Continuous Control Report

Motivation

The navigation project utilized DQNs. DQNs handle problems with high dimensional observation spaces, but need discrete, low-dimensional action spaces. This is untrue for many tasks of interest, most notably physical control tasks, which have continuous and high dimensional action spaces (action space for this problem is between -1,1).

Because DQNs cannot be straight forwardly applied to continuous domains, a different approach is required. DDPG (Deep Deterministic Policy Gradient) relies on an actor-critic architecture with two NNs: actor and critic. An actor is used to tune $\boldsymbol{\theta}$ for the policy function, and the critic evaluates the actions chosen by the actor (evaluates the policy function estimated by the actor by using the temporal difference error).

Additionally, the DDPG here uses experience replay, soft updating between the local and target NNs for both the agent and critic, and a noise process to induce exploration. Note: soft updating is a blending of the local NN weights to the target NN.

Learning Algorithm

Below is the DDPG algorithm, the hyperparameters used, and the NN architecture for both the agent and critic.

DDPG algorithm copied from: https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287

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

Hyperparameters

```
# parameters
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 256        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 0.0002       # learning rate of the actor
LR_CRITIC = 0.0004      # learning rate of the critic
WEIGHT_DECAY = 0.0001   # L2 weight decay
UPDATE_EVERY = 20       # how often network is updated
EPSILON = 1.0           #
```

NN

Agent:

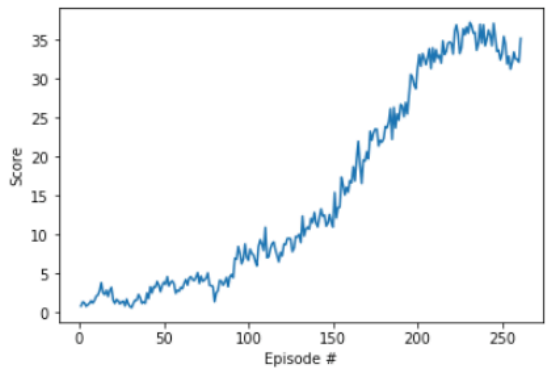
```
nn.Linear(state_size, fc1_units),  # 128 units, fully connected layer
nn.ReLU(),
nn.BatchNorm1d(fc1_units),
nn.Linear(fc1_units, fc2_units),  # 128 units, fully connected layer
nn.ReLU(),
nn.BatchNorm1d(fc2_units),
nn.Linear(fc2_units, action_size),
nn.Tanh()  # delivers output between -1,1
```

Critic:

```
nn.Linear(state_size, fc1_units),  # 128 units, fully connected layer
nn.ReLU(),
nn.BatchNorm1d(fc1_units)
nn.Linear((fc1_units + action_size), fc2_units),  # 128 units + 4 actions, fully
connected layer
nn.ReLU(),
nn.Linear(fc2_units, 1)
```

Plot of Rewards

```
Episode 100     Average Score Last 100 Episodes: 3.20   Avg. Score (All Agents) Last Episode: 6.70
Episode 200     Average Score Last 100 Episodes: 15.72  Avg. Score (All Agents) Last Episode: 31.27
Episode 261     Average Score Last 100 Episodes: 30.07  Avg. Score (All Agents) Last Episode: 35.16
Environment solved in 161 episodes!     Average Score: 30.07
```



Ideas for Future Work

- Implement PPO algorithm
- Implement A3C algorithm
- Implement D4PG algorithm
- Implement Grid Search for Hyperparameter tuning
- Test Different Neural Net architectures for the Agent and Critic