Continuous Control Project Report

Marc Poirier

May 9, 2025

1 Introduction

In this project, we address the Continuous Control task using the Reacher environment from Unity ML-Agents. The environment consists of 20 double-jointed robotic arms, each tasked with tracking a moving target by applying torques to its joints. The objective is to keep the arm's end-effector at the target location for as many time steps as possible, earning a reward of +0.1 per step when successful. The environment is considered solved when the average reward over 100 consecutive episodes, across all 20 agents, reaches at least +30.

We implement the Deep Deterministic Policy Gradient (DDPG) algorithm, a model-free, off-policy reinforcement learning method designed for continuous action spaces. Our implementation successfully solves the environment in 104 episodes, achieving the target average reward.

2 Implementation Description

The Reacher environment features 20 identical agents operating simultaneously. Each agent observes a 33-dimensional continuous state space, which includes joint positions, rotations, velocities, angular velocities, and the target's position. The action space is 4-dimensional and continuous, with each value ranging from -1 to 1, representing torques applied to the arm's four joints.

The DDPG algorithm is employed to train the agents. It combines an actor network, which maps states to actions, and a critic network, which estimates the Q-value of state-action pairs. Training leverages a replay buffer for off-policy learning and Ornstein-Uhlenbeck noise for exploration. The implementation uses PyTorch for neural network construction and optimization, with computations performed on a GPU.

3 Learning Algorithm and Hyperparameters

The DDPG algorithm consists of the following key components:

- **Actor Network**: Predicts actions from states using a neural network.

- **Critic Network**: Evaluates actions by estimating Q-values.
- **Replay Buffer**: Stores experience tuples (state, action, reward, next state, done) for off-policy learning.
- **Target Networks**: Maintain stable learning via soft updates to separate actor and critic targets.
- **Ornstein-Uhlenbeck Noise**: Adds temporally correlated noise to actions for exploration.

The chosen hyperparameters are:

- Replay Buffer Size: 1,000,000 (capacity for experience tuples).
- Batch Size: 1024 (samples per learning update).
- Discount Factor (γ): 0.99 (weights future rewards).
- Soft Update Parameter (τ): 0.001 (controls target network update rate).
- Actor Learning Rate: 10^{-3}.
- Critic Learning Rate: 10^{-3}.
- Weight Decay: 0 (no L2 regularization for the critic).
- Noise Parameters: $\mu = 0$, $\theta = 0.15$, $\sigma = 0.2$ (Ornstein-Uhlenbeck process).
- Learning Frequency: Every 20 steps, with 10 updates per cycle.

The agent updates its networks using experiences sampled from the replay buffer, with the critic minimizing mean squared error and the actor maximizing Q-values. Gradient clipping is applied to the critic to ensure stability.

4 Model Architectures

4.1 Actor Network

The actor network maps the 33-dimensional state to a 4-dimensional action vector:

- **Input Layer**: 33 units (state size).
- **Hidden Layer 1**: 400 units with ReLU activation and batch normalization.
- **Hidden Layer 2**: 300 units with ReLU activation.
- **Output Layer**: 4 units with tanh activation (scaled to [-1, 1]).

Weights are initialized using a uniform distribution based on layer fan-in, with the output layer using a smaller range ($[-3 \times 10^{-3}, 3 \times 10^{-3}]$).

4.2 Critic Network

The critic network estimates Q-values from state-action pairs:

- **Input Layer**: 33 units (state size).
- **Hidden Layer 1**: 400 units with leaky ReLU activation and batch normalization.
- **Concatenation**: Combines 400-unit output with 4-unit action input.
- **Hidden Layer 2**: 300 units with leaky ReLU activation.
- **Output Layer**: 1 unit (Q-value).

Weights follow a similar initialization strategy as the actor, with leaky ReLU enhancing gradient flow.

5 Results

The environment was solved in 104 episodes, with the average reward over 100 consecutive episodes exceeding +30 across all 20 agents. The training process used a maximum of 2000 episodes and 1000 time steps per episode, with progress monitored every 10 episodes. The plot below illustrates the rewards per episode:

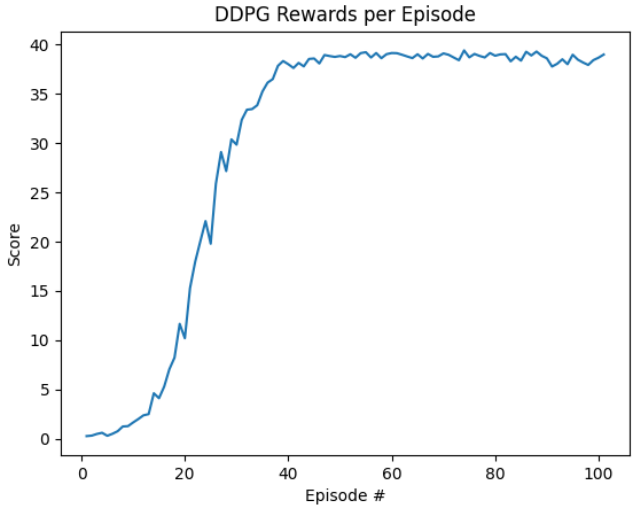

Figure 1: DDPG Rewards per Episode

The plot shows the agent's learning curve, confirming that the target score was achieved efficiently. The environment was solved in 101 episodes !

6 Future Ideas for Improvement

To enhance the agent's performance, we suggest the following:

- **Improved Exploration**: Replace Ornstein-Uhlenbeck noise with parameter space noise or adaptive noise scaling to better balance exploration and exploitation.
- **Algorithm Enhancements**: Adopt Twin Delayed DDPG (TD3) or Soft Actor-Critic (SAC) to address overestimation bias and improve sample efficiency.
- **Hyperparameter Optimization**: Conduct grid search or Bayesian optimization to fine-tune learning rates, batch sizes, and noise parameters.
- **Prioritized Experience Replay**: Prioritize high-error experiences in the replay buffer to accelerate convergence.

These strategies could reduce the number of episodes required to solve the environment and improve robustness.