Topics in Reinforcement Learning

Paper Presentation: Deterministic Policy Gradient and Goal Relabelling

Group 7

Himanshu Singh Vikaskumar Kalsariya

April 15, 2025

Table of Contents

Recap: Policy Gradients

- Previous policy gradient methods (e.g., REINFORCE) primarily assumed stochastic policies.
- Stochastic policies inherently perform exploration by sampling actions.
- Resulting algorithms are often on-policy.
- Limitations of on-policy algorithms:
  - Sample inefficiency (data used once).
  - Exploration can be "unguided" and slow.

The Challenge with Continuous Actions

In continuous action spaces, i.e. $a \in \mathbb{R}^m$, finding $\text{argmax}_a Q^{\mu_\theta}(s, a)$ (needed for Q-learning or greedy policy improvement) requires optimization at each step, which is often computationally expensive, especially with complex function approximators.

Alternative: Deterministic Policies

- Consider a deterministic policy $\mu : \mathcal{S} \to \mathcal{A}$, parameterized by θ:

$$a = \mu_\theta(s)$$

- **Advantage:** For a given state s, the action is fixed. No sampling needed for action selection.
- **Challenge:** How to ensure exploration? Deterministic policies don't explore naturally.
- **Solution:**
  - Use an off-policy approach. Enables techniques like experience replay.
  - Use a separate *stochastic behavior policy* $b(a|s)$ for exploration during training (e.g., adding noise to $\mu_\theta(s)$).

Deterministic Policy Gradient Theorem

Theorem (Deterministic Policy Gradient [1])

Let $J(\theta)$ be the performance objective (expected discounted return) for a deterministic policy $\mu_\theta(s)$. Assuming the MDP satisfies the regularity conditions of continuity and boundedness in all parameters and variables, the gradient of $J(\theta)$ is given by:

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

- **Proof Sketch:**
  - Differentiate the value function to get a recursive equation.
  - Unroll the above equation to get an infinite sum.
  - Represent the objective in terms of probability of visiting a state and its value.
  - Differentiate the objective and substitute the derivative of value function.

Exploration with DPG

- Since the learned policy $\mu_\theta(s)$ is deterministic, exploration must be added *externally*.
- **Standard Approach:** Add noise to the actor's output action during training.

$$a_t = \mu_\theta(s_t) + \mathcal{N}_t$$

  where $\mathcal{N}_t$ is a noise process.
- **Alternate Approach:** Ornstein-Uhlenbeck (OU) Process.
  - Generates temporally correlated noise, suitable for physical control problems with inertia.
  - Discrete-time update: $\nu_{k+1} = \lambda\nu_k + \sigma\epsilon_k$, where $\epsilon_k \sim \mathcal{N}(0, I)$.
  - Parameters: λ (mean reversion), σ (volatility).
- Learning is off-policy because the data is generated by $a_t \sim b(a|s_t)$, not $a_t = \mu_\theta(s_t)$.

DPG Algorithm

Algorithm 1 Deterministic Policy Gradient

Require: Differentiable policy $\mu_\theta(s)$, action-value function $Q(s, u, w)$.

Require: Step sizes α_θ, α_w. Discount γ.

Initialize actor parameters θ, critic parameters w.
for each episode **do**
  Initialize state s_0.
  for $k = 0, 1, \ldots, T-1$ **do**
    Select action $u_k = \mu_\theta(s_k) + \text{Noise}$ {Exploration}
    Execute u_k, observe reward r_k and next state s_{k+1}.
    Choose next action for target: $u' = \mu(s_{k+1}, \theta)$ {Deterministic target action}
    Calculate TD Error: $\delta \leftarrow r_k + \gamma Q(s_{k+1}, u', w) - Q(s_k, u_k, w)$.
    Update Critic: $w \leftarrow w + \alpha_w \delta \nabla_w Q(s_k, u_k, w)$. {TD Learning}
    Update Actor: $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \mu_\theta(s_k) \nabla_u Q(s_k, u, w)|_{u=\mu_\theta(s_k)}$. {DPG Update}
  end for
end for

Motivation: DPG + Deep Learning

- DPG provides a policy gradient for deterministic policies, suitable for continuous actions.
- Deep Q-Networks (DQN) demonstrated success using deep neural networks for value function approximation in high-dimensional state spaces (e.g., pixels) [2].
- **Goal:** Stable and efficient off-policy learning for continuous control in complex environments.
  - Combine the DPG actor-critic approach with the techniques that stabilized DQN [3].
  - Use deep neural networks for both actor ($\mu_\theta(s)$) and critic ($Q(s, a, w)$).
  - Adapt DQN's stabilization techniques: Experience Replay Buffer, Target Networks.

DDPG: Key Ideas

Idea #1: Experience Replay Buffer

- Store transitions $(s_t, u_t, r_{t+1}, s_{t+1})$ in a finite-sized buffer $\mathcal{D}$.
- Sample mini-batches uniformly from $\mathcal{D}$ to update networks.
- Breaks correlations between consecutive samples $\Rightarrow$ more stable training.
- Improves sample efficiency by reusing past experience.

Idea #2: Target Networks

- Use copies of actor and critic networks: $\mu'(s, \theta')$ and $Q'(s, a, w')$ to compute the TD target.

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

- Stabilizes learning by decoupling target calculation from rapidly changing main network weights.
- Soft updates are used for target networks (w', θ'), with a small update coefficient $\tau \ll 1$.

$$w' \leftarrow \tau w + (1 - \tau) w'$$
$$\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$$

Idea #3: Batch Normalization

- Normalize feature scales across different environments using the statistics computed from mini-batches.
- Keep track of running mean and variance for scaling during inference time.
- Borrows from research in deep learning on reducing covariate shift and speeding up training [4].

Idea #4: Exploration Noise

- Treat the problem of exploration independently from the learning algorithm.
- Similar to DPG, add noise to actions during training.

$$u_t = \mu(s_t, \theta) + \mathcal{N}$$

DDPG: Visual Summary

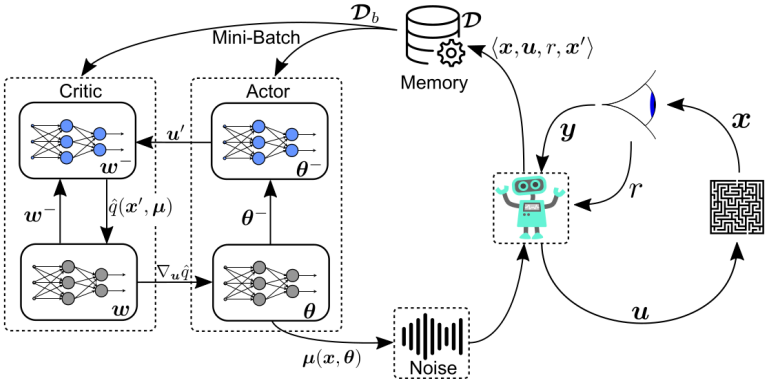


Figure 1: DDPG Structure (Source: Oliver Wallscheid's slides on Reinforcement Learning).

DDPG Algorithm

Algorithm 2 Deep Deterministic Policy Gradient

Initialize replay buffer $\mathcal{D}$, critic $Q(s, u, w)$ and actor $\mu_\theta(s)$ with random weights w, θ.
Initialize target networks Q' and μ' with weights $w' \leftarrow w$, $\theta' \leftarrow \theta$.
for episode = 1, M **do**
    Initialize noise process $\mathcal{N}$. Receive initial state s_1.
    for t = 1, T **do**
        Select action $u_t = \mu(s_t, \theta) + \mathcal{N}_t$.
        Execute u_t, observe reward r_t and next state s_{t+1}.
        Store transition (s_t, u_t, r_t, s_{t+1}) in $\mathcal{D}$.
        Sample a random mini-batch of N transitions (s_i, u_i, r_i, s_{i+1}) from $\mathcal{D}$.
        Set target $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}, \theta'), w')$.
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, u_i, w))^2$.
        Update the actor policy using the sampled policy gradient:

$$\nabla_\theta J \approx \frac{1}{N} \sum_i \nabla_\theta \mu(s_i, \theta) \nabla_u Q(s_i, u, w)|_{u=\mu(s_i, \theta)}$$

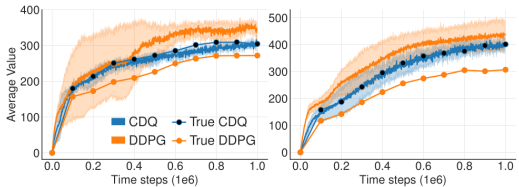
        Update the target networks:

$$w' \leftarrow \tau w + (1 - \tau) w'$$
$$\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$$

    end for
end for

Problem 1: Overestimation Bias

- **Q-Learning (Recap):** Maximization step in target calculation leads to systematic overestimation bias if initial value estimates are noisy.
- **Does this happen in Actor-Critic?** Yes!
  - Function Approximation Error: Neural network critics $Q_w(s, a)$ inevitably have errors.
  - Policy Updates Interact with Error: If Q_w has positive errors in some regions, the actor might exploit these errors, leading to suboptimal policies that work with the flawed critic.



(a) Hopper-v1 (b) Walker2d-v1

Figure 2: DDPG often overestimates Q-values compared to true Monte Carlo returns [5].

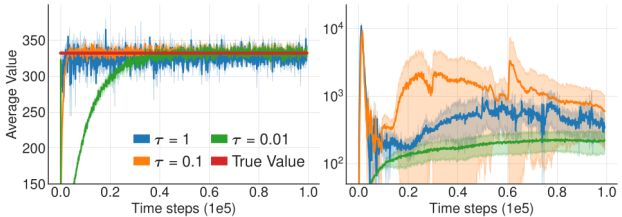
Problem 2: High Variance from TD Errors

- Bellman equation is rarely satisfied due to functional approximation. There's always some residual TD error $\delta(s, u)$. This error accumulates over time during bootstrapping.

$$\begin{aligned} Q_w(s_t, u_t) &= r_t + \gamma \mathbb{E}_{s_{t+1}}[Q_w(s_{t+1}, u_{t+1})] - \delta_t \\ &= \mathbb{E}_\pi \left[\sum_{k=t}^{T} \gamma^{k-t} (r_k - \delta_k) \right] \end{aligned}$$

- The variance of the Q-estimate Q_w depends on the variance of future rewards *and* the variance of future TD errors. High variance would lead to noisy actor gradients and unstable policy updates.
- Target networks help by slowing down changes in the target value, reducing the error introduced at each step, but variance can still accumulate.
- Frequent policy updates can exacerbate this by constantly changing the "true" value target Q^π, making it harder for the critic to track.

Problem 2: High Variance from TD Errors



(a) Fixed Policy (b) Learned Policy

Figure 3: Frequent policy and target updates (high value of τ) can increase instability of the estimated value [5].

.

TD3: Addressing DDPG's Flaws

Goal: Improve DDPG by specifically targeting overestimation bias and high variance.

Three Key Modifications:

1. **Clipped Double Q-Learning:** Mitigate actor exploitation of critic errors (address overestimation).
2. **Delayed Policy Updates:** Update policy less frequently than critic (reduce variance, stabilize targets).
3. **Target Policy Smoothing:** Regularize critic by smoothing Q-estimates w.r.t. actions (reduce variance, prevent sharp peaks).

Modification 1: Clipped Double Q-Learning

- Problem: Standard target network update in DDPG ($y = r + \gamma Q'(s', \mu'(s', \theta'))$) can still overestimate. Double DQN idea (using current policy μ with target critic Q') is ineffective because μ and μ' are too similar.
- **TD3 Approach:**
  - Learn **two** independent critic networks Q_{w_1}, Q_{w_2}, along with their target networks $Q'_{w'_1}, Q'_{w'_2}$.
  - When computing the TD target y, use the minimum of the two target critic values:

$$y = r + \gamma \min_{i=1,2} Q'_{w'_i}(s', \tilde{a})$$

  - Both critics Q_{w_1}, Q_{w_2} are updated towards this same target y.
- **Why?**
  - One critic might overestimate, but it's less likely both overestimate identically.
  - Helps select actions with lower variance Q-value estimates (minimum of two random variables is lower if variance is high).

Modification 2: Target Policy Smoothing

- Problem: Deterministic policies μ can overfit to narrow peaks in the value estimate Q_w. Small errors in Q_w can lead to large errors in the target y.
- **TD3 Approach:** Smooth the target value calculation *before* evaluating the target critics.
  - Compute target action with noise:

$$\tilde{a} = \mu'(s', \theta') + \epsilon$$
$$\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2), -c, c)$$

  - Use this noisy action $\tilde{a}$ in the Clipped Double Q target:

$$y = r + \gamma \min_{i=1,2} Q'_{w'_i}(s', \tilde{a})$$

- **Why?**
  - This allows our deterministic policy to explore more states as well.
  - Forces the Q-function to be smoother in the action dimension around the policy's actions.

Modification 3: Delayed Policy Updates

- Problem: Actor (policy) and Critic (value) can interact poorly. If the Q-estimate is temporarily inaccurate, updating the policy based on it can lead the policy astray. This bad policy then generates poor data, making it harder for the critic to improve. Feedback loop.
- **TD3 Approach:** Update the policy (actor) and target networks less frequently than the value (critic) networks.
  - Update critic networks (Q_{w_1}, Q_{w_2}) at every step.
  - Update actor network (μ_θ) and target networks ($Q'_{w'_1}, Q'_{w'_2}, \mu'_{\theta'}$) only every d critic updates (e.g., $d = 2$).
- **Why?**
  - Allows the critic(s) more time to converge to a better value estimate based on a relatively stable policy before the policy itself is updated.
  - Prioritizes reducing value error before updating the policy.
  - Leads to higher quality policy updates and overall more stable learning. Reduces variance propagation.

TD3 Algorithm Summary

Algorithm 3 Twin Delayed Deep Deterministic Policy Gradient

Initialize critic networks Q_{w_1}, Q_{w_2}, actor network μ_θ.
Initialize target networks $w_1' \leftarrow w_1, w_2' \leftarrow w_2, \theta' \leftarrow \theta$.
Initialize replay buffer $\mathcal{D}$.
for t = 1, T **do**
    Select action with exploration noise: $u_t = \mu(s_t, \theta) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma_{explore})$.
    Execute u_t, observe r_{t+1}, s_{t+1}. Store $(s_t, u_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$.
    Sample mini-batch of N transitions (s, u, r, s') from $\mathcal{D}$.
    Compute target action with noise: $\tilde{a} \leftarrow \mu'(s', \theta') + \epsilon'$, $\epsilon' \sim \text{clip}(\mathcal{N}(0, \sigma_{target}), -c, c)$. Clip $\tilde{a}$ to valid action range.
    Compute target Q value: $y \leftarrow r + \gamma \min_{i=1,2} Q'_{w_i'}(s', \tilde{a})$. {Clipped Double Q + Target Smoothing}
    Update critics Q_{w_1}, Q_{w_2} using gradient descent on $\frac{1}{N}\sum(y - Q_{w_i}(s, u))^2$.
    if t mod d == 0 **then**
        Update actor μ_θ using the deterministic policy gradient w.r.t. Q_{w_1}:

$$\nabla_\theta J \approx \frac{1}{N} \sum \nabla_\theta \mu_\theta(s) \nabla_u Q_{w_1}(s, u)|_{u=\mu_\theta(s)}$$

        Update target networks using soft updates ($\tau \ll 1$):

$$w_i' \leftarrow \tau w_i + (1 - \tau) w_i' \quad \text{for } i = 1, 2$$
$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$

    end if
end for

TD3 Convergence Sketch (Tabular Clipped Double Q)

- **Foundation:** Leverages standard stochastic approximation theory (conditions ensuring iterative updates converge).
- **Goal:** Show the Q-value error $\Delta_t = Q_t - Q^*$ converges towards 0.
- **Proof Outline:**
  1. Show the update rule fits the stochastic approximation framework (requires specific learning rate conditions).
  2. Prove that the difference between the two critic estimates ($\Delta^{BA} = Q^B - Q^A$) converges to zero.
  3. The update step pulls both $Q^A(s, a)$ and $Q^B(s, a)$ towards this common target y.
  4. Argue that once $Q^A \approx Q^B$, the update resembles standard Q-learning (known to converge under tabular conditions).

Motivation: The Sparse Reward Problem

- Many real-world tasks (especially robotics) have sparse rewards: reward is 0 until the task is fully completed, then maybe 1.
- Example: Robot arm needs to push a block to location G. Reward is 0 unless the block is exactly at G.
- Standard RL algorithms (DQN, DDPG, TD3) struggle immensely with sparse rewards. Why?
  - Learning signal is almost always 0.
  - Extremely unlikely to achieve the goal by random exploration initially.
  - Agent gets no feedback on whether it's making progress.
- Traditional Solution: Reward Shaping [6].
  - Manually design dense reward functions (e.g., reward proportional to negative distance to goal).
  - Requires significant domain expertise and careful tuning.
- **Question:** Can we learn efficiently from sparse (binary) rewards without manual shaping?

HER: The Core Idea - Learning from Failure

- Consider an episode where the agent tried to achieve goal g, took actions $a_1, \ldots, a_T$, visited states $s_1, \ldots, s_T$, and failed (never received positive reward).
- Standard RL: This trajectory provides little learning signal (all rewards were 0 or negative).
- **HER Insight:** The trajectory still contains useful information about how to reach the states that *were* visited.
- **Hindsight Replay Mechanism:**
  1. Execute an episode with the original goal g. Store the trajectory $(s_0, a_0, r_0, s_1, ..., s_T)$.
  2. For each transition (s_t, a_t, r_t, s_{t+1}) in the trajectory:
     - Store it in the replay buffer $\mathcal{D}$ with the original goal g.
     - Sample a set of *additional* goals $\mathcal{G}'$ based on states achieved *in the same episode*.
     - For each additional goal $g' \in \mathcal{G}'$:
     - Calculate a new reward $r'_t = r(s_t, a_t, g')$.
     - Store the modified transition $(s_t, a_t, r'_t, s_{t+1})$ in $\mathcal{D}$ with the hindsight goal g'.
- The agent learns how to achieve goals it accidentally encountered, even when failing at the original task.

HER Requirements: Multi-Goal RL

- HER naturally operates in a multi-goal setting.
- We need policies and value functions that are conditioned on the goal g:
  - Policy: $\pi(s, g)$ or $\mu(s, g)$
  - Q-function: $Q(s, a, g)$
- The reward function $r(s, a, g)$ must be computable for any state-action pair (s, a) and *any potential goal* g.
  - For sparse binary rewards: $r(s, a, g') = 0$ if s_{t+1} satisfies goal g', and -1 otherwise. (Or 1 and 0).
- Need a mapping $m : \mathcal{S} \to \mathcal{G}$ that extracts the achieved goal aspect from a state s.
  - Example: If states s include object position s_{obj} and goals g are desired object positions, then $m(s) = s_{obj}$.

HER Algorithm

Algorithm 4 Hindsight Experience Replay

Require: Off-policy RL algorithm $\mathcal{A}$ (e.g., DDPG, TD3).
Require: Reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$.
Require: Strategy $\mathcal{S}$ for sampling hindsight goals.
Require: Hindsight-to-original goal ratio k.

Initialize algorithm $\mathcal{A}$, replay buffer $\mathcal{R}$.
for episode = 1 to M **do**
    Sample initial state s_0 and goal g. Store episode trajectory $\tau = (s_0, a_0, ..., s_T)$.
    for $t = 0$ to $T - 1$ **do**
        Compute reward $r_t = r(s_t, a_t, g)$.
        Store original transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in $\mathcal{R}$. {Store with original goal}
        Sample set of additional goals $\mathcal{G}' = \mathcal{S}(\text{episode } \tau)$.
        for $g' \in \mathcal{G}'$ **do**
            Compute hindsight reward $r'_t = r(s_t, a_t, g')$.
            Store hindsight transition $(s_t||g', a_t, r'_t, s_{t+1}||g')$ in $\mathcal{R}$. {Store with hindsight goal}
        end for
    end for
    for optimization step = 1 to N **do**
        Sample mini-batch B from $\mathcal{R}$.
        Perform optimization step using $\mathcal{A}$ on B.
    end for
end for

HER: Hindsight Goal Sampling Strategies

How to choose the additional goals g' for replay from an episode $(s_0, ..., s_T)$?

final Sample only the goal corresponding to the final state: $g' = m(s_T)$. Simplest version.

future For each transition (s_t, a_t, s_{t+1}), sample k states $s_{t'}$ from the *rest of the trajectory* $(t' > t)$. Use $g' = m(s_{t'})$ as hindsight goals. Rationale: These are goals achieved "soon" after the transition. $k = 4, 8$ often performs best.

episode For each transition (s_t, a_t, s_{t+1}), sample k states $s_{t'}$ randomly from the *entire episode* $(0 \leq t' \leq T)$. Use $g' = m(s_{t'})$.

random Sample k states $s_{t'}$ encountered previously across *all episodes* stored in the replay buffer. Use $g' = m(s_{t'})$.

HER: Hindsight Goal Sampling Strategies

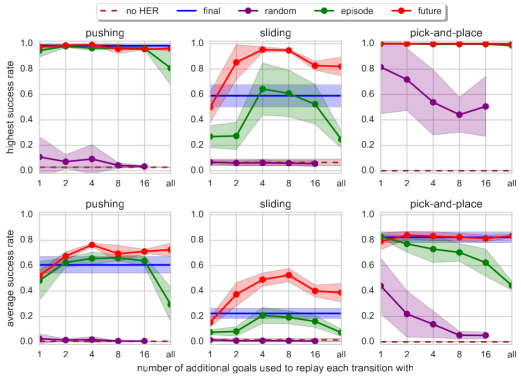


Figure 4: Comparison of HER strategies [7]. The hyperparameter k controls the ratio of HER transitions to original transitions.

TD3 Performance

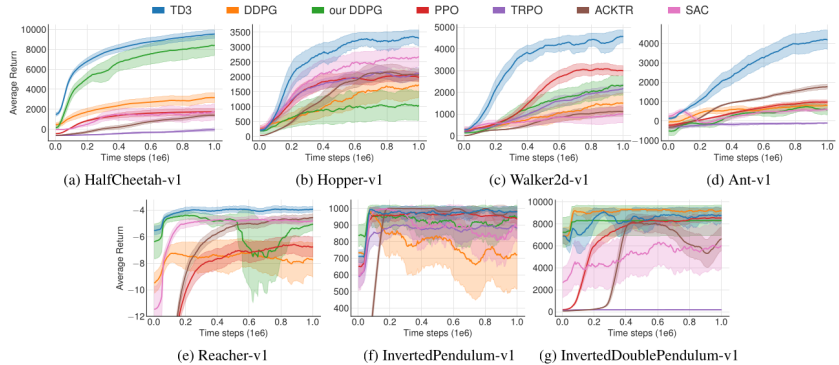


Figure 5: TD3 performance compared to DDPG, PPO, ACKTR, TRPO, SAC on OpenAI Gym MuJoCo tasks [5]. TD3 generally achieves state-of-the-art performance and stability.

HER Performance

Bit Flipping

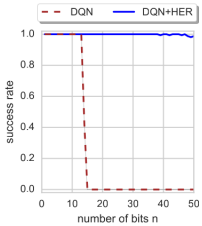


Figure 6: DQN vs DQN+HER on n-bit flipping. HER enables learning for large n [7].

Robotics Tasks (Sparse Reward)

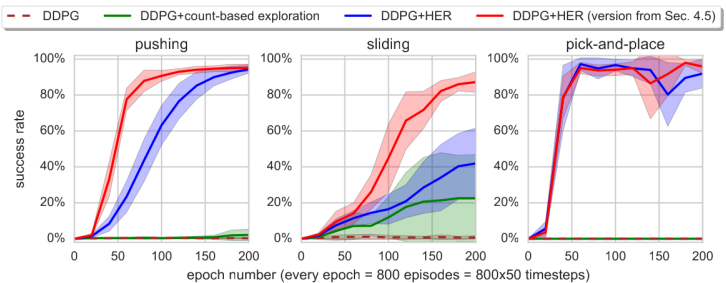


Figure 7: DDPG+HER vs DDPG and DDPG+Counts. HER is crucial for learning complex manipulation with sparse rewards [7].

Summary: What We Learned Today

- **Deterministic Policy Gradient (DPG):** An alternative to stochastic policy gradients, suitable for continuous actions, enabling off-policy learning. Requires external exploration.
- **Deep DPG (DDPG):** Combines DPG with DQN techniques (replay buffer, target networks) for deep RL in continuous control. Often suffers from Q-value overestimation and high variance.
- **Twin Delayed DDPG (TD3):** Addresses DDPG's issues via Clipped Double Q-Learning, Delayed Policy Updates, and Target Policy Smoothing for improved stability and performance.
- **Hindsight Experience Replay (HER):** A technique for learning with sparse rewards by replaying trajectories with achieved goals as desired goals. Compatible with off-policy algorithms like DDPG/TD3. Avoids reward shaping.

Extending DPG/TD3/HER: Future Ideas

- **Synergy between TD3 and HER:**
  - How do TD3's stability improvements interact with HER's relabeling? Does TD3 reduce potential noise introduced by HER?
  - Can the TD3 components (twin critics, smoothing, delay) be adapted specifically for goal-conditioned settings used with HER?
- **Sample Efficiency & Model Use:**
  - Can HER be integrated with model-based RL to generate imagined trajectories that are then relabeled with hindsight goals?
  - More efficient HER sampling: Smarter selection of hindsight goals beyond 'future'? Prioritizing hindsight transitions?

Extending DPG/TD3/HER: Future Ideas (Cont.)

- **Theoretical Understanding:**
  - Convergence analysis for DDPG/TD3 combined with HER, especially with function approximation.
  - Analysis of the implicit curriculum generated by different HER sampling strategies.
- **Beyond Simple Goal Achievement:**
  - Using HER for tasks with more complex success criteria than reaching a specific state/configuration?
  - Hierarchical RL where HER operates at different levels of abstraction? Relabeling sub-goals?
- **Multi-Agent HER:**
  - Applying HER in multi-agent settings. How to define and relabel joint goals or individual goals based on collective outcomes?

References I

[1] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*, Pmlr, 2014, pp. 387–395.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1312.5602.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[4] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: 1502.03167 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1502.03167.

[5] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018. arXiv: 1802.09477 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1802.09477.

[6] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287, ISBN: 1558606122.

[7] M. Andrychowicz, F. Wolski, A. Ray, *et al.*, *Hindsight experience replay*, 2018. arXiv: 1707.01495 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1707.01495.

Thank You!

Questions?