Lab 4: Q-learning (table)

exploit&exploration and discounted future reward

Reinforcement Learning with TensorFlow&OpenAI Gym

Sung Kim <hunkim+ml@gmail.com>

Exploit VS Exploration: decaying E-greedy

for i in range (1000)

    e = 0.1 / (i+1)

    if random(1) < e:

        a = random

    else:

        a = argmax(Q(s, a))

```
for i in range(num_episodes):

e = 1. / ((i / 100)+1)  # Python2

# The Q-Table learning algorithm
while not done:
    # Choose an action by e_greedy
    if np.random.rand(1) < e:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q[state, :])
```

Exploit VS Exploration: add random noise

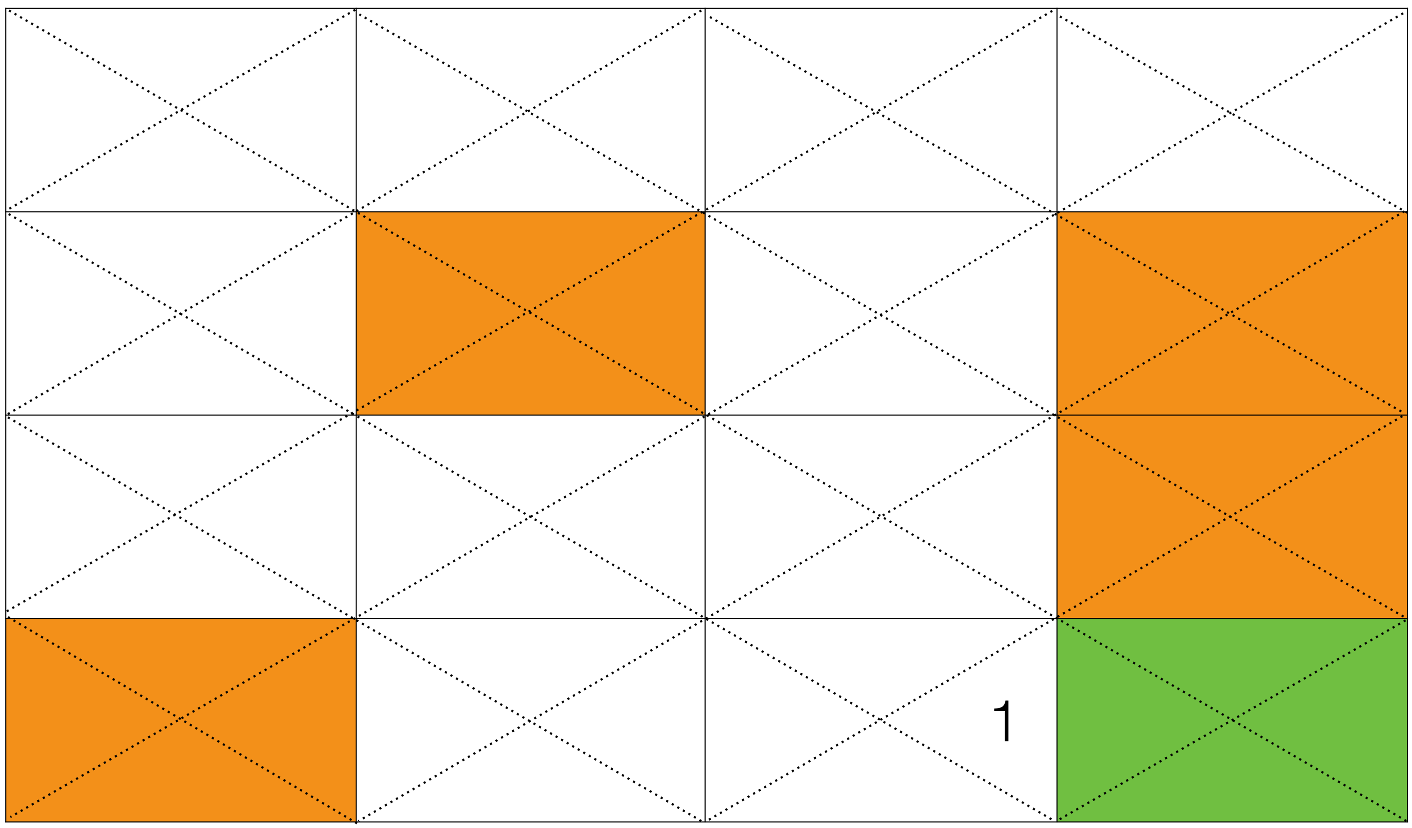
for i in range (1000)

a = argmax(Q(s, a) + random_values / (i+1))

```
# 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))
```

0.5 0.6 0.3 0.2 0.5

Discounted reward γ = 0.9)



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:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

- $s \leftarrow s'$

```
# Discount factor
dis = .99
```

0.9

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

Machine Learning, T. Mitchell, McGraw Hill, 1997

Code: setup

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

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

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

# Initialize table with all zeros
Q = np.zeros([env.observation_space.n,env.action_space.n])
# Discount factor
dis = .99
num_episodes = 2000

# create lists to contain total rewards and steps per episode
rList = []
```

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0#.pjz9g59ap

Code: Q learning

```
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:
        # 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)

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

        rAll += reward
        state = new_state

    rList.append(rAll)
```

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0#.pjz9g59ap

Code: results

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

```
Success rate: 0.9635
Final Q-Table Values
[[ 0.          0.          0.95099005  0.        ]
 [ 0.          0.          0.96059601  0.        ]
 [ 0.          0.970299    0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.9801      0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.970299    0.        ]
 [ 0.          0.9801      0.          0.        ]
 [ 0.          0.99        0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.99        0.        ]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          0.        ]]
```

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0#.pjz9g59ap

Code: e-greedy

//

```
e = 1. / ((i // 100)+1)

# The Q-Table learning algorithm
while not done:
    # Choose an action by e greedy
    if np.random.rand(1) < e:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q[state, :])
```

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0#.pjz9g59ap

Code: e-greedy results

```
Success rate: 0.828
Final Q-Table Values
[[ 0.94148015  0.95099005  0.95099005  0.94148015]
 [ 0.94148015  0.          0.96059601  0.95099005]
 [ 0.95099005  0.970299    0.          0.96059601]
 [ 0.96059601  0.          0.          0.        ]
 [ 0.95099005  0.96059601  0.          0.94148015]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.9801      0.          0.96059601]
 [ 0.          0.          0.          0.        ]
 [ 0.96059601  0.          0.970299    0.95099005]
 [ 0.96059601  0.9801      0.9801      0.        ]
 [ 0.970299    0.99        0.          0.970299  ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.9801      0.99        0.970299  ]
 [ 0.9801      0.99        1.          0.9801    ]
 [ 0.          0.          0.          0.        ]]
```

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0#.pjz9g59ap

Next

Nondeterministic/
Stochastic worlds