Tennis Rally – Decentralized DDPG
Project Submission for Udacity's Deep Reinforcement Learning Nanodegree
Submitted by Patrick Nalepka

Background

1. Multiagent Reinforcement Learning

Multiagent Reinforcement Learning (MARL) is an extension of reinforcement learning techniques involving the training multiple agents in a shared task context. This presents additional challenges to training as now other agents (which can be treated as additional environmental state inputs) are reactive to the behaviors of each other. A GitHub repository of papers related to this area can be found here: https://github.com/LantaoYu/MARL-Papers

2. Deep Deterministic Policy Gradient (DDPG) – Applied to Multiple Agents

To solve the cooperative Tennis Rally task, an Actor-Critic-like algorithm called DDPG was utilized. DDPG involves two neural network architectures. The Actor network architecture takes in the environmental state and outputs the best possible action. This information is inputted to the Critic network, along with the current state, and the Critic provides feedback to the Actor regarding the action-value estimate of the Actor's decision. The original DDPG paper that discusses the algorithm can be found here: https://arxiv.org/abs/1509.02971. A snippet of their algorithm is found below:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Learning Algorithm

The algorithm in this repository implements the DDPG algorithm for each agent in the multiagent Tennis Rally Task. The DDPG algorithm and helper classes have been adapted from the ddpg-pendulum example from Udacity: https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum. Here, each agent's Actor and Critic

networks have full access to the states of the environment received by each agent (8 variables including the position and velocity of the ball and racket × 2 agents = 16 total variables). The components of the algorithm were the same as the algorithm used for the Udacity Reacher assignment, with appropriate modifications due to state and action input sizes.

1. Neural Network Architecture

The Actor and Critic were represented via a neural network. The Actor had an input layer size of 16 (dimensions of the environment), which fed to a fully connected layer of 256 nodes. This layer was then fed to a second hidden layer of 128 nodes, which outputted to 4 output nodes (representing the action dimensions of the Reacher agent). ReLU activation functions were used for the hidden layers, whereas a tanh function was used for the output layer. The Critic had a similar network architecture with the following exceptions. First, an additional 4 nodes were added to the first hidden layer which were the actions of the Actor. Second, the output layer was a simple linear operation to one output node.

2. Replay Buffer

During reinforcement learning, each (S_0,A_0,R_1,S_1) experience tuple is added to a queue (with maximum length **buffer size** [see below]). A mini-batch of experiences are sampled at random and used to train the Actor and Critic neural network.

3. Soft Network Updating

To provide more stable learning, a "local" and "target" network are created. Fast time-scale updates are applied to the "local" networks of the Actor and Critic and are gradually updated to the "target" network so as to have a stable reference point for learning.

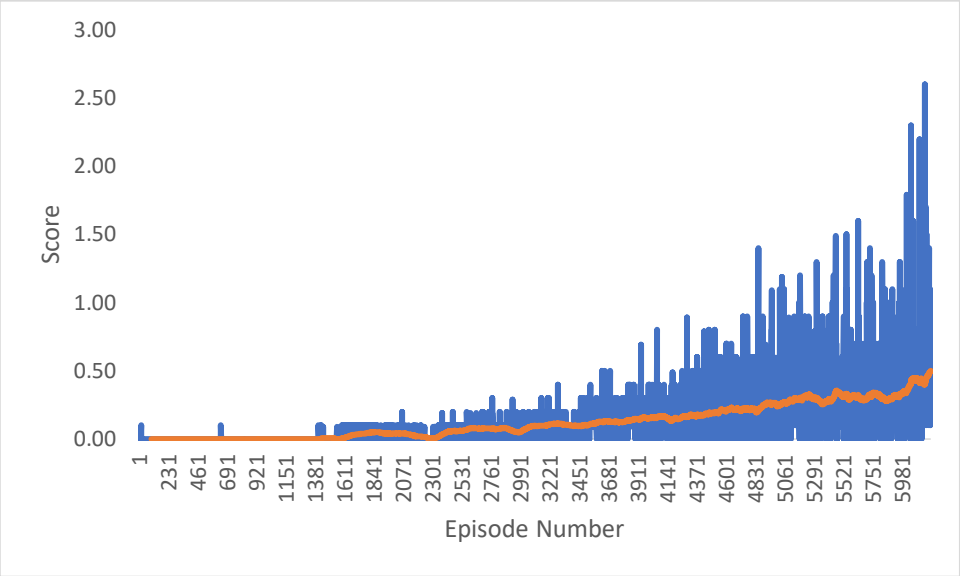
4. DDPG Hyperparameters:

- **Replay Memory Buffer Size:** 2×10^5
- **Mini-batch Size:** 128
- **τ (tau/network interpolation):** 1×10^{-3}
- **γ (gamma/discount factor):** 0.99
- **α (alpha/learning rate):** 1×10^{-4}
- **Maximum number of episodes:** 10000 (with the potential to quit early if required score reached.

Plot of Rewards

Below is an example plot following training. Over 100 consecutive episodes, the required average score of +0.5 was achieved after 6,208 episodes. The score reached at this point was 0.5011, with early stopping applied.

Tennis Rally – Decentralized DDPG
Project Submission for Udacity's Deep Reinforcement Learning Nanodegree
Submitted by Patrick Nalepka



Ideas for Future Work

The downside with the approach utilized in this project is that each agent has full information about the states of the other agents. Future work will seek to utilize MADDPG to provide centralized training via the Critic, but then localized control in the Actor network during test time. The original MADDPG algorithm can be found here: https://arxiv.org/abs/1706.02275 Additionally, reward shaping would be an exciting possibility to alter the behaviors observed in the agents. Modifications in the reward function, for example, can promote more competitive, as opposed to cooperative behaviors. Utilizing more complex task environments, investigating reinforcement learning approaches in task scenario where the environment (including other agents) are partially observable can be an interesting challenge. Finally, some members of the Udacity DRLND Course reported better performance than was shown here (with some reporting the task solved within 3000 episodes, half of what was achieved here). Further tweaks to the hyperparameter settings and network architecture is needed to determine if similar findings can be achieved here.