homework2

December 14, 2020

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
from collections import defaultdict

from tictactoe import TicTacToe
from utils import plot_board, get_and_print_move, plot_test_game
```

[2]:
```
def running_mean(x, N):
    cumsum = np.cumsum(np.insert(x, 0, 0))
    return (cumsum[N:] - cumsum[:-N]) / float(N)
```

[3]:
```
class Policy:
    def __init__(self):
        self.Q = defaultdict(lambda: defaultdict(lambda: 0))
    def getActionGreedy(self, s, na):
        if s in self.Q  and len(self.Q[s]) > 0:
            return max(self.Q[s], key=self.Q[s].get)
        else:
            return np.random.randint(na)
    def getActionEpsilonGreedy(self, s, na, epsilon):
        if np.random.rand() > epsilon:
            return self.getActionGreedy(s, na)
        else:
            return np.random.randint(na)
    def getMaxActionScore(self, s):
        if (s not in self.Q) or (len(self.Q[s]) == 0):
            return 0
        return max(self.Q[s].values())
```

[4]:
```
def play_test_game(env, policy_x, policy_o):
    done = False
    env.reset()

    state, avail_steps = env.getHash(), env.getEmptySpaces()

    while not done:
        if env.curTurn == 1:
```

```
            action_idx = policy_x.getActionGreedy(state, len(avail_steps))
            action = avail_steps[action_idx]
        else:
            action_idx = policy_o.getActionGreedy(state, len(avail_steps))
            action = avail_steps[action_idx]

        (state, avail_steps, _), reward, done, _ = env.step(action)

    return reward
```

[5]:
```
def q_learning(env, policy_x, policy_o, alpha, epsilon):
    env.reset()
    done = False
    state, actions  = env.getHash(), env.getEmptySpaces()

    while not done:
        if env.curTurn == 1:
            state_x = state
            action_idx_x = policy_x.getActionEpsilonGreedy(state, len(actions),␣
 ↪epsilon)
            action = actions[action_idx_x]
            (state, actions, _), reward, done, _ = env.step(action)
            policy_x.Q[state_x][action_idx_x] = policy_x.
 ↪Q[state_x][action_idx_x] + alpha*( reward - policy_x.
 ↪Q[state_x][action_idx_x])
            policy_o.Q[state_o][action_idx_o] = policy_o.
 ↪Q[state_o][action_idx_o] + alpha*(-reward + max(policy_x.Q[state].values())
        else:
            state_o = state
            action_idx_o = policy_o.getActionEpsilonGreedy(state, len(actions),␣
 ↪epsilon)
            action = actions[action_idx_o]
            (state, actions, _), reward, done, _ = env.step(action)
            policy_o.Q[state_o][action_idx_o] = policy_o.
 ↪Q[state_o][action_idx_o] + alpha * (-reward - policy_o.
 ↪Q[state_o][action_idx_o])
            policy_x.Q[state_x][action_idx_x] = policy_x.
 ↪Q[state_x][action_idx_x] + alpha * ( reward + max(policy_x.Q[state].values())
```

[6]:
```
env_3_3 = TicTacToe(n_rows=3, n_cols=3, n_win=3)
env_4_4 = TicTacToe(n_rows=4, n_cols=4, n_win=4)
env_5_5 = TicTacToe(n_rows=5, n_cols=5, n_win=5)
```

[7]:
```
policy_x = Policy()
policy_o = Policy()
policy_random = Policy()
```

[8]:
```
episodes = 400_000
x_hist = []
o_hist = []

for i in tqdm(range(1, episodes +1)):
    q_learning(env_3_3, policy_x, policy_o, alpha=0.05, epsilon=0.4)
    if episodes % 500 == 0:
        x_hist.append(play_test_game(env_3_3, policy_x, policy_random) > 0)
        o_hist.append(play_test_game(env_3_3, policy_random, policy_o) < 0)
x_hist = running_mean(x_hist, 1000)
o_hist = running_mean(o_hist, 1000)
```

```
100%|      | 400000/400000 [14:03<00:00, 474.20it/s]
```

[9]:
```
plt.figure(figsize=(12, 8))
plt.plot(x_hist, label ='X vs Random')
plt.plot(o_hist, label ='O vs Random')
plt.legend()
plt.show()
```

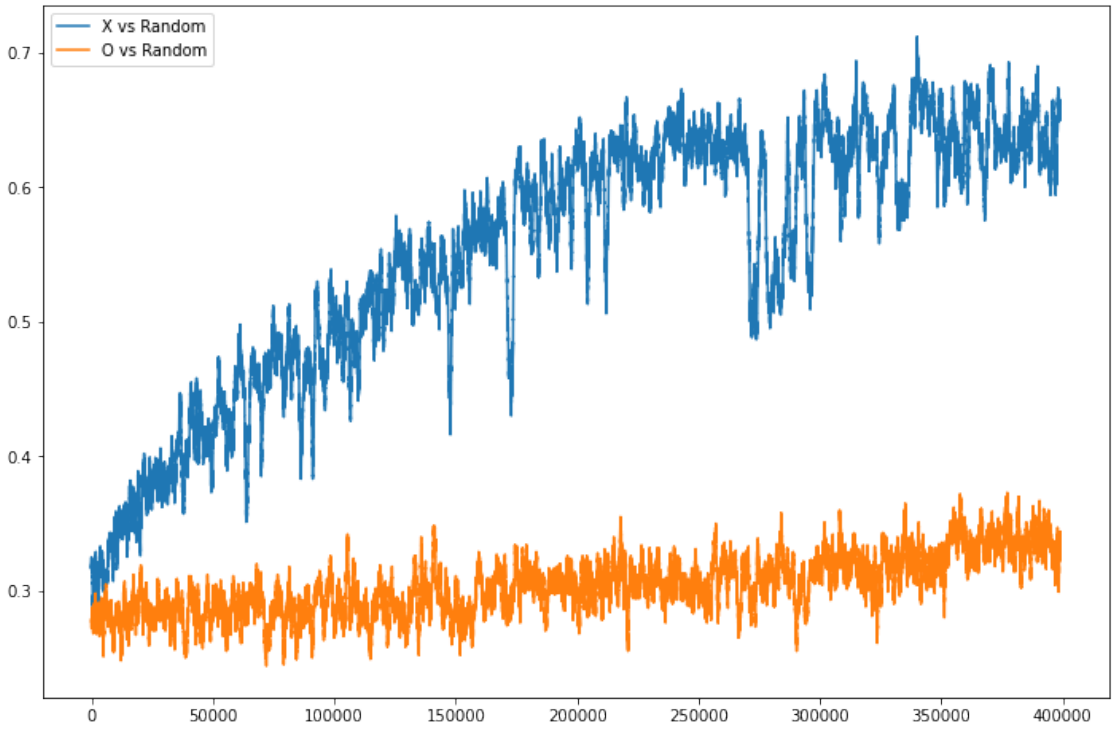
[10]:
```
policy_x = Policy()
policy_o = Policy()
policy_random = Policy()
```

[11]:
```
episodes = 400_000
x_hist = []
o_hist = []

for i in tqdm(range(1, episodes +1)):
    q_learning(env_4_4, policy_x, policy_o, alpha=0.05, epsilon=0.4)
    if episodes % 500 == 0:
        x_hist.append(play_test_game(env_4_4, policy_x, policy_random) > 0)
        o_hist.append(play_test_game(env_4_4, policy_random, policy_o) < 0)
x_hist = running_mean(x_hist, 1000)
o_hist = running_mean(o_hist, 1000)
```

```
100%|      | 400000/400000 [35:47<00:00, 186.22it/s]
```

[12]:
```
plt.figure(figsize=(12, 8))
plt.plot(x_hist, label ='X vs Random')
plt.plot(o_hist, label ='O vs Random')
plt.legend()
plt.show()
```



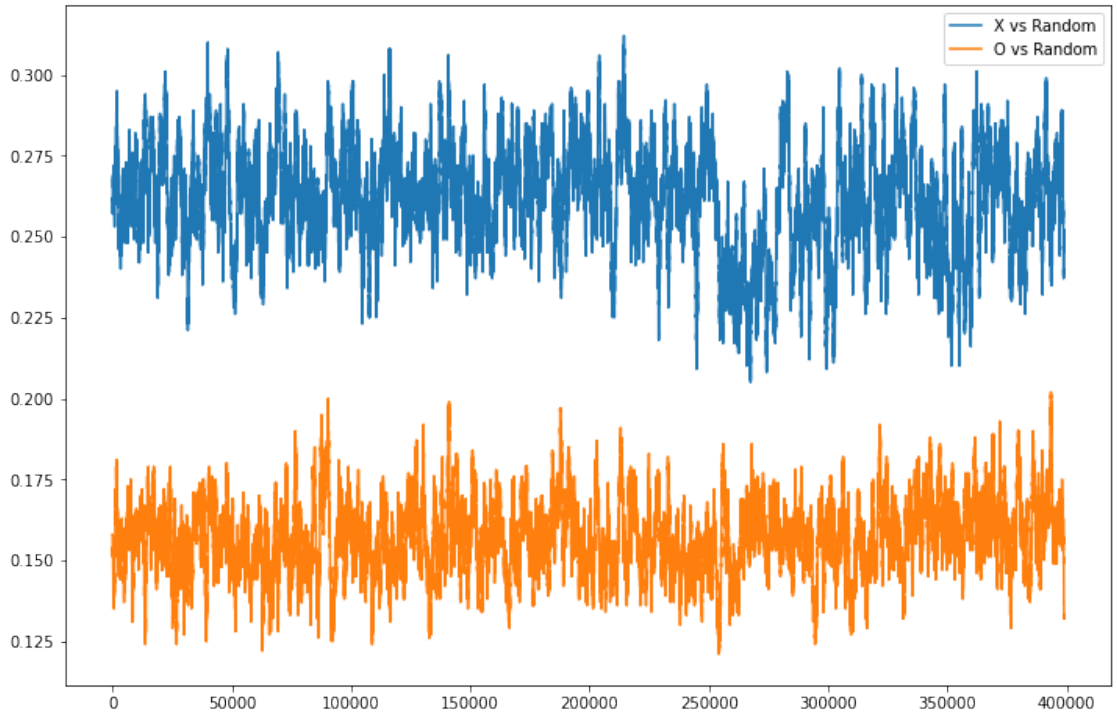
[13]:
```
policy_x = Policy()
policy_o = Policy()
policy_random = Policy()
```

[14]:
```
episodes = 400_000
x_hist = []
o_hist = []

for i in tqdm(range(1, episodes +1)):
    q_learning(env_5_5, policy_x, policy_o, alpha=0.05, epsilon=0.4)
    if episodes % 500 == 0:
        x_hist.append(play_test_game(env_5_5, policy_x, policy_random) > 0)
        o_hist.append(play_test_game(env_5_5, policy_random, policy_o) < 0)
x_hist = running_mean(x_hist, 1000)
o_hist = running_mean(o_hist, 1000)
```

```
100%|      | 400000/400000 [1:14:07<00:00, 89.94it/s]
```

[15]:
```
plt.figure(figsize=(12, 8))
plt.plot(x_hist, label ='X vs Random')
plt.plot(o_hist, label ='O vs Random')
plt.legend()
plt.show()
```



[16]:
```
episodes = 50_000
x_hist = []
o_hist = []
```

```
for i in tqdm(range(1, episodes +1)):
    q_learning(env_5_5, policy_x, policy_o, alpha=0.05, epsilon=0.4)
    if episodes % 500 == 0:
        x_hist.append(play_test_game(env_5_5, policy_x, policy_random) > 0)
        o_hist.append(play_test_game(env_5_5, policy_random, policy_o) < 0)
x_hist = running_mean(x_hist, 1000)
o_hist = running_mean(o_hist, 1000)
```

```
100%|      | 50000/50000 [10:17<00:00, 80.94it/s]
```

[17]:
```
plt.figure(figsize=(12, 8))
plt.plot(x_hist, label ='X vs Random')
plt.plot(o_hist, label ='O vs Random')
plt.legend()
plt.show()
```