Q-learning for Multiple Stage Problem

Extend Classic Cart-Pole Project on multi-tasking

Group 12

Chen Sun, 20100804

Shouyue Hu, 20095119

James Wang, 20119632

1. Abstract

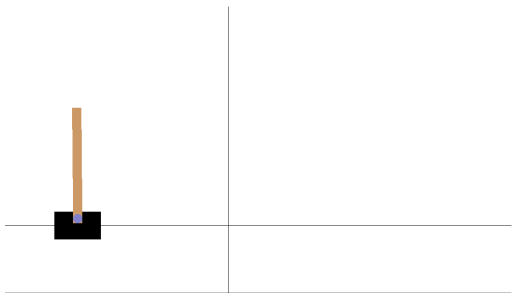
We implemented a variation of the CartPole problem raised by Barto, Sutton, and Anderson[1]. The Q-learning approach is used to solve this continuous time-space problem where different periodic goals require different strategies. In our experiment, the cart has to balance the pole and is supposed to drive from left to right and oscillate near a single position. Since Q-learning is designed for solving discrete time-space problems, a function is introduced to transform continuous time-space to discrete with the idea of binning. Our result indicates that with the help of discrete-binning, Q-learning can better solve the CartPole parking problem in a generalized way.

2. Problem Formulation

2.1 Introduction to Cart-Pole problem

CartPole is a classic reinforcement learning experiment. The cart moves along a frictionless track employing a powerless joint attached to the cart. The system is controlled by applying a force of +1 or -1 to the trolley. The goal is to prevent the pendulum from falling over. A +1 is awarded for each time step the rod remains upright. The episode ends when the rod leaves the vertical by more than 15 degrees, or the trolley moves more than 2.4 units from the center.
To explore how Q-learning can be applied to a multiple-stage problem that requires a general solution. Our work presents an extension based on the cart-pole problem in OpenAI gym.

At the beginning of each episode, the cart respawns on the plane's left side instead of in the middle position. Its goal is to move towards the destination area(where the two lines cross) on the plane without falling over. The purpose of the cart is to oscillate around the destination and thus gain rewards.



2.2 State Space

In this problem, the set is defined as:

1. Position of the cart [-4.8, 4.8]
2. The momentum of the cart [-infinite, infinite]
3. The angle between the pole and horizontal. [-15,15]

To use a Q-table to achieve these three stages, a targeted reward needed to be set so that the agent can use different strategies at different stages, such as trying to speed up while maintaining balance when moving away from the target point, and slowing down if it has already passed the target point or is close to the target point.

2.3 Binning

The CartPole problem is a continuous problem, in which the number of states is infinite, so Q-learning cannot be used directly. Q-learning is designed for discrete state space whereas the cart-pole problem has continuous state space, such as the relative position of the cart over time. In the first Q-learning algorithm *(see attachment simplified-q-learning.py)*[2][4] we use, the continuous-to-discrete time-space transformation was defined by a function that calculates the continuous values to discrete values in numerical methods. This makes the Q-learning implementation suffer from a very long episode learning process.

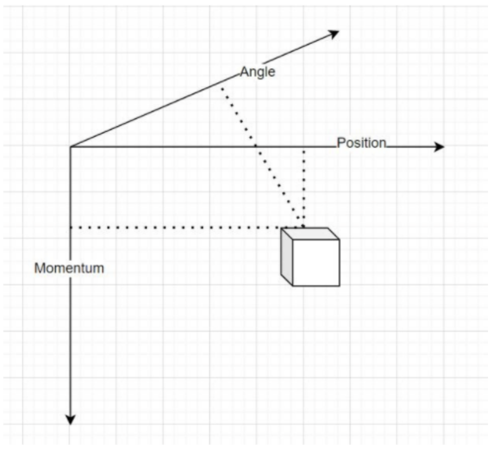


Figure2. A bin can be represented as The intersection of a segment of the three dimensions in the 3D space.

In order to enable the use of Q-learning, binning is adopted to convert continuous variables into discrete ones by partitioning, thus allowing the use of Q-learning. Binning divides the continuous space into several small intervals called bins. Each bin represents a range of continuous values and is labelled as a discrete value. To represent the position of the cart, its position in the state will be mapped to the bins with corresponding discrete values. E.g. A bin of 3 for Position(range(-4.8-4,8) will be [-4.8,1.6][-1.6,1.6][1.6,4.8].

Similarly, for the time state, every 0.02 second is considered a one-time state. This effectively reduces state space, shortens the episodes of learning. Detailed binning tuning strategy will be studied later in the chapter on *4.3 Final Bin Number setting(Parameter tuning)*.

2.4 Action space

Only two actions are permitted, exerting unit force to the cart from left or right.

2.5 Reward

In the code, the destination is at pixel 500 of the screen, the reward, or cost is set mainly according to the distance between the cart and the destination in quadratic. For example, if a state is at pixel 100, so the cost is -(100-500)^2/10000 = -4. Staying in this position will cost -4 every time step. Also, reaching the destination will directly reward 100, losing the balance of the pole will be punished -1000, and the cart hitting the left and right boundary will be punished -10000. Meanwhile, losing balance and hitting boundaries would lead to the end of the episode.

3. Algorithm and implementation

3.1Training Environment

The new environment, “CartPole-v474”, is modified from the default “cart-pole-v0” environment in python OpenAI-Gym. [3]

3.2 Reward function

The goal of the modified CartPole problem is to balance the pole as long as possible and expects the cart to move to the target point (destination position) and eventually stay on the target point.

Thus, the cart agent has three periodic goals:

1. Balancing the cart,
2. Driving to the right without failing
3. Trying to pass through the destination as many times as possible.

Using the Q-learning method[4], we proposed two ways of implementing the reward function as an incentive.
1. Positive reward will be given if the cart experiences speed to the right without falling.
2. Rewards will be given depending on the relative distance between the cart and the destination, if around the destination, positive, if away from the destination, negative, and increase with distance.
For both cases, the same amount of negative reward(penalty) will be given if the cart hits the edge of the pole and falls below 75 degrees to the ground. The same amount of positive awards will also be given if staying near the destination.

To reach the goal, we initially set rewards according to the absolute position of the cart represented by the pixel on the screen. The further the cart moves right, the higher reward it will gain.

3.3 Problem in Reward function Implementation

At the start of setting up the reward function, we face a dilemma on balancing the reward return.If the reward given for maintaining balance is too large, the cart will only move slightly to the right and maintain balance for longer alive reward without approaching the destination; If the weight on the reward given for the absolute position is too large(way larger than the negative reward given for falling), the cart will move to the right with any cost and give up maintaining balance.

The solution to this problem is that we modified the positive reward to be given for the velocity to right instead of giving reward on the current absolute position, also letting the pole falling over will receive a relatively large negative reward and end the game instantly.The velocity Reward is an incentive for the agent to move right and reach the destination in terms of learning.

3.4 Optimize for the third task: oscillation around destination

After some episode of training(around 1000), the cart straightly passes the destination and hits the right bound instead of oscillating around or sticking on the destination line without changing its strategy. The reason for that is the bin size was too large, the destination is defined within the bin so the bin has to be small enough that the destination area range is close to the bin size. After increasing the bin size to 6, we can see a reasonably long-standing pole cart oscillates more than 15 times around the destination area after 2000 episodes of training.

3.5 Q-learning algorithm:

The state of the algorithm is updated by the Bellman Equation

$$Q(S_t, A_t) = (1 - \alpha)\, Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a\, Q(S_{t+1}, a))$$

Alpha is the learning rate, lambda is the discount on the function return, could be set to 1.

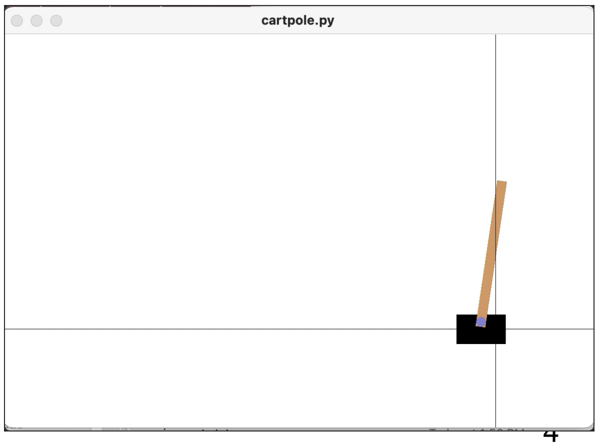
Apply the epsilon-greedy algorithm:

```
WHILE NOT FINISHED:
    On a probability of epsilon:
        choose random action
    Else:
        Choose action that maximizes the Qvalue
    Update Q-value using Bellman's Equation
```

4. Result

4.1 Training result

The result of Q-learning is that the cart will move right starting at the initial point to the destination, then oscillate around it at an episode of approximately 1000.



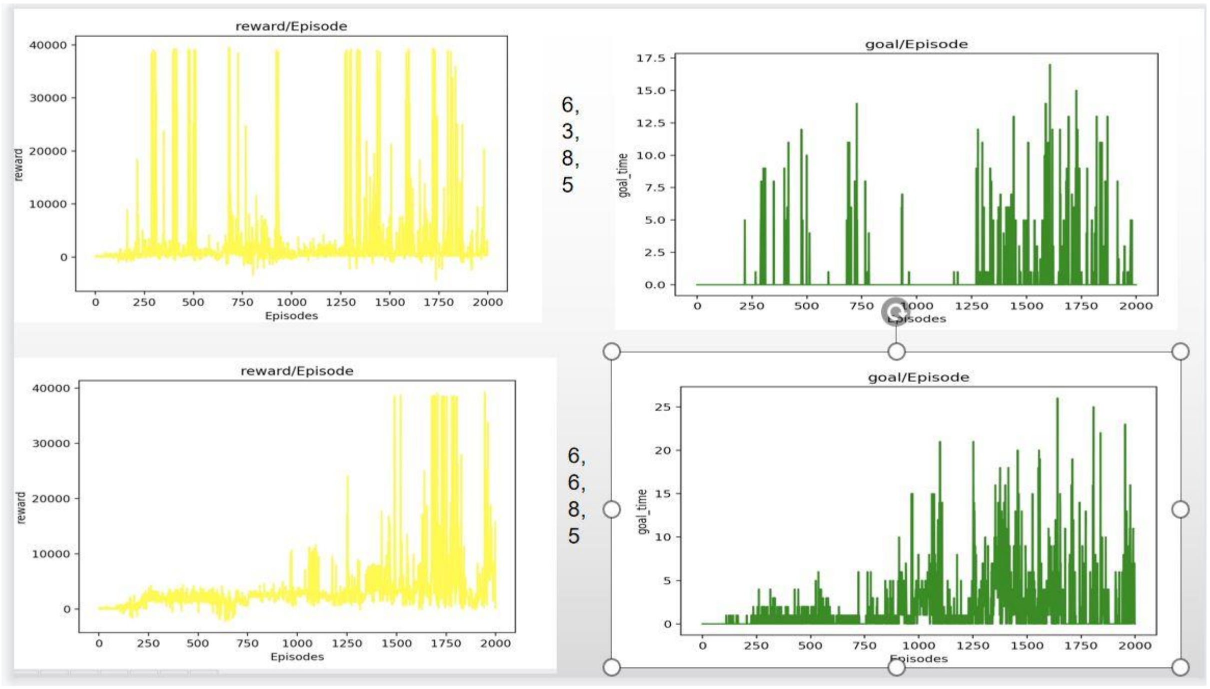
4.2 Final Parameter setting

Epsilon is the probability of taking random action despite action policy which decays by episodes as optimal action is found and no random action is needed.

For the decay of the learning rate, the case is more complex. Linear decay was tested but turned out to be performance bad as the learning rate may decrease unexpectedly before or after the optimal strategy is discovered. Converging too early would slow down the learning procedure, while too late may skip the optimal point. After the linear decay, the condition decay is tried which yields that the learning rate decreases if the condition is met that the cart reaches the destination once and can maintain balance in considerable time. However, the problem with that method is that the condition is ambiguous. More precisely, compared to the original rule where the cart needs to minimize the angle of the pole, the goal of the variant cart pole is compound - the cart needs to be as close to the goal as possible while maintaining the balance of the pole. That produces difficulty to catch the proper time of the decay. In conclusion, log decay is applied here, and the decay happens when a strategy is discovered most of the time in statistics and experiments. Specifically, the learning rate will drop to 0.1 when the average score of an episode is around 100.

Epsilon decays with time:0.01<Epsilon<=1-(log10)(time/4+1)/25
Learning rate (Alpha) decay:0.1<Alpha<=1-(log10)(time/3+1)/25

4.3 Final Bin Number setting(Parameter tuning)



The yellow graph denotes the reward/episode, the green graph denotes the number of passes through the destination/Episodes. The learning curve becomes stochastic when the binning is unbalanced. When the binning partition for positions and velocity is the same [6,6], we can spot a regular pattern of the learning curve, which is from low to high corresponding to an increase in episodes.

4.4 Evaluation of two approaches

Speed reward Approach: Setting the penalty for clashing the edge of the screen to a small level makes the cart move to the right in a rush. (see attachment video1) The reason would be the cost of going off the edge overweighs the cost of failing due to losing balance.

Positional reward Approach: The cart finally evaluates to a state to swing across the destination area but doesn't stay here for a noticeable amount of time.

For both approaches: As the number of bins increases, the interval between each interval decreases. After about 2000 episodes of training, we could see the cart oscillate near the destination for a noticeable amount of time. (see attachment video2).

**The bin partition could not be increased by more than 10 since when the bin interval becomes too small, the destination critical interval (position+-0.5 for example), would be partitioned into multiple bins that causes confusion for the agent(cart-pole).
Changing the binning partition would affect the learning curve of the agent, as evaluated in section 7.2. Under the new environment, Q-learning without binning takes a large learning episode number to train (over 8000 episodes) and doesn't reach an optimum result, whereas Q-learning with binning takes 2000 episodes on average to achieve an acceptable result.

5. Discussion on the approaches

Though we see a large number of oscillations around the destination, the cart doesn't always stay around the destination for a long time but deviates far and heads back. This indicates that the state space doesn't converge. One of the possible reasons is that though binning solved the continuous-to-discrete state-space transformation beautifully, it has a limit of 10 bins for partitioning the input attributes. The interval of the bins at the limit level is still larger than the destination hit critical interval, which means not every step taken by the agent in the bin containing the destination would receive a reward of reaching the destination.
The second possible reason for non-convergence is that it is hard to represent a multi-stage strategy using a single q-table.
At each stage, the optimum strategy should change, like when the cart passes the destination it should head back immediately. However, in the early stage of training, the reward for moving right stimulates the cart to go right even if it passes the destination for the first time. We can see that for Q-learning, it's hard to switch to its optimum strategy.

Chen Sun is responsible for hyperparameter tuning and optimizing, exploring the relationship between bin number and performance, tuning the reward equation, writing the plt function, training most of the graphs on a laptop, and outputting them. Sun Chen is also responsible for the code comment and readme.md. and collaborates with team members to write reports.

Shouyue Hu contributed to registering the training environment, explaining the variables in env, designing reward value, and hyperparameter initialization as well as learning-rate decay.

Donghao Wang contributed to the training for the simplified and binning Q-learning model, parameter optimizing and the first draft of the report.

Through this project, we explored the possibility of applying Q-learning in solving multi-tasking problems. Although the Reinforcement learning agent can learn to perform very complex tasks, it seems to have poor generalization ability to different tasks, compared to supervised deep neural networks which have better generalization ability.

6. Conclusion

In terms of Q-learning, binning improves its performance significantly. As long as the continuous state space could be converted into discrete state-space, Q-learning could be a simple, cheap, and fast approach to solving this problem.

With another component added to the Classic Cartpole problem, the agent's task increases, the reinforcement learning techniques used for the original problem all required extension to adapt to the new environment, such as binning.

A further extension on this project should focus on parameter tuning to explore the possibility of the cart staying longer around the final destination rather than exploiting it. For example, the learning rate could be decaying or increasing depending on the outcome of the episode before.

7. References

[1]AG Barto, RS Sutton and CW Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem", IEEE Transactions on Systems, Man, and Cybernetics, 1983.

[2]Using q learning for OpenAI's cartpole-v1,*https://medium.com/swlh/using-q-learning-for-openais-cartpole-v1-4a216ef237df*

[3]Original CartPole-v1 environment for gym, *https://gym.openai.com/envs/CartPole-v1/*

[4]Classic cart-pole problem via Q-learning,
https://github.com/icoxfog417/cartpole-q-learning