CSE 574

Project4: Supporting material

UBname :De Guo

UBnumber :50289395

Abstract

The project is to do classification for MNIST dataset (for training, validation and testing) and USPS data(for testing).

We have four independent models: Multiclassification for Logistic Regression, Neural Network, Support Machine Vector and Random Forest. By doing training then tuning hyper parameters and we could get accuracies for each model with MNIST or USPS dataset.

After we do combination of every model and make an evaluation for above five models.

1 Coding tasks

1.1 Build a 3-layer neural network using Keras library

```
model.add(Dense(units=128,activation='relu',input_dim=4))

model.add(Dense(units=128,activation='relu'))

model.add(Dense(units=4,activation='linear'))
```

We use Keras library to build our three-layer neural network for reinforcement learning. The first layer has 4 inputs which corresponding to 4 features and we use ReLU as our activation function so we get a 4*128 weight matrix. The second layer is 128 inputs which are the first layer's outputs and also equals to the number of neurons in first layer. We use ReLU as our activation function and we get a 128*128 weight matrix. For the third layer which is the output layer, the activation function is linear, because we want get a linear output and the number of output is 4.

1.2 Implement exponential-decay formula for epsilon

```
self.epsilon=self.min_epsilon+(self.max_epsilon-self.min_epsilon)*np.exp(-(self.lamb*self.steps))
```

Here, we could think that current episode equals to minimum episode +(maximum episode – minimum episode)*(current episode/number of episodes)

1.3 Implement Q-function

```
if st_next is None:
    t[act]=rew
else:
        t[act]=rew+self.gamma*np.max(q_vals_next[i][act])
```

we have our formula as follows:

$$Q_t = \begin{cases} r_t, & \text{if episode terminates at step } t+1 \\ r_t + \gamma max_a Q(s_t, a_t; \Theta), & \text{otherwise} \end{cases}$$

formula as follows This means we have reward(t) at step t+1 and before that, we have the second equation to calculate Q(t) and it follows the content which is covered in lecture. It means the Q(t) is reward in that state plus gamma multiply the maximum Q values of next state with 4 actions in this problem. Gamma is discounted coefficient.

1.4 Report

When I just use original code, the performance is not bad. The number of episode is 10000:

```
Episode 9400
Time Elapsed: 879.23s
Epsilon 0.062295237405542485
Last Episode Reward: 7
Episode Reward Rolling Mean: 6.023653370605311
----------
Episode 9500
Time Elapsed: 888.35s
Epsilon 0.061767741122092365
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.039251143495373
----------
Episode 9600
Time Elapsed: 897.41s
Epsilon 0.06126400208585532
Last Episode Reward: 7
Episode Reward Rolling Mean: 6.050310493632249
----------
Episode 9700
Time Elapsed: 906.32s
Epsilon 0.06079153450455659
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.063222580981148
----------
Episode 9800
Time Elapsed: 915.23s
Epsilon 0.06033681696444667
Last Episode Reward: 7
Episode Reward Rolling Mean: 6.079167096175652
----------
Episode 9900
Time Elapsed: 924.10s
Epsilon 0.059901754707281575
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.092949699010305
```

Here, we could find that each episode time is about 9 seconds and total time is about 924 seconds for 10000 episodes. The final mean reward is about 6.0929.And for the final result we could find that the last episode reward for each episode has a little fluctuation. It may go to 8(the biggest reward) or sometimes go to 6(small rewards when the number of episodes is large).

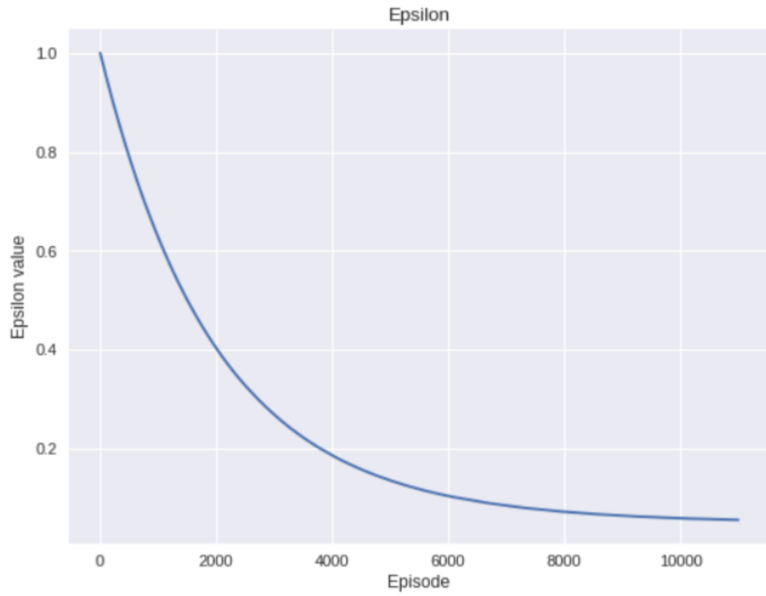
I think this is because the agent was trying to make the rewards more bigger(In effect, it couldn't get more rewards bigger than 8, but no one tells it the maximum reward), this attitude I am thinking about is called exploration which means it was trying to do new different actions so as to find the biggest reward and the corresponding reward is a little different from the original, showing a fluctuation.

And I change the number of episodes to 11000:

```
Episode 10400
Time Elapsed: 946.66s
Epsilon 0.05790947956138873
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.140083487040093
----------
Episode 10500
Time Elapsed: 955.06s
Epsilon 0.05757620174179914
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.149600999903855
----------
Episode 10600
Time Elapsed: 963.29s
Epsilon 0.05725914450121l015
Last Episode Reward: 7
Episode Reward Rolling Mean: 6.161603656794591
----------
Episode 10700
Time Elapsed: 971.92s
Epsilon 0.05694111194699674
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.175266484293934
----------
Episode 10800
Time Elapsed: 980.47s
Epsilon 0.056641992439017394
Last Episode Reward: 7
Episode Reward Rolling Mean: 6.182973553873469
----------
Episode 10900
Time Elapsed: 989.24s
Epsilon 0.05635068057670004
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.1919266734561615
```

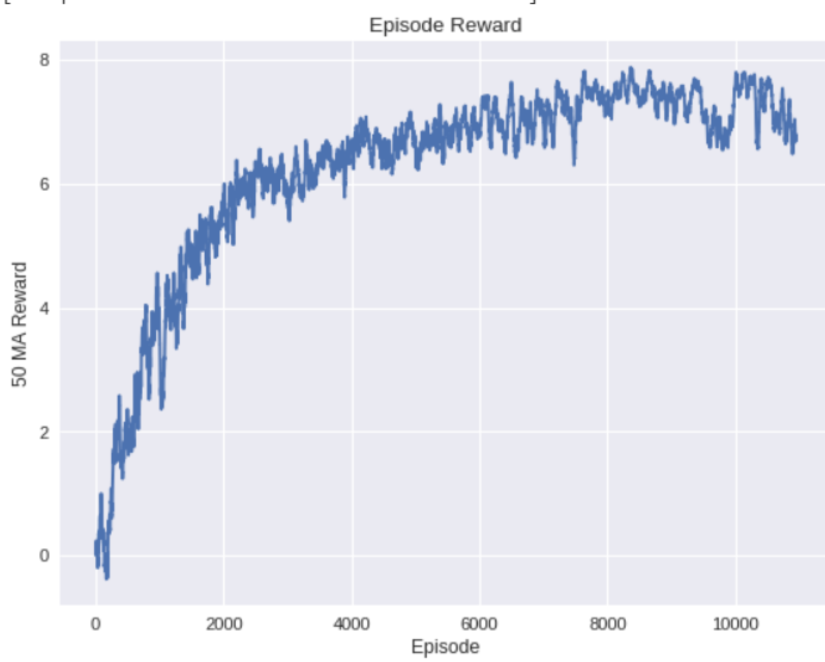
Here, we could find that the reward fluctuation which I already explained in the previous trial. And the final result(mean episode reward) is 6.1919 which is good comparing with last one.

```
[<matplotlib.lines.Line2D at 0x7ff04ac72668>]
```



The epsilon value is decreasing when the number of episode is going up.

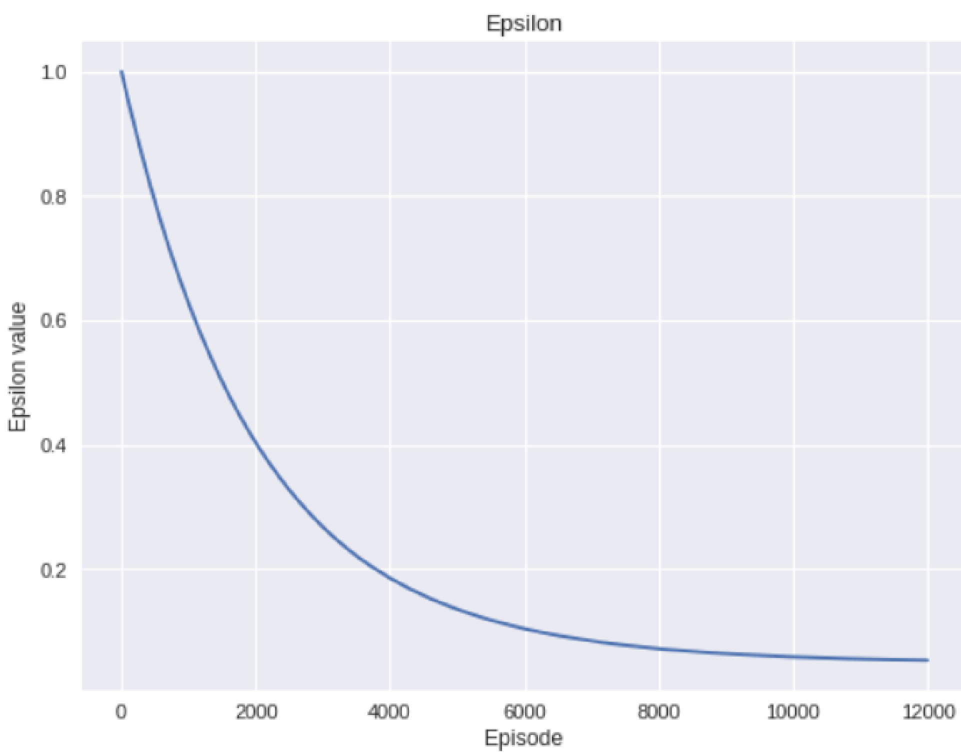
```
[<matplotlib.lines.Line2D at 0x7ff04abd0470>]
```

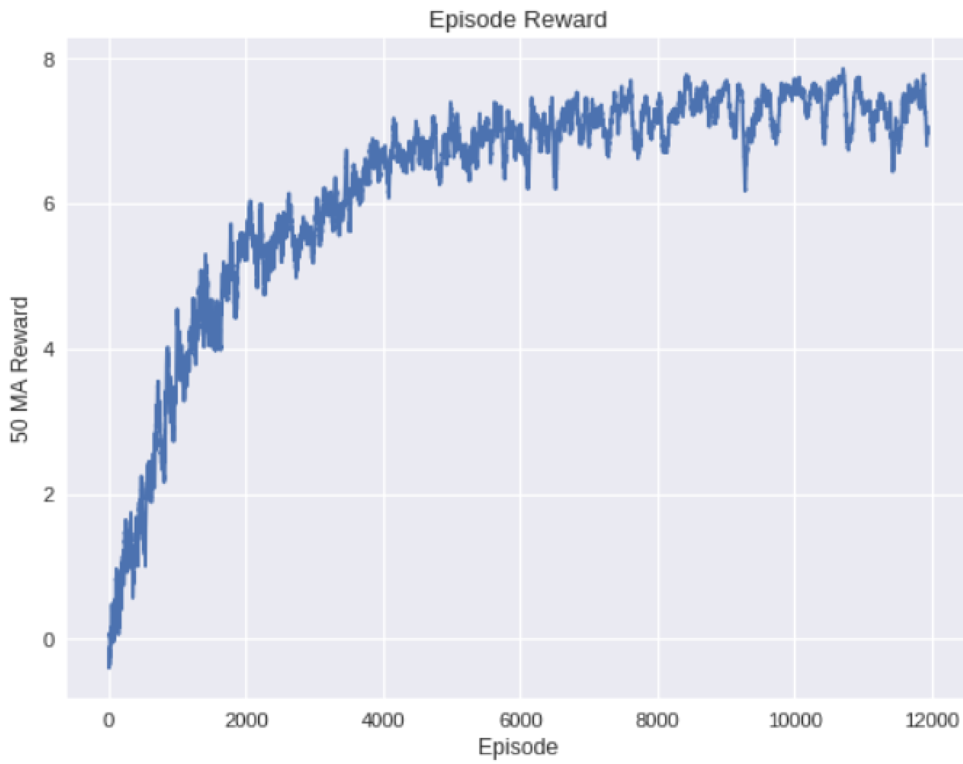


Episode reward is very high with a big episode. It already gets very close to the maximum reward value 8.

And again, I change the number of episode to 12000:

```
----------
Episode 11500
Time Elapsed: 1036.81s
Epsilon 0.054892786534259774
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.193140952548022
----------
Episode 11600
Time Elapsed: 1045.66s
Epsilon 0.05468076699627626
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.199634814363969
----------
Episode 11700
Time Elapsed: 1054.29s
Epsilon 0.05448174278188649
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.2063615205585725
----------
Episode 11800
Time Elapsed: 1062.93s
Epsilon 0.05429225392148467
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.216477224168875
----------
Episode 11900
Time Elapsed: 1071.45s
Epsilon 0.054110982226865154
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.227184136937548
```

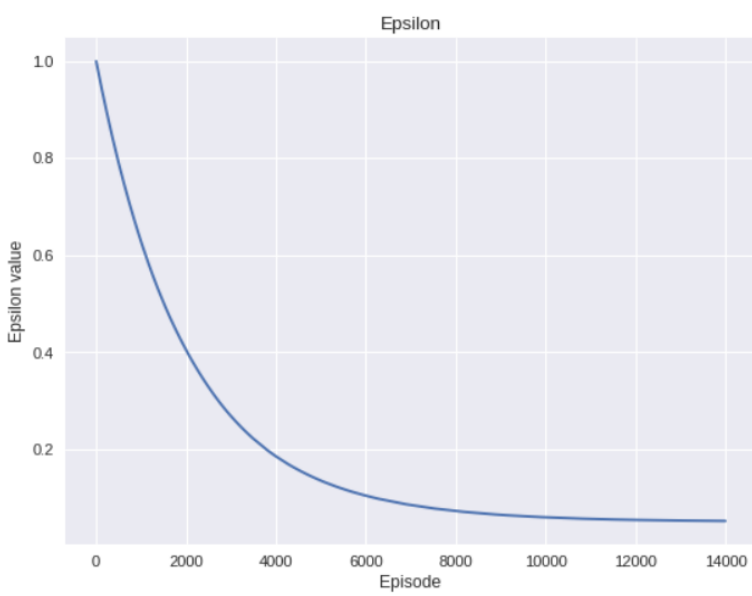




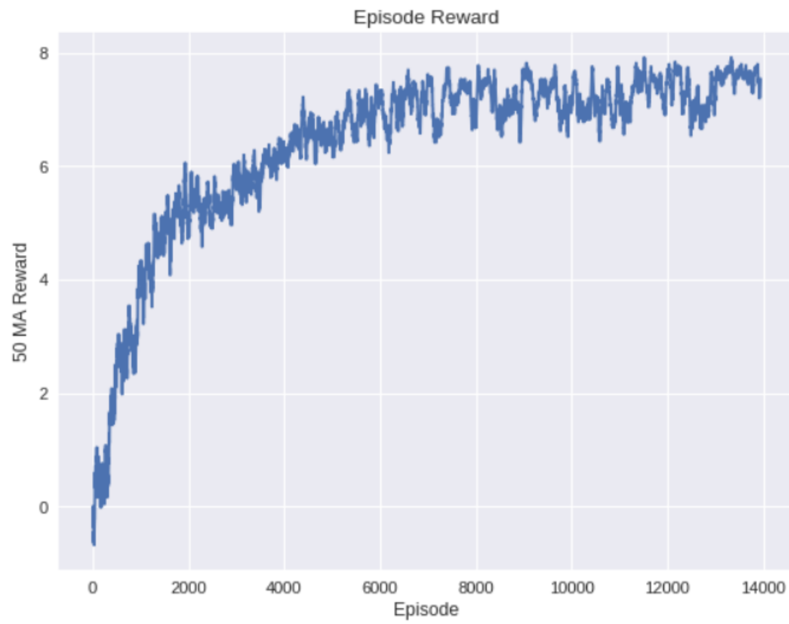
The final mean reward is 6.2272.

And I set the number of episode as 14000:

```
Episode 13500
Time Elapsed: 1240.03s
Epsilon 0.05200563753863029
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.307663607193493
----------
Episode 13600
Time Elapsed: 1248.27s
Epsilon 0.051922952921667105
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.3162728686763945
----------
Episode 13700
Time Elapsed: 1256.59s
Epsilon 0.051843031895062404
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.326299536798765
----------
Episode 13800
Time Elapsed: 1264.82s
Epsilon 0.05176643251529657
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.3356689292752355
----------
Episode 13900
Time Elapsed: 1273.09s
Epsilon 0.051693101382319424
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.344757626258967
```
77



78



And my final mean reward is 6.345.

(The original lambda is 0.00005,gamma is 0.99,max epsilon is 1 and min epsilon is 0.01(these parameters are pretty good as default values when _init_ function has been defined.)Actually I think I could improve the model performance more deeply but I don't have enough time for this because I have a lot of other course to review and it's in exam week. I'm very sorry about that and I will post my all effort for this model in this pdf.)

2 Writing tasks

2.1 First question

Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore. [20 points]

If the agent always choose the maximized Q-value, then it could get the maximized Q from current actions all the time and it will keep that value and do no explorations. Maybe it could still get a good result(mean reward) but it will stop doing exploration and it won't get a higher reward without taking the risk of exploring new actions.

One way to force the agent to explore is to combine random selection and brain training with the help of memory with some probabilities. When it make random selections, this means the agent is doing something new which is the meaning of exploration.

The second way is to set a specific step number (before some number of episode) for agent which is in early stage of learning. And the third way is limiting the number of epsilon(By increasing the number of lambda, epsilon will be small more faster).

And another interesting thing is I find a little unaccuracy in the statement of the code.

5 - Agent

np.amax - Returns the maximum of an array

Epsilon

Our agent will randomly select its action at first by a certain percentage, called 'exploration rate' or 'epsilon'. This is because at first, it is better for the agent to try all kinds of things before it starts to see the patterns. When it is not deciding the action randomly, the agent will predict the reward value based on the current state and pick the action that will give the highest reward. We want our agent to descrese the number of random action, as it goes, so we indroduce an exponential-decay epsilon, that eventually will allow our agent to explore the evironment.

I think there is something wrong with last epsilon in the last sentence. The right answer is eventually epsilon will be very small with the number of episode increasing. Because there is -lambda|S|.Finally, epsilon becomes small which means the agent prefer to do exploitation rather than exploration, which also means the agent will focus on the current actions and doesn't find any new actions anymore.

And I made the comment in my code the same time. Thanks.

And go back to my second way. Maybe you can set this step number to be very large, which means you force the agent to keep doing exploration as the prolonged exploration time, or you can also set this number be small, which means you do not prefer to make the agent to do exploration that long time but of course you also make agent to do exploration a little longer time than the regular exploration time. The specific number is up to you. I think it's the second way to improve the exploration time comparing the first random way which is already executed in code.

2.2 Second question

Calculate Q-value for the given states and provide all the calculation steps. [20 points] Consider an environment which is a 3x3 grid, where one space of the grid is occupied by the agent (green square) and another is occupied by a goal (yellow square). The agent's action space consists of 4 actions: UP, DOWN, LEFT, and RIGHT. The goal is to have the agent move onto the space that the goal is occupying in as little moves as possible. Initially, the agent is set to be in the upper-left corner and the goal is in the lower-right corner. The agent receives a reward of: 1 when it moves closer to the goal -1 when it moves away from the goal 0 when it does not move at all (e.g., tries to move into an edge) Consider the following possible optimal set of actions and their resulting states, that reach the goal in the smallest number of steps:

	1	2	3
1	s_{11}	s_{12}	s_{13}
2	s_{21}	s_{22}	s_{23}
3	s_{31}	s_{32}	s_{33}

	up	down	left	right
s_{11}	3.901	3.940	3.901	3.940
s_{12}	2.940	2.970	2.901	2.970
s_{13}	1.970	1.99	1.940	1.970
s_{21}	2.901	2.970	2.940	2.970
s_{22}	1.940	1.99	1.940	1.99
s_{23}	0.970	1	0.970	0.99
s_{31}	1.940	1.970	1.970	1.99
s_{32}	0.970	0.99	0.970	1
s_{33}	0	0	0	0

$\gamma = 0.99$ and the target is in s_{33} are already given. We could start calculating Q-function from the last state which is s_{33}. We initialized it to 0 for 4 actions in s_{33}.

$q(s_{32}, up) = -1 + 0.99 \times \max_{a'}(q(s_{22}, a'))$, Here for $q(s_{22}, a')$, the agent could go →↓ or go ↓→, actually the Q-value is the same. $\max_{a'}(q(s_{22}, a'))$ is $1 + 0.99 \times 1 = 1.99$.

so $q(s_{32}, up) = -1 + 0.99 \times 1.99 \approx 0.970$. The same with $q(s_{32}, \cancel{right}$ left$)$ is 0.970.

129 $q(s_{32}, down) = 0 + 0.99 \times \max_{a'}(q(s_{32}, a')) = 0 + 0.99 \times 1 = 0.99$. $q(s_{32}, right) = 1 +$

s_{33} | 0 | 0 | 0 | 0

$\gamma = 0.99$ and the target is in s_{33} are already given. We could start calculating Q-function from the last state which is s_{33}. We initialized it to 0 for 4 actions in s_{33}.

$q(s_{32}, up) = -1 + 0.99 \times \max_{a'}(q(s_{22}, a'))$, Here for $q(s_{22}, a')$, the agent could go →↓ or go ↓→, actually the Q-value is the same. $\max_{a'}(q(s_{22}, a'))$ is $1 + 0.99 \times 1 = 1.99$.

so $q(s_{32}, up) = -1 + 0.99 \times 1.99 \approx 0.970$. The same with $q(s_{32}, \cancel{right}$ left$)$ is 0.970.

$q(s_{32}, down) = 0 + 0.99 \times \max_{a'}(q(s_{32}, a')) = 0 + 0.99 \times 1 = 0.99$. $q(s_{32}, right) = 1 +$
$0.99 \times \max_{a'}(q(s_{33}, a')) = 1$.

$q(s_{31}, up) = -1 + 0.99 \times \max_{a'} q(s_{21}, a') = -1 + 0.99 \times (1 + 0.99 \times 1 + 0.99^2 \times 1) \approx -1 + 0.99 \times 2.9701 \approx 1.940$ $q(s_{31}, down) = 0 + 0.99 \times \max_{a'} q(s_{31}, a') = 0 + 0.99 \times (1 + 0.99 \times 1) \approx 1.970 = q(s_{31}, left)$

$q(s_{31}, right) = 1 + 0.99 \times \max_{a'} q(s_{32}, a') = +1 + 0.99 \times 1 = 1.99$

$q(s_{23}, up) = -1 + 0.99 \times \max_{a'} q(s_{13}, a') = -1 + 0.99 \times (1 + 0.99) \approx 0.970 = q(s_{23}, left)$

$q(s_{23}, down) = 1 + 0.99 \times \max_{a'} q(s_{33}, a') = 1$ $q(s_{23}, right) = 0 + 0.99 \times \max_{a'} q(s_{23}, a') = 0.9$

$q(s_{22}, up) = -1 + 0.99 \times \max_{a'} q(s_{12}, a') = -1 + 0.99 \times (1 + 0.99 + 0.99^2) = -1 + 0.99 \times 2.9701 \approx 1.940$.

130 $q(s_{22}, down) = q(s_{22}, right) = 1 + 0.99 \times 1 = 1.99$ $q(s_{22}, left) = -1 + 0.99$

$$q(s_{13}, up) = 0 + 0.99 \times \max_{a'} (q(s_{13}, a')) = 0.99 \times (1 + 0.99 \times 1) \approx 1.970 = q(s_{13}, right)$$

$$q(s_{13}, left) = -1 + 0.99 \times \max_{a'} q(s_{12}, a') = -1 + 0.99 \times (1 + 0.99 \times 1 + 0.99^2) \approx 1.940$$

$$q(s_{13}, down) = 1 + 0.99 \times 1 = 1.99.$$

$$q(s_{12}, up) = 0 + 0.99 \times \max_{a'} q(s_{12}, a') = 0.99 \times (1 + 0.99 \times 1 + 0.99^2) \approx 2.940.$$

$$q(s_{12}, left) = -1 + 0.99 \times \max_{a'} q(s_{11}, a') = -1 + 0.99 \times (1 + 0.99 \times 1 + 0.99^2 \times 1 + 0.99^3 \times 1) \approx 2.901$$

$$q(s_{12}, right) = 1 + 0.99 \times \max_{a'} q(s_{13}, a') = 1 + 0.99 \times (1 + 0.99) \approx 2.970.$$

$$q(s_{12}, down) = 1 + 0.99 \times \max_{a'} q(s_{22}, a') = 1 + 0.99 \times (1 + 0.99) \approx 2.970$$

$$q(s_{11}, up) = 0 + 0.99 \times \max_{a'} q(s_{11}, a') = 0.99 \times (1 + 0.99 + 0.99^2 + 0.99^3) \approx 3.901 \quad = q(s_{11}, left)$$

$$q(s_{11}, right) = q(s_{11}, down) = 1 + 0.99 \times \max_{a'} q(s_{12}, a') \approx 3.940$$

Sorry again about couldn't finish the bonus part because I don't have enough time for this project in the exam week. I think it's not very difficult to understand the procedures but hard to understand the essence of it and reinforcement learning when confronting with more complicated and difficult problems. Thank you for giving us a good opportunity to learn and review these stuff.