Report on Continuous Control Project

This project has been split into four modules due to its length, logical structure, and reusability. The modules are as follows:

- DDPG_agent.py: Implementation of DDPG agent that interacts with and learns from the environment.
- PPO_agent.py: Implementation of PPO agent that interacts with and learns from the environment.
- Network.py: Implementation of Actor (Policy) model and Critic (Value) model.
- dqn_agent.py: This file implements the agent that interacts with and learns from the environment. The file also has normal replay buffer and prioritized replay buffer.
- utils.py: This file contains helper functions for the project
- main.py: This is the main module with the training loop.

Proximal Policy Optimization (PPO) Algorithm

PPO is an on-policy algorithm, and it can be used for environments with either discrete or continuous action spaces.

PPO-clip updates policies via

$$\theta_{k+1} = argmax_{\theta} E_{s,a\sim\pi_{\theta_k}}[L(s,\ \alpha,\ \theta_k,\ \theta)]\,,$$

typically taking multiple steps of (usually minibatch) SGD to maximize the objective. Here L is given by

$$L(s,\ \alpha,\ \theta_k,\ \theta)\ =\ min(\frac{\pi_\theta(\alpha|s)}{\pi_{\theta_k}(\alpha|s)}A^{\pi_{\theta_k}}(s,\ \alpha),\ clip(\frac{\pi_\theta(\alpha|s)}{\pi_{\theta_k}(\alpha|s)},\ 1-\varepsilon,\ 1+\varepsilon)A^{\pi_{\theta_k}}(s,\ \alpha))\,,$$

in which ε is a (small) hyperparameter which roughly says how far away the new policy is allowed to go from the old.

There is a considerably simplified version

$$L(s,\ \alpha,\ \theta_k,\ \theta)\ =\ min(\frac{\pi_\theta(\alpha|s)}{\pi_{\theta_k}(\alpha|s)}A^{\pi_{\theta_k}}(s,\ \alpha),\ g(\varepsilon,\ A^{\pi_{\theta_k}}(s,\ \alpha)))\,,$$

where $g(\varepsilon,\ A) = (1+\varepsilon)A$ when $A \geq 0$ and $g(\varepsilon,\ A) = (1-\varepsilon)A$ when $A < 0$.

Pseudocode for PPO-Clip Algorithm is shown below:

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Deep Deterministic Policy Gradients (DDPG) Algorithm

DDPG is a policy gradient algorithm that uses a stochastic behavior policy for good exploration but estimates a deterministic target policy, which is much easier to learn. DDPG primarily uses two neural networks, one for actor and one for the critic. The input of the actor network is the current state, and the output is a single real value representing an action chosen from a continuous action space. The critic's output is imply the estimated Q-value of the current state and of the action given by the actor. The deterministic policy gradient theorem provides the update rule for the weights of the actor network. The critic network is updated from the gradients obtained from the TD error signal.

We want to maximize the rewards (Q-values) received over the ampled mini-batch. The gradient is given as:

PPO is an on-policy algorithm, and it can be used for environments with either discrete or continuous action spaces.

$$\nabla_{\theta^\mu} J \approx E_{s_t \sim \rho^\beta}[\nabla_{\theta^\beta} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}]$$

applying the chain rule

$$= E_{s_t \sim \rho^\beta}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_t}]$$

Pseudocode for DDPG Algorithm is shown 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:

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

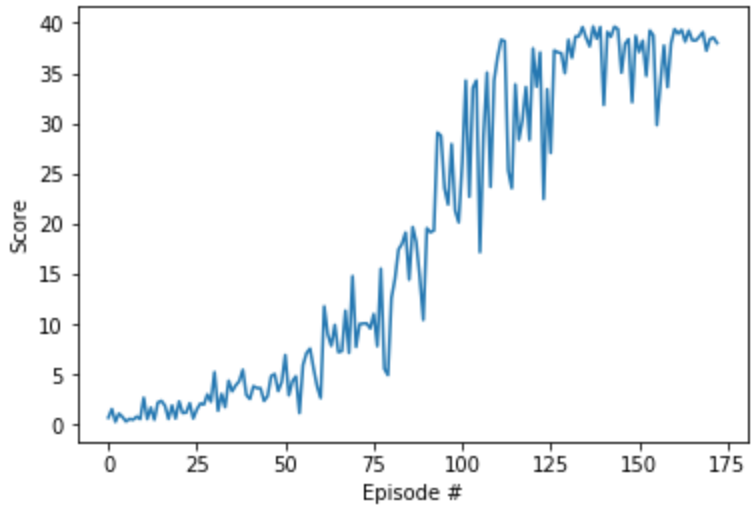
    Update the target networks:

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

  end for
end for

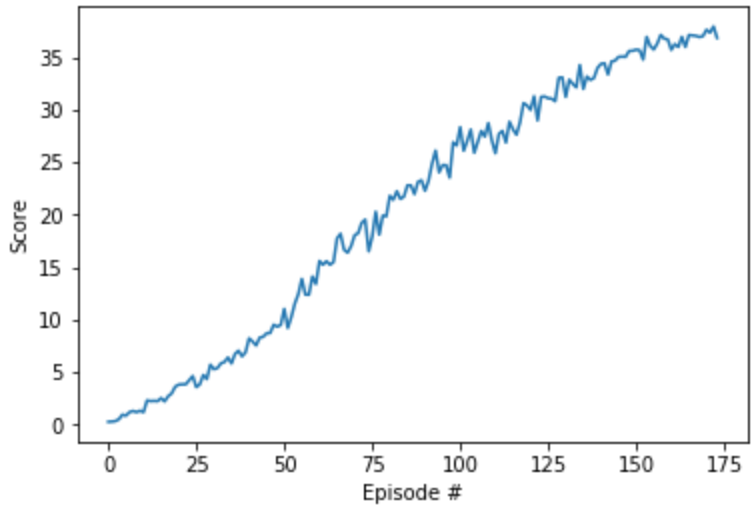
Plot of Rewards

Version 1 single agent environment (DDPG implementation)



Environment solved in 172 episodes with an average reward (over 100 episodes) of at least +30

Version 2 twenty agents environment (PPO implementation)



Environment solved in 176 episodes with an average reward (over 100 episodes) of at least +30

Ideas for Future Work

- Implement Asynchronous Advantage Actor Critic (A3C) algorithm
- Implement Advantage Actor Critic (A2C) algorithm
- Implement Distributed Distributional Deterministic Policy Gradients (D4PG) algorithm