01_Q_learning-Improve

October 11, 2019

0.1 ### Install Package

[1]:

```
import numpy as np
import gym
import random
```

0.2 ### Créer la env

- Here we'll create the FrozenLake environment.
- OpenAI Gym is a library composed of many environments that we can use to train our agents.
- In our case we choose to use Frozen Lake.

[5]:

```
from gym.envs.registration import register
register(
    id="FrozenLakeNotSlippery-v0",
    entry_point = 'gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name': '4x4','is_slippery':False},
    max_episode_steps=100,
    reward_threshold=0.8196, #optimum = 0.8196
)

env = gym.make("FrozenLakeNotSlippery-v0")
```

0.3 ### Créer la Q-Table

- Now, we'll create our Q-table, to know how much rows (states) and columns (actions) we need, we need to calculate the action_size and the state_size
- OpenAI Gym provides us a way to do that: env.action_space.n and env.observation_space.n

[6]:

```
# C'est cols
action_size = env.action_space.n

# C'est rows
state_size = env.observation_space.n

# Créer la Q-table:
```

```
qtable = np.zeros((state_size, action_size))
print(qtable)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

0.4 ### Créer la parameters:

[7]:

```
total_episodes = 20000        # Total episodes
learning_rate = 0.8           # Learning rate
max_steps = 99                # Max steps per episode
gamma = 0.95                  # Discounting rate

# Exploration parameters
epsilon = 1.0                 # Exploration rate
max_epsilon = 1.0             # Exploration probability at start
min_epsilon = 0.01            # Minimum exploration probability
decay_rate = 0.001             # Exponential decay rate for exploration prob
```

0.5 ### Créer la code de simulation:

[10]:

```
# List of rewards
rewards = []

# 2 Loop tout episodes:
for episode in range(total_episodes):
    # Reset the environment
    state = env.reset()
    step = 0
    game_over = False
    total_rewards = 0

    for step in range(max_steps):
        # On faire le random-number
        exploration_exploitation_flag = random.uniform(0,1)

        # Si la flag >  epsilon, on faire la exploitation:
        # Prendre la gros value pour cette state.
        if exploration_exploitation_flag > epsilon:
            action = np.argmax(qtable[state,:])
        # Si la flag <  epsilon, on faire la exploration:
        # Prendre la random-action
        else:
            action = env.action_space.sample()

        # Prendre la action, obtenir la prochain state (s), obetenir la reward (r)
        new_state , reward, done, info = env.step(action)

        # Update Q(s,a) = Q(s,a) + lr [ R(s,a) + gamma * max Q(s',a') - Q(s,a) ]
        qtable[state,action] = qtable[state, action] + learning_rate *(reward + gamma * np.max(qtable[new_state,:])-qtable[state,action])

        # [total_reward]: Mise à jour
        total_rewards += reward
        state = new_state

        # Si game_over, on arrete:
        if game_over == True:
            break

    # réduire epsilon. (on a besoin de moin de epsilon, apres beaucoup de epsiodes)
    epsilon = min_epsilon + (max_epsilon - min_epsilon)* np.exp(-decay_rate*episode)
    rewards.append(total_rewards)

print("Score average over time: " + str(sum(rewards)/total_episodes))
print(qtable)
```

```
Score average over time: 0.93455
[[0.73509189 0.77378094 0.77378094 0.73509189]
 [0.73509189 0.         0.81450625 0.77378094]
 [0.77378094 0.857375   0.77378094 0.81450625]
 [0.81450625 0.         0.77378094 0.77378097]
 [0.77378094 0.81450625 0.         0.73509189]
 [0.         0.         0.         0.        ]
 [0.         0.9025     0.         0.81450625]
 [0.         0.         0.         0.        ]
 [0.81450625 0.         0.857375   0.77378094]
 [0.81450625 0.9025     0.9025     0.        ]
 [0.857375   0.95       0.         0.857375  ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.9025     0.95       0.857375  ]
 [0.9025     0.95       1.         0.9025    ]
 [0.         0.         0.         0.        ]]
```

[11]:

```python
# Afficher la action:
# gauche: 0, bas: 1, droit: 2, haut: 3

env.reset()
env.render()
print(np.argmax(qtable,axis=1).reshape(4,4))
```

```
SFFF
FHFH
FFFH
HFFG
[[1 2 1 0]
 [1 0 1 0]
 [2 1 1 0]
 [0 2 2 0]]
```

[12]:

```python
env.reset()
max_steps = 99
for episode in range(5):
    state     = env.reset()
    step      = 0
    game_over = False
    msg  = "--------------------------------------------------\n"
    msg += "Dans la episonde [%d]\n"%episode
    print(msg)

    for step in range(max_steps):

        action = np.argmax(qtable[state,:])

        new_state, reward, game_over, info = env.step(action)


        if game_over:
            env.render()
            print("Number of steps ",step)
            print(info)
            break

        state = new_state


print("C'est fini....")
env.close()
```

```
--------------------------------------------------
Dans la episonde [0]

  (Right)
SFFF
FHFH
FFFH
HFFG
Number of steps  5
{'prob': 1.0}
--------------------------------------------------
Dans la episonde [1]

  (Right)
SFFF
FHFH
FFFH
HFFG
Number of steps  5
{'prob': 1.0}
--------------------------------------------------
Dans la episonde [2]

  (Right)
SFFF
FHFH
FFFH
HFFG
Number of steps  5
{'prob': 1.0}
--------------------------------------------------
Dans la episonde [3]

  (Right)
SFFF
FHFH
FFFH
HFFG
```

```
Number of steps  5
{'prob': 1.0}
---------------------------------------------------
Dans la episonde [4]

  (Right)
SFFF
FHFH
FFFH
HFFG
Number of steps  5
{'prob': 1.0}
C'est fini...
```

0.6 ### JA_Test

[27]:
```
qtable
```

[27]:
```
array([[0.73509189, 0.77378094, 0.77378094, 0.73509189],
       [0.73509189, 0.        , 0.81450625, 0.77378094],
       [0.77378094, 0.857375  , 0.77378094, 0.81450625],
       [0.81450625, 0.        , 0.77378094, 0.77378097],
       [0.77378094, 0.81450625, 0.        , 0.73509189],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.9025    , 0.        , 0.81450625],
       [0.        , 0.        , 0.        , 0.        ],
       [0.81450625, 0.        , 0.857375  , 0.77378094],
       [0.81450625, 0.9025    , 0.9025    , 0.        ],
       [0.857375  , 0.95      , 0.        , 0.857375  ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.9025    , 0.95      , 0.857375  ],
       [0.9025    , 0.95      , 1.        , 0.9025    ],
       [0.        , 0.        , 0.        , 0.        ]])
```

[29]:
```
qtable.shape
```

[29]:
```
(16, 4)
```

[26]:
```
cest_quoi = np.argmax(qtable,axis=1)
cest_quoi
```

[26]:
```
array([1, 2, 1, 0, 1, 0, 1, 0, 2, 1, 1, 0, 0, 2, 2, 0])
```

[16]:
```
row_beaucoup = cest_quoi.reshape(8,2)
row_beaucoup
```

```
[16]: array([[1, 2],
              [1, 0],
              [1, 0],
              [1, 0],
              [2, 1],
              [1, 0],
              [0, 2],
              [2, 0]])
```

```
[19]: row_beaucoup[7][0]
```

```
[19]: 2
```

```
[17]: rows_size = int(10) # states
      cols_size = int(4)   # actions

      check_my_np = np.zeros((rows_size,cols_size))
      check_my_np
```

```
[17]: array([[0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.]])
```

```
[24]: check_my_np[9][3]
```

```
[24]: 0.0
```

0.7 #### np.argmax Chercher la max action dans la Q-Table:

```
[8]: state_size  = 10
     action_size = 4     # (haut, bas, gauche, droite)
     qtable      = np.zeros((state_size,action_size))
     qtable
```

```
[8]: array([[0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.]])
```

[9]:

```
# update state[1]
for col in range(4):
    qtable[1,col] = col
qtable
```

[9]:

```
array([[0., 0., 0., 0.],
       [0., 1., 2., 3.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[17]:

```
qtable[9,:] = 55
qtable[2][0]=3.1
qtable[2][1]=2.7
qtable[2][2]=3.3
qtable[2][3]=0.9
qtable[3][:] = 2.1
qtable
```

[17]:

```
array([[ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  1. ,  2. ,  3. ],
       [ 3.1,  2.7,  3.3,  0.9],
       [ 2.1,  2.1,  2.1,  2.1],
       [ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ],
       [55. , 55. , 55. , 55. ]])
```

[18]:

```
state = 1
get_max_col = np.argmax(qtable[1,:])
get_max_col
```

[18]:

```
3
```

[19]:
```
state = 2
get_max_col = np.argmax(qtable[state,:])
get_max_col
```

[19]:
```
2
```

[20]:
```
qtable
```

[20]:
```
array([[ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  1. ,  2. ,  3. ],
       [ 3.1,  2.7,  3.3,  0.9],
       [ 2.1,  2.1,  2.1,  2.1],
       [ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ],
       [55. , 55. , 55. , 55. ]])
```

[21]:
```
qtable[2,3]   # Q(state,action)
```

[21]:
```
0.9
```

[22]:
```
qtable[3,:]   #Q (new_stat, all)
```

[22]:
```
array([2.1, 2.1, 2.1, 2.1])
```

[24]:
```
what_is_this = qtable[3,:] - qtable[2,3]
what_is_this
```

[24]:
```
array([1.2, 1.2, 1.2, 1.2])
```

[ ]:
```

```

[ ]:
```

```