Lab 5: Windy Frozen Lake
Nondeterministic world!

Reinforcement Learning with TensorFlow & OpenAI Gym
Sung Kim <hunkim+ml@gmail.com>
Deterministic

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

env = gym.make('FrozenLake-v3')
```
Stochastic (non-deterministic)

```python
env = gym.make('FrozenLake-v0')
```

```python
# is_slippery True

('State: ', 0, 'Action: ', 2,
'SFFF
FHFH
FFFH
HFFG
    (Right)
('State: ', 0, 'Action: ', 2,
'SFFF
FHFH
FFFH
HFFG
    (Right)
('State: ', 4, 'Action: ', 2,
'SFFF
FHFH
FFFH
HFFG
    (Right)
('State: ', 5, 'Action: ', 1,
('Finished with reward', 0.0)
```
Q-learning algorithm for deterministic

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:

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

- $s \leftarrow s'$

---

Our previous Q-learning does not work

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

env = gym.make('FrozenLake-v0')

Score over time: 0.0165
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:
  \[
  Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]
  \]
- \( s \leftarrow s' \)
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:
  
  $$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

- $s \leftarrow s'$

# Update Q-Table with new knowledge using learning rate

$$Q[\text{state, action}] = (1-\text{learning\_rate}) \times Q[\text{state, action}] \ \& \ \text{learning\_rate} \times (\text{reward} + \text{dis} \times \text{np\_max}(Q[\text{new\_state, :}]))$$
Code: Setup

```python
import gym
import numpy as np
import matplotlib.pyplot as plt

env = gym.make('FrozenLake-v0')

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

# Set learning parameters
learning_rate = .85
dis = .99
num_episodes = 2000
```
Code: Q-learning

```python
# 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:
        # 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 learning rate
        Q[state, action] = (1- learning_rate) * Q[state, action] \
        + learning_rate*(reward + dis * np.max(Q[new_state, :]))

        rAll += reward
        state = new_state

    rList.append(rAll)
```
Code: Report results

```python
print("Score over time: ", str(sum(rList)/num_episodes))
print("Final Q-Table Values")
print(Q)
plt.bar(range(len(rList)), rList, color="blue")
plt.show()
```

Score over time: 0.653
Next
(Deep) Q-Network with TensorFlow!