Algorithm Description:

Algorithm used: A varient of DDPG - Multi Agent Deep Deterministic Policy Gradient (MADDPG)

About MADDPG (taken from the original article):

"We then present an adaptation of actor-critic methods that considers action policies of other agents and is able to successfully learn policies that require complex multiagent coordination. Additionally, we introduce a training regimen utilizing an ensemble of policies for each agent that leads to more robust multi-agent policies. We show the strength of our approach compared to existing methods in cooperative as well as competitive scenarios, where agent populations are able to discover various physical and informational coordination strategies."

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

Networks used:

- Actor: Two hidden layers (400, 300), Relu activation in the hidden layers with BatchNorm and Tanh at the output layer
- Critic: Two hidden layers (400, 300), Relu activation in the hidden layers with BatchNorm and Linear activation at the output layer

Hyperparams:

SEED = 10 # Random seed

NB_EPISODES = 10000 # Max nb of episodes

NB_STEPS = 1000 # Max nb of steps per episodes

UPDATE_EVERY_NB_EPISODE = 4 # Nb of episodes between learning process

MULTIPLE_LEARN_PER_UPDATE = 3 # Nb of multiple learning process performed in a row

BUFFER_SIZE = int(1e5) # replay buffer size

BATCH_SIZE = 200 # minibatch size

ACTOR_FC1_UNITS = 400 # Number of units for the layer 1 in the actor model

ACTOR_FC2_UNITS = 300 # Number of units for the layer 2 in the actor model

CRITIC_FCS1_UNITS = 400 # Number of units for the layer 1 in the critic model

CRITIC_FC2_UNITS = 300 # Number of units for the layer 2 in the critic model

NON_LIN = F.relu # Non linearity operator used in the model

LR_ACTOR = 1e-4 # learning rate of the actor

LR_CRITIC = 5e-3 # learning rate of the critic

WEIGHT_DECAY = 0 # L2 weight decay

GAMMA = 0.995 # Discount factor

TAU = 1e-3 # For soft update of target parameters

CLIP_CRITIC_GRADIENT = False # Clip gradient during Critic optimization

ADD_OU_NOISE = True # Add Ornstein-Uhlenbeck noise

MU = 0. # Ornstein-Uhlenbeck noise parameter

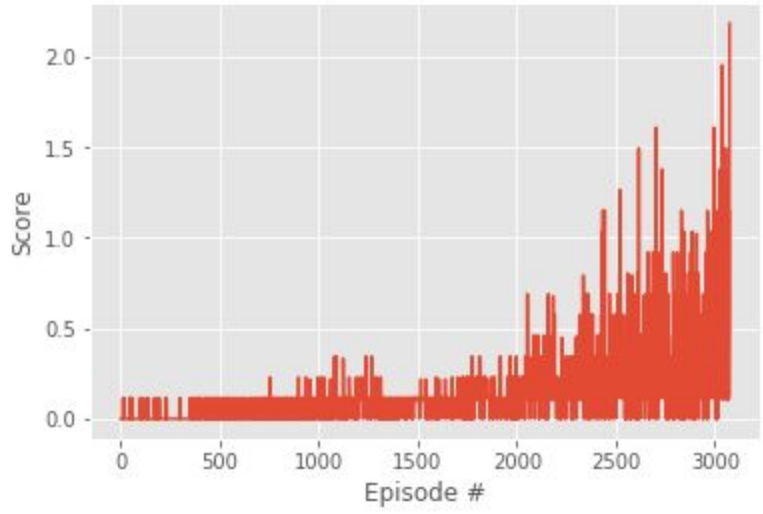
THETA = 0.15 # Ornstein-Uhlenbeck noise parameter

SIGMA = 0.2 # Ornstein-Uhlenbeck noise parameter

NOISE = 1.0 # Initial Noise Amplitude

NOISE_REDUCTION = 1.0 # Noise amplitude decay ratio

Rewards Plot:



Future Work:

Optimization of hyperparams through grid search or SMBO. Optimization of network architectures with deeper and more robust neural nets