CS403/603 - ML Project Proposal

INVERTED PENDULUM: CARTPOLE BALANCING USING Q LEARNING

Submitted

In partial fulfillment of the requirements of

CS 432 Reinforcement Learning

by

Ayush Kumar Sinha (200001012)

Submitted to

Prof. Dr. Surya Prakash

Department Of Computer Science and Engineering

Indian Institute of Technology Indore

A.Y 2022-2023

Abstract:

This project aims to train an agent to balance a pole on a cart using reinforcement learning techniques, specifically *Q-learning*. The agent interacts with the CartPole-v1 environment provided by *OpenAI Gym*, where it receives observations and rewards based on its actions. The goal is to develop a policy that maximizes the time the pole remains balanced on the cart. We propose a solution that utilizes discretization of the state space, *epsilon-greedy exploration*, and Q-learning algorithm to update action-value estimates. The performance of the agent is evaluated based on the time the pole is balanced within the bounds of a window and the average reward obtained across testing episodes.

Objectives:

- Train an agent to learn a policy for balancing the pole on the cart using Q-learning.
- Develop a solution that efficiently explores the state space while maximizing long-term rewards.
- Evaluate the performance of the agent based on its ability to balance the pole for extended durations.
- Assess the average reward obtained by the agent across testing episodes as a measure of its effectiveness.

Proposed Solution:

The proposed solution involves discretizing the continuous state space of the CartPole environment to facilitate Q-table learning. We employ an epsilon-greedy exploration strategy to balance between exploration and exploitation during training. The Q-learning algorithm is utilized to update action-values based on observed rewards and state transitions. The agent learns a policy that aims to maximize the time the pole remains balanced on the cart.

Methodology:

Detailed step wise description for the CartPole.py :

STEP 1: Import Required Libraries:

- Import necessary libraries such as `numpy`, `time`, `gym`, `cv2`, `pickle`, `KBinsDiscretizer` **from** `sklearn.preprocessing`, `math`, **and** `random`.

STEP 2: Create CartPole Environment:

- Initialize the CartPole environment using `gym.make('CartPole-v1', render_mode='rgb_array')`.

STEP 3: Define Parameters for Discretization:

- Define parameters for state discretization, including the number of bins, lower and upper bounds for angle and pole velocity.

STEP 4: Define Discretizer Function:

- Define a discretizer function to convert continuous states into discrete states using `KBinsDiscretizer`.

STEP 5: Main Algorithm:

- Initialize Q-Table:
    - Initialize the Q-table with zeros based on the discretized state and action space dimensions.
- Define Policy Function:
    - Define a policy function to choose actions based on the current state by selecting the action with the highest Q-value.
- Define New Q-Value Function:
    - Define a function to calculate the new Q-value using the temporal difference formula.
- Define Learning Rate Function:
    - Define a function to compute the decaying learning rate based on the episode number.
- Define Exploration Rate Function:

    - Define a function to compute the decaying exploration rate based on the episode number.
- Training Loop:
    - Set the number of training episodes (`n_episodes`**).**
    - Enter a loop to iterate through each episode.
    - Reset the environment and discretize the initial state.
    - Enter an episode loop to interact with the environment until the episode is done.
    - Choose an action based on the current state using the policy.
    - Implement epsilon-greedy exploration to occasionally choose a random action.
    - Take a step in the environment and get the next observation, reward, and done status.
    - Discretize the new state.
    - Compute the learning rate and update the Q-value using the Bellman equation.
    - Update the current state for the next iteration.
    - Render the environment and display the frame.
    - Break the episode loop if 'q' is pressed.
    - Save the Q-table to a file after training.
- Print Final Q-Table:
    - Print the final Q-table after training.
- Save Q-Table to File:
    - Save the final Q-table to a file using `pickle.dump`**.**
    - This is later used in the testing phase.

NOTE: The following CODE SNIPPET -

```
Q_table[current_state][action] = (1 - lr) * old_value + lr * learnt_value
# Update Q-value using the Bellman equation
```

tells us that we used the Bellman Equation of Q-Learning algorithm in the code to update the Q-table.

Detailed step wise description for the CartPole_Test.py :

STEP 1: Import Required Libraries:

- Import necessary libraries such as `numpy`, `cv2`, `gym`, `pickle`, `KBinsDiscretizer` from `sklearn.preprocessing`, `typing`, `math`, `random`, and `time`.

STEP 2: Load Q-Table from File:

- Attempt to load the Q-table from the file 'Q_table.pkl' using `pickle.load`.
- If the file does not exist, print a message indicating the absence of the Q-table and exit the program.

STEP 3: Print Q-Table for Verification:

- Print the loaded Q-table to verify its contents.

STEP 4: Create CartPole Environment:

- Initialize the CartPole environment using `gym.make('CartPole-v1', render_mode='rgb_array').`

STEP 5: Set Testing Parameters:

- Define the number of testing episodes (`n_testing_episodes`) and discretization parameters (`n_bins`, `lower_bounds`, `upper_bounds`).

STEP 6: Define Discretizer Function:

- Define a function `discretizer` to convert continuous states into discrete states using `KBinsDiscretizer`.

STEP 7: Define Policy Function:

- Define a function `policy` to choose actions based on the current state by selecting the action with the highest Q-value.

STEP 8: Initialize Evaluation Metrics:

- Initialize empty lists to store total rewards and episode durations (`total_rewards`, `episode_durations`).

STEP 9: Testing Loop:

- Enter a loop to iterate through each testing episode.
- Reset the environment and discretize the initial state.
- Initialize episode reward and start time.
- Enter the main loop for the episode:
    - Choose an action based on the current state using the policy.
    - Take a step in the environment and get the next observation, reward, and done status.
    - Discretize the new state.
    - Update the current state for the next iteration.
    - Render the environment and display the frame using `cv2.imshow`.
    - Accumulate the episode reward.
    - Break the loop if 'q' is pressed.

Calculate and Print Episode Metrics:

- Calculate the episode duration and print the episode number, duration, and total reward.

Store Episode Metrics:

- Store the episode reward and duration in the respective lists.

Print Average Metrics:

- Print the average total reward and average episode duration over all testing episodes.

Close OpenCV Windows and Environment:

- Close all OpenCV windows using `cv2.destroyAllWindows()`.
- Close the CartPole environment using `env.close()`.

Pseudocode For Q Learning ALgorithm:

```
Initialize Q-table with zeros or small random values
Initialize environment

For each episode from 1 to N:
    Reset environment to initial state
    Discretize the current state to get state s
    Set done to False
    Initialize episode reward to 0

    While not done:
        Choose action a based on the epsilon-greedy policy from state s
        Take action a in the environment
        Observe reward r and new state s'
        Discretize new state s' to get new state s'

        Calculate new Q-value using the Bellman equation:
        Q(s, a) = (1 - learning_rate) * Q(s, a) + learning_rate * (reward + discount_factor * max(Q(s', :)))

        Update Q-table with the new Q-value for the state-action pair (s, a)
        s = s'
        Add reward to episode reward

        If episode ends (done is True):
            Break the loop

    Store the episode reward

Print average episode rewards over all episodes
```

Dataset Information:

The primary source of data for training and testing the agent is the CartPole-v1 environment provided by OpenAI Gym. This environment provides observations consisting of the cart's position, velocity, pole angle, and angular velocity, along with rewards obtained by the agent for its actions. The dataset is generated dynamically through interactions between the agent and the environment during training and testing phases.

Performance Metrics:

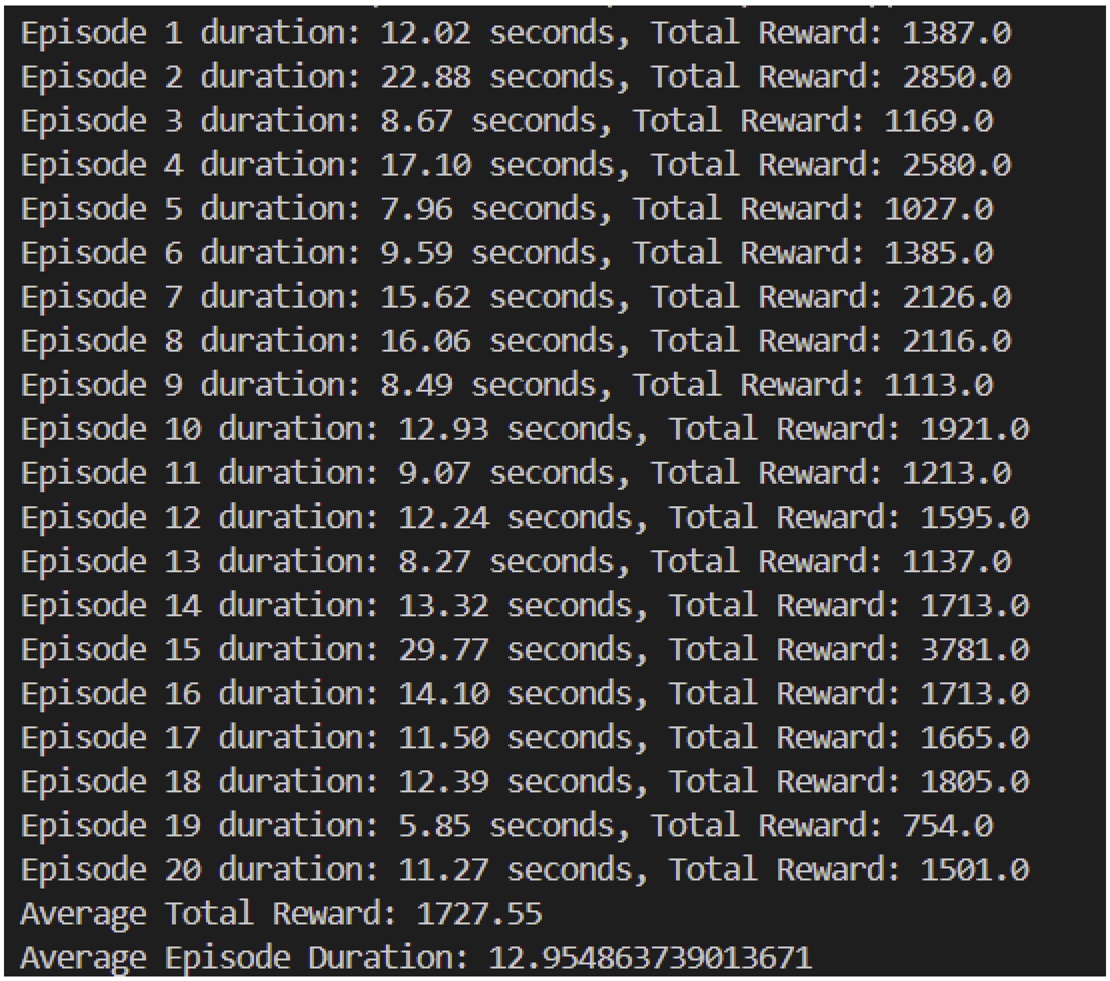
Two main performance metrics are considered for evaluating the agent:

- Time for Which the Pole Was Balanced within Bounds of Window: This metric measures the duration for which the pole remains within predefined bounds, indicating the effectiveness of the learned policy in balancing the pole.
- Average Reward for All Episodes in Testing: The average reward obtained by the agent across testing episodes provides insight into its overall performance and ability to achieve positive outcomes in the environment.

By analyzing these performance metrics, we can assess the effectiveness and efficiency of the proposed reinforcement learning approach for the CartPole balancing task.

Evaluation Results:

The main advantage of using a Q-table stored in a pickle file is that it allows you to reuse a trained Q-table without having to train the agent again. This can be particularly useful for evaluating the agent's performance or deploying the trained agent in different environments or scenarios.



```
Episode 1 duration: 12.02 seconds, Total Reward: 1387.0
Episode 2 duration: 22.88 seconds, Total Reward: 2850.0
Episode 3 duration: 8.67 seconds, Total Reward: 1169.0
Episode 4 duration: 17.10 seconds, Total Reward: 2580.0
Episode 5 duration: 7.96 seconds, Total Reward: 1027.0
Episode 6 duration: 9.59 seconds, Total Reward: 1385.0
Episode 7 duration: 15.62 seconds, Total Reward: 2126.0
Episode 8 duration: 16.06 seconds, Total Reward: 2116.0
Episode 9 duration: 8.49 seconds, Total Reward: 1113.0
Episode 10 duration: 12.93 seconds, Total Reward: 1921.0
Episode 11 duration: 9.07 seconds, Total Reward: 1213.0
Episode 12 duration: 12.24 seconds, Total Reward: 1595.0
Episode 13 duration: 8.27 seconds, Total Reward: 1137.0
Episode 14 duration: 13.32 seconds, Total Reward: 1713.0
Episode 15 duration: 29.77 seconds, Total Reward: 3781.0
Episode 16 duration: 14.10 seconds, Total Reward: 1713.0
Episode 17 duration: 11.50 seconds, Total Reward: 1665.0
Episode 18 duration: 12.39 seconds, Total Reward: 1805.0
Episode 19 duration: 5.85 seconds, Total Reward: 754.0
Episode 20 duration: 11.27 seconds, Total Reward: 1501.0
Average Total Reward: 1727.55
Average Episode Duration: 12.954863739013671
```

Based on the maximum balancing of the pole time of more than 29 seconds shows that the trained Q-table is performing well in the CartPole-v1 environment.

Conclusion:

In conclusion, the Q-learning approach implemented for the CartPole-v1 environment has demonstrated promising results. Leveraging a pre-trained Q-table, the evaluation phase showcased the agent's capability to effectively balance the pole, achieving a high episode duration and total reward across multiple testing episodes. This success underscores the effectiveness of reinforcement learning techniques, specifically Q-learning, in mastering complex control tasks like balancing an unstable system.

The discretization of the state space and the epsilon-greedy policy selection allowed the agent to generalize its learned strategies to unseen states, contributing to its robust performance. Additionally, the visualization through OpenCV provided valuable insights into the agent's decision-making process, offering a clear representation of its interactions with the environment.

While this implementation yielded satisfactory results, there's always room for further optimization and exploration. Future work could focus on tuning hyperparameters, exploring different discretization techniques, or experimenting with alternative reinforcement learning algorithms to potentially enhance the agent's performance and adaptability to various scenarios.

Overall, this project serves as a testament to the potential of reinforcement learning in solving challenging control problems and paves the way for future endeavors in advancing autonomous systems and robotics applications.