# Homework2_B1128019

# Smart Home Energy Optimization using Q-Learning

## 1. Introduction

> This project implements a reinforcement learning (Q-Learning) approach to train an agent for smart home energy management. The agent learns to switch electrical appliances on or off based on the time of day, electricity price, and user presence to minimize energy costs while preserving user comfort. By learning from historical data, the agent develops context-aware behavior that adapts to real-life usage scenarios.

## 2. Dataset and Preprocessing

- **Dataset:** Includes hourly data over 24 hours
- **Features:**
  - Time (0–23 hours)
  - Electricity Price
  - User Activity (1 = active/present, 0 = inactive/away)
  - Appliance State (1 = on, 0 = off)
- **State Representation:**

```
state = (Time, UserActivity, ApplianceState)
```

## 3. Reinforcement Learning Design

### MDP Setup:

- **State:** Tuple of (time, user activity, appliance state)
- **Actions:**
  - 0: Turn off appliance
  - 1: Turn on appliance

### Reward Function:

#### In the first time, I am code this:

```
def get_reward(user_activity, appliance_state, electricity_price):
    if user_activity == 1 and appliance_state == 0:
        return -10
    elif user_activity == 1 and appliance_state == 1:
        return -electricity_price
    elif user_activity == 0 and appliance_state == 1:
        return -electricity_price - 2
    else:
        return 1
```

But it has a big issue, agent thought the rewards too high when the person who is turn off the light, that way will make the decision plot looks so conservative.

#### After the many trials, the this function's plot

```
def get_reward(user_activity, appliance_state, electricity_price):
    if user == 1 and device == 0:
        return -3
    elif user == 1 and device == 1:
        return -price
    elif user == 0 and device == 1:
        return -price - 0.5
    else:
        return 2
```

> This reward structure balances energy cost and user comfort by penalizing inappropriate actions and rewarding efficient behavior.

## 4. Q-Learning Agent and Training Setup

- **Agent:** Q-table with ε-greedy strategy
- **Key parameters:**
  - alpha = 0.001
  - gamma = 0.95
  - initial_epsilon = 0.3
- **Exploration decay:**

```
agent.epsilon = max(0.01, 0.3 * (0.995 ** epoch))
```

- 
- **Training loop:** 3000 epochs using tqdm for progress visualization

## 5. Results and Learning Curve

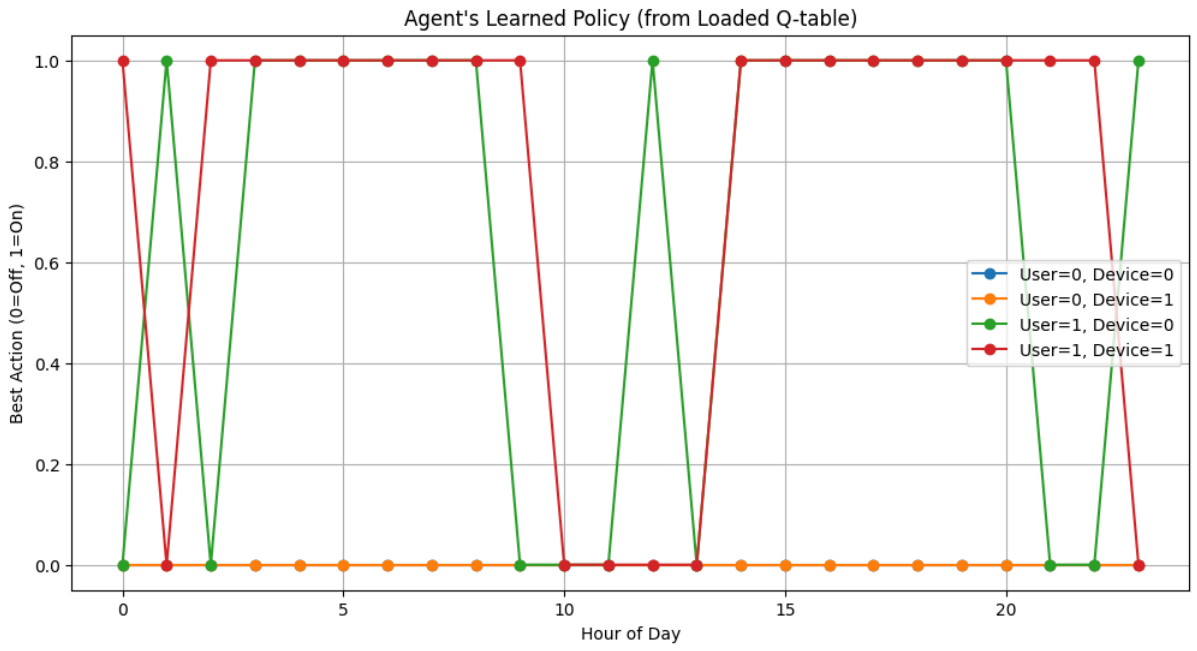
### Training Reward Trend:

> The agent's total reward started negative and gradually improved over time, stabilizing after ~1000 epochs. This indicates successful learning and convergence.

### 📊 Strategy Map:

> The agent learns context-sensitive behaviors:

- Keeps appliances on during active hours
- Turns them off during inactive periods
- Makes conditional decisions during transitional hours (e.g., early morning or late night)




## 6. Performance Evaluation

| Model | Total Reward |
|---|---|
| Q-Learning Agent | +570.03 |
| Random Strategy | −331.26 |

> The trained agent significantly outperforms the random baseline, showing an improvement of nearly +900 in total reward.

## 7. Q-table Storage

- Saved using pickle:

```
with open('q_table.pkl', 'wb') as f:
    pickle.dump(dict(agent.q_table), f)
```

- 
- Reloadable for later testing or deployment.

## Discussion

> The agent initially adopted a conservative strategy (always on or always off) to avoid penalties. After tuning the reward function and using decay in exploration, it began making more nuanced decisions. The learned policy reflects a good balance between minimizing electricity costs and respecting user presence.

## Conclusion

> This project demonstrates the effectiveness of Q-Learning for smart home energy control. The trained agent adapts to user behavior and dynamic pricing to optimize energy usage while maintaining user comfort. The approach is promising for real-world smart energy systems.