Report

The implementation of this reacher (single/multi-agent) solution was done in a python 3.6, with GPU (cuda) support in a local (linux ubuntu 24.04) environment.

Learning Algorithm

The learning algorithm was an implementation of actor-critic DDPG method, with various combinations of hyperparameter and model structure attempted to achieve the solution. While the notebook has solution for the single entity (version 1) environment, the same can be applied to solve the multi-entity (version 2) environment, by just changing the selected Reacher unity environment.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

How it works:

The agent has a pair of Actor neural nets (actor_local, actor_target) with identical structure, and another pair of Critic neural nets (critic_local, critic_target) with identical structure, and one replay buffer

Actor neural network:

> 33 inputs for each dimension → 128 unit hidden layer → Batch Normalization → ReLu activation → 256 unit hidden layer → ReLu activation → 4 unit output for each action → tanh activation (to bound it between -1 and +1)

Critic neural network:

> 33 inputs for each dimension → 128 unit hidden layer → Batch Normalization → ReLu activation + action (concatenated) → 256 unit hidden layer → ReLu activation → 1 unit output

In every step of iteration, actor_local produces action (with added noise, following the Ornstein-Uhlenbeck process and parameters: mu=0., theta=0.15, sigma=0.08) for each action types for each agent, and trains itself. With these actions fed back into the environment, which returns the next states corresponding to each action for each agent, the whole individual experiences (state, action, reward, next_state, done for all agents combined) get added to the replay buffer.

All the algorithms use a fixed buffer of 1,000,000 size, and was trained over 1,000 episode with each episode having 1,000 steps. The model was trained 7 times in every 20 steps, and the experience from each step was added to the fixed buffer (which deleted oldest experience when full to accommodate for newest experiences). For each step of learning (7 times in every 20 steps), 128 experiences were sampled randomly from the replay buffer, which captures state, action, reward, next_state, done (for all agents combined, not by individual agents, to facilitate cross-learning).

In each training iteration, the agent learns by updating the critic, by getting predicted next state actions from actor_target, and then getting the Q values for next states by feeding those next state and next state actions into the critic_target network. The Q targets for current states are calculated as follows: rewards + (gamma * Q values for next states * (1 – dones)). In other words, Q targets for current states = rewards + gamma * critic_target(next_state, actor_target(next_state))

Q expected is calculated using current state and action from critic_local. Using Q target and Q expected, losses are calculated, which is then used for training the critic_local network through back-propagation of the losses.

Then actor is updated, by back-propagating the actor losses (derived by putting current states in actor_local to get predicted actions and then feeding these predicted actions and current states into critic_local) through the actor_local.

Once the local networks (critic_local and actor_local) are updated, then it is followed by soft update of the target network parameters (for both critic and actor with parameter TAU (target = TAU * local + (1 - TAU)*target)

Other hyperparameters

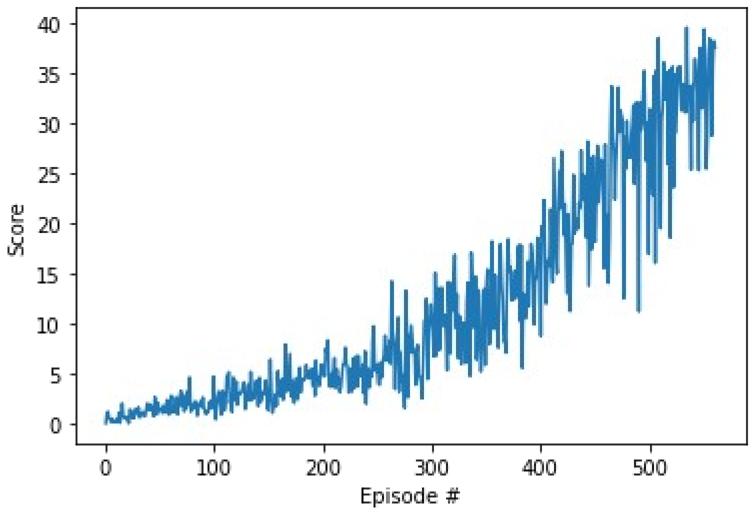
```
BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 128       # minibatch size
GAMMA = 0.95           # discount factor
TAU = 1e-3             # for soft update of target parameters
LR_ACTOR = 1e-4        # learning rate of the actor
LR_CRITIC = 1e-3       # learning rate of the critic
WEIGHT_DECAY = 0       # L2 weight decay
TRAIN_EVERY = 20       # How many iterations to wait before updating target networks
TRAIN_TIMES = 7        # Number of times training, every time training happens
NUM_EPISODES = 1000
MAX_T = 1000
PRINT_EVERY = 100
```

Various hyperparameters were tried by changing the hyperparameters, but best results (below) were achieved with the hyperparameter values above.

Plot of Rewards

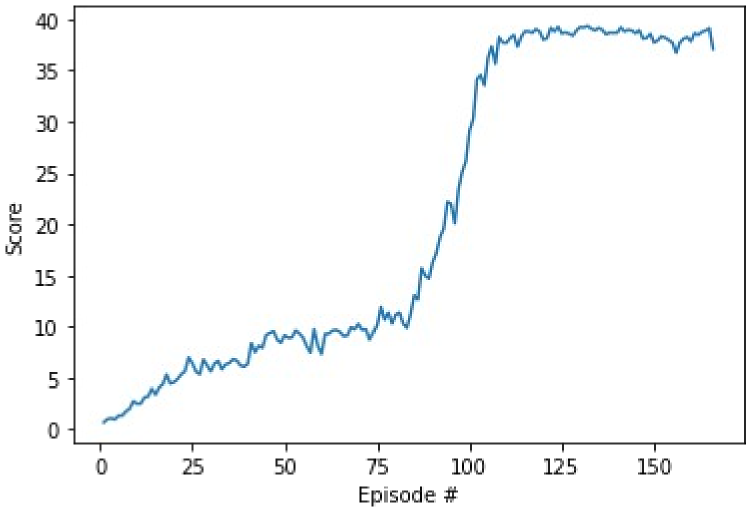
- ***[version 1]*** The notebook represents this version

```
Episode 100     Average Score: 1.41
Episode 200     Average Score: 3.48
Episode 300     Average Score: 6.17
Episode 400     Average Score: 11.98
Episode 500     Average Score: 23.23
Episode 560     Average Score: 30.19
Environment solved in 560 episodes!     Average Score: 30.19
```



- ***[version 2]***

```
Episode 100	Average Score: 9.01
Episode 166	Average Score: 30.09
Environment solved in 166 episodes!	Average Score: 30.09
```



<u>Ideas for Future Work</u>

To enhance the performance various alternatives such as, REINFORCE, TNPG, RWR, REPS, TRPO, CEM, CMA-ES, one-by-one or in combination, can be attempted as extension of the current work.