عملکرد عامل یادگیری Q با حالتی که در آن شیر تحت کنترلPID عمل می کند، مقایسه می شود. نتایج نشان می دهد که کنترل مبتنی بر یادگیری Q به طور قابل توجهی مصرف انرژی را در مقایسه با کنترل کلاسیک PID کاهش می دهد.

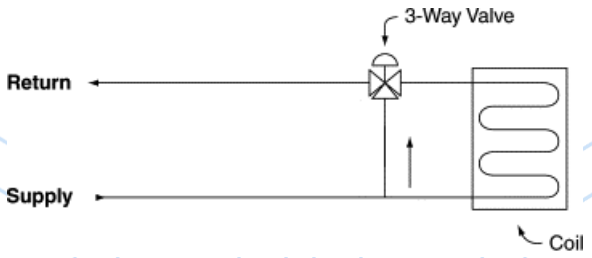
مقدمه

هواساز (AHU) تجهیز اصلی یک سیستم HVAC شناخته می شود. هواساز شامل قسمت های دمپر برقی(Damper)، فیلتر، میکروسوئیچ، شیر سه راهه یا دوراهه، سنسور آنتی فریز(Antifreeze)، سنسور دما و رطوبت(RHT :Relative Humidity Temperature) و فن و موتور است. در این پروژه به کنترل وضعیت شیر سه راهه با در دست داشتن داده های مربوط به دمای هوای بیرون و دمای اتاق در ساعات مشخص ضبط داده و تحت کنترل PID می‌پردازیم و در نهایت با تعریف یک معیار برای مصرف انرژی، میزان مصرف انرژی را برای کنترلر PID و کنترل مبتنی بر یادگیری Q می سنجیم.

شیر سه راهه(3-Way Valve)

کویل های هواساز با توجه به نوع سیال جریان یافته داخل ان ها متفاوت می باشد و به صورت آب سرد، آب گرم و... می باشند. نحوه عملکرد کویل ها در دستگاه های هواساز بدین صورت می باشد که ابتدا هوا توسط دریچه دمپرها وارد دستگاه شده و با عبور از بستر فیلترها پاکسازی می شوند. سپس با جریان یافتن برروی کویل های گرم (Heating Coil)و یا سرد(Cooling Coil) باعث گرم و یا سرد شدن هوای عبوری می شود.

به منظور کنترل دما از شیر سه راهه استفاده می شود. شیر سه راهه میزان جریان آب را در یک سیستم کنترلی تنظیم می کند. این شیر کنترلی شامل یک شیر و یک اکچوئیتور (actuator) است.



شکل ۱ -۰-۱ نمایش مسیر ارتباط کویل و شیر سه راهه

شکل۱ شیر سه راهه متصل به کویل را نشان می دهد.با توجه به این شکل اگر شیر سه راهه مسیر مشخص شده را ببندد بنابراین آب وارد مسیر کویل شده و سپس از مسیر Return خارج می شود.یعنی ۱۰۰٪ آب وارد کویل می شود.و هوای اتاق بسته به نوع کویل (سرمایشی یا گرمایشی)،سرد یا گرم می شود.

سنسور دما و رطوبت(RHT :Relative Humidity Temperature)

سنسور RHT بصورت Duct Mode در خط Return Air یا Exhaust Air وجود دارد که دمای خروجی Zone را اندازه می گیرد و دمای اندازه گیری شده را به شیر سه راهه می دهد.شیر سه راهه بر اساس دمای دریافتی از سنسور تصمیم می گیرد کدام مسیر را باز یا بسته کند.برای مثال اگر دمای اتاق کمتر از حد مطلوب شد، شیر سه راهه مسیر مشخص شده در شکل ۲۷ را باز می کند تا آب وارد این مسیر شده و داخل کویل نچرخد. وقتی آب وارد کویل نشود هوا دیگر سرد یا گرم(بسته به نوع کویل) نمی شود.شیر سه راهه مجددا با دریافت داده از سنسور RHT اقدام مورد نیاز را انجام می دهد. به این ترتیب شیر سه راهه مدام در حال باز یا بسته شدن است.

شیر سه راهه در عمل یک شیر تدریجی است.یعنی با توجه به دمای دریافتی از سنسور RHT درصدی از آب درون کویل می چرخد و درصدی در مسیر مشخص شده جریان دارد.

شکل ۱-۰-۲سنسور **RHT**

مراحل انجام پروژه

نمایش دیتاست

ابتدا دیتاست با فرمت csv را با دستور gdown در محیط کولب بارگزاری می کنیم پس از حذف ستون Day چند سطر دیتا را با دستور()head. نمایش می دهیم.

```
!gdown 1gDNUJYw-aUos0NTxTjrJTYenAFA_5jNs
```

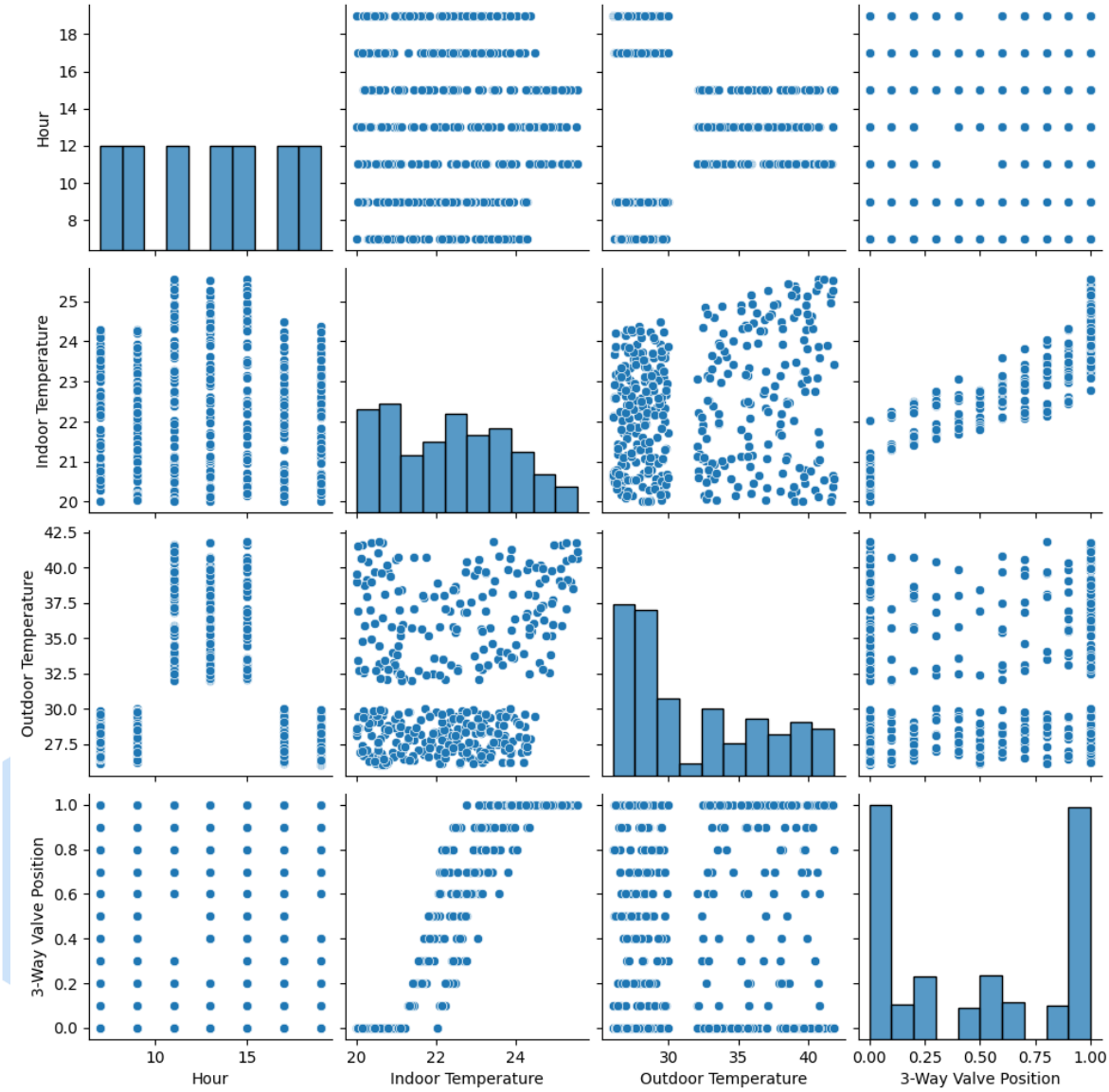
```
Downloading...
From: https://drive.google.com/uc?id=1gDNUJYw-aUos0NTxTjrJTYenAFA_5jNs
To: /content/hvac_valve_data.csv
100% 15.3k/15.3k [00:00<00:00, 41.3MB/s]
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('/content/hvac_valve_data.csv')

# Remove the 'day' column
data = data.drop(columns=['Day'])

# Display first five rows of the dataset
data.head()
```

	Hour	Indoor Temperature	Outdoor Temperature	Setpoint Temperature	3-Way Valve Position
0	7	20.790136	28.797435	22	0.0
1	9	20.052228	28.673260	22	0.0
2	11	24.408804	35.449545	22	1.0
3	13	25.523063	41.765079	22	1.0
4	15	24.446614	36.021703	22	1.0

در ستون اول ساعتی از روز که داده ضبط شده است نشان داده می شود و در ستون های بعد به ترتیب دمای داخل اتاق، دمای هوای بیرون، دمای مطلوب اتاق و موقعیت شیر سه راهه نشان داده شده است.

برای نمایش داده های از کتابخانه seaborn استفاده می کنیم:

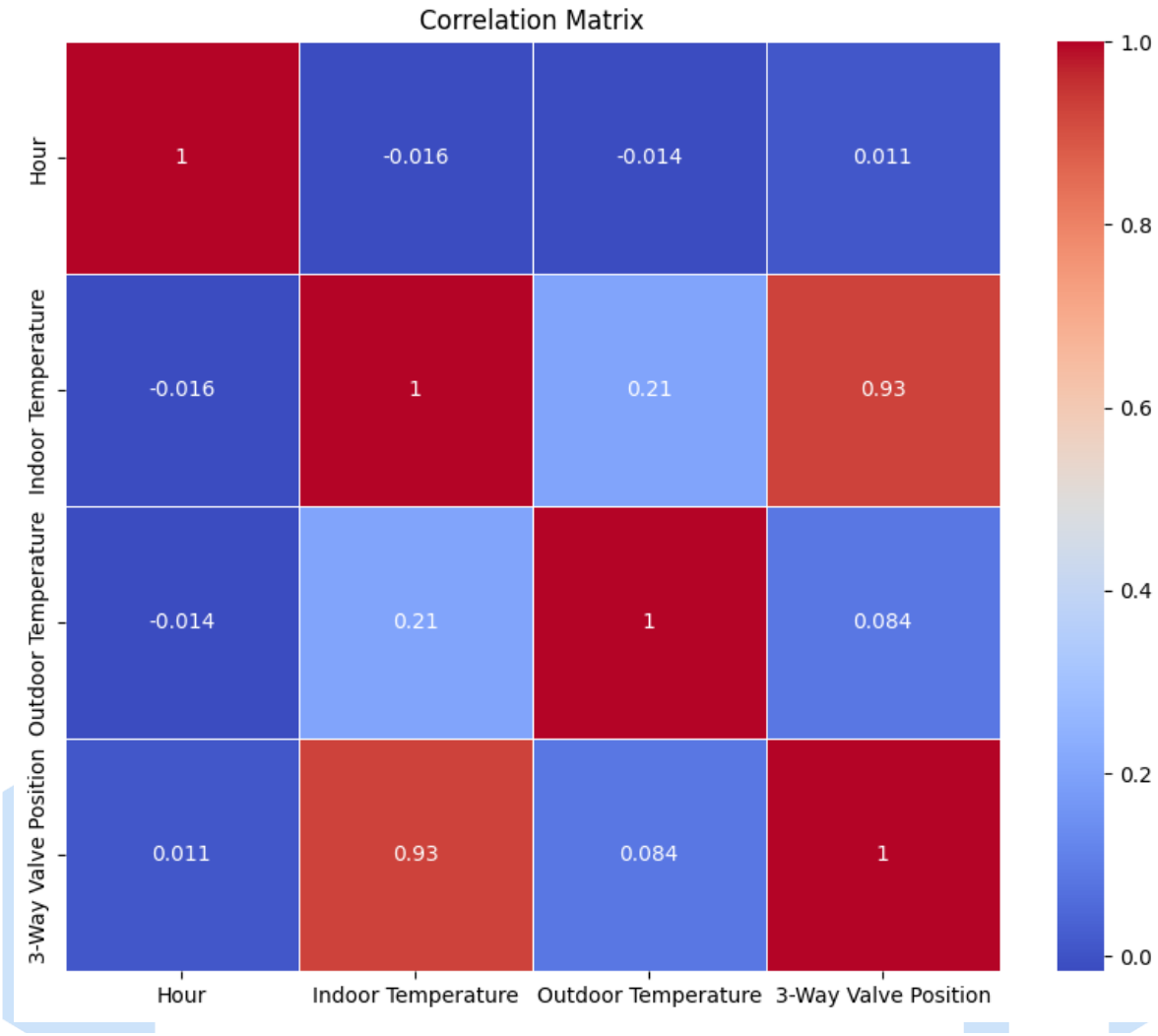


شکل ۰-۱ نمایش پراکندگی داده ها

برای بررسی میزان همبستگی ستون های دیتاست ماتریس همبستگی را رسم می کنیم:

```
# Compute the correlation matrix
correlation_matrix = data1.corr()
# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

بررسی ماتریس همبستگی نشان می دهد دمای هوای داخل اتاق بیشترین همبستگی را با موقعیت شیر سه راهه دارد.

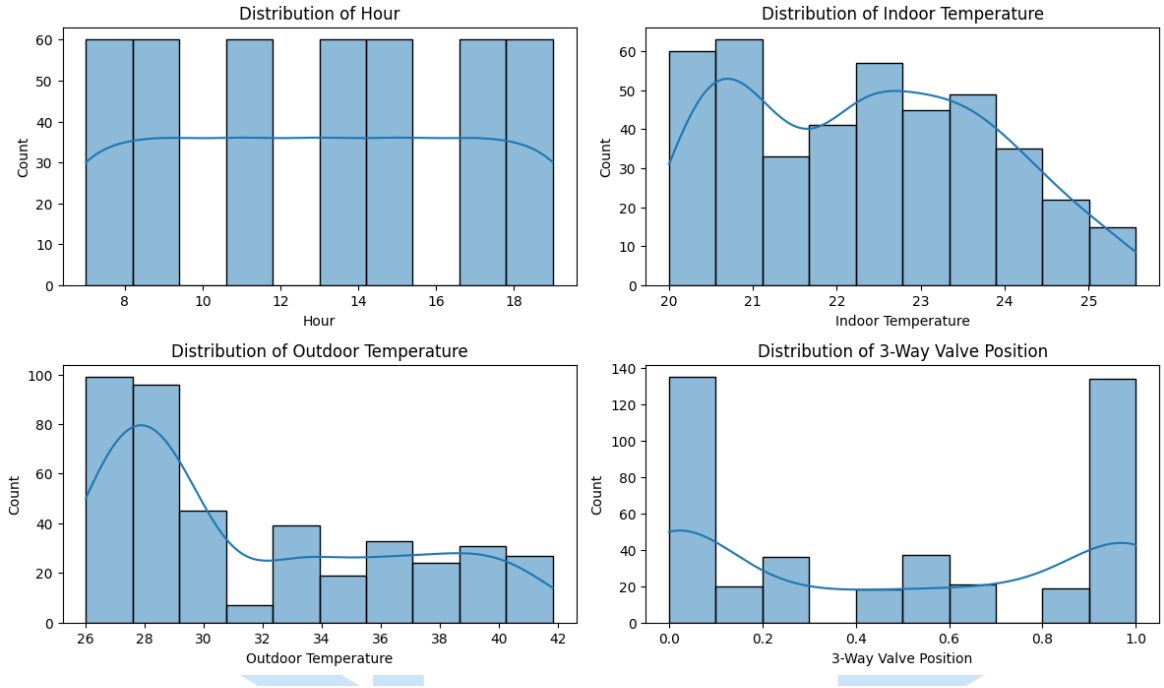


شکل ۲-۰۱ ماتریس همبستگی

همانطور که انتظار می رود، دمای هوای داخل اتاق و دمای بیرون به ترتیب بیشترین میزان همبستگی با موقعیت شیر سه راهه را دارد.

نمودار توزیع داده ها با استفاده از sns.histplot به صورت زیر رسم می شود:

```
plt.figure(figsize=(12, 10))
for i, column in enumerate(data1.columns, 1):
    plt.subplot(3, 2, i)
    sns.histplot(data[column], kde=True)
    plt.title(f'Distribution of {column}')
plt.tight_layout()
plt.show()
```



شکل ۰-۳ نمودار پراکندگی داده ها

پیاده سازی الگوریتم Q-Learning

ابتدا محیط سیستم HVAC را به‌گونه ای پیاده سازی می‌کنیم که در آن عامل با تنظیم موقعیت شیر تعامل دارد. این محیط برای مدل سازی برهمکنش های دینامیکی بین موقعیت شیر سه راهه و دمای داخل اتاق طراحی شده است. کلاس HVACEnvironment، برای شبیه سازی رفتار سیستم HVAC ایجاد شد. این کلاس مسئول تنظیم مجدد محیط، گذر از حالت ها بر اساس اعمال انجام شده توسط عامل، و محاسبه دمای داخلی و مصرف انرژی حاصل است.

حالت و عمل در این محیط به صورت زیر تعریف شده است:

State	Action
Indoor Temperature **Outdoor Temperature** **Hour**	**Valve Position**

$$\text{statet} = (\text{Indoor Temperaturet}_t.\,\text{Outdoor Temperaturet}_t.\,\text{Hourt}_t)$$

$$\text{actiont} \in \{0.0.1.0.2.\,..\,..\,1.0\}$$

معیار مصرف انرژی به این‌صورت است که موقعیت شیر بین 0 تا 1تغییر می کند و هرچه بیشتر باشد مصرف انرژی بیشتر است چرا که شیر بازتر شده است.

میخواهیم دمای اتاق در ۲۲ درجه بماند و در عین حال کمترین میزان مصرف انرژی را شاهد باشیم. بنابراین معیار مصرف انرژی به صورت زیر در نظر گرفته شده است:

$$\text{Energy} = |\text{Setpoint Temperature} - \text{Next Indoor Temperature}| \times \text{Valve Position} + |\text{Setpoint Temperature} - \text{Next Indoor Temperature}|$$

تاثیر موقعیت شیر بر دمای داخل اتاق توسط ضریب انتقال حرارتی شبیه سازی شده است. دمای اتاق به صورت زیر در محیط وارد می شود:

$$\Delta\text{Temperature} = (\text{Outdoor Temperature} - \text{Indoor Temperature}) \times \text{Heat Transfer Coefficient} \times \text{Valve Position}$$

$$\text{Next Indoor Temperature} = \text{Indoor Temperature} + \Delta\text{Temperature}$$

$$\text{next\_state} = (\text{Next Indoor Temperature}. \text{Outdoor Temperature}. \text{Hour})$$

تابع پاداش بصورت Energy Consumption–در نظر گرفته می شود.

$$\text{Reward} = -\text{Energy Consumption}$$

```python
class HVACEnvironment:
    def __init__(self, data, timestep=1, sim_steps=5):
        self.data = data
        self.current_index = 0
        self.timestep = timestep
        self.sim_steps = sim_steps
        self.current_temp = None

    def reset(self):
        self.current_index = 0
        row = self.data.iloc[self.current_index]
        self.current_temp = row['Indoor Temperature']
        return (self.current_temp, row['Outdoor Temperature'], row['Hour'])

    def step(self, action):
        row = self.data.iloc[self.current_index]
        outdoor_temp = row['Outdoor Temperature']
        hour = row['Hour']

        heat_transfer_coefficient = 0.01
        for _ in range(self.sim_steps):
            temp_change = (outdoor_temp - self.current_temp) * heat_transfer_coefficient * action
            self.current_temp += temp_change

        energy = abs(row['Setpoint Temperature'] - self.current_temp) * action + abs(row['Setpoint Temperature'] - self.current_temp)
        reward = -(energy)

        self.current_index += self.timestep
        if self.current_index >= len(self.data):
            done = True
            next_state = (self.current_temp, outdoor_temp, hour)
        else:
            next_row = self.data.iloc[self.current_index]
            next_state = (self.current_temp, next_row['Outdoor Temperature'], next_row['Hour'])
            done = False

        return next_state, reward, done
```

لازم به ذکر است، sim_steps مشخص می کند که عامل چقدر می تواند تصمیم بگیرد و تغییرات محیط را مشاهده کند. تعداد بیشترsim_steps به معنای تصمیم گیری ها و به روز رسانی های مکرر است. سیستم‌های تهویه مطبوع در یک مرحله زمانی عمل نمی‌کنند، بلکه به طور مداوم در طول زمان تنظیم

می‌شوند. شبیه سازی چند مرحله ای امکان تقریب واقعی تری از نحوه واکنش سیستم به اقدامات کنترلی در طول زمان را فراهم می کند.

طراحی عامل (Q-Learning Agent)

حال عامل را با استفاده از Q Learning طراحی می کنیم. برای این منظور کلاس QLearningAgent را تشکیل می دهیم. عامل یک تابع ارزشQ، Q(s,a) را می آموزد که پاداش تجمعی مورد انتظار را برای انجام عملa در حالت s و دنبال کردن سیاست بهینه تخمین می زند. تابع ارزش Q بوسیله معادله بلمن بروز می شود.

$$Q(s.a) \leftarrow Q(s.a) + \alpha(r + \gamma max_{a'} Q(s'.a') - Q(s.a))$$

α نرخ یادگیری، γ ضریب تخفیف،r پاداش دریافتی پس از انجام عمل a در حالت s و s' حالت بعدی است. همچنین انجام هر عمل در این عامل از استراتژی *epsilon greedy* پیروی می کند. بدین صورت که نرخ اکتشاف را ۱ در نظر می گیریم به این معنی که عامل با احتمال بالایی برای انجام عمل تصادفی شروع می کند. نرخ کاهش آن را 0.995 در نظر می گیریم به این معنی که پس از هر قسمت، نرخ اکتشاف در ۰,۹۹۵ ضرب می شود و به آرامی احتمال انجام عمل تصادفی کاهش می دهد.

مقادیر هایپر پارامتر ها در ابتدا به صورت زیر است:

α	**0.1**
γ	**0.9**
exploration_rate	**1**
exploration_decay	**0.995**

در نهایت توابعی برای رسم نمودار پاداش تجمعی و همچنین اجرای شبیه سازی با عامل آموزش دیده تعریف می شوند.

```python
class QLearningAgent:
    def __init__(self, env, learning_rate=0.1, discount_factor=0.9, exploration_rate=1.0, exploration_decay=0.995):
        self.env = env
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_rate = exploration_rate
        self.exploration_decay = exploration_decay
        self.q_table = {}
        self.cumulative_rewards = []
        self.actions = np.arange(0, 1.1, 0.1)

    def get_q_value(self, state, action):
        return self.q_table.get((state, action), 0.0)

    def update_q_value(self, state, action, reward, next_state):
        best_next_action = max(self.actions, key=lambda x: self.get_q_value(next_state, x))
        td_target = reward + self.discount_factor * self.get_q_value(next_state, best_next_action)
        td_error = td_target - self.get_q_value(state, action)
        new_q_value = self.get_q_value(state, action) + self.learning_rate * td_error
        self.q_table[(state, action)] = new_q_value

    def choose_action(self, state):
        if random.random() < self.exploration_rate:
            return random.choice(self.actions)
        else:
            return max(self.actions, key=lambda x: self.get_q_value(state, x))

    def train(self, episodes, max_steps_per_episode):
        for episode in range(episodes):
            state = self.env.reset()
            total_reward = 0

            for step in range(max_steps_per_episode):
                action = self.choose_action(state)
                next_state, reward, done = self.env.step(action)
                self.update_q_value(state, action, reward, next_state)
                state = next_state
                total_reward += reward

                if done:
                    break

            self.exploration_rate *= self.exploration_decay
            self.cumulative_rewards.append(total_reward)
            print(f"Episode {episode + 1}: Total Reward: {total_reward}, Exploration Rate: {self.exploration_rate}")

    def plot_cumulative_rewards(self):
        plt.plot(self.cumulative_rewards)
        plt.title('Cumulative Reward during Training')
        plt.xlabel('Episode')
        plt.ylabel('Cumulative Reward')
        plt.grid(True)
        plt.show()

    def play(self, episodes, max_steps_per_episode):
        for episode in range(episodes):
            state = self.env.reset()
            total_reward = 0

            for step in range(max_steps_per_episode):
                action = self.choose_action(state)
                next_state, reward, done = self.env.step(action)
                state = next_state
                total_reward += reward

                if done:
                    break

            print(f"Episode {episode + 1}: Total Reward: {total_reward}")
```

آموزش و تست عامل

حال داده را به دو بخش آموزش و تست تقسیم و عملکرد عامل را می سنجیم. به این ترتیب که دیتای مربوط به روز اول که نیمه اول دیتاست را برای آموزش عامل و دیتای روز دوم را برای تست عامل اختصاص می دهیم:

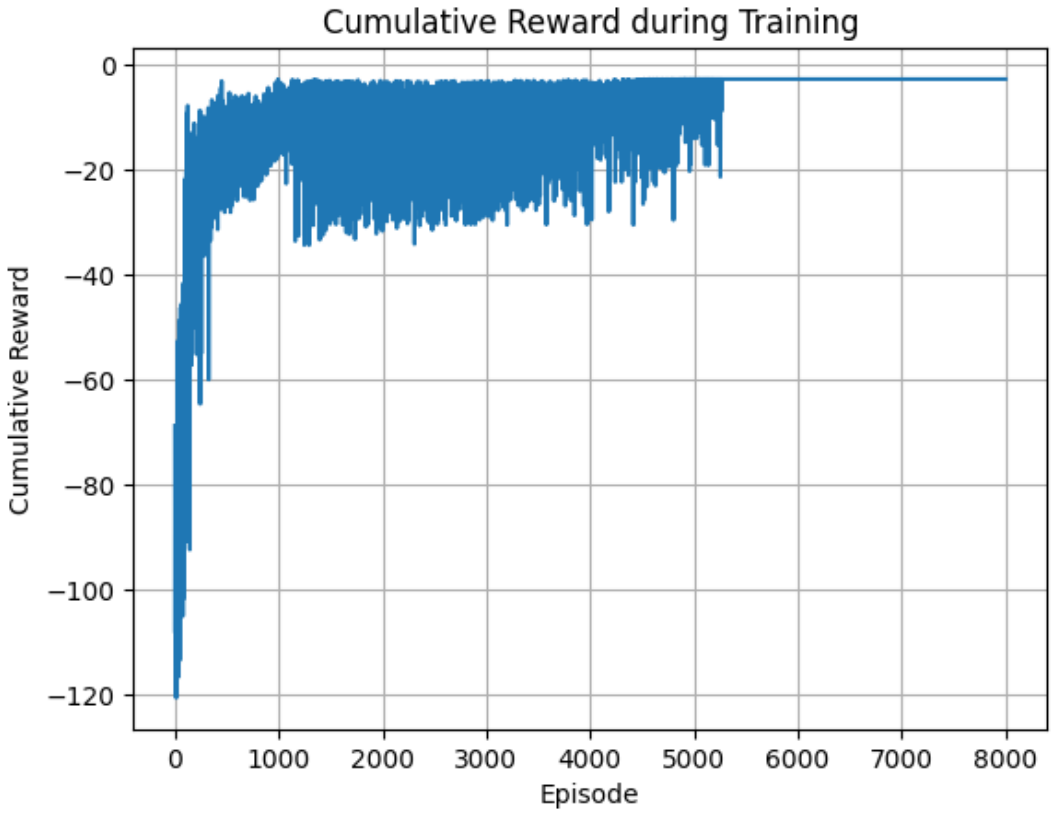
```python
train_data = data[:len(data) // 2]
test_data = data[len(data) // 2:]
```

حال عامل را آموزش داده و سپس آن را تست می کنیم:

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

    # Create the environment
    env = HVACEnvironment(data)

    # Create the Q-learning agent
    agent = QLearningAgent(env)

    # Training the agent
    agent.train(episodes=8000, max_steps_per_episode=30)

    # Plot cumulative rewards and average rewards
    agent.plot_cumulative_rewards()

    # Create the environment for testing
    test_env = HVACEnvironment(test_data)

    # Create the Q-learning agent with the same environment
    test_agent = QLearningAgent(test_env)

    # Playing the environment with the trained agent
    agent.play(episodes=10, max_steps_per_episode=50)
```

نتایج به صورت زیر قابل مشاهده است:

```
Episode 5264: Total Reward: -2.7134443001623203, Exploration Rate: 3.472938122212649e-12
Episode 5265: Total Reward: -2.7134443001623203, Exploration Rate: 3.455573431601586e-12
Episode 5266: Total Reward: -2.7134443001623203, Exploration Rate: 3.438295564443578e-12
Episode 5267: Total Reward: -2.7134443001623203, Exploration Rate: 3.42110408662136e-12
Episode 5268: Total Reward: -2.7134443001623203, Exploration Rate: 3.4039985661882534e-12
Episode 5269: Total Reward: -2.7134443001623203, Exploration Rate: 3.386978573357312e-12
Episode 5270: Total Reward: -2.7134443001623203, Exploration Rate: 3.3700436804905254e-12
Episode 5271: Total Reward: -2.7134443001623203, Exploration Rate: 3.3531934620880727e-12
Episode 5272: Total Reward: -2.7134443001623203, Exploration Rate: 3.3364274947776323e-12
Episode 5273: Total Reward: -2.7134443001623203, Exploration Rate: 3.3197453573037443e-12
Episode 5274: Total Reward: -2.7134443001623203, Exploration Rate: 3.3031466305172254e-12
Episode 5275: Total Reward: -2.7134443001623203, Exploration Rate: 3.286630897364639e-12
```



شکل ۰-۴ نمودار همگرایی پاداش تجمعی در فرآیند آموزش

همانطور که پیداست پاداش تجمعی در فرآیند آموزش عامل پس از حدود ۵۲۰۰ اپیزود همگرا شده است.

نتایج تست عامل به صورت زیر است:

```
Episode 1: Total Reward: -2.8211151688657004
Episode 2: Total Reward: -2.8211151688657004
Episode 3: Total Reward: -2.8211151688657004
Episode 4: Total Reward: -2.8211151688657004
Episode 5: Total Reward: -2.8211151688657004
Episode 6: Total Reward: -2.8211151688657004
Episode 7: Total Reward: -2.8211151688657004
Episode 8: Total Reward: -2.8211151688657004
Episode 9: Total Reward: -2.8211151688657004
Episode 10: Total Reward: -2.8211151688657004
```

پیداست عامل توانسته است شیر را به خوبی کنترل کند. پاداش تجمعی در مرحله تست مقدار 0.1 با پاداش تجمعی فرآیند آموزش اختلاف دارد.

بهینه سازی هایپر پارامترها با استفاده از GridSearch

چند مقدار را برای هر یک از هایپر پارامترها در نظر می گیریم. سپس با استفاده از GridSearh آموزش را انجام داده و تریبی از هایپر پارامتر ها که بیشترین مقدار پاداش را گزارش کند به عنوان هایپر پارامتر بهینه معرفی می شود.

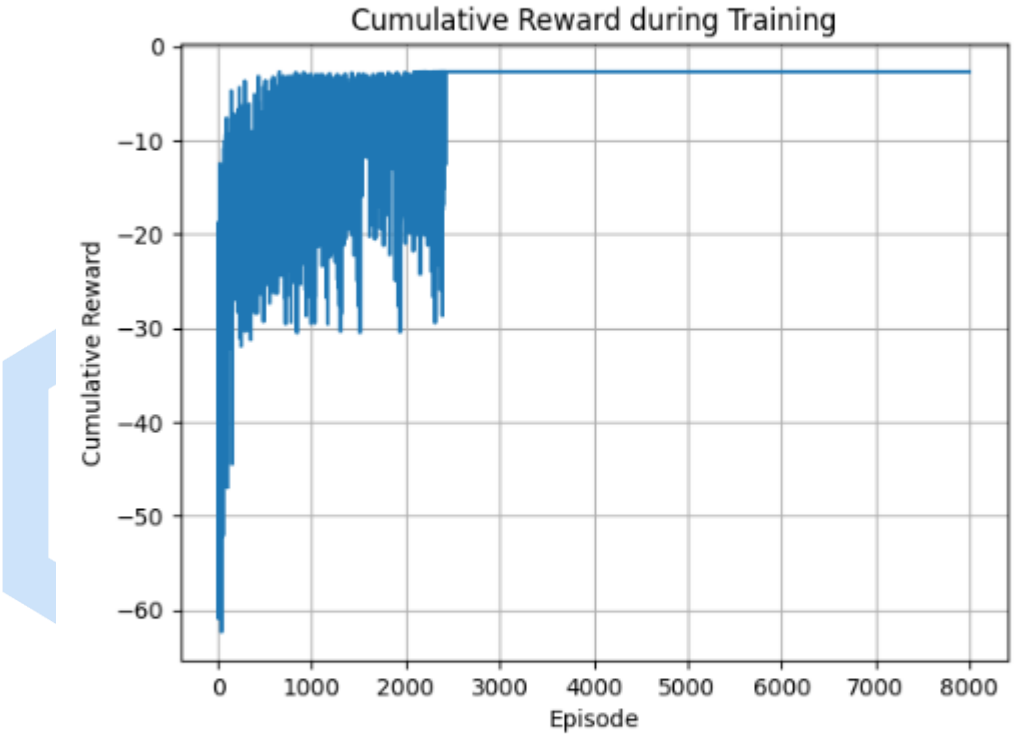
```python
# Define parameter grid
param_grid = {
    'learning_rate': [0.01, 0.1, 0.5],
    'discount_factor': [0.9, 0.95, 0.99],
    'exploration_rate': [1.0, 0.8, 0.5],
    'exploration_decay': [0.995, 0.99, 0.9]
}

best_score = -float('inf')
best_params = {}

for params in ParameterGrid(param_grid):
    print(f"Testing parameters: {params}")
    env = HVACEnvironment(data)
    agent = QLearningAgent(env, **params)
    avg_reward = agent.train(episodes=200, max_steps_per_episode=30)  # Adjust episodes and steps as needed

    if avg_reward > best_score:
        best_score = avg_reward
        best_params = params

    print(f"Average Reward: {avg_reward}")

print(f"Best parameters found: {best_params}")
```

نتیجه به صورت زیر است:

```
Best parameters found: {'discount_factor': 0.95, 'exploration_decay': 0.99, 'exploration_rate': 0.5, 'learning_rate': 0.5}
```

حال مجددا عامل را با هایپر پارامترهای فوق آموزش داده و نتایج را مقایسه می کنیم:

```
Episode 7993: Total Reward: -2.7134443001623203, Exploration Rate: 6.472624645660527e-36
Episode 7994: Total Reward: -2.7134443001623203, Exploration Rate: 6.407898399203922e-36
Episode 7995: Total Reward: -2.7134443001623203, Exploration Rate: 6.343819415211883e-36
Episode 7996: Total Reward: -2.7134443001623203, Exploration Rate: 6.280381221059765e-36
Episode 7997: Total Reward: -2.7134443001623203, Exploration Rate: 6.217577408849168e-36
Episode 7998: Total Reward: -2.7134443001623203, Exploration Rate: 6.155401634760676e-36
Episode 7999: Total Reward: -2.7134443001623203, Exploration Rate: 6.09384761841307e-36
Episode 8000: Total Reward: -2.7134443001623203, Exploration Rate: 6.03290914222894e-36
```




```
Episode 1: Total Reward: -2.8211151688657004
Episode 2: Total Reward: -2.8211151688657004
Episode 3: Total Reward: -2.8211151688657004
Episode 4: Total Reward: -2.8211151688657004
Episode 5: Total Reward: -2.8211151688657004
Episode 6: Total Reward: -2.8211151688657004
Episode 7: Total Reward: -2.8211151688657004
Episode 8: Total Reward: -2.8211151688657004
Episode 9: Total Reward: -2.8211151688657004
Episode 10: Total Reward: -2.8211151688657004
```

همانطور که پیداست نمودار پاداش تجمعی نسبت به حالت قبل بسیار سریعتر همگرا شده است و آموزش عامل در حدود ۲۵۰۰ اپیزود کامل می شود.

مقایسه مصرف انرژی

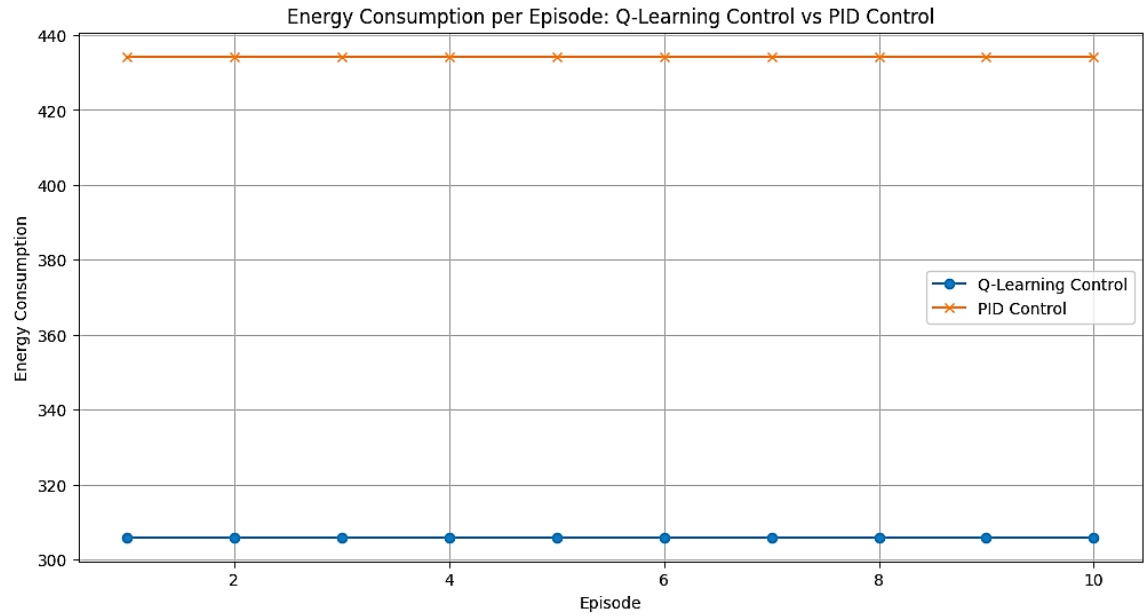
در این قسمت ابتدا برای دیتاست که دارای مقادیر موقعیت شیر سه راهه و دمای داخل و بیرون اتاق است، معیار مصرف انرژی پیاده سازی می شود. در واقع هر جا برای محاسبه به مقدار دمای فعلی نیاز شد از دمای داخل اتاق و موقعیت شیر در آن حالت در دیتاست استفاده می شود. سپس مصرف انرژی برای کنترلQ-Learning باتوجه به اعمال ا نجام شده توسط عامل برای کنترل شیر سه راهه محاسبه می شود.

```
def calculate_energy_consumption(env, data, agent=None, num_episodes=1):
    energy_consumption_per_episode = []

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

        while not done:
            if agent is None:
                # Control without agent (use data directly)
                action = data.iloc[env.current_index]['3-Way Valve Position']
                next_state = (data.iloc[env.current_index]['Indoor Temperature'],
                              data.iloc[env.current_index]['Outdoor Temperature'],
                              data.iloc[env.current_index]['Hour'])
                reward = -(abs(data.iloc[env.current_index]['Setpoint Temperature'] - next_state[0]) * action +
                           abs(data.iloc[env.current_index]['Setpoint Temperature'] - next_state[0]))
                done = env.current_index >= len(data) - 1
                env.current_index += 1
            else:
                # Control with agent
                action = agent.choose_action(state)
                next_state, reward, done = env.step(action)

            total_energy_consumption += -reward
            state = next_state

        energy_consumption_per_episode.append(total_energy_consumption)
```

حال مصرف انرژی را در داده های تست برای کنترلر پایه PID و کنترل مبتنی بر Q-Learning گزارش و درصد بهبود بهینه سازی مصرل انرژی را محاسبه می کنیم:

```
num_test_episodes = 10

# Calculate energy consumption with and without control
energy_consumption_with_control = calculate_energy_consumption(test_env, test_data, agent, num_episodes=num_test_episodes)
energy_consumption_without_control = calculate_energy_consumption(test_env, test_data, num_episodes=num_test_episodes)

# Plot energy consumption
plt.figure(figsize=(12, 6))
plt.plot(range(1, num_test_episodes + 1), energy_consumption_with_control, label='With Control', marker='o')
plt.plot(range(1, num_test_episodes + 1), energy_consumption_without_control, label='Without Control', marker='x')
plt.xlabel('Episode')
plt.ylabel('Energy Consumption')
plt.title('Energy Consumption per Episode: With Control vs Without Control')
plt.legend()
plt.grid(True)
plt.show()

# Calculate percentage optimization
avg_energy_with_control = np.mean(energy_consumption_with_control)
avg_energy_without_control = np.mean(energy_consumption_without_control)
percentage_optimization = ((avg_energy_without_control - avg_energy_with_control) / avg_energy_without_control) * 100

print(f"Average energy consumption with control: {avg_energy_with_control}")
print(f"Average energy consumption without control: {avg_energy_without_control}")
print(f"Percentage energy consumption optimization: {percentage_optimization:.2f}%")
```

نتیجه به صورت زیر است:




```
Average energy consumption Q-Learning control: 305.9206311000007
Average energy consumption PID control: 434.2789410820001
Percentage energy consumption optimization: 29.56%
```

همانطور که پیداست، با معیار تعریف شده برای مصرف انرژی، کنترل مبتنی بر Q-Learning ۲۹/۵۶٪ نسبت به کنترل پایه PID در مصرف انرژی موفق تر است.

نتیجه گیری

این پروژه کاربرد Q-learning را در بهینه سازی موقعیت شیر سه راهه در یک سیستم HVACبه نمایش گذاشته است که منجر به کاهش ۲۹/۵۶درصدی می شود.

برای افزایش عملکرد عامل یادگیری Q، بهینه‌سازی هایپرپارامتر با استفاده از جستجوی شبکه‌ای GridSearch انجام شد. این فرآیند شامل جستجوی سیستماتیک ترکیبات مختلف فراپارامترها، مانند نرخ یادگیری، ضریب تخفیف، نرخ اکتشاف و نرخ کاهش اکتشاف است. هدف جستجوی شبکه شناسایی هایپرپارامترهای بهینه ای است که کارایی عامل را از نظر سرعت یادگیری و صرفه جویی در مصرف انرژی به حداکثر می رساند.

کارهای آتی

۱. پیاده سازی الگوریتم یادگیری Q در PLC
۲. پیاده سازی DQN و مقایسه عملکرد و هزینه آن نسبت به Q-Learning

منابع

۱. اصول و مبانی سیستمهای هوشمند کنترل و BMS/مولف رسول حدادی‌نیستانک.

2. Faddel, S., Tian, G., Zhou, Q., & Aburub, H. (2020, March). Data driven q-learning for commercial hvac control. In *2020 SoutheastCon* (pp. 1-6). IEEE.
3. Siraskar, R. (2021). Reinforcement learning for control of valves. *Machine Learning with Applications*, *4*, 100030.