Lab 3: Dummy Q-learning (table)

Reinforcement Learning with TensorFlow&OpenAI Gym
Sung Kim <hunkim+ml@gmail.com>
Learning $Q(s, a)$: Table

Initial $Q$ values are 0

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Learning $Q(s, a)$ Table (with many trials)
initial $Q$ values are 0
Learning $Q(s, a)$ Table: one success!

initial $Q$ values are 0

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Learning $Q(s, a)$ Table: one success!

$$\pi^*(s) = \operatorname*{argmax}_a Q(s, a)$$
Dummy Q-learning algorithm

For each \( s, a \) initialize table entry \( \hat{Q}(s, a) \leftarrow 0 \)

Observe current state \( s \)

Do forever:

- Select an action \( a \) and execute it
- Receive immediate reward \( r \)
- Observe the new state \( s' \)
- Update the table entry for \( \hat{Q}(s, a) \) as follows:
  \[
  \hat{Q}(s, a) \leftarrow r + \max_{a'} \hat{Q}(s', a')
  \]
- \( s \leftarrow s' \)

Dummy Q-learning algorithm

For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state $s$

Do forever:

- Select an action $a$ and execute it
- Receive immediate reward $r$
- Observe the new state $s'$
- Update the table entry for $\hat{Q}(s, a)$ as follows:
  \[
  \hat{Q}(s, a) \leftarrow r + \max_{a'} \hat{Q}(s', a')
  \]
- $s \leftarrow s'$

```
# Initialize table with all zeros
Q = np.zeros([env.observation_space.n, env.action_space.n])
# Set learning parameters
num_episodes = 2000

# create lists to contain total rewards and steps per episode
rList = []
for i in range(num_episodes):
    # Reset environment and get first new observation
    state = env.reset()
    rAll = 0
    done = False
    while not done:
        action = rargmax(Q[state, :])
        # Get new state and reward from environment
        new_state, reward, done, _ = env.step(action)
        newQ = max(Q[new_state, :])
        # Update Q-Table with new knowledge using learning rate
        Q[state, action] = reward + np.max(Q[new_state, :])
    rList.append(rAll)

```
import gym
import numpy as np
import matplotlib.pyplot as plt
from gym.envs.registration import register
import random as pr

def rargmax(vector):  # https://gist.github.com/stober/1943451
    """Argmax that chooses randomly among eligible maximum indices. """
    m = np.amax(vector)
    indices = np.nonzero(vector == m)[0]
    return pr.choice(indices)

register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name': '4x4',
            'is_slippery': False}
)

env = gym.make('FrozenLake-v3')

Code: (dummy) Q-learning

```
# Initialize table with all zeros
Q = np.zeros([env.observation_space.n, env.action_space.n])

# Set learning parameters
num_episodes = 2000

# create lists to contain total rewards and steps per episode
rList = []
for i in range(num_episodes):
    # Reset environment and get first new observation
    state = env.reset()
    rAll = 0
    done = False

    # The Q-Table learning algorithm
    while not done:
        action = np.argmax(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

    rList.append(rAll)
```

Code: result reporting

```python
print("Success rate: " + str(sum(rList)/num_episodes))
print("Final Q-Table Values")
print("LEFT DOWN RIGHT UP")
print(Q)
plt.bar(range(len(rList)), rList, color="blue")
plt.show()
```

Success rate: 0.95
```python
Q = np.zeros([env.observation_space.n, env.action_space.n])

print(Q)
```

```
LEFT DOWN RIGHT UP
[[ 0.  0.  1.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  0.]]
```
Next

Exploit & exploration and
discounted future reward