Collaboration and Competition project

Introduction

This report describes my implementation of a multi-agent Deep Reinforcement learning approach, which is able to play tennis. This project is part of the Udacity Deep Reinforcement learning Nanodegree program. The task was to train two agents to play table tennis together and keep the ball in play as long as possible (See Figure 1). More details about the environment and its setup are described in the readme file of this repository.

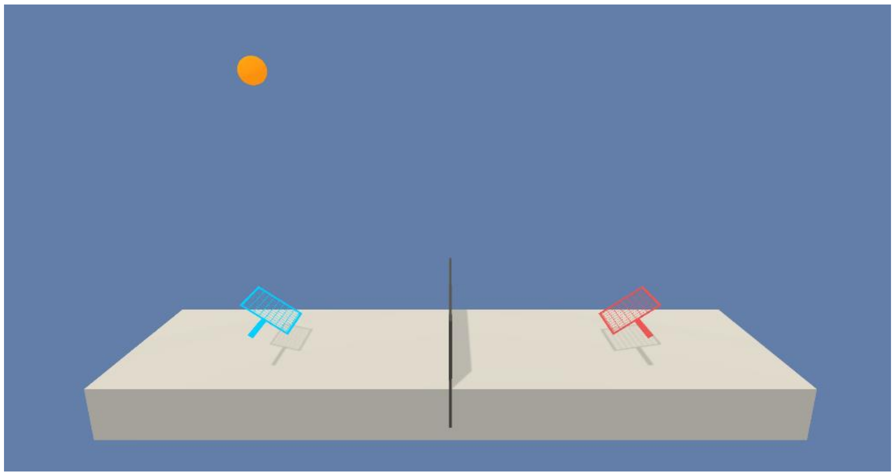

Figure 1: Example screen of the environment

Multi-agent Reinforcement learning

Overview

There are multiple approaches to apply Reinforcement learning to multi-agent environments. One of them is the multi-agent version of the DPPG[1] (Deep Deterministic Policy Gradient) Algorithm called MADDPG[2]. The basic idea of this approach is to collect all experiences centrally and apply them to train the Critic network of all the DPPG agents. This way, all the Agents are able to access all the information of taken actions and observed states. However, each Agent has only a local Actor that only applies the information that was experienced by himself. This framework allows having a different reward structure for each agent, which makes it applicable to collaborative and competitive multi-agent environments.

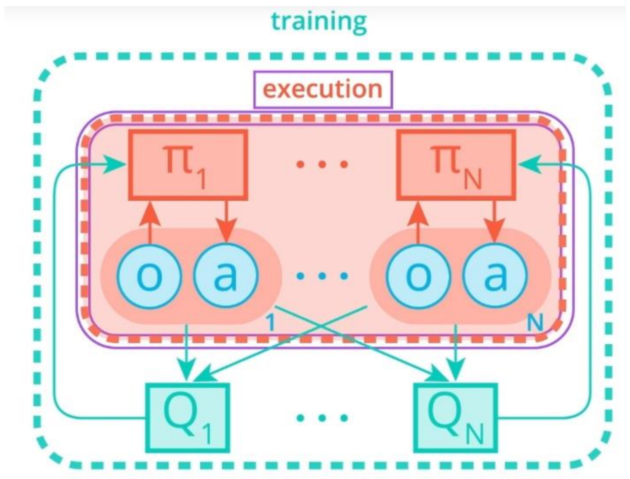

Figure 2: Sketch of MADDPG algorithm

Implementation

The implemented MADDPG algorithm contains a main class to handle multiple agents and their shared replay buffer. The Actor and Critic network of the agents have a similar neural network setup with 3 layers each. Actor and Critic do also have a local network used for training and target network for actions, which receives updates from the local network via a soft update function. An Ornstein-Uhlenbeck noise function is added to boost the exploration of each agent. The noise function generates a larger noise value in the first 1000 episodes which is reduced to half of it for further episodes to focus more on exploitation of the learned strategies. There are multiple parameters that can be tuned to adjust the training of the agents. This is the parameter set which I have applied in my implementation:

Parameter type	**Parameter**	**Value**
OU noise parameter	Mu	0.00
OU noise parameter	Theta	0.15
OU noise parameter	Sigma	0.19/0.38
Neural network setup	Number of hidden units 1	256
Neural network setup	Number of hidden units 2	196
Neural network setup	L2 weight decay critic	0
Memory buffer setup	Memory buffer size	10^5
Memory buffer setup	Batch size for learning	256
Learning speed factor	Discount factor	0.99
Learning speed factor	Network learning rate actor	1e-4
Learning speed factor	Network learning rate critic	1e-3
Learning speed factor	Target network soft update factor	1e-3

Results

Evaluation methods

The score per episode of the agents is the basis to measure the performance of the trained agents. For each episode, the maximum score out of all scores achieved by all agents is used as the score that defines the performance for each episode. To avoid fluctuations the random score over 100 consecutive episodes is used as measure. The environment is considered to be solved once this average score is larger than 0.5.

Agent scores

The training scores of the final agent that learned to act in the tennis environment is presented in Figure 3. In the episode range until around 1000 the agent is mainly in the exploration phase and not yet learning to steadily hit the ball in order to receive a positive reward score. After that, the agents starts learning and the average score improves continuously. The learning is also boosted by the reduction of the noise parameter after episode 1000. Towards episode 2000 the scores get much larger since the agents are able to return the ball multiple times. The task was solved in episode 1909 with an Average score of 0.5170.

Plot of rewards

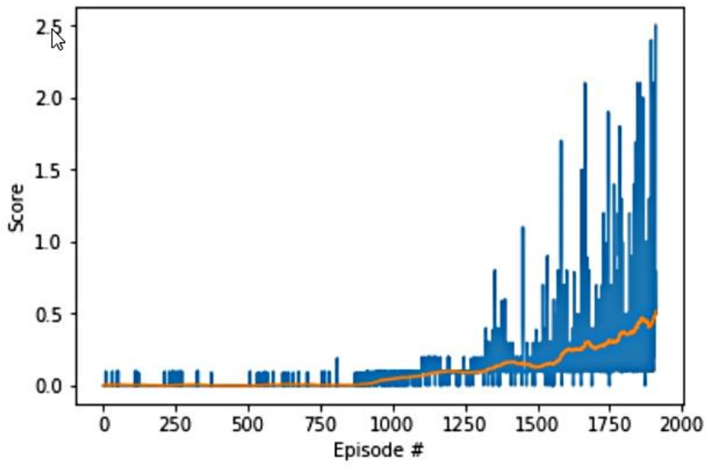

Figure 33: Training scores of the final agents (blue = 1 episode, orange = avg. 100)

Extensions for performance improvements

Parameter optimization

The implementation of the agents has shown a strong dependence on the selected parameters and especially on the noise parameter. A systematic search for the parameters is likely to improve the learning speed of the agent.

Application of prioritized experience Replay

I have experienced that score fluctuations are large and that there are rare events in which the agent performs well. Therefore, I would expected that a prioritized experience replay is able to boost the learning speed of the agent.

Explore other multi-agent algorithms

It would be interesting to also explore other algorithms that are applicable to solve multi-agent reinforcement learning environments. One of these approaches could be Proximal Policy Optimization[3] (PPO), which has also be successfully applied to multi-agent problems.

Sources

1) https://arxiv.org/abs/1509.02971
2) https://arxiv.org/abs/1706.02275
3) https://arxiv.org/abs/1707.06347