DPRL Assignment 4: Q-learning Method for Solving Chain Problem Group 65

Tian Xia
2703936
Vrije University Amsterdam
t.xia@student.vu.nl

Haohui Zhang
2722930
Vrije University Amsterdam
h17.zhang@student.vu.nl

1 Introduction

The target we intend to accomplish is to use reinforcement learning algorithm to find the optimal Q-values of the chain system.

Specifically, this problem requires two different methods of calculating the optimal Q value. First, we use tabular Q-learning and ϵ-greedy method to provide the optimal Q-values under different learning rate and the epsilon value. Next, we use function approximation algorithm to provide the optimal Q-values and discuss about the influence of hyper-parameters on the convergence results.

2 Explanation of Environment and Building Tabular Q-learning Method

Consider a chain system that is made up of 5 discrete states and 2 discrete actions, where state 1 get a reward on 0.2 and 1 at state 5, the other side of chain. The transition graph is shown by **Figure 1**, and the discount factor is $\gamma = 0.9$.

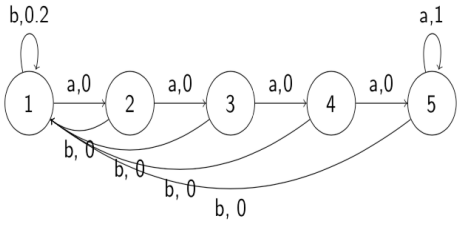

Figure 1: Figure 1. The transition graph of each state and reward of each action under two different actions

In the case of part 1(a), the chain system has a state space size of 5 and an action space size of 2. Here we can give out the transition matrix and the reward matrix. Since the transition of state is trivial, we directly combine the action and state transition:

$$P_a = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad P_b = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$r = \begin{bmatrix} 0.2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad a = \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 5 & 1 \end{bmatrix}$$

where where r is the reward on each state under two actions , and a denotes the result of state transition under two actions.

2.1 Greedy Case

Before we go to the ϵ-greedy condition, let's do a greedy tabular Q-learning method to give a solution, where we set a fixed learning rate $\alpha = 0.9$.

Greedy method means that "always choose the action with the best current Q-value", therefore we can write down the Q-table update rule:

$$Q(x_{t+1}, a_{t+1}) \leftarrow Q(x_t, a_t) + \alpha_t \left[r_t + \gamma \max_{a' \in \mathcal{A}} Q_t(x_{t+1}, a') - Q_t(x_t, a_t) \right]$$

Do the iteration until the value of Q-table does not change anymore, meaning: L2 norm $(Q(x_{t+1}, a_{t+1}) - Q(x_t, a_t))^2 \leq e$, where $e = 1e^{-20}$

```
q_value*:
[[ 6.561   6.1049]
 [ 7.29    5.9049]
 [ 8.1     5.9049]
 [ 9.      5.9049]
 [10.      5.9049]]
```

We observe that it is converged after 257 times iteration, when $\alpha = 0.9$, $\epsilon = 0$.

2.2 ϵ-greedy Case

Under ϵ-greedy condition, we have a probability of $1-\epsilon$ to do an action following $\max_a Q(x,a)$, while a probability of ϵ to randomly choose an action and get the $Q(x,a_r)$. Therefore, the Q-table update rule becomes to:

$$Q(x_{t+1},a_{t+1}) = \begin{cases} Q(x_t,a_t) + \alpha_t\left[r_t + \gamma \max_{a'\in\mathcal{A}} Q_t(x_{t+1},a') - Q_t(x_t,a_t)\right], & \text{if } p \geq \epsilon \\ Q(x_t,a_t) + \alpha_t\left[r_t + \gamma Q_t(x_{t+1},a_r) - Q_t(x_t,a_t)\right], & \text{otherwise} \end{cases}$$

where a_r stands for the random action.

Now, at this time, we no longer fix the learning rate α and the greedy index ϵ. We set both of them as variables to see their influence on the convergence speed. In this case, we range the parameters in:

$$\alpha \in [0.8, 1], \quad \epsilon \in [0, 0.2]$$

$$\Delta \leq 1e^{-5} \quad as\ converged$$

Since the randomness of ϵ, for each Q-table, we do 10 times of experiment and take the average iteration number as the real required converging steps. **Figure 2** and **Figure 3** are the line graph that show the rate of convergence under different learning rates and epsilons.

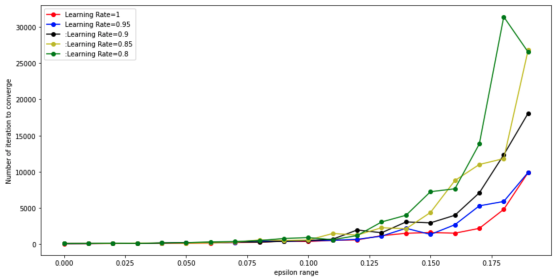

Figure 2: Figure 2. Number of iteration to converge on 5 states chain

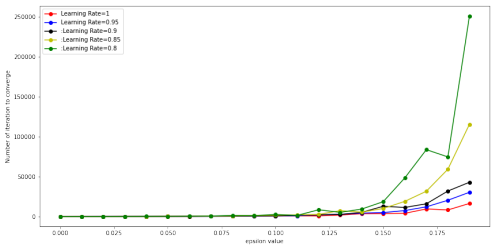

Figure 3: Figure 3. Number of iteration to converge om 10 states chain

2.3 Discussion of learning rate α and ϵ

According to Figure 2 and Figure 3, we observe that as epsilon goes high, the needed iteration time raises exponential, which means the more random actions, the much more effort is required to reach convergence in this situation.

Similar but different, as the learning rate decreases, the number of steps required to reach convergence does not increase significantly. We judge that this is because there is only one global optimal solution in this case, no local optimal point, so even if the step length of the gradient descent becomes shorter, we can still reach the optimal point smoothly.

Next, comparing 10 states case and 5 states case, there is no significant difference on the trend of lines. But with the states space enlarging, the influence of the change of ϵ on the number of iterations required for convergence is far greater than the influence of the change of the learning rate. Also, as the total number of states increases, this influence is further increased.

2.4 10 States Result

In question part 1(b), the size of the chain states increases from 5 to 10 while keeping the same rewards at both end of the chain. #State_space $= 10$ and #Action_space $= 2$

$$r = \begin{bmatrix} 0.2 & 0 \\ 0 & 0 \\ \cdots & \cdots \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad a = \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ \cdots & \cdots \\ 8 & 1 \\ 9 & 1 \\ 9 & 1 \end{bmatrix}$$

Now we re-set up the environment, the space is widened but the algorithm remains same, where r is the reward on each state under two actions , and a denotes state transition result under two actions. Both r and a is of dimension 10 x 2.

```
print(Q_learning())
```

```
(337, array([[ 3.87420489,  3.6867844 ],
       [ 4.3046721 ,  3.4867844 ],
       [ 4.782969  ,  3.4867844 ],
       [ 5.31441   ,  3.4867844 ],
       [ 5.9049    ,  3.4867844 ],
       [ 6.561     ,  3.4867844 ],
       [ 7.29      ,  3.4867844 ],
       [ 8.1       ,  3.4867844 ],
       [ 9.        ,  3.4867844 ],
       [10.        ,  3.4867844 ]]))
```

By using the same Q-table iteration method, we get the optimal Q-value after 337 times iteration.

3 Value Function Approximation Methods

In this part, we try to solve the chain problem using function approximators for $\gamma = 0.9$ and $state = 10$.

3.1 Construct Environment

In this section, we construct the environment following the request. In the environment class, we have two method, respectfully environment reset and move-to-next-step.

3.2 Deep Q learning with NN network

In order to deal with the continuous state, we represent the value function with the function approximators and parameters θ, the approximate function is as follow:

$$Q(x, a; \theta) \approx Q(x, a)$$

The parameters θ are updated such that:

$$\theta := \theta + \alpha \frac{d}{d\theta}(Q(x, a; \theta) - \mathcal{B}K$$

, where $\mathcal{B}$ is the Bellman operator mapping any function $K : \mathcal{X}$ x $A \rightarrow R$ and into another function $\mathcal{X}$ x $A \rightarrow R$ and is defined as follow:

$$\mathcal{B}Q(x, a) = \sum_{x' \in \mathcal{S}} T*(R(x, a, x') + \gamma \max_{a' \in \mathcal{A}} Q(x', a'; \theta_k) \tag{1}$$

In our MDP, the transition is deterministic, so that the $T(x, a, x') = 1$.
In this model, we design two full-connected layer to learn the feature. We use the keras package provided by tensorflow to construct our model. The first full-connected layer, we use the sigmod function as our activation function, as well as use the linear function for the second full-coneected layer, these two activation functions are chosen empirically. The input of model is a 1x10 one-hot vector which represents the state. The outputs of the model are two value, which represent the Qvalue for the input state of two actions. Besides, we use the cross entropy as the loss function and the Adam as the optimal function. The detailed information of the model is shown in the Figure 4. Pseuducode is attached in Appendix part.

3.3 Experiments

In our model, we have six hyper-parameters, separately ϵ, γ, learning_rate, batch_size, decay_factor, episode =1000 and step=1000 in each episode. Except for the γ is a fixed number, the other five are all related to the convergence. We will discuss these five one by one by providing the comparison result and analysis.

3.3.1 batch_size

First, we set the batch_size is 1, which means our model will update after each step. The result is terrible. It is truly difficult to converge to the right action. These is because we start our MDP in the state 0, and in almost all episode and step, for all 10 state the action 2 will have a better q value. This is because except the state 0 and state 9 have a reward, all the other state rewards is 0, this leads to it is very difficult for us to get to the state 9 by randomness and select the action 1, even if we suppose select all the action randomly, it will be 0.5^10 probability to get to the state 9 to select the action 1. Thus, in the normal case, the q_learning table will eventually converge to the action 1. The simulation is shown in Figure 5.

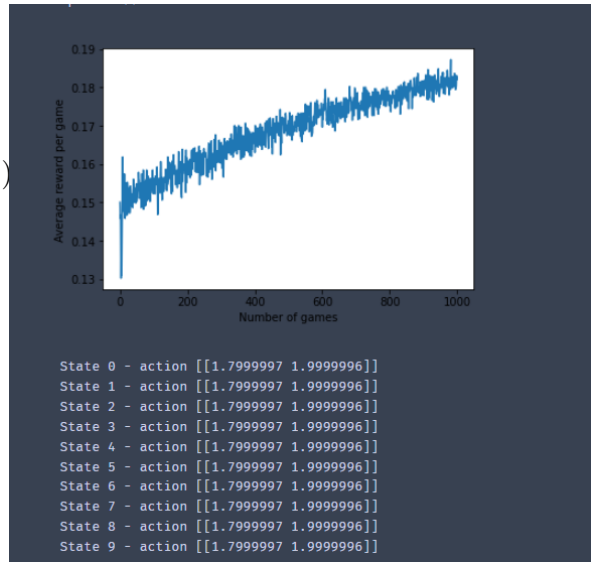

Figure 5: The simulation which batch_size=1, ϵ= 0.5, decay_factor=0.999, learning_rate=1, episode=1000, step=1000

Also, because with deep learning, the update usually uses a mini-batch (e.g., 32 elements). Thus, we set the batch_size = 32. Which means, after 32 steps, we will update our model.

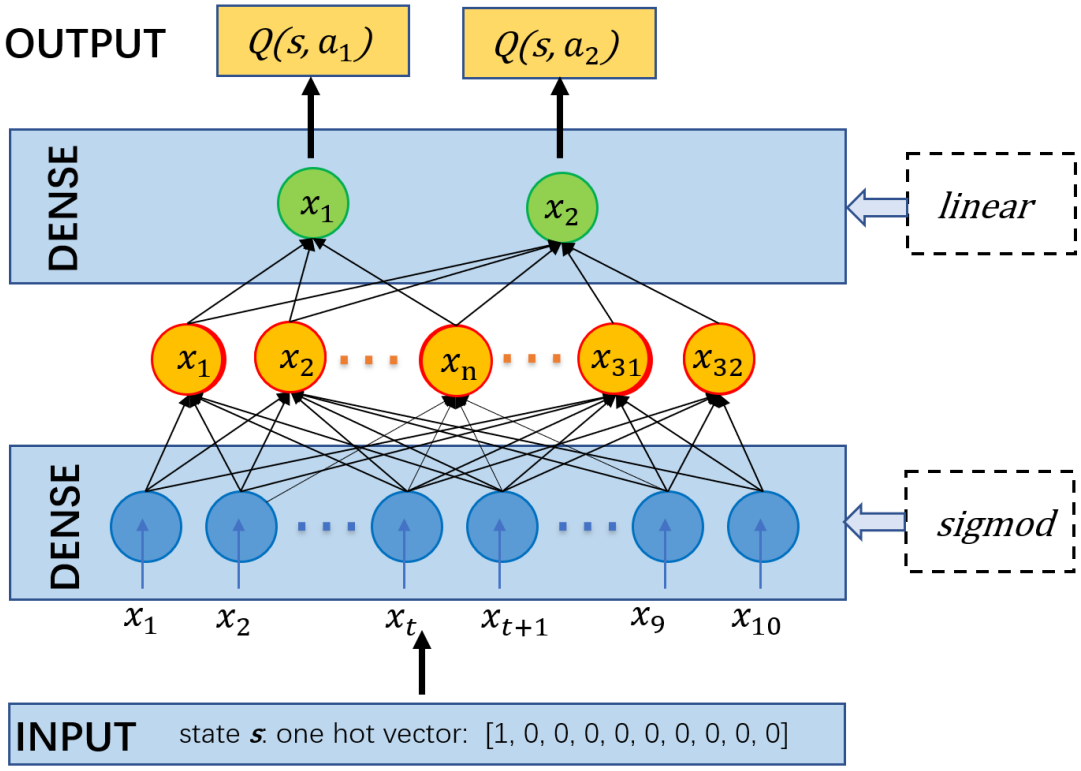

Figure 4: NN model

3.3.2 ϵ

The ϵ is the most important hyper-parameter. We do a lot of comparison experiment to elaborate its influence to our system. The two figures in Figure 6 and 7. Through the comparison, we can figure out the when the ϵ equals to 0.5 it is successfully converge to the right direction which is action 0. However, if we adjust the ϵ to 0.9, it can still converge but do not converge to the right direction except for the state 9. The reason is because the random probability is to high and according to the state matrix, more random probability means bigger probability to return to state 0. And as a result, for instance, in state 8, even if it knows move to state 9 is a better action, but the there is still a high probability to go back to state 0. That's the reason why after the ϵ decrease through the steps, it will at last converge to the action 1. What's more, if the ϵ is to small like 0.1, there is still a high probability all states converge to action 1 except for the state 9.

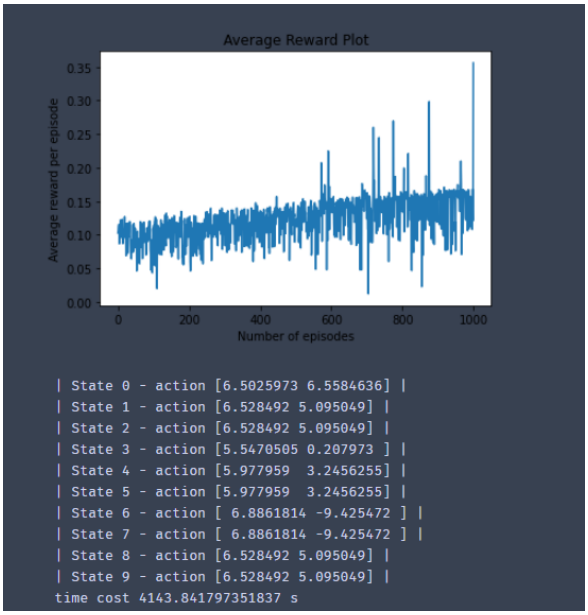

Figure 6: The simulation which batch_size=1, ϵ= 0.5, decay_factor=0.999, learning_rate=1, episode=1000, step=1000

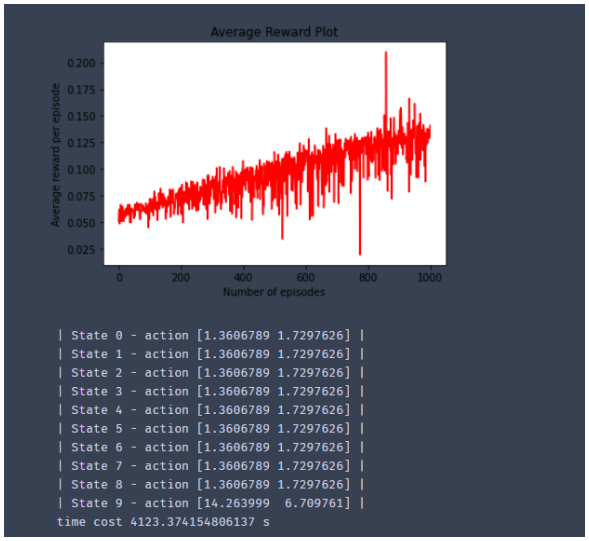

Figure 7: The simulation which batch_size=1, ϵ= 0.9, decay_factor=0.999, learning_rate=1, episode=1000, step=1000

3.3.3 decay_factor

Comparing the figure in Figure 8 and figure in the ϵ part, we can get, decrease the decay factor will let the our system easier to converge but may not in the right direction. You can see in Figure 6, the system eventually converge to the action 0, however, after decreasing the decay factor, the system at last converge to the action 1.

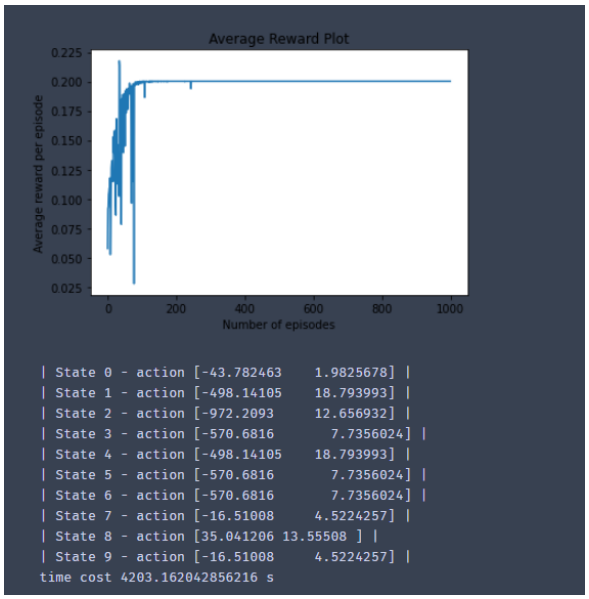

Figure 8: The simulation which batch_size=1, ϵ= 0.5, decay_factor=0.95, learning_rate=1, episode=1000, step=1000

3.3.4 The rest hyper-parameters

Because of the training time, we do not have enough time to modify the rest hyper-parameter and train our model for a lot of times. In the first part, we have done a sufficient analysis of how the learning rate affects the convergence. I convince using the function approximators will get the same conclusion. Besides, increasing the step number in each episode will also definitely increase the probability of convergence.

The conclusion for our experiment is that when the three hyper-parameters batch_size = 32, ϵ=0.5, decay_factor=0.999, our system will have the best performance.

4 Conclusion

In this assignment, we used tabular Q-learning algorithm and deep Q-learning algorithm to solve the chain problem.

Considering that this is a simple state transition situation within low-dimension, and there is only one global optimal solution. We can easily use Q-table method to deduce the convergence procedure. Though the solution seems trivial, we have learned: when faced with adjustable parameters, such as learning rate and greedy index, we still need to choose them carefully. A learning rate that is too large may jump back and forth around the optimal point but never reach it. A learning rate that is too low may greatly affect the convergence speed of the algorithm. As for the ϵ-greedy method, if there is a long state transition without positive rewards, then we should pick a relatively small ϵ, which will accelerating convergence process.

What's more, in the case that the Q-table is easy to calculate, using value approximation methods to predict the Q-value is not necessarily a good choice, since most of the time will be spent in the early stages of model training, especially when using deep learning models.

5 Appendix

Algorithm 1 NN Q Learning

$D = < s, a, r, s' >$

Input: learning_rate, episode, batch_size, γ, ϵ, decay_factor:β

Output: Q_value table

Initialization: MDP environment

procedure MODEL $NN(s)$
  for $i = 0, 1, \ldots,$episode **do**
    $\epsilon \leftarrow \epsilon * \beta$
    $size \leftarrow 0$;
    while $step < 1000$ **do**
      $Q\_value(s, a) \leftarrow NN(s.to\_vector())$
      if $random < \epsilon$ **then**
        Choose random a
      else
        $a = \arg\min_a(Q\_value(s, a))$
      end if
      $s', r \leftarrow MDP$
      $Q\_value(s', a') \leftarrow NN(s'.to\_vector())$
      $\mathcal{B}Q(s, a) \leftarrow r + \gamma * max(Q\_value(s', a'))$
      $Q(s, a) \leftarrow \mathcal{B}Q(s, a)$
      if $size = batchsize$ **then**
        Fit model $NN(s, Q(s, a))$
      end if
    end while
  end for
end procedure