UDACITY Machine Learning Nanodegree
Capstone Project

Reinforcement Learning to Improve Reaction Dynamics

Lalit A Patel

LLSR@att.net
July 15, 2016

Motivation

Controlled thermonuclear fusion, the best option for source of energy, has not been achieved even after extensive work since 1950. This is because it is difficult to make deuterium nuclei fuse when they are randomly flying in the high-temperature plasma. Nuclear reactions in this plasma are much more difficult than normal chemical reactions. Some scientists [http://www.euro-fusionscipub.org/wp-content/uploads/2014/11/EFDP12021.pdf] now believe that a machine learning approach can be of great help in the study of controlled thermonuclear fusion.

A steam engine has a speed-controlling valve. If the engine gets excess [/ less] fire, the valve's opening becomes narrow [/ wide], so that less [/ more] fuel enters and the fire is decreased [/ increased].

A controlled thermonuclear fusion reactor should have a feedback mechanism, somewhat similar to the steam engine's speed-controlling valve. Depending upon the levels and temperatures of items, this feedback mechanism should change the settings of various parameters affecting the reactions.

This project is an attempt towards a possible design for a feedback mechanism, which can improve and control dynamics of reactions in a controlled thermonuclear fusion reactor.

Resources

The project makes use of reinforcement learning, which is based on concepts of actions, rewards, and Q values. Apart from some basic concepts of and assumptions for reinforcement learning, there are no datasets involved here.

Simplification

Instead of dealing with the complicated problem of controlled thermonuclear fusion, let us deal with a simple and generic problem of chemical and nuclear reactions.

Suppose that we have six types of items: items 1, 2, 3, 4, 5, and 6. It is possible to convert item 1 into item 2 (by using changer 12) or item 4 or item 6, item 2 into item 1 or item 3, item 4 into item 1 or item 5, item 5 into item 4 or item 6. Other conversions are not possible. We want to enhance the production of desired item 6 and avoid the production of waste item 3.

In order to understand this case, we may take an example of hydrogen and oxygen reacting to produce water [2 H2 + O2 => 2 H2O], and calcium and water reacting to produce calcium hydroxide [CaO + H2O => Ca(OH)2]. In this example, hydrogen is item 1, oxygen is changer 12, water is item 2, calcium is changer 23, and calcium hydroxide is item 3.

This case can be treated like a treasure hunt problem. It can be depicted in the form of a house with 6 rooms, 4 bidirectional doors, and 2 unidirectional doors, as shown in figure 1. This depiction can be improved by using a flow diagram, as shown in figure 2.

Figure 1 (Six items depicted as a house with six rooms)

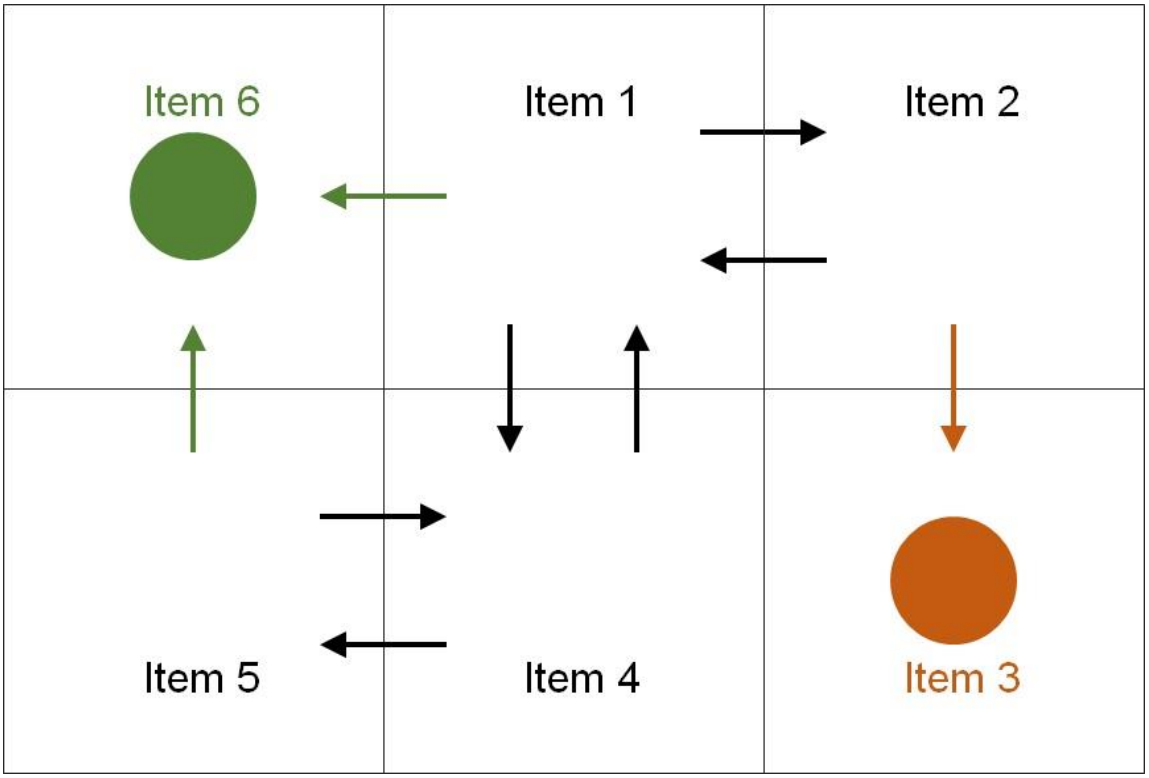
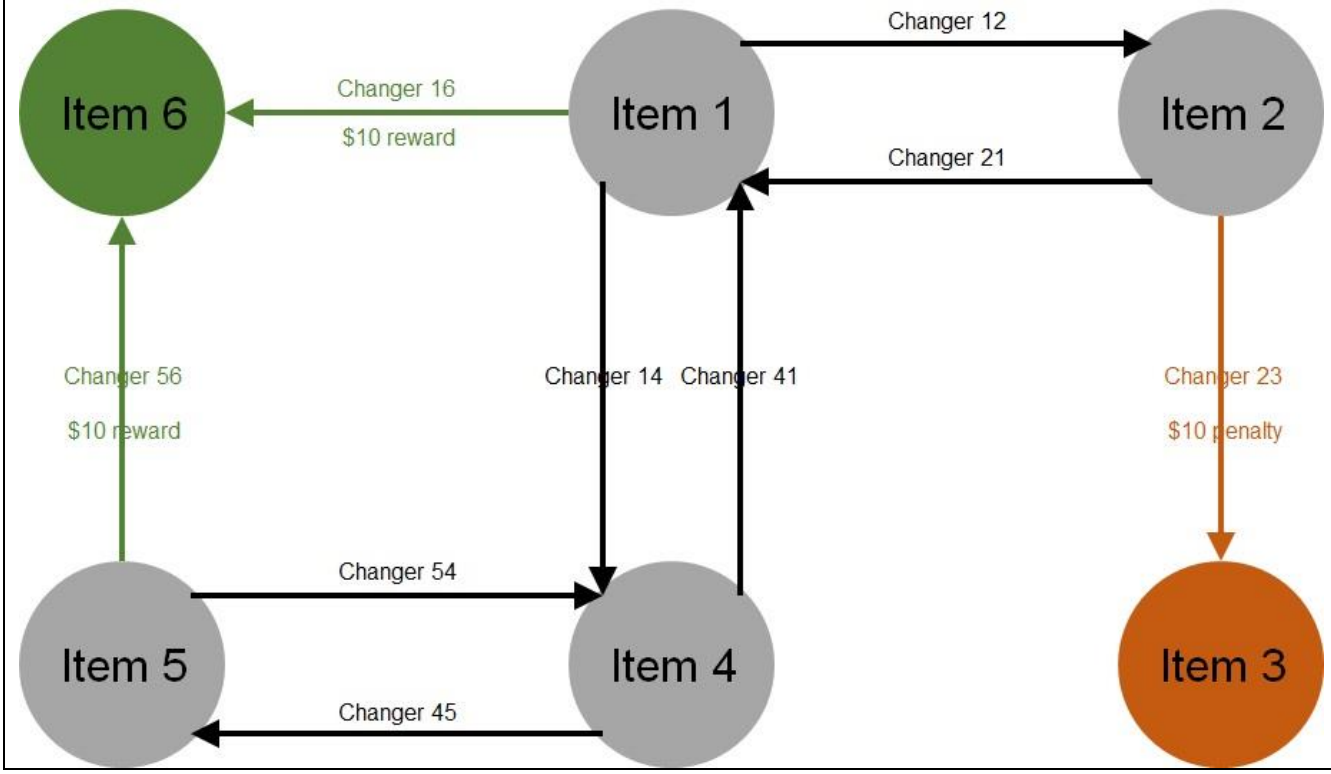


Figure 2 (Flow diagram of the case of six items)



Q-Learning Problem

In the above problem, suppose the environment assigns the agent a reward of $10 when the desired item 6 is produced and a penalty of $10 when the waste item 3 is produced.

Suppose the agent is not informed upfront about the reward scheme. Initially, therefore, the agent has to make random moves. As it goes through a series of steps, it receives rewards and penalties, learns by using reinforcement learning, and starts making calculated moves.

Input items and the reward system make up the environment, and changers (converters of items) make up the agent. Input items constitute states, and changers constitute actions.

Overall process in this problem can be considered a Markov process. There is no simple formula or model available for the agent to take actions. Under these circumstances, this problem as a reinforcement learning problem based on Q-learning algorithm.

Q-learning is a popular reinforcement learning. In this, the agent keeps the history of state-action-reward sequences in the form of a Q table, and learns an optimal policy for taking an action for a given state by reviewing the Q table.

Q-learning is based on Q values, which depend on the environment-agent's state and the agent's action. When the agent starts afresh, all Q values have 0 or some other fixed default value. Q values keep getting updated as the agent goes through various states, makes actions, gets rewards, and learns from rewards.

Q values also depend on learning rate alpha and discount factor gamma. Learning rate alpha "relates" Q values to immediate rewards; Q value of a state-action is linearly proportional to reward for that state-action in the absence of discount factor gamma. Discount factor gamma "relates" Q values to future rewards; Q value of a state-action is linearly proportional to reward for the forthcoming state-action in the absence of learning rate alpha.

Q values are updated according to the following formula:

Q_new_value_of_previous_state_action = (1 – alpha) * Q_old_value_of_ previous_state_action + alpha * (Reward_of_current_state_action + gamma * Q_maximum_value_for_current_state_action)

Single-Entity Problem

To simplify this case, let us assume that only one entity of item 1, 2, 3, 4, 5, or 6 can exist at any given time.

The table of rewards and Q values for this case will be somewhat similar to table 1.

Table 1 (Rewards and Q values of single-entity problem)

Input item	Changer	Output item	Reward	Initial Q value	Subsequent Q values
1	12	2	0	0	
1	14	4	0	0	
1	16	6	10	0	
2	21	1	0	0	
2	23	3	-10	0	
3	-	-	0	0	
4	41	1	0	0	
4	45	5	0	0	

5	54	4	0	0	
5	56	6	10	0	
6	-	-	0	0	

Since the number of possible Q records is small, it is easy to simulate this case manually without using a code. Table 2 shows the history of states, actions, rewards, and computations from one such simulation. For this simulation, we have used a learning rate alpha of 0.9 and a discount factor gamma of 0.4 .

Table 2 (Manual simulation of single-entity problem)

Item source	Input item	Changer source	Changer	Output item	Reward	Old Q value	Output's Max Q	New Q value	Computation
Add	2	Random	23	3	-10	0	0	-9	0.1 * 0 + 0.9 * (-10 + 0.4 * 0)
Add	4	Random	45	5	0	0	0	0	0.1 * 0 + 0.9 * (0 + 0.4 * 0)
Convert	5	Random	54	4	0	0	0	0	0.1 * 0 + 0.9 * (0 + 0.4 * 0)
Convert	4	Random	45	5	0	0	0	0	0.1 * 0 + 0.9 * (0 + 0.4 * 0)
Convert	5	Random	56	6	10	0	0	9	0.1 * 0 + 0.9 * (10 + 0.4 * 0)
Add	1	Random	12	2	0	0	0	0	0.1 * 0 + 0.9 * (0 + 0.4 * 0)
Convert	2	Q value	21	1	0	0	0	0	0.1 * 0 + 0.9 * (0 + 0.4 * 0)
Convert	1	Random	16	6	10	0	0	9	0.1 * 0 + 0.9 * (10 + 0.4 * 0)
Add	5	Q value	56	6	10	9	0	10	0.1 * 9 + 0.9 * (10 + 0.4 * 0)
Add	4	Random	41	1	0	0	9	3	0.1 * 0 + 0.9 * (0 + 0.4 * 9)
Convert	1	Q value	16	6	10	9	0	10	0.1 * 9 + 0.9 * (10 + 0.4 * 0)
Add	2	Q value	21	1	0	0	10	4	0.1 * 0 + 0.9 * (0 + 0.4 * 10)
Convert	1	Q value	16	6	10	10	0	10	0.1 * 10 + 0.9 * (10 + 0.4 * 0)
Add	5	Q value	56	6	10	10	0	10	0.1 * 10 + 0.9 * (10 + 0.4 * 0)
Add	2	Q value	21	1	0	4	10	4	0.1 * 4 + 0.9 * (0 + 0.4 * 10)
Convert	1	Q value	16	6	10	10	0	10	0.1 * 10 + 0.9 * (10 + 0.4 * 0)

The above simulation clearly demonstrates that the agent progresses from random and unrewarding actions to calculated and rewarding actions by using reinforcement learning.

Two-Entity Problem

Let us now extend the above single-entity problem. We now assume that two entities of item 1 or 2 or 3 or 4 or 5 or 6 can co-exist at any given time.

If we abbreviate items "i & j" as "ij", possible input items and possible output items are: 11, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 33, 34, 35, 36, 44, 45, 46, 55, 56, and 66.

If we abbreviate changers "ij to mn" as "ijmn", possible changers are: 1122, 1124, 1126, 1142, 1144, 1146, 1166, 1221, 1223, 1241, 1234, 1216, 1236, 1323, 1334, 1336, 1421, 1425, 1441, 1445, 1416, 1456, 1524, 1526, 1544, 1546, 1566, 1626, 1646, 1666, 2211, 2213, 2233, 2313, 2333, 2411, 2415, 2413, 2435, 2514, 2516, 2534, 2536, 2616, 2636, 3431, 3435, 3534, 3536, 4411, 4415, 4455, 4514, 4516, 4545, 4556, 4616, 4656, 5544, 5546, 5566, 5646, 5666.

The table of Rewards and Q values for this scenario will be similar to table 3.

Table 3 (Rewards and Q values of two-entity problem)

Input items	Changers	Output items	Reward	Initial Q value	Subsequent Q value
11	1124	24	0	0	

11	1126	26	10	0	
11	1142	42	0	0	
11	1144	44	0	0	
11	1146	46	10	0	
11	1166	66	20	0	
12	1216	16	10	0	
12	1221	21	0	0	
12	1223	23	-10	0	
12	1234	34	-10	0	
12	1236	36	0	0	
12	1241	41	0	0	
13	1323	23	-10	0	
13	1334	34	-10	0	
13	1336	36	0	0	
14	1416	16	10	0	
14	1421	21	0	0	
14	1425	25	0	0	
14	1441	41	0	0	
14	1445	45	0	0	
14	1456	56	10	0	
15	1524	24	0	0	
15	1526	26	10	0	
15	1544	44	0	0	
15	1546	46	10	0	
15	1566	66	20	0	
16	1626	26	10	0	
16	1646	46	10	0	
16	1666	66	20	0	
22	2211	11	0	0	
22	2213	13	-10	0	
22	2233	33	-20	0	
23	2313	13	-10	0	
23	2333	33	-20	0	
24	2411	11	0	0	
24	2413	13	-10	0	
24	2415	15	0	0	
24	2435	35	-10	0	
25	2514	14	0	0	
25	2516	16	10	0	
25	2534	34	-10	0	
25	2536	36	0	0	
26	2616	16	10	0	
26	2636	36	0	0	
34	3431	31	-10	0	

34	3435	35	-10	0	
35	3534	34	-10	0	
35	3536	36	0	0	
44	4411	11	0	0	
44	4415	15	0	0	
44	4455	55	0	0	
45	4514	14	0	0	
45	4516	16	10	0	
45	4545	45	0	0	
45	4556	56	10	0	
46	4616	16	10	0	
46	4656	56	10	0	
55	5544	44	0	0	
55	5546	46	10	0	
55	5566	66	20	0	
56	5646	46	10	0	
56	5666	66	20	0	

Unlike table 1 of single-entity problem, this table 3 is huge. As a result, in this case, the agent will need to go through a large number of steps before it shows some effect of reinforcement learning. It is, therefore, difficult to simulate this case manually. Let us simulate this case and create a history table by using a python code.

Benchmark

An agent, which does not learn, may take any state-action from table 3 (of rewards and Q values of two-entity problem). Since there are 62 records with total reward of 110, approximate average reward is 2. At this stage, we can therefore estimate that a non-learning agent will have an approximate average reward of 2.

An agent, which does learn, avoids a state-action leading to a penalty (i.e. negative reward). Since there are 48 such records (of non-negative rewards) with total reward of 270, approximate average reward is 5. At this stage, we can therefore estimate that a learning agent will have an approximate average reward of 5.

In view of this, it will be fair to take average reward as the benchmark for the agent's learning. It will be fair to take average reward of 5 as the benchmark criterion to determine whether the agent has learnt.

Simulation of Two-Entity Problem

Capstone-LalitAPatel-R4-Part1Simulate.py is an illustrative python code to simulate this case of two entities.

At the start of a simulation session, a Q table with 0 q values is created. Another empty table is also created to hold the history of simulation records. The session is divided into several episodes. In the first episode, the agent's actions are random. In subsequent episodes, the agent's actions are based on Q learning, which is governed by learning rate alpha and discount factor gamma. In all but the last episode, alpha and gamma are selected randomly. In the last episode, alpha and gamma are selected based on best results.

Each episode is divided into several trials. Trials are independent from Q learning perspective, in the sense that the agent's Q learning is not carried forward from one trial to another trial. Each trial consists of many steps. In each step, the agent is provided with an item, which is either derived from the previous step or selected randomly. The agent then applies a changer to act on this item. This action leads to another item as listed in the

Q table. The agent is rewarded or penalized for its action, as listed in the Q table. In all except the first episode, the agent's action (by selecting a changer) is based on the Q learning algorithm.

Number of episodes in the simulation should be such that a wide spectrum of alpha from the range 0 to 1 and a wide spectrum of gamma from 0 to 1 are covered. The simulation presented here was rum for 30 episodes.

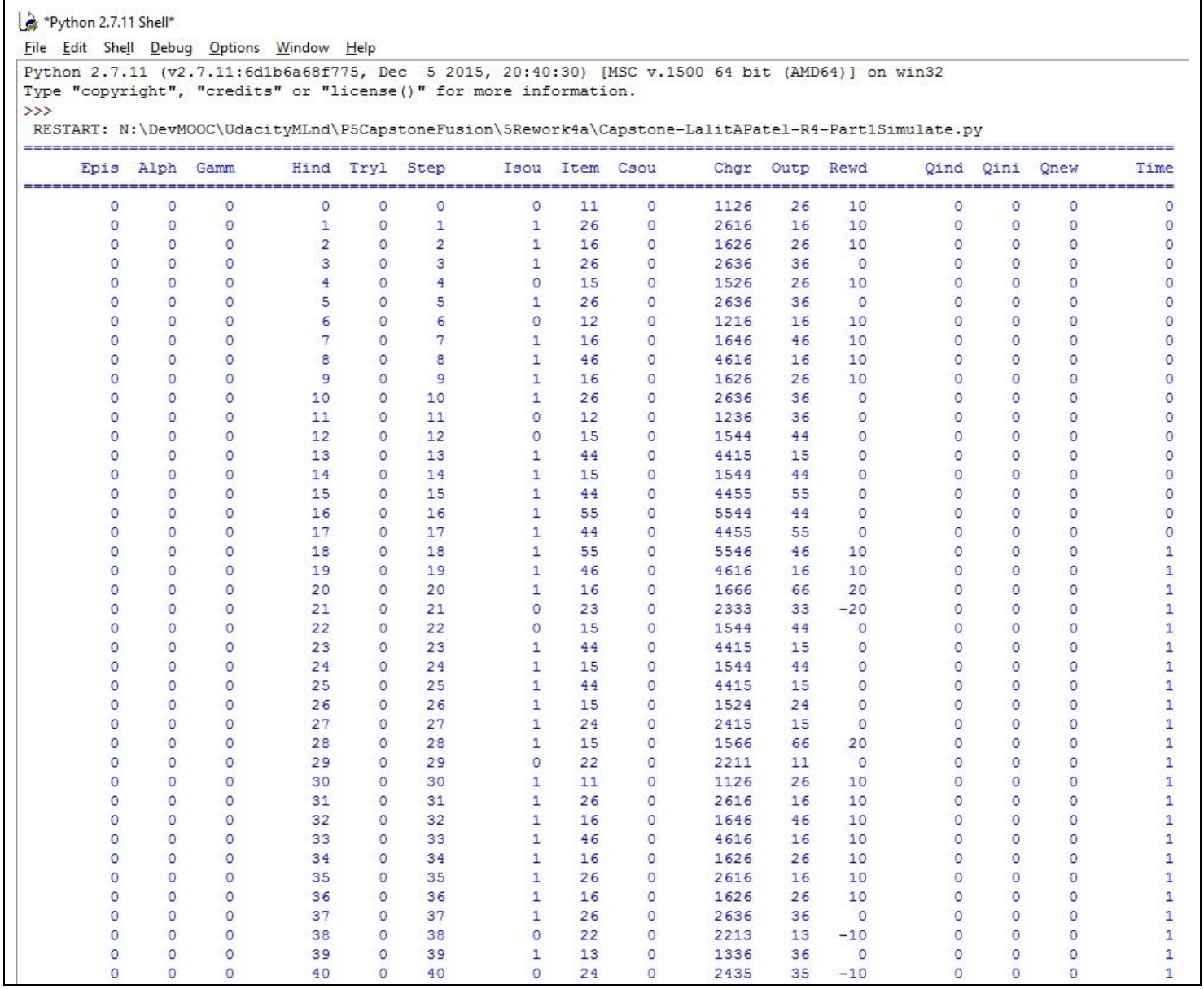
Number of trial in each episode should be such that average reward from the trials run can be safely taken as average reward of the episode. Upon understanding the criterion of the central limit theorem, the simulation presented here was run for 30 trials in each episode.

Number of steps in each trial should be such that the Q table has been reasonably well updated and the agent has shown signs of having learnt. The simulation presented here was run for 300 steps in each trial.

The simulation presented here was run a computer with: Intel Core i7 processor, 16 GB memory, Windows 10 64-bit operating system, and Python 2.7. As can be seen from the last line of Part1_Text.txt file, this simulation took 58,374 seconds (16.2 hours).

Part1_Screen.jpg is a screenshot taken manually from the screen from this simulation. Part1_Text.txt is the text of the same screen saved as a txt file.

Figure 3 (Part1_Screen.jpg - Screenshot from simulation)



Python 2.7.11 Shell

File Edit Shell Debug Options Window Help

```
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:40:30) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: N:\DevMOOC\UdacityMLnd\P5CapstoneFusion\5Rework4a\Capstone-LalitAPatel-R4-Part1Simulate.py
```

Epis	Alph	Gamm	Hind	Tryl	Step	Isou	Item	Csou	Chgr	Outp	Rewd	Qind	Qini	Qnew	Time
0	0	0	0	0	0	0	11	0	1126	26	10	0	0	0	0
0	0	0	1	0	1	1	26	0	2616	16	10	0	0	0	0
0	0	0	2	0	2	1	16	0	1626	26	10	0	0	0	0
0	0	0	3	0	3	1	26	0	2636	36	0	0	0	0	0
0	0	0	4	0	4	0	15	0	1526	26	10	0	0	0	0
0	0	0	5	0	5	1	26	0	2636	36	0	0	0	0	0
0	0	0	6	0	6	0	12	0	1216	16	10	0	0	0	0
0	0	0	7	0	7	1	16	0	1646	46	10	0	0	0	0
0	0	0	8	0	8	1	46	0	4616	16	10	0	0	0	0
0	0	0	9	0	9	1	16	0	1626	26	10	0	0	0	0
0	0	0	10	0	10	1	26	0	2636	36	0	0	0	0	0
0	0	0	11	0	11	0	12	0	1236	36	0	0	0	0	0
0	0	0	12	0	12	0	15	0	1544	44	0	0	0	0	0
0	0	0	13	0	13	1	44	0	4415	15	0	0	0	0	0
0	0	0	14	0	14	1	15	0	1544	44	0	0	0	0	0
0	0	0	15	0	15	1	44	0	4455	55	0	0	0	0	0
0	0	0	16	0	16	1	55	0	5544	44	0	0	0	0	0
0	0	0	17	0	17	1	44	0	4455	55	0	0	0	0	0
0	0	0	18	0	18	1	55	0	5546	46	10	0	0	0	1
0	0	0	19	0	19	1	46	0	4616	16	10	0	0	0	1
0	0	0	20	0	20	1	16	0	1666	66	20	0	0	0	1
0	0	0	21	0	21	0	23	0	2333	33	-20	0	0	0	1
0	0	0	22	0	22	0	15	0	1544	44	0	0	0	0	1
0	0	0	23	0	23	1	44	0	4415	15	0	0	0	0	1
0	0	0	24	0	24	1	15	0	1544	44	0	0	0	0	1
0	0	0	25	0	25	1	44	0	4415	15	0	0	0	0	1
0	0	0	26	0	26	1	15	0	1524	24	0	0	0	0	1
0	0	0	27	0	27	1	24	0	2415	15	0	0	0	0	1
0	0	0	28	0	28	1	15	0	1566	66	20	0	0	0	1
0	0	0	29	0	29	0	22	0	2211	11	0	0	0	0	1
0	0	0	30	0	30	1	11	0	1126	26	10	0	0	0	1
0	0	0	31	0	31	1	26	0	2616	16	10	0	0	0	1
0	0	0	32	0	32	1	16	0	1646	46	10	0	0	0	1
0	0	0	33	0	33	1	46	0	4616	16	10	0	0	0	1
0	0	0	34	0	34	1	16	0	1626	26	10	0	0	0	1
0	0	0	35	0	35	1	26	0	2616	16	10	0	0	0	1
0	0	0	36	0	36	1	16	0	1626	26	10	0	0	0	1
0	0	0	37	0	37	1	26	0	2636	36	0	0	0	0	1
0	0	0	38	0	38	0	22	0	2213	13	-10	0	0	0	1
0	0	0	39	0	39	1	13	0	1336	36	0	0	0	0	1
0	0	0	40	0	40	0	24	0	2435	35	-10	0	0	0	1

The simulation code generates fusionQ.csv and fusionH.csv files. fusionQ.csv is the Q table of updated q values, and fusionH.csv is the history table of simulation records.

Analysis of Simulation Results

Capstone-LalitAPatel-R4-Part2Analyze.py is an illustrative python code to analyze fusionH.csv file of history table of the simulation records.

This code computes and plots average rewards [averaged over steps] by episodes of different alpha and gamma values. It also computes and plots average cumulative rewards by steps in a few trial runs.

Part2_Screen.jpg is a screenshot taken manually from the screen from a typical simulation session. Part2_Text.txt is the text of the same screen saved as a txt file.

Figure 4 (Part2_Screen.jpg – Screenshot from analysis)

```
11      11      0.53    0.80    9000    2040    6960    60820   6.757778        11
12      12      0.03    0.67    9000    2794    6206    20040   2.226667        12
13      13      0.52    0.62    9000    2269    6731    54160   6.017778        13
14      14      0.46    0.37    9000    2112    6888    60210   6.690000        14
15      15      0.46    0.88    9000    2220    6780    55810   6.201111        15
16      16      0.39    0.45    9000    2223    6777    56530   6.281111        16
17      17      0.61    0.82    9000    2026    6974    61930   6.881111        17
18      18      0.42    0.68    9000    2172    6828    57970   6.441111        18
19      19      0.37    0.91    9000    2063    6937    61890   6.876667        19
20      20      0.77    0.86    9000    1989    7011    63360   7.040000        20
21      21      0.59    0.94    9000    2104    6896    58940   6.548889        21
22      22      0.72    0.87    9000    2117    6883    58540   6.504444        22
23      23      0.91    0.11    9000    2232    6768    55880   6.208889        23
24      24      0.55    0.60    9000    2075    6925    59380   6.597778        24
25      25      0.91    0.70    9000    2133    6867    58700   6.522222        25
26      26      0.78    0.51    9000    2034    6966    61620   6.846667        26
27      27      0.39    0.76    9000    2017    6983    62190   6.910000        27
28      28      0.61    0.22    9000    1964    7036    63030   7.003333        28
29      29      0.65    0.47    9000    2190    6810    56280   6.253333        29
30      30      0.30    0.84    9000    2237    6763    54700   6.077778        30
31      31      0.77    0.86   18000    3956   14044   124860   6.936667        31

Warning (from warnings module):
  File "N:\DevMOOC\UdacityMLnd\P5CapstoneFusion\5Rework4a\Capstone-LalitAPatel-R4-Par
    zHdc = zHdb.sort(columns='AvgRewd', ascending=False)
FutureWarning: sort(columns=....) is deprecated, use sort_values(by=.....)
================================================================================
Summary of rewards (average per step):

 alpha of max reward   gamma of max rewd      Maximum reward   Random act reward
            0.770000            0.860000            7.040000            2.000000
================================================================================
Summary of rewards (average per step):
```

This analysis code generates fusion_RewardByAlphaGamma.jpg, fusion_RewardByStep_Episode00.jpg, and fusion_RewardByStep_Episode31.jpg files.

Rewards versus alpha-gamma

Figure 5 (fusion_RewardByAlphaGamma.jpg)

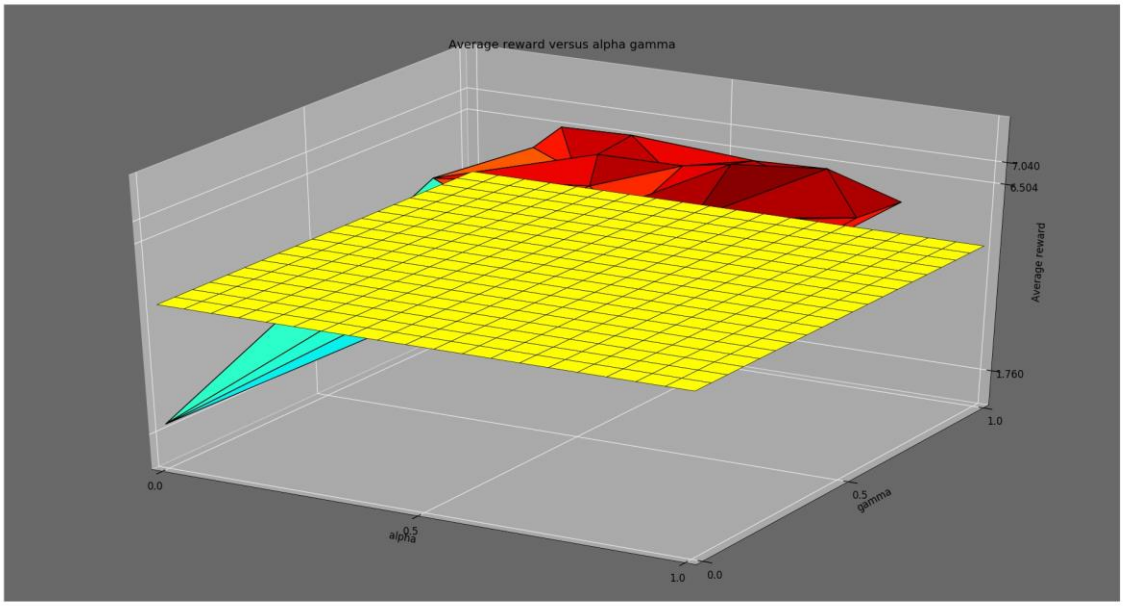


Figure 5 shows average rewards [averaged over steps] versus alpha-gamma. In this, the 0 alpha 0 gamma point corresponds to the agent making random actions without using Q learning, and other (>0 alpha >0 gamma) points correspond to the agent making calculated actions by using Q learning. The yellow plane corresponds to the benchmark of average reward of 5.

Figure 5 and the Part2_text show that the average reward for the 0 alpha 0 gamma case is approximately 2, which lies below the benchmark plane. This is as expected as explained in the benchmark section. In the 0 alpha 0 gamma case, the agent's actions are not geared towards getting better rewards.

Figure 5 and the Part2_text show that the average reward for each >0.2 alpha >0 gamma case is approximately 5, which lies above the benchmark plane. This is as expected as explained in the benchmark section. In the >0.2 alpha >0 gamma case, the agent's actions are geared towards getting better rewards.

Figure 5 and the Part2_text show that the average reward for alpha less than 0.2 is approximately 2, which lies below the benchmark plane. This is because the agent's learning rate is so less that it does not reach an adequate level of learning within 300 steps.

Figure 5 and Part2_text show that average reward for the 0 alpha 0 gamma case is 2.00. Minimum average reward is 1.76. Median average reward is 6.50. Maximum average reward is 7.04 for 0.77 alpha 0.86 gamma.

Rewards versus Steps of Random Actions

Figure 6 (fusion_RewardByStep_Episode00.jpg)

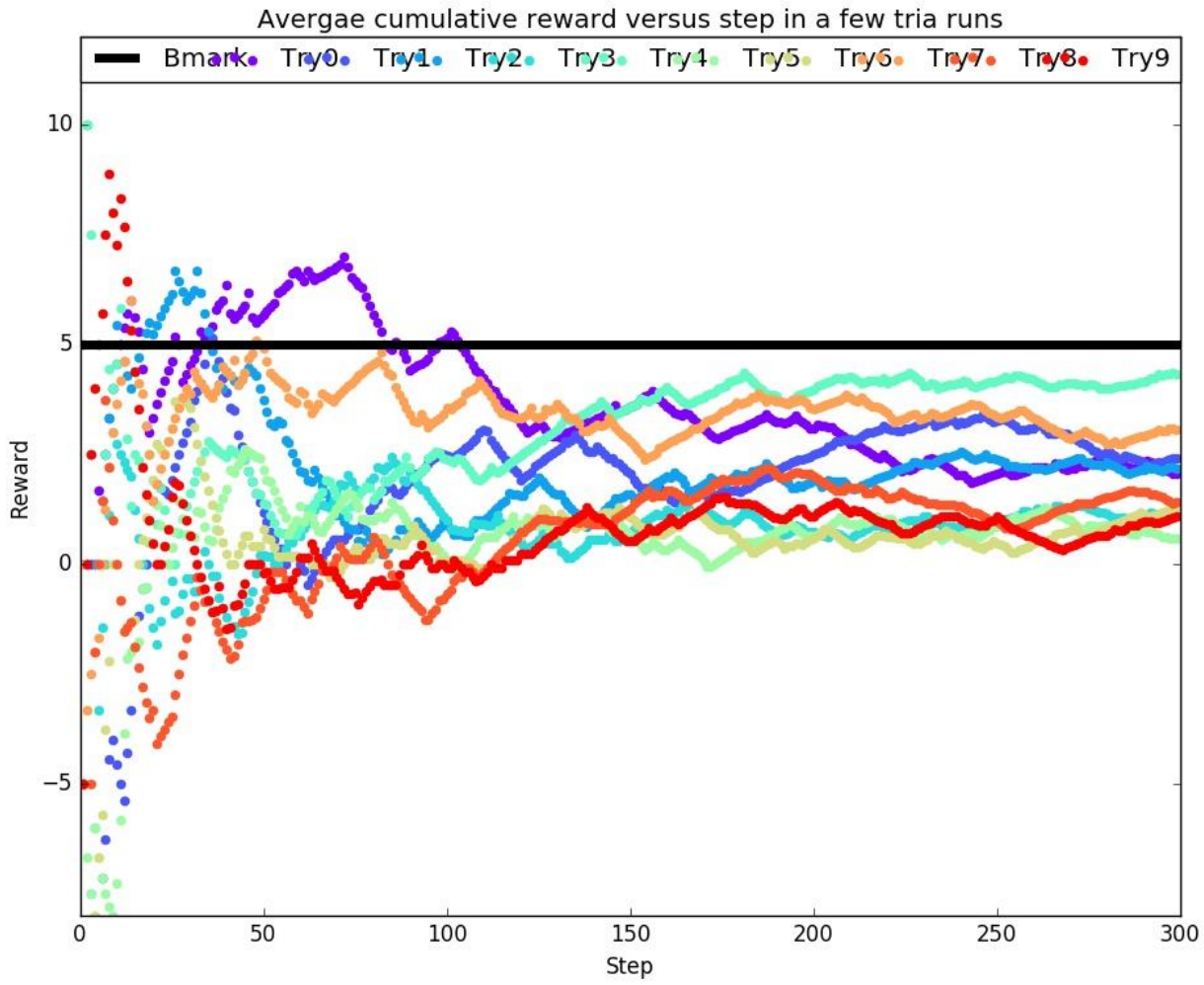


Figure 6 shows average cumulative rewards [cumulated and averaged over steps] versus steps in a few trials with 0 alpha 0 gamma. The black line shows the benchmark.

This figure shows that average rewards in these trials remain below the benchmark of average reward of 5.

This figure does not show any improvement in the average cumulative reward as the agent makes more steps. This is as expected. The agent does not learn from rewards or penalties and makes random actions. As a result, the agent's chances of winning rewards are dim and do not improve as time goes.

Rewards versus Steps of Q-Learning Actions

Figure 7 (fusion_RewardByStep_Episode21.jpg)

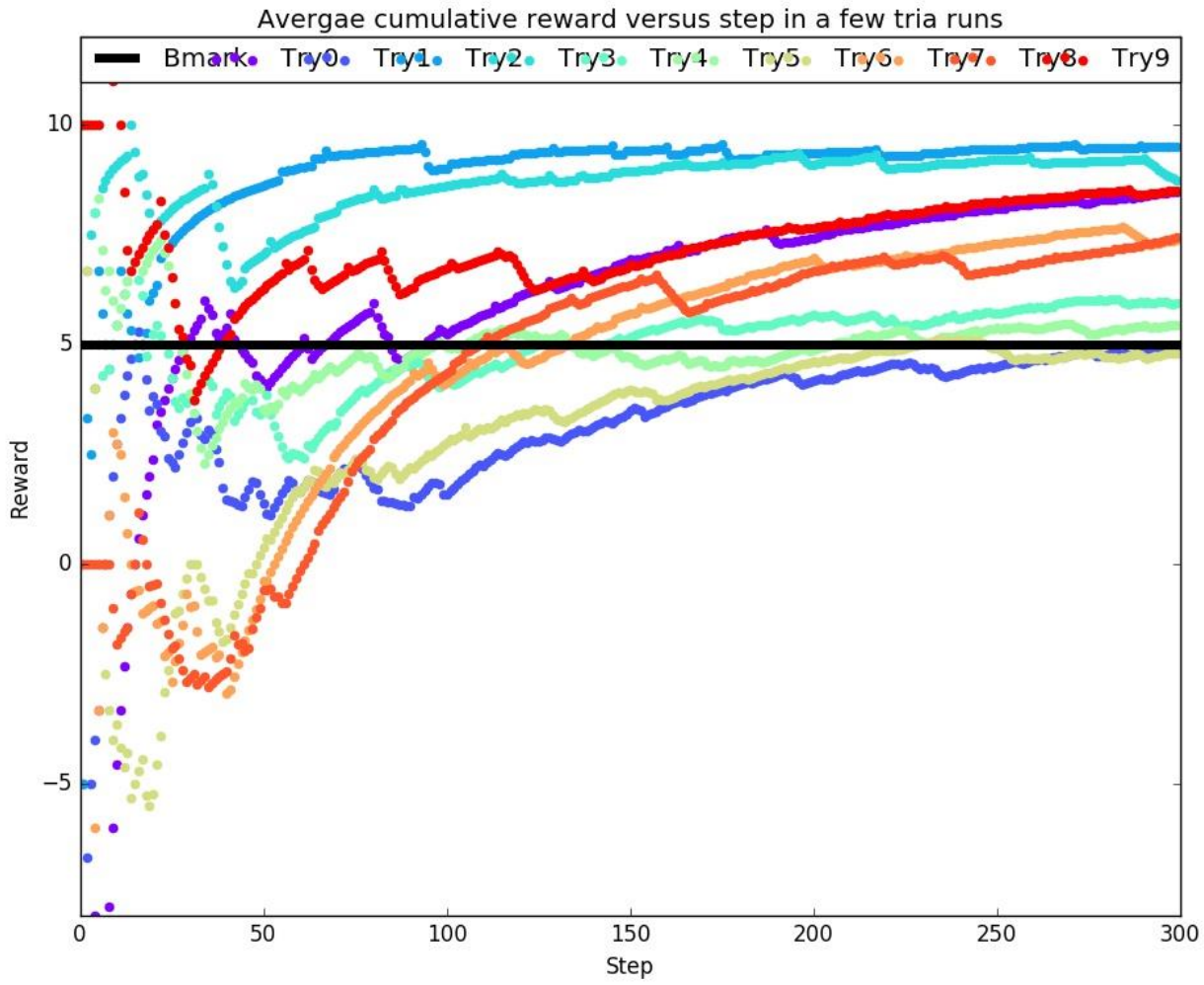


Figure 7 shows average cumulative rewards [cumulated and averaged over steps] versus steps in a few trials with 0.55 alpha 0.51 gamma. The black line shows the benchmark.

This figure shows that average rewards in these trials are mostly above the benchmark of average reward of 5. For 2 trials, more steps may be needed to cross the benchmark.

This figure does show clear improvement in the average cumulative reward as the agent makes more steps. This is as expected. The agent learns from rewards and penalties and makes calculated actions. As a result, the agent's chances of winning rewards are high and keep improving as time goes.

In this figure, some trials do not show improvement as clearly and as high as expected. This is mainly because the agent may not have learn enough during the low number [300] of steps. If the agent were allowed to pass through a much higher number [>1000] of steps, the agent would learn enough and start receiving better rewards step after step, thereby showing better average cumulative rewards.

Conclusion

It is possible to apply Q learning based reinforcement learning to the problem of reaction dynamics. The method and code provided in this project can be used as a guide towards studying this problem.

In a typical problem of reaction dynamics, input items and the reward system make up the environment, and changers (converters of items) make up the agent. Input items constitute states, and changers constitute actions.

It is fair to take average reward as the benchmark for the agent's learning. A non-learning random-action agent has an average reward of 2, whereas a learning calculated-action agent has an average reward of 5.

If the agent does not learn, there is no improvement in the average cumulative reward as the agent makes more steps. On the other hand, if the agent learns, there is a clear improvement in the average cumulative reward as the agent makes more steps.

For this problem, it was difficult to arrive at a reasonable definition of the state. Another big problem was that lots of trial runs are needed before the agent starts referring to previously-used Q records and start achieving good rewards.