Project: Investigating Reinforcement Learning

Student name: Van Tri Nguyen
Student ID: 220622865

Taxi problem

Introduction to the Taxi problem

The Taxi problem was introduced in [3 - Dietterich2000] to illustrate some issues in hierarchical reinforcement learning. In the map of size 5 by 5, there are four cells labeled by letters R,G,B,Y, representing the possible pick-up and drop-off locations for the passenger. The AI’s job is controlling a taxicab. It receives +20 points for each successful dropoff. The penalty for mistakes such as driving into a wall/illegal pickup and drop-off actions is 10 points. There is also a variant of this problem where the AI loses 1 point for every timestep it takes, representing the idea that it should take the shortest path available. We want our AI to maximize the net score.

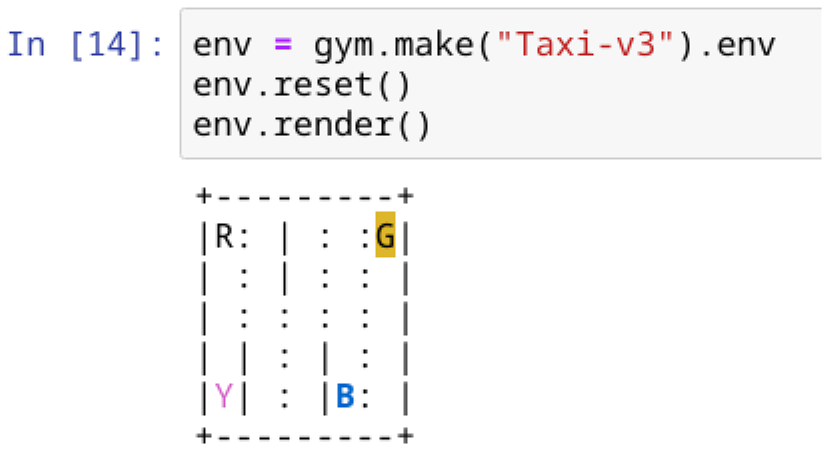
Mathematical description of the reinforcement learning problem

The problem is encoded as a 2D map of size 5 by 5. In this map there are four special cells named R,G,B,Y which will be possible pick-up/drop-off cells, but this information should not be directly given to our AI but must be inferred during the training. Even though, when we print the map, this information is shown to us for debugging purposes.

At each timestep our taxi will be at exactly one cell. For each episode (which means each simulation run), the taxi should, optimally:

through a series of **driving actions** go to the pick-up cell
do the **picking up action**
through another series of **driving actions** go to the drop-off cell
do the **dropping off action**

So in total, there are six possible actions the AI could choose from at any time step:
go down, go up, go right, go left, picking up, dropping off. There would be 500 states of the simulation in total, calculated as a combination of possible places the taxi can be (5x5), the place it needs to be (4), and whether the passenger is on it or not (1+4). Each state is then indexed from 0 and served.



```
In [14]: env = gym.make("Taxi-v3").env
         env.reset()
         env.render()
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

The reward mentioned in the introduction of the taxi problem is encoded as a dictionary of lists, as the form of

```
{action : [(probability, nextstate, reward, done)]}
```

With the meaning as followed:

- **action** is the action could be taken (e.g. go up).
- **probability** is the probability that the action will successfully transform the current state to **nextstate** (in the taxicab problem this is always 1).
- **nextstate** is the state that occurs if **action** is done.
- **done** is a boolean value indicating whether the episode is over.

The taxi problem is phrased in terms of actions and states, totally suitable for reinforcement learning problems.

Q-learning algorithm

The reinforcement algorithm for the taxi problem is Q-learning. The idea of the algorithm is to establish a causal relationship between pairs of action and state. This casual relationship in this case is over the expected reward - the total score the AI thinks it would have at the end of the episode if it executed a particular action in the current state. This expected reward is represented as a 2D table Q.

$$Q(\text{state}, \text{action}) := (1-\alpha)\ Q(\text{state},\text{action}) + \alpha\ Q(\text{reward} + \gamma\ \max_{\alpha} Q(\text{next state, all actions}))$$

The assumption is that this self-referential assignment will converge over a large enough exploration over state space. The meaning of each variable is as followed

- **α** is the learning rate, ranging from 0 to 1.
- **γ** is the discount factor, ranging from 0 to 1 If γ is 0, the term it multiplies with canceled out and there is no future reward taken into account in the causal inference process.

- **next state** is the state the AI ended up with after the action.
- **$\max_{\alpha}$Q(next state, all actions)** means a single maximum Q score of a pair (next state, a) with a is a variable ranging over all possible actions.

There is also this variant of the formula with the addition of parameter ε stands for exploration factor of the AI. With each move, we will randomize a number and if it's smaller than ε we the AI will not follow the strategy proposed by the Q table but randomly execute an action.

In the taxi problem, the state has the domain in the mentioned 500 states and the action has the domain in 6 available actions. We will train the model through 100000 episodes and will not terminate early under any condition.

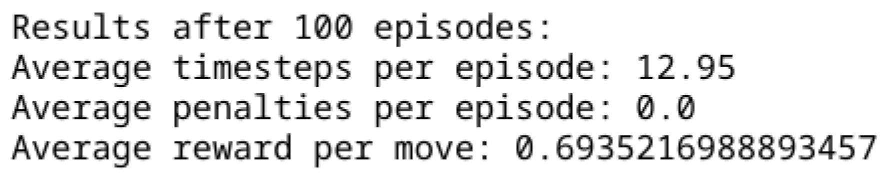
Compare the quality of solution of a random policy versus the optimal policy obtained by Q-learning

There are three key aspects to consider. The reason for each of these can be found at [1].

- Average number of penalties per episode.
- Average number of timesteps per trip.
- Average reward per move.

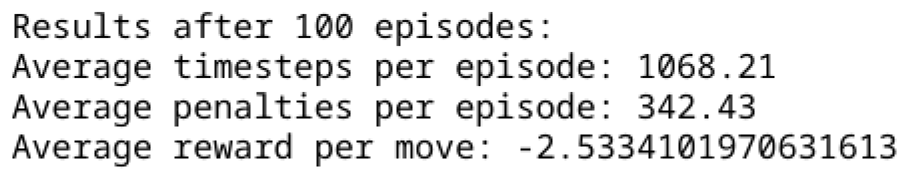
The result is that the Q-learning algorithm produces a significantly better solution than that of random policy.

Q table solution:



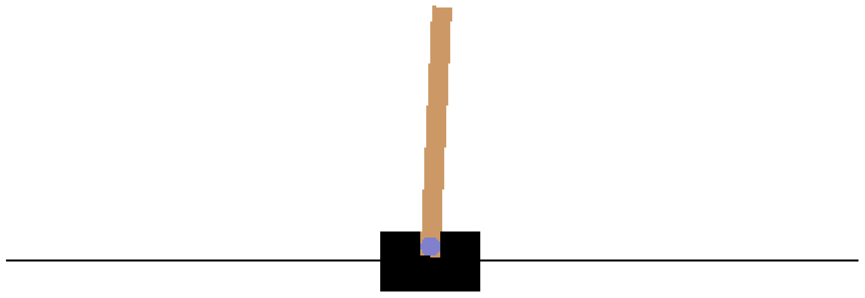
```
Results after 100 episodes:
Average timesteps per episode: 12.95
Average penalties per episode: 0.0
Average reward per move: 0.6935216988893457
```

Random policy solution:



```
Results after 100 episodes:
Average timesteps per episode: 1068.21
Average penalties per episode: 342.43
Average reward per move: -2.5334101970631613
```

CartPole problem



CartPole description

The action of the AI is applying a force of +1 or -1 to the cart. The objective is to keep the pole from falling over. For each timestep where the pole remains standing, the AI is rewarded +1 point. The episode ends if the pole falls over (more than 15 degrees from vertical) or the cart moves too far away (2.4 units from the center)

CartPole differences with the Taxi problem

Instead of 6 as in the Taxi problem, in the CartPole problem there are only possible 2 actions to take:

- push the cart to the left
- push the cart to the right

In this problem, the state space is continuous as it represents a tuple of the cart position, velocity, the pole angle and velocity at tip.
This poses some changes needed in the Q learning algorithm. The implementation of the Q table for the Taxi problem is indexed by 2 discrete keys: state and action, which is totally fine. We cannot do the same for this problem, indexing using a continuous variable is not valid and even if it is (e.g. with python dictionary), it's not memory efficient and updating is almost impossible because the likelihood of hitting the same continuous value of state twice is unlikely. The solution could be either clipping the state value to turn it discrete or Deep Q Learning - where instead of a Q table we have a neural network approximating said table. In this project, to keep the original spirit of Q learning with a table, I will try the first approach: clipping the value. The state space of the cart pole problem is therefore also much larger than that of the taxi.

Comparing the performance of Q learning in both problems

The simulation takes significantly longer for the CartPole problem so training using Q learning takes longer. It can be seen that clipping the value to turn it discrete can have a major effect on the performance of the Q learning algorithm. Also, the large state space requires more than 100000 for the table to converge. In the cart pole problem, Q learning did not converge fast over 1000 episodes training like in the taxi problem, as can be seen in the following graph of the obtained reward.

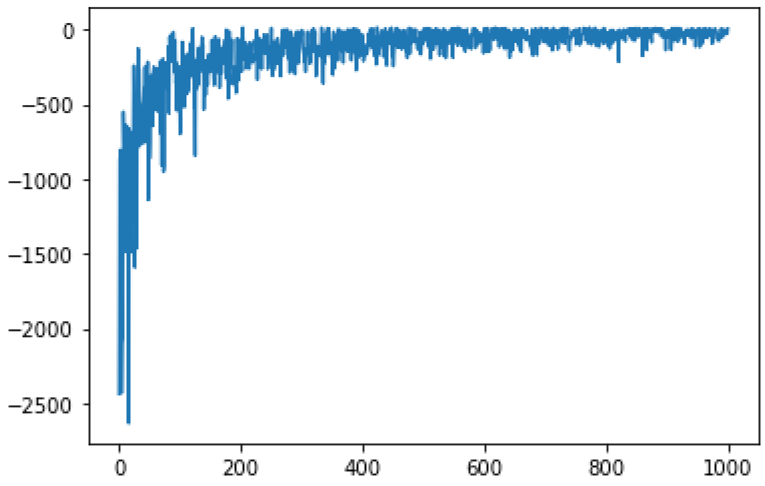


Fig. Reward obtained of 1000 episodes (Taxi)

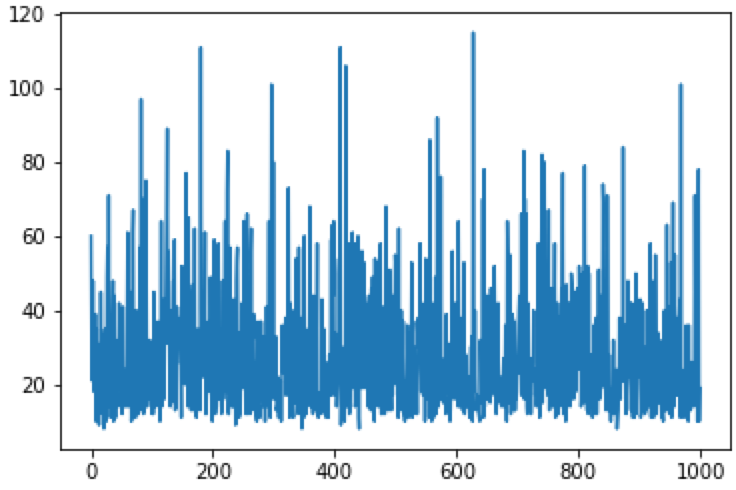


Fig. Reward obtained of 1000 episodes (CartPole)

The length of each episode has also not improved fast. As can be seen below.

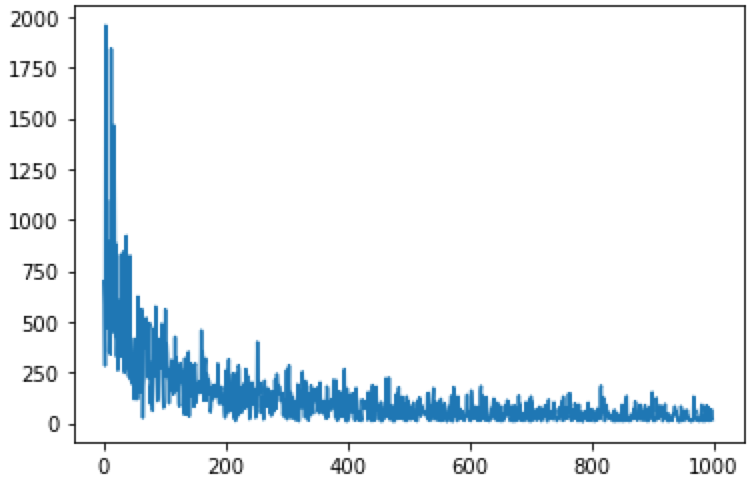


Fig. Length of episode (Taxi),
the length shinked as the taxi learns to take the shortest path

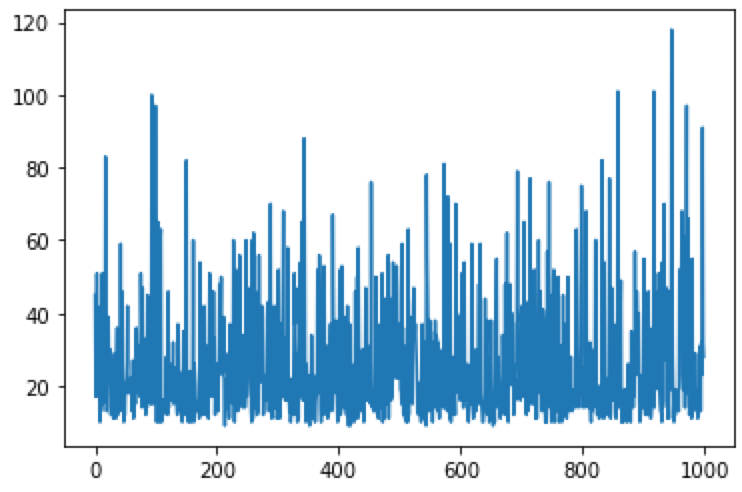


Fig. Length of episode (Cart Pole)

References

- Kansai, S., & Martin, B. (n.d.). Reinforcement Q-Learning from Scratch in Python with OpenAI Gym. Learn DataSci. Retrieved May 29, 2021, from https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
- Gym: A toolkit for developing and comparing reinforcement learning algorithms. (n.d.). Open AI Gym. Retrieved May 29, 2021, from https://gym.openai.com/envs/Taxi-v2/
- Dietterich, T. G. (2000). Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. Journal of Artificial Intelligence Research, 13, 227–303. https://doi.org/10.1613/jair.639
- OpenAI Gym: the CartPole-v0 environment. (n.d.). Open AI Gym. Retrieved May 29, 2021, from https://gym.openai.com/envs/CartPole-v0/