PROJECT SPECIFICATION

Collaboration and Competition

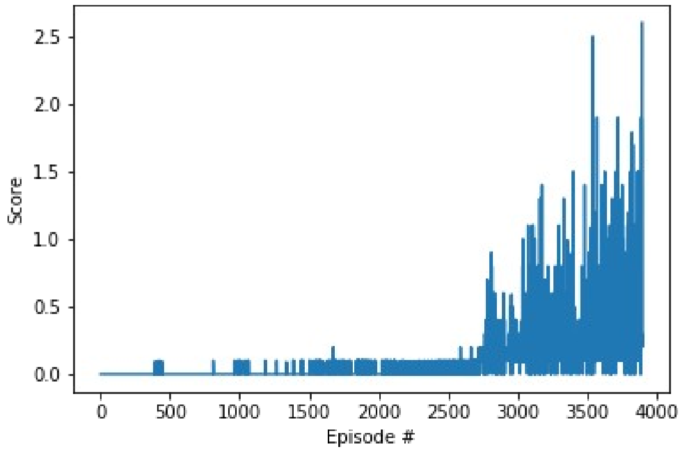
Training Code

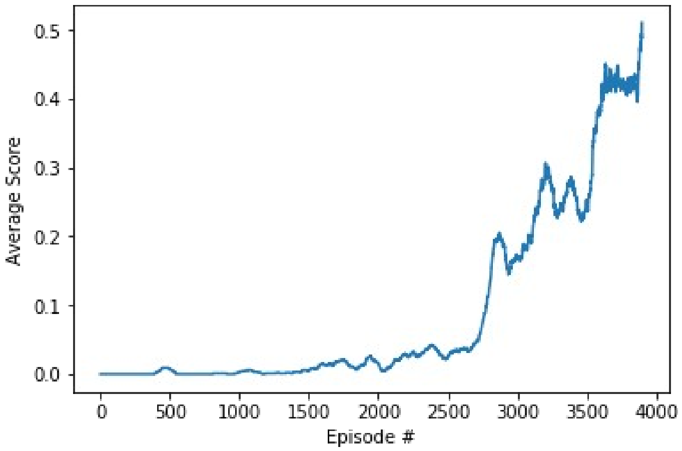
CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
Training code	The repository includes functional, well-documented, and organized code for training the agent.	The code is mainly in 3 files. 1. The ipython notebook (Tennis.ipynb) 2. multi_ddpg_agent.py 3. model.py
Framework	The code is written in PyTorch and Python 3.	The code is written in PyTorch and Python3
Saved Model Weights	The submission includes the saved model weights of the successful agent.	The model weights are saved to: agent1_checkpoint_actor.pth, agent2_checkpoint_actor.pth, agent1_checkpoint_critic.pth, agent2_checkpoint_critic.pth

README

CRITERIA	MEETS SPECIFICATIONS	STUDENT COMMENTS
`README.md`	The GitHub submission includes a `README.md` file in the root of the repository.	The README.md file is present in the main directory of the git repo.
Project Details	The README describes the the project environment details (i.e., the state and action spaces, and when the environment is considered solved).	These are documented under "Project Details" section of README.md
Getting Started	The README has instructions for installing dependencies or downloading needed files.	These are documented under "Getting Started" section of README.md
Instructions	The README describes how to run the code in the repository, to train the agent. For additional resources on creating READMEs or using Markdown, see here and here.	These are documented under "Instructions" section of README.md

Report

CRITERIA	MEETS SPECIFICATIONS	
Report	The submission includes a file in the root of the GitHub repository (one of `Report.md`, `Report.ipynb`, or `Report.pdf`) that provides a description of the implementation.	The Report.pdf (this file) is provide in the main directory of the repository
Learning Algorithm	The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.	The learning algorithm used is from the paper https://arxiv.org/pdf/1706.02275.pdf It is mentioned below this table.
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.	A sample plot of rewards is as below: 

CRITERIA	MEETS SPECIFICATIONS	
		
Ideas for Future Work	The submission has concrete future ideas for improving the agent's performance.	In addition to the current work, we can do the following to improve performance: 1. Implement the other Multi-Agent algorithms such as: a). Multi Agent PPO as presented in this paper (https://arxiv.org/pdf/1710.03748.pdf) b). Multi Agent DQN as presented in this report (http://cs231n.stanford.edu/reports/2016/pdfs/122_Report.pdf). While using MADQN, we can try various combinations of DQN algorithms and as

CRITERIA	MEETS SPECIFICATIONS	
		we know the most effective one is the rainbow method. This significantly improves performance on DQN networks. c). We can implement the suggestion of Gaussians mixture for action-value distribution as described in the D4PG paper along with the MAD4PG algorithm. (https://arxiv.org/pdf/1804.08617.pdf) 2. In addition to the above papers, we can use the traditional optimizations for a deep neural network by finding out the optimal learning rates, batch sizes and other hyper parameters.

Learning Algorithm:

Multi-Agent Deep Deterministic Policy Gradient Algorithm

For completeness, we provide the MADDPG algorithm below.

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

---

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^j)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$$

  end for
end for
