DRL Continuous Control

Introduction

This project aims to train an agent to control a double-jointed arm to move to target locations and maintain its position there for as many time steps as possible.

A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the agent aims to maintain its position at the target location for as many time steps as possible.

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

There are 20 identical copies of the agent. It has been shown that having multiple copies of the same agent-sharing experience can accelerate learning, and the same is applied to this project.

The environment is considered solved when the average (over 100 episodes) of those average scores is at least +30. This problem will be solved using the Deep Q-Network Algorithm using PyTorch and Python.

Methodology

If a DRL algorithm uses a deep neural network to approximate value function, then the agent is said to be value-based. If an agent uses a deep neural network to approximate a policy directly, then it is a policy-based algorithm.

Both methods have advantages and disadvantages. Value-based methods have high variance since one state can give multiple rewards, and the bias is low because the rewards are not assumed or estimated. Policy-based methods have a high bias as the rewards are estimated after the agent has explored a set of states, and since this method does not lead to multiple rewards, the variance is low.

So, the actor-critic uses the best of both methods, The actor is responsible for selecting actions based on the current state, while the critic evaluates the actions the actor took and provides feedback on their quality.

DDPG is an example of an actor-critic method. It stands for Deep Deterministic Policy Gradient. It is an extension of the DQN (Deep Q-Network) algorithm used for discrete action spaces.

The key idea behind DDPG is to use deep neural networks to approximate the actor and critic functions. The actor-network takes the current state as input and outputs the best action to take. The critic network takes both the state and action as input and estimates the Q-value, which represents the expected return of taking that action in that state.

The actor update equation is used to update the weights of the actor-network. It aims to maximize the expected return by adjusting the actor's policy. The equation is as follows:

$$\nabla\theta\mu \approx 1/N \sum(\nabla aQ(s, a|\theta Q)|s=s, a=\mu(s|\theta\mu))\ \nabla\theta\mu(s|\theta\mu)$$

Here, $\nabla\theta\mu$ represents the gradients of the actor network's weights, $\theta\mu$, with respect to the expected return. N is the batch size, s is the state, a is the action, $Q(s, a|\theta Q)$ is the Q-value estimated by the critic network, and $\mu(s|\theta\mu)$ is the action selected by the actor-network.

The critic update equation is used to update the weights of the critic network. It aims to minimize the mean squared error between the estimated and target Q values. The equation is as follows:

$$L = 1/N * \sum(y - Q(s, a|\theta Q))^2$$

Here, L represents the loss function, N is the batch size, y is the target Q-value, and $Q(s, a|\theta Q)$ is the Q-value estimated by the critic network.

The target Q-value, y, is calculated using the target actor and target critic networks. It is given by:

$$y = r + \gamma * Q'(s', \mu'(s'|\theta\mu')|\theta Q')$$

Here, r is the reward received, γ is the discount factor, s' is the next state, μ' is the action selected by the target actor-network, and $Q'(s', \mu'(s'|\theta\mu')|\theta Q')$ does the target critic network estimate the Q-value.

To improve the performance of the DDPG algorithm, two techniques are used: replay buffer and soft updates to the target networks.

1. The **replay buffer** is a memory that stores the agent's experiences. It randomly samples experiences from the buffer to break the correlation between consecutive experiences and improve training stability. This allows the agent to learn from a diverse set of experiences.
2. **Soft updates** to the target networks are used to slowly update the target actor and critic networks. Instead of copying the weights from the regular networks to the target networks abruptly, DDPG blends the weights gradually. This helps stabilize the learning process and prevents sudden changes that could lead to instability.

The neural network architecture for the actor has two hidden layers and an output layer.
The ReLu activation function was used for both layers and TanH for the output layer.
The critic also has two hidden layers and an output layer, ReLu and Linear respectively.

Hyperparameters

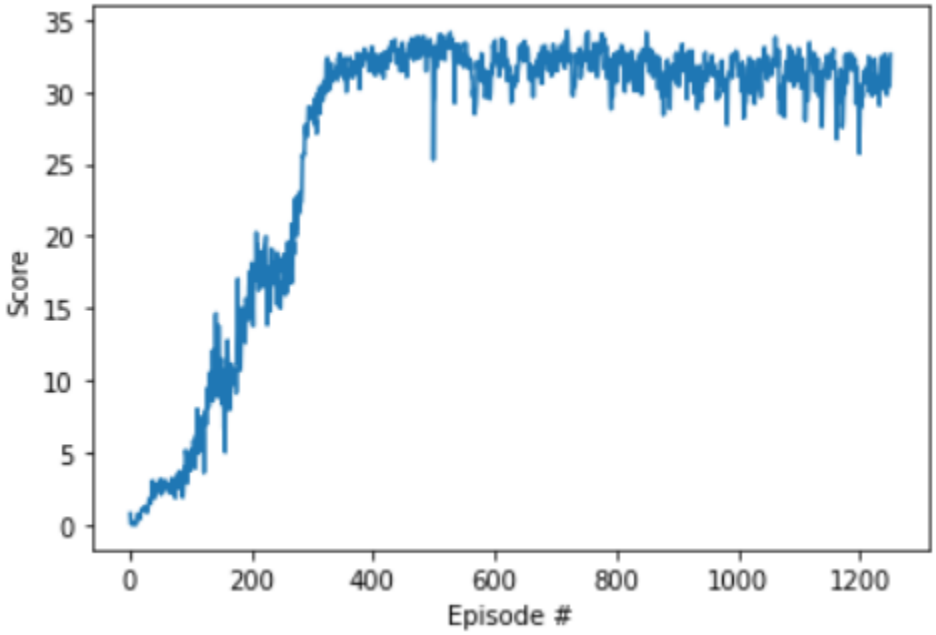
For the training, the following hyperparameters were used.

Parameter	Value
Number of Episodes	1250
Maximum Time Steps	900
Print Every	100
Num_agents	20
Random Seed	35
Replay buffer size	int(1e5)
Batch Size	184
Discount Factor	0.99
Soft update of target Parameters	1e-3
Learning Rate Actor	1e-3

Learning Rate Critc	1e-3
Average value of the noise	0
standard deviation of the noise	0.2
Rate at which the noise reverts to the mean	0.15

Results

The following reward plot was obtained when the agent achieved an average reward of 30 for 100 continuous cycles. The environment was solved in 488 episodes.



Future Work

In the future, I intend to do the following:-

1. Implement Prioritized experience replay and try different algorithms to improve the results obtained in this project.
2. To complete the 'Crawl' challenge, where the goal is to teach a creature with four legs to walk forward without falling.