Aug 22, 2022

Udacity Banana Project Report

YoonChae Na

1. Introduction

For this project, you will train an agent to navigate (and collect bananas!) in a large, square world. A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:

0 - move forward. **1** - move backward. **2** - turn left. **3** - turn right.

The task is episodic, and in order to solve the environment, your agent must get an average score of +13 over 100 consecutive episodes.

2. Examine the State and Action Spaces

```
# reset the environment
env_info = env.reset(train_mode=True)[brain_name]

# number of agents in the environment
print('Number of agents:', len(env_info.agents))

# number of actions
action_size = brain.vector_action_space_size
print('Number of actions:', action_size)

# examine the state space
state = env_info.vector_observations[0]
print('States look like:', state)
state_size = len(state)
print('States have length:', state_size)
```

```
Number of agents: 1
Number of actions: 4
States look like: [ 1.          0.          0.          0.          0.84408134  0.          0.
  1.          0.          0.0748472   0.          1.          0.          0.
  0.25755     1.          0.          0.          0.          0.74177343
  0.          1.          0.          0.          0.25854847  0.          0.
  1.          0.          0.09355672  0.          1.          0.          0.
  0.31969345  0.          0.        ]
States have length: 37
```

3. Take Random Actions in the Environment

```
env_info = env.reset(train_mode=True)[brain_name] # reset the environment
state = env_info.vector_observations[0]            # get the current state
score = 0                                          # initialize the score
while True:
    action = np.random.randint(action_size)        # select an action
    env_info = env.step(action)[brain_name]        # send the action to the environment
    next_state = env_info.vector_observations[0]   # get the next state
    reward = env_info.rewards[0]                   # get the reward
    done = env_info.local_done[0]                  # see if episode has finished
    score += reward                                # update the score
    state = next_state                             # roll over the state to next time step
    if done:                                       # exit loop if episode finished
        break

print("Score: {}".format(score))
```

```
Score: 3.0
```

4. Q-network Parameter

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

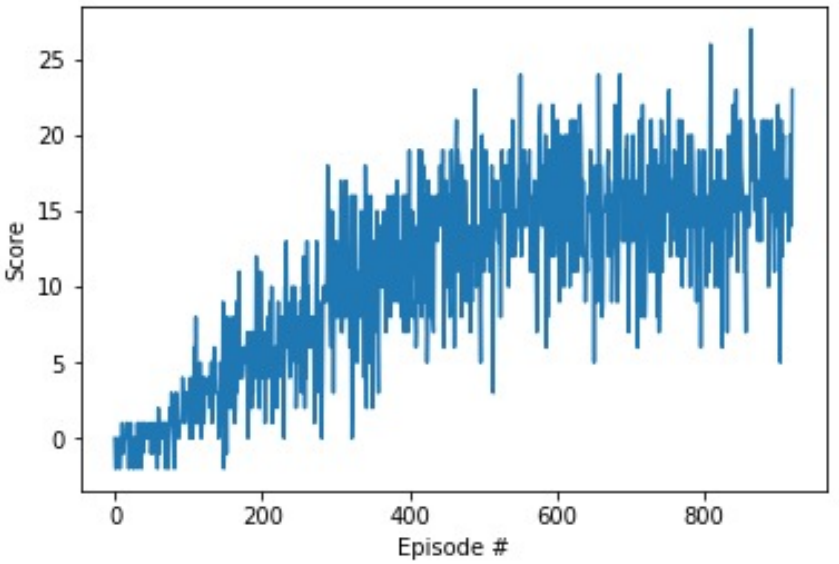
```
P_seed=0
P_model=QNetwork(state_size,action_size,P_seed)
print(P_model)
```

```
QNetwork(
  (fc1): Linear(in_features=37, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=4, bias=True)
)
```

5.Result

```
Episode 100     Average Score: 0.30
Episode 200     Average Score: 3.79
Episode 300     Average Score: 7.07
Episode 400     Average Score: 10.79
Episode 500     Average Score: 13.02
Episode 600     Average Score: 15.14
Episode 700     Average Score: 14.91
Episode 800     Average Score: 15.39
Episode 900     Average Score: 15.82

Environment solved in 821 episodes!     Average Score: 16.11
```



Appendix. Success agent Weight

OrderedDict([('fc1.weight', tensor([[-1.0245e-01, 9.0063e-03, -1.9052e-01, ..., -2.8144e-02, -5.6943e-02, -5.9848e-03], [1.2959e-01, 2.2108e-01, -4.8796e-02, ..., -1.1046e-01, 6.8787e-02, 3.7634e-02], [6.6845e-02, 1.7236e-01, 3.4386e-02, ..., -2.1106e-01, -2.5309e-01, -6.6552e-02], ..., [4.2899e-02, 8.7614e-02, 8.7021e-02, ..., 1.4176e-01, -9.0148e-03, 1.5390e-03], [-1.5071e-01, -1.6737e-02, -9.9000e-03, ..., -3.4237e-01, -1.5824e-01, -3.8230e-02], [-4.2095e-02, -4.8843e-01, 4.1845e-02, ..., 4.9399e-02, -2.8405e-02, -5.2707e-02]], device='cuda:0')), ('fc1.bias', tensor([0.0920, -0.0282, -0.1990, 0.0444, 0.2565, -0.0254, -0.2069, 0.0105, 0.0973, -0.2148, 0.3044, 0.2506, 0.0609, 0.1595, -0.0720, -0.1674, -0.0267, -0.0390, -0.0112, -0.2420, -0.0032, 0.0922, 0.0526, -0.1007, -0.0589, 0.1142, 0.0744, -0.2503, 0.1493, 0.1157, -0.0549, -0.0861, 0.2642, -0.0218, -0.0984, 0.0961, 0.0132, 0.0250, -0.1139, -0.2001, -0.1057, 0.0587, -0.0288, 0.1145, -0.0537, -0.0535, 0.0982, -0.0242, -0.1471, -0.0979, -0.1804, -0.0138, 0.0894, -0.0555, -0.1997, -0.0180, 0.1305, -0.1405, -0.1824, -0.1820, -0.0445, 0.0356, -0.0072, 0.0226], device='cuda:0')), ('fc2.weight', tensor([[0.0316, 0.0497, 0.0242, ..., 0.5882, 0.1210, 0.2496], [-0.0663, 0.1255, -0.2569, ..., 0.0059, -0.2011, -0.1763], [-0.0200, -0.1314, 0.0403, ..., 0.3366, -0.0246, -0.1796], ..., [0.0051, 0.0043, -0.1004, ..., 0.0524, 0.0258, 0.0761], [-0.1507, -0.1041, -0.6102, ..., 0.3159, -0.2024, -0.0448], [-0.3464, -0.3371, 0.1851, ..., -0.1240, 0.1484, 0.4227]], device='cuda:0')), ('fc2.bias', tensor([0.0167, 0.3042, 0.4013, -0.0538, 0.1697, -0.2588, -0.1187, -0.0408, 0.0599, 0.5651, -0.0345, 0.2785, 0.2086, -0.1520, -0.0915, 0.2336, 0.0222, 0.0592, -0.0004, 0.2111, 0.1200, 0.5608, -0.1385, 0.0942, -0.1632, -0.2290, 0.0025, 0.0207, -0.0338, 0.4769, -0.0515, 0.1240, -0.1189, -0.0159, 0.0649, -0.0360, -0.0729, -0.2146, 0.5396, -0.0442, -0.1261, 0.6433, 0.0968, 0.0595, 0.3720, -0.1779, 0.0830, -0.1852, -0.0866, -0.1932, -0.2419, 0.2122, -0.1102, -0.0541, 0.2547, -0.0519, -0.2803, -0.1597, -0.0031, -0.0612, 0.0305, 0.4050, 0.1381, -0.1076], device='cuda:0')), ('fc3.weight', tensor([[0.5002, 0.2099, 0.3344, -0.4774, 0.0766, -0.0910, -0.1732, -0.0182, 0.3106, 0.1748, -0.0087, 0.2184, 0.1314, -0.2438, -0.0289, 0.1220, -0.6086, 0.3433, -0.3332, 0.2894, 0.4468, 0.1581, -0.1924, 0.1199, -0.2649, -0.2144, -0.3233, 0.4124, 0.0772, 0.1550, -0.1506, 0.1265, 0.2002, 0.0735, 0.0798, 0.0967, 0.2773, 0.1231, 0.1908, -0.0236, -0.1235, 0.1634, 0.3458, 0.2542, 0.1348, -0.2982, 0.1424, -0.1869, -0.3312, 0.1401, -0.0616, 0.0620, -0.2853, 0.3920, 0.1781, 0.2379, -0.2728, -0.1131, 0.3663, -0.0462, -0.1684, 0.1523, -0.0271, -0.0381], [0.4665, 0.1601, 0.2180, 0.1377, -0.0151, -0.1595, -0.2033, -0.3028, 0.3098, 0.1903, 0.0101, 0.1757, 0.0865, -0.1643, 0.0092, 0.1108, 0.2188, 0.2171, -0.1791, 0.1489, 0.4834, 0.2327, -0.2075, 0.2156, 0.0180, -0.0709, -0.0666, 0.4670, -0.3974, 0.1771, -0.0828, 0.1073, -0.0285, -0.5913, 0.1660, 0.2053, 0.4133, -0.0096, 0.1750, 0.2299, 0.3986, 0.2218, -0.1776, 0.3588, 0.0296, -0.3817, 0.1969, -0.1966, -0.1270, 0.2236, -0.1634, 0.1838, -0.0308, 0.3617, 0.0995, 0.0099, -0.2779, -0.2674, 0.3011, -0.4538, -0.0374, 0.0731, -0.3596, 0.1927], [0.3020, 0.1474, 0.2096, -0.2617, 0.2212, -0.1409, -0.1091, -0.1997, 0.2406, 0.1333, -0.1100, 0.2688, 0.1412, -0.5688, -0.0606, 0.2152, -0.1740, -0.6739, -0.2129, 0.0229, 0.3329, 0.2907, -0.5646, 0.1731, 0.0412, -0.1917, 0.1194, 0.4278, 0.2469, 0.2236, -0.0953, -0.0192, -0.6010, -0.1452, 0.3878, 0.5641, 0.1094, 0.0565, 0.1609, 0.0196, 0.0372, 0.2335, 0.3736, 0.0638, 0.0760, -0.3599, 0.1680, 0.1922, 0.1586, 0.1115, -0.2074, 0.2804, 0.2595, 0.8118, 0.1141, 0.3093, -0.0926, -0.1119, -0.1769, -0.0854, -0.4905, 0.1916, -0.6095, 0.4256], [0.5865, 0.2545, 0.2142, -0.2488, 0.2136, -0.1280, -0.4202, -0.1214, 0.4266, 0.2315, 0.2374, -0.0481, 0.1204, -0.1386, 0.0593, 0.2923, 0.3243, 0.4334, -0.1434, 0.1413, 0.3417, 0.2043, -0.3257, 0.2985, -0.3258, -0.4187, -0.1530, 0.1028, -0.2991, 0.0655, 0.0423, 0.0506, 0.0085, 0.1096, 0.1930, -0.0481, 0.4925, -0.2920, 0.1728, 0.7050, -0.0478, 0.1994, 0.3664, 0.1011, 0.1807, -0.0772, 0.1004, -0.4252, -0.0319, -0.5476, -0.1451, 0.1165, -0.3536, 0.4644, 0.1662, 0.1912, -0.3316, -0.4092, -0.3658, -0.5215, -0.3621, 0.2119, 0.0993, 0.1435]], device='cuda:0')), ('fc3.bias', tensor([0.4092, 0.3767, 0.2888, 0.2641], device='cuda:0'))])