Report

CRITERIA

Learning Algorithm

The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.

1) Learning algorithm

MADDPG is used – **2 pairs of the actor and critic** (~ 1 brain, 1 external brain, **num_agents = 2**) are included in multi_agent in this project. Unlike a single-agent RL, **multiple agents**(interacting with the environment) **interact with each other**. Here, a **shared replay buffer** is implemented for the that multi agent systems(MAS). It is important to notice that many real problems should be designed by multi agents. It is known that MAS can enhance overall system performance, specifically along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse.

From now on, **most of the characteristics below** were actually also **made use of in the project 2**.

Basically, as in the project 2, here we also deal with a continuous task. This is the reason why DDPG agents are used to compose MAS. In DDPG algorithm, there still remain some important characteristics in DQN algorithm such as 1**) the critic's** implementing the network structure similar to the Q-Network in DQN and **following the same paradigm as in Q-learning** for training and 2) implementing a **replay buffer** in order to get experiences from the agent. In this project, the actor implements a current policy to try optimally mapping states to a desired action (to be used for the critic's input). With the gradients from maximizing the estimated Q-value from the critic, the actor is trained. In this way, **the critic and actor are trained interactively**.

Here, **Ornstein-Uhlenbeck noise**(to encourage exploration during training) is added to the actor's selected actions. Unlike for discrete action spaces (where exploration is done via epsilon-greedy or Boltzmann exploration, for instance), exploration is done adding noise to the action itself.

(Reference: T. Lillicrap et al., Continuous control with deep reinforcement learning (2015).)

Additionally, **Epsilon decay** is implemented to decrease an average scale of noise applied to actions.

On the actor's last output, the **Tanh activation** is used, which ensures that the range of every entry in the action vector is between -1 and 1.

For an optimizer, **Adam** is used for both actor and critic networks(actor learning rate: 1e-4, critic learning rate: 3e-4).

Using a **soft update** strategy, target networks are updated, consisting of slowly blending in your regular network weights(actually trained, most up to date) with your target network weights(used for prediction to stabilize strain).

As in the project 2, a **learning interval** is defined to speed up the learning process – performing the learning step after a specific learning interval is passed.

Lastly, I applied **dropout for critic** - nn.Dropout(p=0.2) – just before the last fc layer.

2) Network architectures

Vector Observation space size (per agent): 8

Number of stacked Vector Observation: 3

Vector Action space size (per agent): 2

➔ **state_size: 24 units, action_size: 2 units**

2-1) Actor Network's architecture (mapping states to actions)

Input(batch normalized, 24 units)

fc layer 1 with Batch Normalization and ReLU (input: 24 units (state_size), output: 512 units)

fc layer 2 with ReLU (input: 512 units, output 256 units)

fc layer 3 with Tanh (input: 256 units, output: 2 units (action_size))

2-2) Critic Network's architecture (mapping (state, action) pairs to Q-values)

Input(batch normalized, 24 units)

fcs layer 1 with Batch Normalization and ReLU (input: 24 units (state_size), output: 512 units)

fc layer 2 with Concatenate[(output from layer 1, action), axis=1] & ReLU

(input: 514 units(512 + action_size), output 256 units)

fc layer 3 (input: 256 units, output: 1 unit)

3) Parameters used in DDPG algorithm

Maximum episodes: 20000

Maximum steps per episode: 2000

First target average score: 6.0

4) Parameters used in DDPG Agent

```
buffer_size = 1e+6    # replay buffer size
batch_size = 256      # minibatch size
gamma = 0.99          # discount factor
tau = 1e-3            # for soft update of target parameters
lr_actor = 1e-4       # learning rate of the actor
lr_critic = 3e-4      # learning rate of the critic
ou_mu = 0.0           # Ornstein-Uhlenbeck noise parameter
ou_sigma = 0.2        # Ornstein-Uhlenbeck noise parameter
ou_theta = 0.15       # Ornstein-Uhlenbeck noise parameter
epsilon_decay = 1e-6  # Decay value for epsilon (epsilon -> epsilon - EPSILON_DECAY)
learn_every = 10      # time-step interval for learning
num_learn = 5         # number of learning with sampleing memory
```

Plot of Rewards

A plot of rewards per episode is included to illustrate that the agents get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).

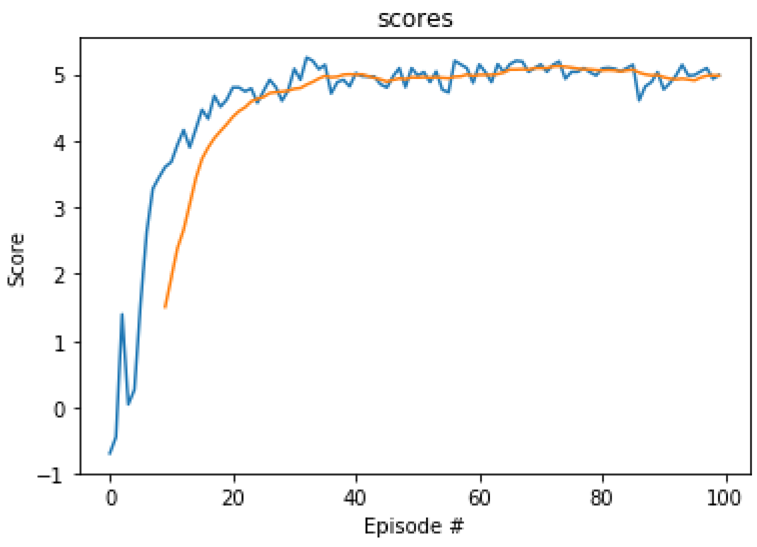
The submission reports the number of episodes needed to solve the environment.

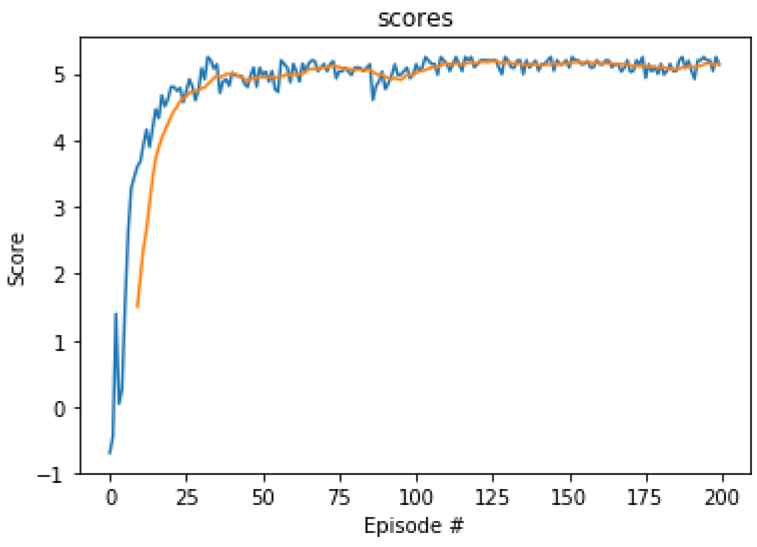
```
scores = ddpg(target_mean_scores_deque=6.0)

# plot the scores
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1, len(scores)+1), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()

plot_scores(scores)
```

```
Episode 100     Average Score: 4.55
Episode 200     Average Score: 5.14
Episode 240     Average Score: 5.15
```





Ideas for Future Work

The submission has concrete future ideas for improving the agent's performance.

1) Utilizing Crawler-DDPG, NAF, PPO, TRPO, TNPG, D4pG, A2C, A3C, Q-Prop, et cetera.

2) Implementing prioritized replay buffer.

3) Weighted sampling for data points with larger errors (from which the model potentially learns more)

4) Optimizing hyperparameters more extensively.

5) Utilizing Ideas in AlphaZero.

6) Trying other weight initialization methods.

(Choosing a good initial condition is often turned out to be very important for achieving high performance!)