Dynamic Programming & Reinforcement Learning

Assignment 4

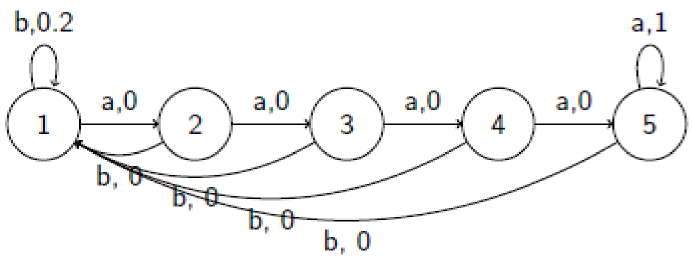
Qiuyang Fu (2721961)

Introduction

This report describes my solutions, including ε-greedy Q-learning with tabular and function approximator, for solving the chain environment. Solutions regarding the several questions mentioned in this assignment will be placed in the following sections.

1. Problem

The chain environment made up of several discrete states and 2 discrete actions, where you get a reward on 0.2 on one end of the chain and 1 at the other end. Below shows a chain environment with 5 states.



If the number of states is just 5, we can solve the problem even by hand. It is obvious to choose action *a* at state *5*, and since it is endless. According to the Bellman equation:

$$Q^{\pi}(s,a) = \mathrm{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right]$$

Since we are in deterministic environment here, we have,

$$Q^{\pi}(5,a) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

Since we choose the discount factor γ to be 0.9, the final Q-value of state *5* and action *a* is 10. With the similar method, the final Q-value of state *1* and action *b* is 2. Then we can construct the whole Q-value table as below:

	a	b
1	6.561	2

2	7.29	1.8
3	8.1	1.62
4	9	1.458
5	10	1.3122

Thus, the resulting policy is to choose action *a* whatever at any state. And since Q(1,b)<Q(1,a), the optimal Q-value of Q(1,b) must be a discount of Q(1,a), i.e., if one selects action *b* at state *1*, he must select action *a* in the next time. And the final Q-value table is

	a	b
1	6.561	6.1049
2	7.29	5.9049
3	8.1	5.9049
4	9	5.9049
5	10	5.9049

However, as the chain becomes longer and longer, the states that is closer to state *1* will receive less and less reward from action *a*. In section 2 and 3, we will demonstrate this with tabular Q-learning and deep Q-learning.

2. tabular Q-learning with ε-greedy

The algorithm's pseudocode is as below:

```
Initialize Q(x,a) arbitrarily
for each episode do
    Initialize x
    for each step in episode do
        Choose a given x using policy derived from Q (e.g., ε greedy)
        Take action a, observe r, x'
        Q(x, a) ← Q(x, a) + α[r + γ max_a' Q(x', a') − Q(x, a)]
return Q(·, ·)
```

My implementation of chain environment is in the file `**Toy_env.py**`, and my implementation of tabular Q learning is in the file `**run_toy_env_simple.py**` (from line 18 to line 35).

2.1 result of 5-state chain

(1) With discount factor $\gamma = 0.9$, learning rate $\alpha = 0.9$, and $\varepsilon = 0.05$ I run Q-learning with 10 episodes, in each episodes with 1000 steps. To ensure the result is fair, I try 10 times and all of them print nearly the same result as below (*from now on, I use 0 to represent action* a, *1 for* b):

Qtable = [[6.561   6.1049],

[7.29   5.9049],

[8.1   5.9049],

[9.   5.9049],

[10.   5.9049]]

Policy = [0 0 0 0 0]

which is consistent with the theoretical result.

(2) Enlarge ε

With discount factor $\gamma = 0.9$, learning rate $\alpha = 0.9$, and $\varepsilon = 0.2$ I run Q-learning with 10 episodes, in each episodes with 1000 steps. To ensure the result is fair, I try 10 times and all of them print nearly the same result with above. **It suggests that allowing a more random exploration of action space will not affect the result of Q-learning.**

(3) Smaller α

With discount factor $\gamma = 0.9$, learning rate $\alpha = 0.1$, and $\varepsilon = 0.05$ I run Q-learning with 10 episodes, in each episodes with 1000 steps. To ensure the result is fair, I try 10 times and all of them print nearly the same result with above. I think that it is because the length of episode is long enough for the Q-learning to convergence. Therefore, I re-run with 100 steps each episode. This time, Q-learning will give some results as below:

Qtable = [[0.17215154 1.92059358]

[0.73712855 0.09183433]

[2.56379402 0.   ]

[6.1841707 0.   ]

[9.9778079 0.45037922]]

Policy = [1 0 0 0 0]

It suggests that a smaller learning rate will slow down the convergence rate of Q-learning.

3. Deep Q learning

Deep Q learning, instead of using Q table, uses a neural network to predict the Q value based on the state and action. Since the chain environment is simple, I use a network with a single hidden layers with 10 neurons and ReLU activator to predict the Q-value (see `**simpleNN.py**`). More detail, given a state as input, the network predicts two values for each action as the Q value.

(1) With discount factor $\gamma = 0.9$, learning rate $\alpha = 0.1$, and $\varepsilon = 0.1$, the result is:

state 0, Q value [0.53668344 0.07955375], best action 0

state 1, Q value [0.53668344 0.07955375], best action 0

state 2, Q value [0.57007617 0.07753744], best action 0

state 3, Q value [1.1075023 0.04508685], best action 0

state 4, Q value [1.6449286 0.01263627], best action 0

Though the Q values are far from what we expect, the policy is right.

(2) Smaller learning rate will give unexpected policy, larger ε also has little effect on the final policy.

4. Longer chain

In this part, we study the longer chain with 10 states. The theoretical Q value of state *1* and action *a* is $10*0.9^9$=3.8742>2, so the optimal policy remains to choose action *a* at any state.

With discount factor $\gamma = 0.9$, learning rate $\alpha = 0.9$, and $\varepsilon = 0.05$, tabular Q learning gives the following result:

Qtable = [[3.87420489 3.68676753]

[4.3046721 3.4867844]

[4.782969 3.48329762]

[5.31441 3.48678437]

[5.9049 3.4867844]

[6.561 3.48678405]

[7.29 3.48678091]

[8.1 3.48329762]

[9.   3.48678091]

[10.   3.4867844]]

Policy = [0 0 0 0 0 0 0 0 0 0 0]

With discount factor $\gamma = 0.9$, learning rate $\alpha = 0.1$, and $\varepsilon = 0.05$, tabular Q learning gives the following result:

Qtable = [[5.10106451e-07 0.00000000e+00]

[1.17356446e-05 0.00000000e+00]

[1.93284319e-04 0.00000000e+00]

[2.34250134e-03 0.00000000e+00]

[2.11347821e-02 0.00000000e+00]

[1.41904950e-01 0.00000000e+00]

[7.01052087e-01 0.00000000e+00]

[2.48965810e+00 0.00000000e+00]

[6.12539095e+00 0.00000000e+00]

[9.99828930e+00 1.50437452e-08]]

Policy = [0 0 0 0 0 0 0 0 0 0 0]

With discount factor $\gamma = 0.9$, learning rate $\alpha = 0.1$, and $\varepsilon = 0.1$, deep Q learning gives the following result:

state 0, Q value [0.74911046 0.03404114], best action 0

state 1, Q value [0.74911046 0.03404114], best action 0

state 2, Q value [0.83649844 0.0373226], best action 0

state 3, Q value [1.3559653   0.05682879], best action 0

state 4, Q value [1.8754319   0.07633498], best action 0

state 5, Q value [2.3948984   0.09584117], best action 0

state 6, Q value [2.9143655   0.11534737], best action 0

state 7, Q value [3.4338322   0.13485356], best action 0

state 8, Q value [3.9532986   0.15435974], best action 0 state 9, Q value [4.4727654   0.17386594], best action 0

5. Conclusion

In this report, I study the chain environment from theory and experiments, including tabular Q-leaning and deep Q-learning.

Remark:

The code requires Numpy and deer package for running.