



Hankuk University of Foreign Studies

QLearning-final term report



Computer Science and Engineering

과 목 명 : 딥러닝

담당 : 최재영 교수님

제 출 일 : 2021-12-23

학 과 : 컴퓨터전자시스템공학부

학 번 :

이 름 :

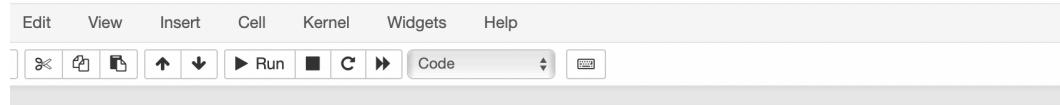
목차

1. 구현
2. 문제에 대한 답안

1. 구현

- Jupyter notebook을 이용해서 학습 자료에 있는 Q-learning.ipynb파일을 이용해서 코드를 구현해주었습니다.
- 아래는 구현한 결과 캡처본 입니다.

pyter Q_learning Last Checkpoint: Last Tuesday at 6:50 PM (unsaved changes)



Deep Learning Q Learning Final Project

학번: 201600765
학과: 컴퓨터전자시스템 공학부
이름: 김주원

- PDF슬라이드에 정의 한 state의 index를 location에 mapping

```
In [1]: location_to_state = {  
    'L1' : 0,  
    'L2' : 1,  
    'L3' : 2,  
    'L4' : 3,  
    'L5' : 4,  
    'L6' : 5,  
    'L7' : 6,  
    'L8' : 7,  
    'L9' : 8  
}  
  
In [2]: #Maps indices to locations  
state_to_location = dict((state,location) for location,state in  
                           location_to_state.items())  
print(state_to_location)  
  
{0: 'L1', 1: 'L2', 2: 'L3', 3: 'L4', 4: 'L5', 5: 'L6', 6: 'L7', 7: 'L8', 8: 'L9'}
```

- Action과 Reward 정의

```
In [3]: import numpy as np  
  
In [4]: actions = [0,1,2,3,4,5,6,7,8]  
  
In [5]: #Define the rewards  
rewards = np.array([  
    [0,1,0,0,0,0,0,0,0],  
    [1,0,1,0,0,0,0,0,0],  
    [0,1,0,0,1,0,0,0,0],  
    [0,0,0,0,0,0,1,0,0],  
    [0,1,0,0,0,0,0,1,0],  
    [0,0,1,0,0,0,0,0,0],  
    [0,0,0,1,0,0,0,1,0],  
    [0,0,0,0,1,0,1,0,0],  
    [0,0,0,0,0,0,0,1,0]])
```

- future reward의 gamma변수와 learning rate의 alpha값 설정

```
In [6]: #Initialize parameters  
gamma = 0.75 # Discount factor  
alpha = 0.9 # Learning rate
```

- 최적의 경로 탐색 : Q_learning 이용

```

In [7]: def get_optimal_route(start_location,end_location):
    rewards_new = np.copy(rewards)
    ending_state = location_to_state[end_location]
    rewards_new[ending_state,ending_state] = 999

    Q = np.array(np.zeros([9,9]))

    # Q-Learning process
    for i in range(1000):
        # Picking up a random state
        current_state = np.random.randint(0,9)
        # Python excludes the upper bound
        playable_actions = []
        # Iterating through the new rewards matrix
        for j in range(9):
            if rewards_new[current_state,j] > 0:
                playable_actions.append(j)
        # Pick a random action that will lead us to next state
        next_state = np.random.choice(playable_actions)
        # Computing Temporal Difference
        TD = rewards_new[current_state,next_state] + gamma * Q[next_state, np.argmax(Q[next_state,])] - Q[current_state,next_state]
        # Updating the Q-Value using the Bellman equation
        Q[current_state,next_state] += alpha * TD

    # Initialize the optimal route with the starting location
    route = [start_location]
    #Initialize next_location with starting location
    next_location = start_location

    # We don't know about the exact number of iterations
    #needed to reach to the final location hence while loop
    #will be a good choice for iterating
    while(next_location != end_location):
        # Fetch the starting state
        starting_state = location_to_state[start_location]
        # Fetch the highest Q-value pertaining to starting state
        next_state = np.argmax(Q[starting_state,:])
        # We got the index of the next state.
        #But we need the corresponding letter.
        next_location = state_to_location[next_state]
        route.append(next_location)

        # Fetch the highest Q-value pertaining to starting state
        next_state = np.argmax(Q[starting_state,:])
        # We got the index of the next state.
        #But we need the corresponding letter.
        next_location = state_to_location[next_state]
        route.append(next_location)
        # Update the starting location for the next iteration
        start_location = next_location

    print(Q)

    return route

```

- 결과 출력

```

In [8]: print(get_optimal_route('L9','L1'))

[[3995.97495013 2249.47521991  0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   2997.98107847  0.          1688.10210338  0.          0.          0.
   0.          0.          0.          0.          0.          0.
   [ 0.          2249.47521217  0.          0.          0.          0.
   1267.07350775  0.          0.          0.          0.          0.
   [ 0.          0.          0.          0.          0.          0.
   0.          951.30167295  0.          0.          0.          0.
   [ 0.          2249.46829682  0.          0.          0.          0.
   0.          0.          1267.06889749  0.          0.          0.
   [ 0.          0.          1688.10597856  0.          0.          0.
   0.          0.          0.          0.          0.          0.
   [ 0.          0.          0.          714.47625467  0.          0.
   0.          0.          1267.06889727  0.          0.          0.
   [ 0.          0.          0.          0.          0.          1688.09186332
   0.          951.30161632  0.          951.30125504]
   [ 0.          0.          0.          0.          0.          0.
   0.          0.          1267.06889749  0.          0.          0.
   ['L9', 'L8', 'L5', 'L2', 'L1']]

```

2. 문제에 대한 답안

Q2-1. Explain how to implement Q-learning process and how to find optimal routes using the final Q-table

이 코드는 다음 문제 상황에 대한 Q-learning process with TD(Temporal Difference)를 구현하고 있습니다. L1~L9까지 factory가 존재하며 robot이 이동하는데 주어진 출발점과 도착점에 대해 최적의 최단 경로를 학습하는 Q-learning을 구현 해야합니다.

따라서 그림의 위치를 숫자를 이용해서 dictionary형태로 mapping을 시켜주었습니다.

이후 위치를 이동하는 action을 1차원 리스트로, action에 대한 reward table은 numpy의 np.array()를 이용해 구현해주었습니다. 이후 딕셔너리를 이용하여 인덱스를 활용할 수 있도록 state_to_location이라는 딕셔너리에 mapping 시켜주어 학습 environment, action, reward에 대한 setting을 마치었습니다.

```
location_to_state = {  
    'L1' : 0,  
    'L2' : 1,  
    'L3' : 2,  
    'L4' : 3,  
    'L5' : 4,  
    'L6' : 5,  
    'L7' : 6,  
    'L8' : 7,  
    'L9' : 8  
}  
  
#Maps indices to locations  
state_to_location = dict((state,location) for location,state in  
    location_to_state.items())  
print(state_to_location)
```

```
import numpy as np  
  
actions = [0,1,2,3,4,5,6,7,8]  
  
#Define the rewards  
rewards = np.array([  
    [0,1,0,0,0,0,0,0,0],  
    [1,0,1,0,1,0,0,0,0],  
    [0,1,0,0,0,1,0,0,0],  
    [0,0,0,0,0,1,0,0,0],  
    [0,1,0,0,0,0,1,0,0],  
    [0,0,1,0,0,0,0,0,0],  
    [0,0,0,1,0,0,0,1,0],  
    [0,0,0,0,1,0,1,0,1],  
    [0,0,0,0,0,0,1,0,0]])
```

<L1~L9 딕셔너리에 숫자로 맵핑한 모습> <Action과 Reward Matrix 리스트, numpy를 이용해 구현>

이후 gamma, alpha 변수를 이용해서 gamma는 future reward에 대한 discount factor(상쇄)값을 그리고 alpha를 이용해 learning rate(상쇄 값)을 지정해주었습니다. 이 코드에서는 각각 값을 0.75와 0.9로 할당하였습니다.

이후 get_optimal_route(start_location, end_location)함수를 이용해서 학습 및 결과를 얻어내도록 하였습니다. get_optimal_route함수는 시작 위치와 도착할 곳에 대한 값을 start_location과 end_location이라는 인자로 받습니다.

이후 rewards_new 변수에 기준에 선언해 두었던 reward matrix를 복사해 가져오고, 최종 목적지를 location_to_state딕셔너리와 인자로 입력 받은 end_location을 통해서 지정해줍니다. 또 시작점과 도착점에 대한 reward value를 999 혹은 값을 크게 reward를 주어 학습할 때 해당하는 값에 도달할 경우 가장 많은 reward를 줄 수 있도록 입력받은 인자와 rewards_new matrix를 이용해 설정해줍니다. 그리고 학습을 통해 Update할 Q table을 numpy의 np.array를 이용하여 0으로 초기화 해줍니다.

```
In [6]: #Initialize parameters
gamma = 0.75 # Discount factor
alpha = 0.9 # Learning rate
```

<gamma, alpha 변수 지정>

```
def get_optimal_route(start_location,end_location):
    rewards_new = np.copy(rewards)
    ending_state = location_to_state[end_location]
    rewards_new[ending_state,ending_state] = 999
    Q = np.array(np.zeros([9,9]))
```

<Q table 0으로 초기화 및 기타 세팅>

그리고 for와 range를 이용해서 1000번의 학습을 진행합니다. 학습을 진행할 때 로봇은 랜덤하게 초기값을 지정 받습니다. np.random.randint()를 이용해 해당 루프가 돌 때마다 current_state에 위치를 지정하는 것이죠. 그리고 reward_new매트릭스를 통해 현재에서 이동하여 갈 수 있는 모든 지점에 대한 reward값을 playable_actions에 append하여 저장해둡니다. 이후 next_state는 np.random.choice(playable_action)함수를 이용하여 다음이동동 공간을 랜덤하게 이동하게 됩니다. 그리고 아래의 해당하는 움직임에 대한 TD값을 아래와 같은 함수를 구현한 식을 이용해서 구해주고, Q table에 학습률 알파를 곱해 값을 저장하는 방식으로 학습을 하게 됩니다. 이후 이러한 방식을 앞서 언급했던 것처럼 1000번을 반복하는 방식으로 학습하여 마무리하게 됩니다.

```
for i in range(1000):
    # Picking up a random state
    current_state = np.random.randint(0,9)
    # Python excludes the upper bound
    playable_actions = []
    # Iterating through the new rewards matrix
    for j in range(9):
        if rewards_new[current_state,j] > 0:
            playable_actions.append(j)
    # Pick a random action that will lead us to next state
    next_state = np.random.choice(playable_actions)
    # Computing Temporal Difference
    TD = rewards_new[current_state,next_state] +
        gamma * Q[next_state, np.argmax(Q[next_state,:])] -
        Q[current_state,next_state]
    # Updating the Q-Value using the Bellman equation
    Q[current_state,next_state] += alpha * TD
```

<학습에 사용된 식 및 Q-learning 학습과정>

학습을 마무리하고 난 뒤, 업데이트된 Q table을 이용해서 주어진 시작점과 도착점에 대한 최적의 최단 거리를 찾는 방법을 진행하게 됩니다.

route=[start_location]을 이용하여 시작지점으로 초기화하고 next_location도 현재 다음 위치를 알지 못하므로 시작점과 동일하게 초기화를 해줍니다.

이후 while문을 돌며 next_location이 end_location과 같을 때, 즉 최적의 경로를 찾을 때까지 루프문을 반복합니다. 여기서는 starting_state를 fetch해준 다음, np.argmax를 이용해서 starting state에서 최종적으로 업데이트된 Q table을 확인하여 현재 위치에서 이동했을 때 reward값이 최대가 되는 위치로 이동하는 위치를 next_state에 저장해줍니다. 그리고 이 인덱스를 이용해서 해당하는 위치를 찾아준다음 route경로에 append를 하여 추가해줍니다. 이러한 방식으로 최종 도착 지점에 이를 때까지 루프문을 반복하고 해당하는 route경로를 append해주고 while문이 종료되면 최적의 경로 route값을 반환하고 함수가 마무리 됩니다.

```

# Initialize the optimal route with the starting location
route = [start_location]
#Initialize next_location with starting location
next_location = start_location

# We don't know about the exact number of iterations
#needed to reach to the final location hence while loop
#will be a good choice for iteratiing
while(next_location != end_location):
    # Fetch the starting state
    starting_state = location_to_state[start_location]
    # Fetch the highest Q-value pertaining to starting state
    next_state = np.argmax(Q[starting_state,:])
    # We got the index of the next state.
    #But we need the corresponding letter.
    next_location = state_to_location[next_state]
    route.append(next_location)
    # Update the starting location for the next iteration
    start_location = next_location

print(Q)

return route

```

<최적의 경로를 찾는 과정>

Q2-2. When starting location = L7, ending location = L1, what is the resulting Q-table (given that gamma = 0.75 (Discounter factor), alpha = 0.9 (Learning rate)) after finishing Q- learning process? In addition, what is a corresponding optimal route?

<Reward 행렬이 파일로 프로젝트PPT설명처럼 주어 졌을 때 Result Q table>

	L1	L2	L3	L4	L5	L6	L7	L8	L9
L1	3995.99089	2249.4993153	0	0	0	0	0	0	0
L2	2997.9908945	0	1688.1243413	0	1688.1243883	0	0	0	0
L3	0	2249.49912177	0	0	0	1267.0932599	0	0	0
L4	0	0	0	0	0	0	951.31992607	0	0
L5	0	2249.49931689	0	0	0	0	0	1267.09323421	0
L6	0	0	1688.12434133	0	0	0	0	0	0
L7	0	0	0	714,48994434	0	0	0	1267.09334006	0
L8	0	0	0	0	1688.12446902	0	951.31988271	0	951.31985278
L9	0	0	0	0	0	0	0	1267.09335059	0

<코드 결과>

```
[ [3995.999089 2249.49931513 0. 0. 0.
  0. 0. 0. 0. 0.
  [2997.99908945 0. 1688.1243413 0. 1688.12433883
  0. 0. 0. 0. 0.
  [ 0. 2249.49912177 0. 0. 0.
  1267.09325599 0. 0. 0. 0.
  [ 0. 0. 0. 0. 0.
  0. 951.31992607 0. 0. 0.
  [ 0. 2249.49931689 0. 0. 0.
  0. 0. 1267.09323421 0. 0.
  [ 0. 0. 1688.12434133 0. 0.
  0. 0. 0. 0. 0.
  [ 0. 0. 0. 714.48994434 0.
  0. 0. 1267.09334006 0. 0.
  [ 0. 0. 0. 0. 1688.12446902
  0. 951.31988271 0. 951.31985278]
  [ 0. 0. 0. 0. 0.
  0. 0. 1267.09335059 0. 0.
  ]] ]]
```

<Reward 행렬이 코드로 구현된 그대로 시행하였을 때 Result Q table>

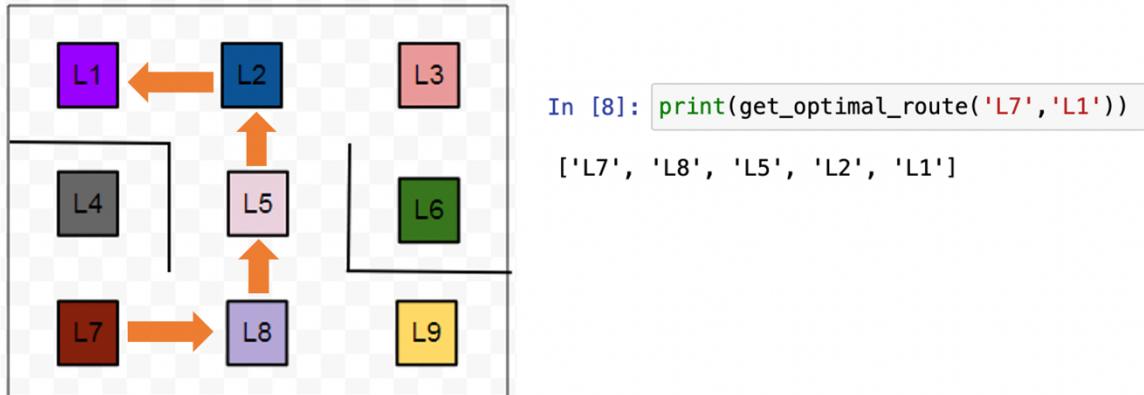
	L1	L2	L3	L4	L5	L6	L7	L8	L9
L1	3995.99908856	2249.4990942	0	0	0	0	0	0	0
L2	2997.99926598	0	1688.12431956	0	0	0	0	0	0
L3	0	2249.4991075	0	0	0	1267.09240356	0	0	0
L4	0	0	0	0	0	0	951.31930665	0	0
L5	0	2249.49910749	0	0	0	0	0	1267.0924129	0
L6	0	0	1688.12430952	0	0	0	0	0	0
L7	0	0	0	714.48840002	0	0	0	1267.09241291	0
L8	0	0	0	0	1688.12321772	0	951.31811759	0	951.3193095
L9	0	0	0	0	0	0	0	1267.09241328	0

<코드 결과>

```
[ [3995.99908856 2249.4990942 0. 0. 0.
  0. 0. 0. 0. 0.
  [2997.99926598 0. 1688.12431956 0. 0.
  0. 0. 0. 0. 0.
  [ 0. 2249.4991075 0. 0. 0.
  1267.09240356 0. 0. 0. 0.
  [ 0. 0. 0. 0. 0.
  0. 951.31930665 0. 0. 0.
  [ 0. 2249.49910749 0. 0. 0.
  0. 0. 1267.0924129 0. 0.
  [ 0. 0. 1688.12430952 0. 0. 0.
  0. 0. 0. 0. 0.
  [ 0. 0. 0. 714.48840002 0.
  0. 0. 1267.09241291 0. 0.
  [ 0. 0. 0. 0. 0.
  0. 951.31811759 0. 951.3193095 0.
  [ 0. 0. 0. 0. 0.
  0. 1267.09241328 0. 0. 0.
  ]] ]]
```

```
['L7', 'L8', 'L5', 'L2', 'L1']
```

<최적의 경로(최단 경로) : L7 – L8 – L5 – L2 – L1>



* 파일 프로젝트 PPT에 주어진 문제 설명 슬라이드의 Reward 행렬과 코드에 구현된 Reward 행렬에 다음 표에 해당하는 체크 해놓은 부분이 다른 것을 체크할 수 있습니다. 따라서 슬라이드 초기 문제 설명에 주어진 Reward 행렬 한번, 과제에 주어진 코드대로 한 번씩 수행해보며 최적의 경로를 체크해보았습니다. 결과는 위와 같이 Q-table에서 다른 값이 존재하지만, 경로는 둘다 올바르게 출력되는 것을 확인할 수 있었습니다.

	L1	L2	L3	L4	L5	L6	L7	L8	L9
L1	0	1	0	0	0	0	0	0	0
L2	1	0	1	0	1	0	0	0	0
L3	0	1	0	0	0	1	0	0	0
L4	0	0	0	0	0	0	1	0	0
L5	0	1	0	0	0	0	0	1	0
L6	0	0	1	0	0	0	0	0	0
L7	0	0	0	1	0	0	0	1	0
L8	0	0	0	0	1	0	1	0	1
L9	0	0	0	0	0	0	0	1	0

```
#Define the rewards
rewards = np.array([[0,1,0,0,0,0,0,0,0,0],
[1,0,1,0,0,0,0,0,0,0],
[0,1,0,0,0,1,0,0,0,0],
[0,0,0,0,0,0,1,0,0,0],
[0,1,0,0,0,0,0,1,0,0],
[0,0,1,0,0,0,0,0,0,0],
[0,0,0,1,0,0,0,0,1,0],
[0,0,0,0,1,0,0,0,1,0],
[0,0,0,0,0,1,0,1,0,0],
[0,0,0,0,0,0,0,1,0,0]])
```

<딥러닝 기말 프로젝트-Q learning 슬라이드 6에 있는 정의된 Reward> <딥러닝 기말 프로젝트-Q learning 슬라이드 6에 있는 정의된 Reward>

문제에서 정의된 로봇은 L2->L5으로 이동할 수 있으나 코드상에서는 해당 움직임에 대한 Reward를 0으로 정의되어 있지만, “딥러닝 기말 프로젝트-Q learning 슬라이드6”에 정의된 Reward는 1로 정의되어 있음을 확인하여 원본 그대로 코드를 돌려보고, 슬라이드에 맞게 수정하여 코드를 돌려보았습니다. 둘 다 동일한 최적의 경로가 출력되었습니다. L5->L2의 Reward값은 동일하여 같은 경로가 나오는 것으로 보입니다.

Q2-3. Explain the policy used during the Q-learning (random policy? or optimal policy?)

Reinforcement Learning에서는 크게 두 가지 학습방법이 있습니다. Behavior policy와 Target Policy가 일치하는 on Policy 학습방법과 Behavior Policy와 Target Policy가 일치하지 않는 Off policy로 나뉩니다. 이 때 **Q-Learning**은 두 Policy 중 **Off Policy** 알고리즘을 이용해 학습을 진행합니다. TD를 이용하는 Q-Learning은 **Target policy**는 **Greedy Policy**를 사용하며, **Behavior Policy**는 **ϵ -Greedy Policy**를 이용합니다.

$$Q(S_t, A_t) = Q(S_t, A_t) +$$

$$Q(S_t, A_t) = (1-\alpha)Q(S_t, A_t) +$$

Current Action(By Behavior Policy)
: ϵ -Greedy Algorithm

$$\alpha(R_{t+1} + \gamma \max Q(S'_{t+1}, A'))$$

$$\alpha(R_{t+1} + \gamma \max Q(S'_{t+1}, A')) - Q(S_t, A_t)$$

Next Action(By Target Policy)
: Greedy Algorithm

조금 더 상세하게 설명하자면 Target Policy는 위식처럼 Q-value TD target이 항상 최적화 하는 방식으로 학습이 됩니다. 항상 immediate reward와 future reward가 최대가 되도록 학습을 하게 됩니다. 반면에 상태를 전이 시켜주는 Action에 대한 Policy를 담당하는 Behavior Policy는 ϵ -Greedy Policy를 따르게 됩니다. 예를 들어, Action을 취할 때 만약 오른쪽으로 이동하는 것이 현재 policy 상 가장 optimal한 방법이라도, 갈 수 있는 다른 방향(왼쪽 혹은 오른쪽)으로 이동하는 exploration을 할 수 있습니다. 즉, 최적의 방법이 존재하더라도 몇 번의 학습 케이스에서는 다른 선택을 하여 움직여 exploration을 해보는 방식의 알고리즘을 이용해 학습을 하는 것입니다.

따라서 ϵ -Greedy Policy는 이처럼 대부분 케이스에서는 Optimal Action(policy)를 통해서 이동을 하지만, 일부 케이스에서는 Random Action(policy)를 따라서 상태를 이동하며 학습을 합니다. 예를 들면 90퍼센트는 Optimal하게 이동하고, 10퍼센트는 Random하게 이동하는 Behavior Policy를 이용하는 것입니다.

이번에 implementation한 코드에서도 random하게 action을 할 수 있도록 구현되어 있는 것을 확인 할 수 있습니다.

```
# Pick a random action that will lead us to next state
next_state = np.random.choice(playable_actions)
```

<Code에서 next_state를 이동할 때 random.choice를
이용해 다음 state를 random하게 이동하도록
implementation이 된 것을 확인할 수 있습니다.>

Q2-4. Explain how to implement Q-value update using temporal difference in the code file

Q-value는 numpy를 이용해서 Q table을 초기화 한 다음, 다음 취할 Action에 대한 Q-value를 업데이트하면서 구현해주었습니다.

Q-value는 현재의 Q-value에 TD(Temporal difference)값을 더해주면서 Q-value값을 업데이트 해줍니다. TD값은 Bellman Equation과 Markov Decision Process를 통해 유도한 공식을 이용하여 공식을 통해서 값을 구해주었습니다. 이 때 확률 값은 제외하고 "Action에 대한 immediate reward" + "new_state에서 얻을 수 있는 최대 reward의 값(future reward)*gamma" - "현재 Q value"를 이용해서 값을 구해주었습니다. 그리고 해당 방식을 1000번의 loop문을 통해 Q table의 값을 업데이트 하여 최종적인 Q table을 구해주었습니다. 식은 아래에서 확인 하실 수 있습니다.

```

for i in range(1000):
    # Picking up a random state
    current_state = np.random.randint(0,9)
    # Python excludes the upper bound
    playable_actions = []
    # Iterating through the new rewards matrix
    for j in range(9):
        if rewards_new[current_state,j] > 0:
            playable_actions.append(j)
    # Pick a random action that will lead us to next state
    next_state = np.random.choice(playable_actions)
    # Computing Temporal Difference
    TD = [rewards_new[current_state,next_state] +
    gamma * Q[next_state, np.argmax(Q[next_state,:])] -
    Q[current_state,next_state]]
    # Updating the Q-Value using the Bellman equation
    Q[current_state,next_state] += alpha * TD

```

The diagram illustrates the calculation of Temporal Difference (TD) and its application in the Bellman equation for Q-value update. It shows the following components:

- TD Calculation:** $TD = R_{t+1} + \gamma \max Q(S'_{t+1}, A') - Q(S_t, A_t)$
- Bellman Equation:** $Q(S_t, A_t) = Q(S_t, A_t) + \alpha TD$
- Code Annotations:**
 - TD:** Points to the term $R_{t+1} + \gamma \max Q(S'_{t+1}, A')$ in the TD formula.
 - Q(S_t, A_t)**: Points to the term $Q(S_t, A_t)$ in the TD formula.
 - Q(S_t, A_t)**: Points to the term $Q(S_t, A_t)$ in the Bellman equation.
 - TD**: Points to the term αTD in the Bellman equation.
 - Q[Q[current_state,next_state]]**: Points to the term $Q[current_state,next_state]$ in the TD calculation code.
 - Q[Q[current_state,next_state]]**: Points to the term $Q[current_state,next_state]$ in the Bellman equation code.
 - Q[current_state,next_state] += alpha * TD**: Points to the final update line in the code.

<학습에 사용된 식 및 Q-learning 학습과정>

이후 최종적으로 업데이트된(학습이된) Q-table을 이용하여 시작위치에 대한 도착위치의 최적의 경로를 현재 위치에서 갈 수 있는 최적의 다음 위치를 정하는 방법을 반복함으로서 route를 구해 주었습니다.