Week 9 Precept Notes: DDPG & PPO

March 28, 2025

Quick Review: PPO

Today we'll be discussing DDPG and how it relates to PPO. But before we get there, let's begin with the review of PPO from last precept — not everything, just the essentials we'll build on today.

We'll go over two things:

1. What is PPO?
2. What are the most important ideas inside PPO?

1. What is PPO?

PPO (Proximal Policy Optimization) is a policy gradient method — it's an actor-critic algorithm that improves a parameterized policy $\pi_\theta(a|s)$ using data collected from environment interaction.

Unlike vanilla policy gradient, PPO introduces a mechanism to prevent the new policy π_θ from changing too much from the old policy π_{old}.

It does this by optimizing a **clipped surrogate objective** — it's not a hard constraint like TRPO's KL penalty, but an approximate way to keep the updated policy in a "trust region".

Mental picture: PPO = stochastic policy + conservative updates.

2. Important components of PPO

1. **Clipped surrogate objective (core idea)**:
   - PPO maximizes:
     $$L^{\text{clip}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right],$$
     where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$
   - This discourages large policy updates when r_t moves far from 1.
2. **Stochastic policy (sampling-based action)**:
   - PPO samples actions from a distribution $\pi_\theta(a|s)$.
   - Exploration is built-in, since sampling adds randomness.
3. **Advantage estimation (usually with GAE)**:
   - We estimate $\hat{A}_t$ using Generalized Advantage Estimation:
     $$\hat{A}_t = \sum_{l=0}^{T}(\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$
   - GAE controls the bias-variance tradeoff.
4. **Training with old samples (importance weighting)**:
   - Data comes from π_{old}, but the loss is on π_θ.
   - We use $r_t(\theta)$ to correct for this distribution mismatch.

Wrap-up

To sum up:

- PPO is a policy gradient method that avoids large updates via clipping.
- It is stochastic and relies heavily on advantage estimates.
- It collects samples under the old policy and applies importance correction.

That's all we need for now. Let's carry these ideas forward as we move to DDPG.

DDPG: Deep Deterministic Policy Gradient

The goal of this lecture is to understand how DDPG works as a continuous control reinforcement learning algorithm. We'll focus on two major aspects: (1) what are the most important ideas and knowledge points inside DDPG; (2) how DDPG relates to PPO. Along the way, we'll break down the algorithm step by step and explain why each part is needed.

1. What is DDPG trying to solve?

In environments with continuous action spaces — such as robotic control, vehicle steering, or manipulation — we can't enumerate all possible actions. So sampling-based methods like DQN don't work directly. One idea is to use a neural network to output the best action directly: a deterministic policy. That's exactly what DDPG does.

DDPG uses an actor network $\mu_\theta(s)$ to deterministically output an action given a state. That is, for every state s, the actor gives exactly one action $a = \mu_\theta(s)$. There's no sampling involved at inference time.

However, training a deterministic actor introduces new challenges. In particular, the actor will not explore by itself. So DDPG introduces explicit noise during data collection. Also, because we're learning with function approximation, training can become unstable — so the algorithm introduces several techniques to stabilize learning.

DDPG is an off-policy actor-critic algorithm, which means it separately maintains a policy (the actor) and a value function (the critic), and it uses experience replay to update them.

****Question:** Why does DDPG need to add noise for exploration? Why is stochasticity not built-in?**

Because the actor is deterministic: $\mu(s)$ always outputs the same action. Without injected noise, the agent will always follow the same trajectory, never exploring alternatives. Unlike stochastic policies that explore naturally via sampling, DDPG must manually inject exploration.

2. Key ideas in DDPG (What makes DDPG work?)

We now walk through the core design ideas that make DDPG effective. These are not architectural components, but conceptual insights that guide the algorithm.

(1) Deterministic Policy Gradient: Instead of optimizing a stochastic $\pi(a|s)$, DDPG uses a deterministic mapping $a = \mu(s)$, and updates the policy by directly differentiating through the Q function:

$$\nabla_\theta J(\theta) = \mathbb{E}_s \left[\nabla_a Q(s,a)|_{a=\mu(s)} \cdot \nabla_\theta \mu_\theta(s) \right]$$

This reduces variance in gradient estimation and avoids sampling overhead.

(2) Off-policy learning with replay buffer: DDPG is off-policy. It collects transitions using a noisy version of the policy, but updates both actor and critic using samples drawn from a replay buffer. This allows reusing old transitions and improves sample efficiency.

(3) Bootstrapped Q-learning for critic: The critic is trained via TD targets:

$$y = r + \gamma Q_{\phi'}(s', \mu_{\theta'}(s'))$$

This introduces a stability–bias tradeoff: learning is faster, but instability is possible.

(4) Stabilization via target networks: Instead of using the latest parameters for targets, DDPG slowly updates separate networks for both actor and critic:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \quad \phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

This prevents divergence from rapidly shifting targets.

(5) Explicit exploration through noise injection: Since the policy is deterministic, it will always output the same action for a given state. So DDPG adds noise during data collection:

$$a = \mu_\theta(s) + \mathcal{N}$$

Typically $\mathcal{N}$ is Gaussian noise, although Ornstein–Uhlenbeck was used in the original paper.

****Question:** What are the trade-offs of using deterministic vs. stochastic policies? Which part of DDPG compensates for the lack of built-in exploration?**

Deterministic policies reduce gradient variance and are more sample-efficient in theory. However, they lack inherent exploration. DDPG compensates for this by adding noise manually during data collection.

****Question:** What would happen if we removed the target networks? Or didn't use a replay buffer?**

Removing target networks would cause instability, as the critic would be trained using targets that change every step. Removing the replay buffer would make learning on-policy — updates would be highly correlated and lead to poor convergence.

3. Step-by-step training procedure

Step 1: Collect data from the environment. At each environment step, the agent observes a state s, produces an action $a = \mu_\theta(s) + \mathcal{N}$, executes the action, receives a reward r and next state s', and stores the transition (s, a, r, s') into the replay buffer.

Step 2: Sample a minibatch from the replay buffer. We randomly select a batch of transitions $\{(s_i, a_i, r_i, s'_i)\}$ to update the networks.

Step 3: Update the critic. The critic is updated to minimize the Bellman error:

$$L(\phi) = \frac{1}{N} \sum_i \left(Q_\phi(s_i, a_i) - y_i\right)^2,$$

where the target y_i is computed using the target networks:

$$y_i = r_i + \gamma Q_{\phi'}(s'_i, \mu_{\theta'}(s'_i)).$$

Step 4: Update the actor. We update the actor using the deterministic policy gradient, by backpropagating through both the critic and actor.

Step 5: Update the target networks. The target actor and critic are softly updated towards the online networks.

4. Understanding the actor gradient

To improve the policy, we want the actor to output actions that yield higher Q values. Since the Q function is differentiable, we can compute how the output of the actor affects Q, and use that to adjust the actor.

This gives:

$$\nabla_\theta Q(s, \mu_\theta(s)) = \nabla_a Q(s, a)|_{a=\mu(s)} \cdot \nabla_\theta \mu_\theta(s)$$

This is why the actor and critic must be differentiable and trained jointly.

This gradient also motivates a supervised-learning perspective: we can interpret DDPG as regressing $\mu(s)$ towards the action that maximizes Q. This helps understand the actor update as "amortized Q maximization."

****Question:** What does the critic gradient tell the actor? How is this different from REINFORCE-style updates?**

The critic gradient tells the actor which direction in action space improves future reward. In REINFORCE, we use sampled return and weight by $\log \pi$ gradient. In DDPG, we use direct gradient from Q, which is lower variance but needs differentiability.

5. Relation to PPO

DDPG and PPO both use actor-critic structures, but differ in core mechanisms:

- **Policy type**: PPO is stochastic, DDPG is deterministic
- **Exploration**: PPO samples from policy; DDPG adds noise externally
- **Update method**: PPO uses clipped surrogate loss; DDPG uses direct gradient through Q
- **Critic**: PPO learns V(s); DDPG learns Q(s,a)

- **Training style**: PPO is on-policy; DDPG is off-policy

****Question:** What stabilizes PPO's updates? What plays that role in DDPG?**
PPO uses a clipped objective to prevent the policy from changing too fast. In DDPG, stability comes from target networks and replay buffer, which prevent divergence in Q-value learning.

6. When would you prefer DDPG over PPO?

DDPG is more sample efficient due to replay, and better suited to fine-grained continuous control. But it's harder to tune, more sensitive to hyperparameters, and requires careful balancing of noise and learning rate.

PPO is often more robust and easier to use, but requires more environment interaction since it's on-policy.

7. Summary

DDPG introduces a deterministic policy gradient method that combines off-policy data usage, Q-learning updates, and direct actor adjustment using differentiable critics.

The core ideas — deterministic updates, replay, bootstrapped targets, target networks, and exploration via noise — make DDPG distinct from other policy gradient methods like PPO.

Understanding these elements gives us a foundation to study stronger extensions like TD3 and SAC.