FrozenLake Q-learning

이은후

Dummy Q-Learning

Q-Table에서 0으로 전부 초기화 시켜둔 Q값들은 학습을 거치면서 업데이트됨.
Goal 지점으로 넘어갈 때에만 reward=1이 주어짐

각 칸의 Q(s,a) = reward + max Q(s',a')

0 S	1	2	3
4	5 H	6	7 H
8	9	10	11 H
12 H	13	14	15 G

1) 처음에는 agent가 어디로 가든 전부 0으로 초기화 되어있는 상태이므로, reward=0, max Q(s',a')도 0임. agent가 random하게 선택하여 가다가, 우연히 s(14)에 도달하였다고 가정
→ Goal지점(s(15))로 넘어갈 때 reward=1을 받으니까 Q(14,right)=1로 업데이트

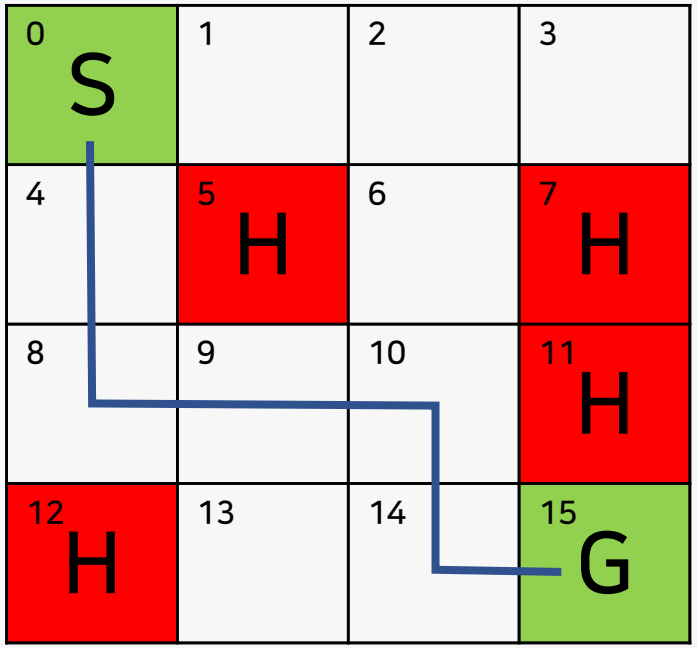
2) Q(14,right) 제외하고 나머지 Q값은 여전히 0. agent가 또다시 random하게 이동하다가 우연히 s(13)에 도달하였다고 가정
→ reward=0이지만, max Q(s',a')는 Q(14,right)덕분에 1이 됨. 따라서 Q(13,right)=1로 업데이트

Dummy Q-Learning

```
# The Q-Table learning algorithm
while not done:
    action = qmax_action(Q[state, :])

    # Get new state and reward from environment
    new_state, reward, done, _ = env.step(action)

    # Update Q-Table with new knowledge using learning rate
    Q[state, action] = reward + np.max(Q[new_state, :])

    rAll += reward
    state = new_state
```

qmax_action함수: 취할 수 있는 4개의 action(상하좌우) 중 가장 큰 값 선택하는 함수. 동일하면 ranom하게 선택

Dummy Q-Learning



	LEFT	DOWN	RIGHT	UP
	0	1	2	3
0	0	1	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	1	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	1	0
9	0	0	1	0
10	0	1	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	1	0
14	0	0	1	0
15	0	0	0	0

정확도: 약 95%

```
In [25]: runfile('C:/Users
learning/dummy_q-learning.
프/강화학습/Q-learning')
Success rate: 0.9525
```

Dummy Q-Learning

- 문제점

가장 optimal 한 경로를 따라 Q 값이 업데이트 되지 않을 수 있는 가능성이 있음.

가끔은 최적의 Q로 이동하는 action 이 아닌, 랜덤한 action을 취해줌.
(Exploitation & Exploration 적용)

기법1) ϵ -Greedy
기법2) add Random noise

1) ϵ-Greedy

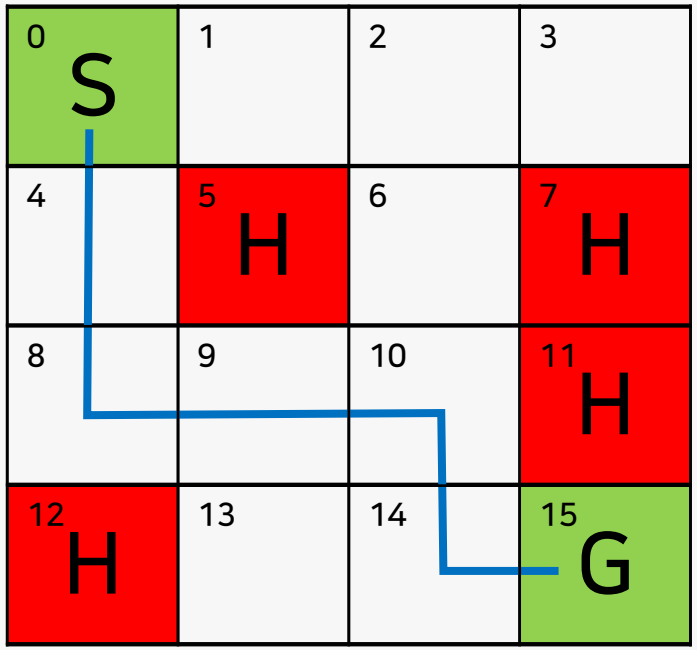
: 랜덤한 확률로 가끔은 최적의 Q값을 따라가지 않도록 설정하는 방법

```
# Q learning 알고리즘
while not done :

    # E-Greedy 알고리즘으로 action 고르기
    if np.random.rand(1) < e :
        action = env.action_space.sample()
    else :
        action = np.argmax(Q[state, :])

    # 해당 Action을 했을 때 environment가 변하고, 새로운 state, reward, done 여부를 반환 받음
    new_state, reward, done, _ = env.step(action)

    # Q = R + Q
    #Q[state, action] = reward + dis * np.max(Q[new_state, :])
    Q[state,action]= Q[state,action]+lr*(reward+dis* np.max(Q[new_state,:])-Q[state,action]


    #env.render()
    rAll += reward
    state = new_state

rList.append(rAll)
```

[설정값]

- discount factor: 0.99
- learning rate: 0.1
- num_episodes: 2000
- ϵ: 1./((i / 100) + 1)

1) ϵ-Greedy



	LEFT	DOWN	RIGHT	UP
	0	1	2	3
0	0.937628	0.95099	0.908781	0.940388
1	0.938075	0	0.0385017	0.167742
2	0.296429	0	0	0
3	0.0230631	0	0	0
4	0.936052	0.960596	0	0.937939
5	0	0	0	0
6	0	0.96452	0	0.0254452
7	0	0	0	0
8	0.954918	0	0.970299	0.948253
9	0.952964	0.967536	0.9801	0
10	0.967226	0.99	0	0.889377
11	0	0	0	0
12	0	0	0	0
13	0	0.266373	0.989816	0.261288
14	0.946416	0.981277	1	0.974534
15	0	0	0	0

정확도: 약 82%

```
In [4]: runfile('C:/Users/Eunhoo/
Desktop/졸프/강화학습/Q-learning/
greedy.py', wdir='C:/Users/Eunhoo/
Desktop/졸프/강화학습/Q-learning')
Success rate : 0.8245
```

2) Random noise

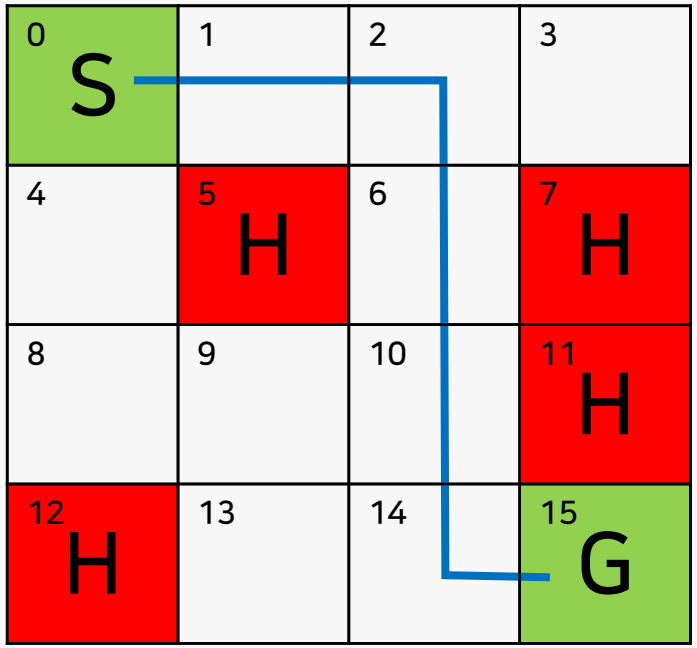
: action을 결정할 때 참고하는 각 Q값에 랜덤한 noise를 주어서 조금 더 랜덤한 action을 선택하게 하는 방법

```
# The Q-Table learning algorithm
while not done:
    # Choose an action by greedily (with noise) picking from Q table
    action = np.argmax(Q[state,:] + np.random.randn(1,env.action_space.n) / (i+1))

    # Get new state and reward from environment
    new_state, reward, done,_ = env.step(action)

    # Get negative reward every step
    if reward == 0 :
        reward=-0.001

    # Q-Learning
    Q[state,action]= Q[state,action]+lr*(reward+y* np.max(Q[new_state,:])-Q[state,action])
    state = new_state
    rAll += reward

rList.append(rAll)
```

[설정값]

- discount factor: 0.99
- learning rate: 0.1
- num_episodes: 2000

2) Random noise



	LEFT	DOWN	RIGHT	UP
	0	1	2	3
0	-0.00137118	-0.00181724	0.946089	-0.00218951
1	-0.00253175	-0.000997781	0.956656	-0.00134422
2	-0.00131321	0.967329	-0.00121858	-0.000777122
3	-0.000688325	-0.000651322	-0.000917253	-0.000861017
4	-0.00181808	0.00136072	-0.000995362	-0.00265816
5	0	0	0	0
6	-0.00040951	0.97811	-0.000771232	-0.000798322
7	0	0	0	0
8	-0.000977637	-0.000814698	0.0240252	-0.00145664
9	-0.000518643	-0.000478459	0.159208	-0.00040951
10	0	0.989	-0.00019	-0.000374264
11	0	0	0	0
12	0	0	0	0
13	-0.00019	-0.0001	-0.0001	-0.000297829
14	0	0	1	0
15	0	0	0	0

정확도: 약 91%

```
In [8]: runfile('C:/Users/Eunhoo/De
random.py', wdir='C:/Users/Eunhoo/D
learning')
Success rate : 0.9133529999999581
```

(참고) 문제해결

파일을 재실행 시킬 때마다
cannot re-register id frozenlake-v3
라는 오류가 발생, 프로그램 자체를 restart해야 정상 작동하는 문제가 발생함.

```
env_dict = gym.envs.registration.registry.env_specs.copy()
for env in env_dict:
    if 'FrozenLake' in env:
        del gym.envs.registration.registry.env_specs[env]
```

기존 세션에 할당되어 있던 환경을 제거하는 코드를 앞부분에 추가하니,
굳이 프로그램을 restart시키지 않더라도 잘 돌아감 !!

Q - network

지금은 4*4 grid world로 Q Table을 작성하였지만,

실전 문제에 적용하려면 엄청난 크기의 state가 발생하며, 그만큼 시간복잡도도 굉장히 늘어남

Q를 Neural Network로 대체하여 해결 (Q-network)

Q - network

[변수설명]

X: input으로 줄 내용인 변수.(Placeholder)

W: 지속적인 업데이트가 필요한 Network의 Weight에 해당하는 변수.(Variable), 16*4크기

Qpred: 추측값. 1*4크기

Y: target값. (즉 4개의 action에 대한 reward값) 1*4크기

손실함수(loss func): 예측값과 타겟값의 차이. (AdamOptimizer 사용)

[설정값]

학습률(learning_rate): 0.1

감가율(dis): 0.99

에피소드(num_episodes): 2000

Q - network

```
with tf.compat.v1.Session() as sess:
    sess.run(init)
    for i in range(num_episodes):
        # Reset environment and get first new observation
        s = env.reset()
        e = 1. / ((i / 50) + 10)
        rAll = 0
        done = False
        local_loss = []

        # The Q-Network training
        while not done:
            # Choose an action by greedily (with e chance of random action)
            # from the Q-network
            Qs = sess.run(Qpred, feed_dict={X: one_hot(s)})
            if np.random.rand(1) < e:
                a = env.action_space.sample()
            else:
                a = np.argmax(Qs)

            # Get new state and reward from environment
            s1, reward, done, _ = env.step(a)
            if done:
                # Update Q, and no Qs+1, since it's a terminal state
                Qs[0, a] = reward
            else:
                # Obtain the Q_s1 values by feeding the new state through our
                # network
                Qs1 = sess.run(Qpred, feed_dict={X: one_hot(s1)})
                # Update Q
                Qs[0, a] = reward + dis * np.max(Qs1)

            # Train our network using target (Y) and predicted Q (Qpred) values
            sess.run(train, feed_dict={X: one_hot(s), Y: Qs})

            rAll += reward
            s = s1
        rList.append(rAll)
```

Qs ← network의 추측값이 저장됨 (입력은 one-hot인코딩 통함)

action은 e-greedy방식으로 결정됨

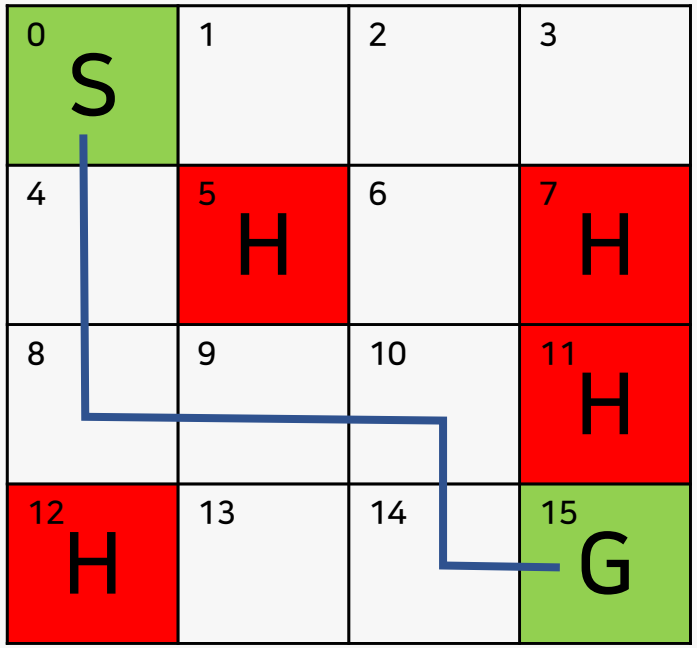
if (Hole/Goal에 도달): Qs는 reward값만으로 업데이트됨

else: 다음 state(s1)에 대한 Qpred를 수행하여 타겟으로 사용할 Qs 얻음

(Qs[0,a]에는 Qpred를 평가한 값이 1*4로 나옴)

X(이전 state)에 대한 Y(타겟, 즉 정답)을 Qs가 가지고 있으므로,
이를 이용해 Network를 훈련시켜 Qpred값과 비교해 Weight를 업데이트시킴

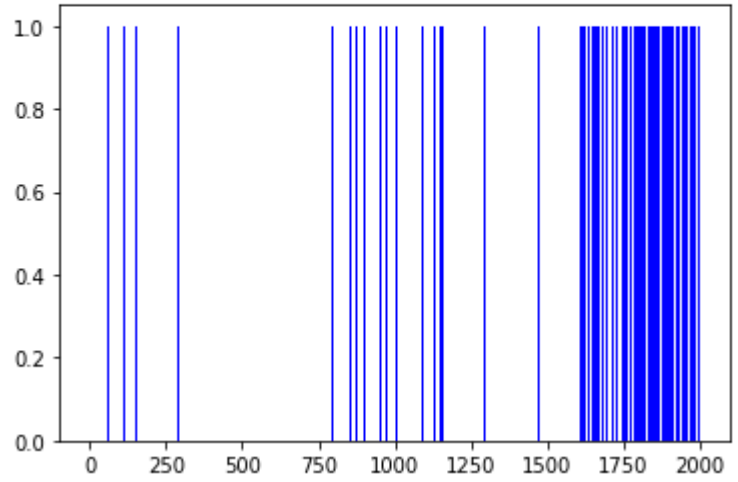
Q - network



	LEFT	DOWN	RIGHT	UP
	0	1	2	3
0	0.940333	0.95099	0.913786	0.939816
1	0.938415	0	0.0379599	0.11563
2	0.270388	0	0	0
3	0.000746674	0	0	0
4	0.948543	0.960596	0	0.938781
5	0	0	0	0
6	0	0.967049	0	0.00481742
7	0	0	0	0
8	0.957742	0	0.970299	0.946447
9	0.950729	0.972956	0.9801	0
10	0.96357	0.99	0	0.90555
11	0	0	0	0
12	0	0	0	0
13	0	0.530911	0.989966	0.361859
14	0.970327	0.989116	1	0.978476
15	0	0	0	0

정확도: 약 27%

```
In [27]: runfile('C:/Users/Eunhoo/Deskt
Users/Eunhoo/Desktop/졸프/강화학습/Q-lear
Percent of successful episodes: 0.273
```



Q - network

- 정확도가 낮은 이유?

Optimal policy로 수렴하지 않기 때문.

1) Network가 얕음
2) 입력 dataset간의 연관성이 너무 큼 → 제대로 된 업데이트X

DQN으로 해결할 수 있음

1) weight와 Hidden layer(은닉층) 추가해서 해결
2) 데이터를 무작위 방법으로 뽑아 dataset간의 높은 연관성을 피함