Deep Deterministic Policy Gradient (DDPG) Tutorial

September 21, 2025

1 Introduction

Deep Deterministic Policy Gradient (DDPG) extends deterministic policy gradients to deep neural networks, combining actor-critic architectures with replay buffers and target networks. DDPG excels in continuous control tasks by learning a deterministic policy while stabilizing training through soft target updates.

2 Theory and Formulas

2.1 Deterministic Policy Gradient

For deterministic policy $\mu_\theta(s)$ and critic $Q_w(s,a)$, the objective gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\big[\nabla_a Q_w(s,a)\big|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)\big]. \tag{1}$$

States are sampled from a replay buffer $\mathcal{D}$, enabling off-policy learning.

2.2 Critic Update

The critic minimizes the temporal-difference loss with target networks $(\mu_{\theta^-}, Q_{w^-})$:

$$L(w) = \mathbb{E}_{(s,a,r,s')}\Big[\big(r + \gamma Q_{w^-}(s', \mu_{\theta^-}(s')) - Q_w(s,a)\big)^2\Big]. \tag{2}$$

Target networks are updated softly: $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$.

2.3 Exploration Noise

Because the policy is deterministic, exploration is introduced by adding noise $\mathcal{N}_t$ to the action during data collection: $a_t = \mu_\theta(s_t) + \mathcal{N}_t$. Ornstein-Uhlenbeck noise or Gaussian noise preserves temporal coherence.

3 Applications and Tips

- **Robotics**: continuous torque control, manipulation, locomotion.
- **Industrial control**: optimize real-valued actuators with safety constraints.
- **Autonomous driving**: steering and throttle in simulation environments.

- **Best practices**: use large replay buffers, normalize observations, clip gradients, tune noise scale, and monitor critic loss for divergence.

4 Python Practice

The script `gen_ddpg_figures.py` trains a lightweight DDPG agent on a one-dimensional continuous control problem. It logs episode returns and the evolution of the deterministic policy across states.

Listing 1: Excerpt from gen$_ddpg_figures.py$

```
# Critic update
q_pred = critic_w @ features(state, action)
q_target = reward + gamma * critic_target_w @ features(next_state,
    actor_target(next_state))
critic_w += critic_lr * (q_target - q_pred) * features(state, action)

# Actor update via deterministic policy gradient
grad_q = critic_grad(state, actor(state))
actor_theta += actor_lr * grad_q
```

5 Result

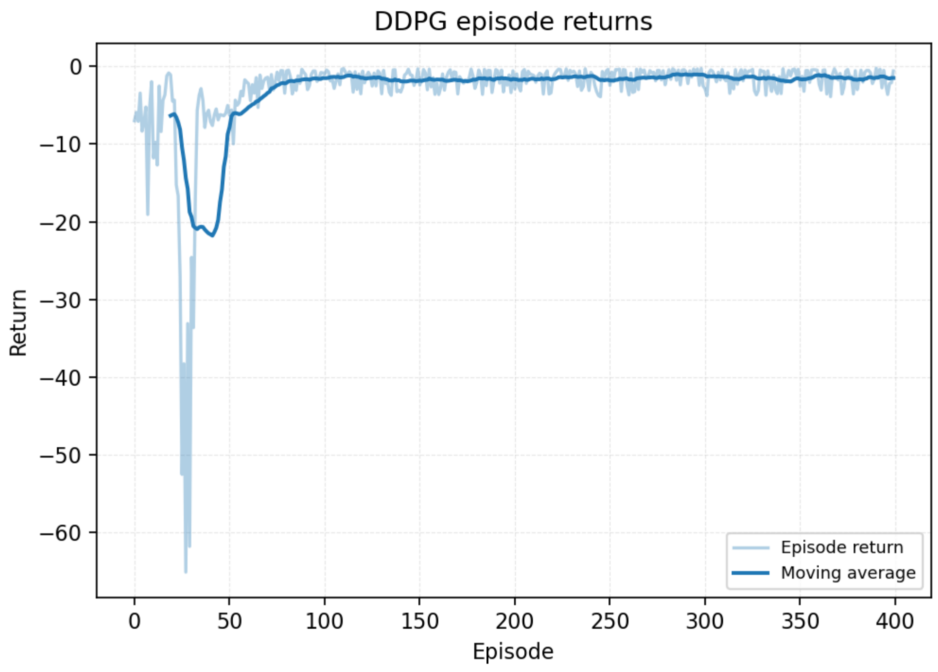

Figure 1: DDPG training returns approaching the optimal control policy

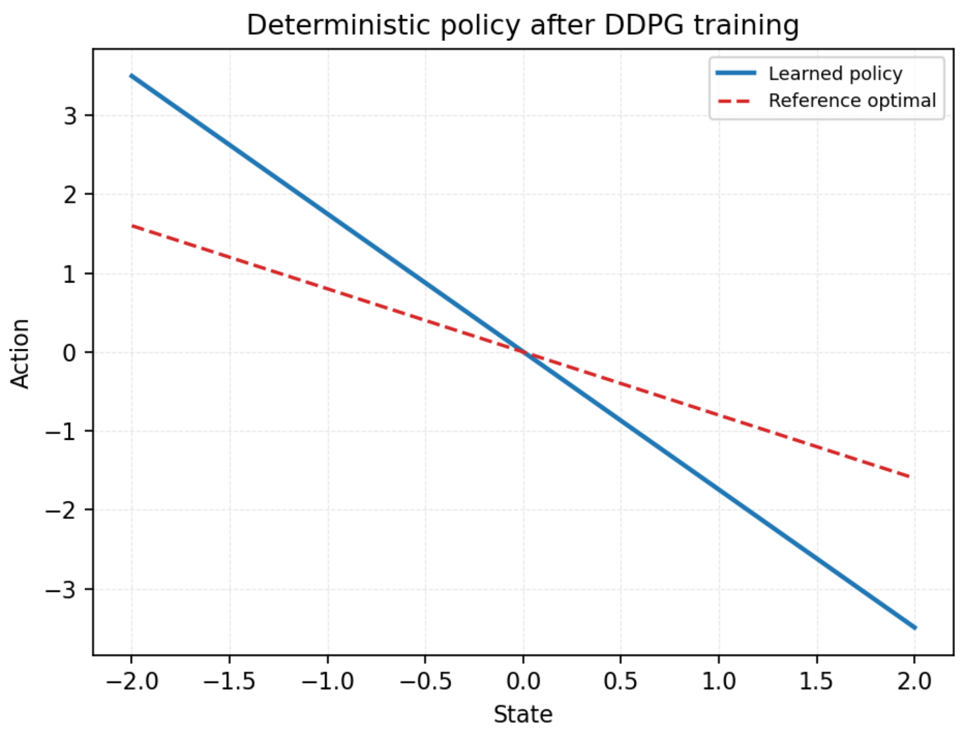

Figure 2: Learned deterministic policy across states compared to the optimal action

6 Summary

DDPG adapts deterministic policy gradients to deep function approximation using target networks and replay buffers for stability. Proper exploration noise, normalization, and critic monitoring are essential. The toy control example demonstrates improving returns and convergence of the actor toward optimal continuous actions.