Topics in Reinforcement Learning

Deep RL Project: Fetch Reach and Push

Team 7

Himanshu Singh Vikaskumar Kalsariya

May 6, 2025

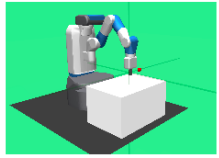
Table of Contents

Environment

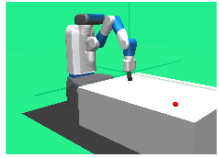
- A block is to be moved to a target position on top of a table by pushing it.
- The manipulator used for this movement has 7-DoF, with a two-fingered parallel gripper.
- The robot is controlled by small displacements of the gripper in Cartesian coordinates. The inverse kinematics are computed internally by the MuJoCo framework.
- The task is also continuing which means that the robot has to maintain the block in the target position for an indefinite period of time.
- The control frequency of the robot is of 25 Hz. The same action is applied in 20 subsequent simulator steps (with a time step of $dt = 0.002s$) before returning the control to the robot.

Environment

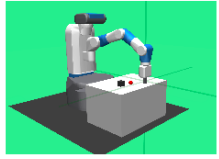
Fetch



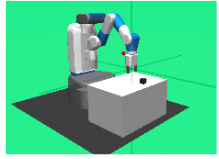
FetchReach-v0: Fetch has to move its end-effector to the desired goal position.



FetchSlide-v0: Fetch has to hit a puck across a long table such that it slides and comes to rest on the desired goal.



FetchPush-v0: Fetch has to move a box by pushing it until it reaches a desired goal position.



FetchPickAndPlace-v0: Fetch has to pick up a box from a table using its gripper and move it to a desired goal above the table.

Figure: The Fetch environment (source: cnblogs).

Action Space

- An action represents the Cartesian displacement dx, dy, and dz of the end effector.
- For some tasks, there is also a last action that controls closing and opening of the gripper.

Action	Control Min	Control Max	Unit
Displacement of the end effector in the x direction	-1	1	position (m)
Displacement of the end effector in the y direction	-1	1	position (m)
Displacement of the end effector in the z direction	-1	1	position (m)

State Space (FetchReach)

- The observation space consists of information about the robot's end effector state and goal.
- Unlike the actions, the observations can take any value in $\mathbb{R}$.

Observation	Unit
Position of the end effector	position (m)
Joint displacement of the left and right gripper finger	position (m)
Linear velocity of the end effector	velocity (m/s)
Linear velocity of the left and right gripper finger	velocity (m/s)
Final goal position of the end effector	position (m)

State Space (FetchPush)

Observation	Unit
Position of the end effector	position (m)
Joint displacement of the left and right gripper finger	position (m)
Linear velocity of the end effector	velocity (m/s)
Linear velocity of the left and right gripper finger	velocity (m/s)
Position of the block	position (m)
Relative block position with respect to the gripper	position (m)
Global rotation of the block in a XYZ Euler frame rotation	angle (rad)
Relative block linear velocity with respect to the gripper	velocity (m/s)
Angular velocity of the block	angular velocity (rad/s)
Final goal position of the block	position (m)

Other Details of the Environment

- The goal is reached if the Euclidean distance between the desired goal and achieved position is lower than 0.05 m.
- The reward can be 'sparse' (0 or -1) or 'dense' (negative Euclidean distance between the achieved position and the desired goal).
- The initial position of the gripper and the base of the robot fixed at $(x, y, z) = [1.34, 0.75, 0.56]m$ and $(x, y, z) = [0.40, 0.48, 0]m$ respectively.
- The orientation (in quaternions) of the gripper and the block is initialized to $(w, x, y, z) = [1.0, 0.0, 1.0, 0.0]$.
- The initial and the target position of the block is set at an offset to the gripper's initial position. The offset is sampled from a uniform distribution with a range of $[-0.15, 0.15]m$.

Hindsight Experience Replay (HER)

Algorithm Hindsight Experience Replay

Require: Off-policy RL algorithm $\mathcal{A}$ (e.g., DDPG, TD3).
Require: Reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$.
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 $\tau = (s_0, a_0, \ldots, s_T)$.
    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 $\mathcal{G}' = \mathcal{S}$(episode τ).
        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

Deep Deterministic Policy Gradient (DDPG)

Algorithm Deep Deterministic Policy Gradient

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

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

13: Update the target networks:

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

14: **end for**
15: **end for**

Twin Delayed DDPG (TD3)

Algorithm Twin Delayed Deep Deterministic Policy Gradient

1: Initialize critic networks Q_{w_1}, Q_{w_2}, actor network μ_θ.
2: Initialize target networks $w_1' \leftarrow w_1, w_2' \leftarrow w_2, \theta' \leftarrow \theta$.
3: Initialize replay buffer $\mathcal{D}$.
4: **for** t = 1, T **do**
5: Select action with exploration noise: $u_t = \mu(s_t, \theta) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma_{explore})$.
6: Execute u_t, observe r_t, s_{t+1}. Store (s_t, u_t, r_t, s_{t+1}) in $\mathcal{D}$.
7: Sample mini-batch of N transitions (s, u, r, s') from $\mathcal{D}$.
8: 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.
9: Compute target Q value: $y \leftarrow r + \gamma \min_{i=1,2} Q'_{w_i'}(s', \tilde{a})$. {Clipped Double Q + Target Smoothing}
10: Update critics Q_{w_1}, Q_{w_2} using gradient descent on $\frac{1}{N}\sum(y - Q_{w_i}(s, u))^2$.
11: **if** t mod d == 0 **then**
12: Update actor μ_θ using the deterministic policy gradient w.r.t. Q_{w_1}:

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

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

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

14: **end if**
15: **end for**

Soft Actor Critic (SAC)

Algorithm Soft Actor-Critic

1: Initialize critic networks Q_{w_1}, Q_{w_2}, actor network $\pi_\theta(a|s)$.
2: Initialize replay buffer $\mathcal{D}$, target networks $w_1' \leftarrow w_1, w_2' \leftarrow w_2$.
3: Initialize temperature parameter α (either fixed or learned). Set target entropy $\mathcal{H}$ if learning α.
4: **for** each interaction step **do**
5: Observe state s_t. Sample action $a_t \sim \pi_\theta(a|s_t)$.
6: Execute a_t, observe r_t, s_{t+1}. Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$.
7: Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{D}$.
8: Sample next action $a' \sim \pi_\theta(a'|s')$ and compute log-probability $\log \pi_\theta(a'|s')$.
9: Compute target value: $y = r + \gamma \left(\min_{i=1,2} Q'_{w_i'}(s', a') - \alpha \log \pi_\theta(a'|s') \right)$.
10: Update critic networks Q_{w_1}, Q_{w_2} by minimizing: $L_Q = \frac{1}{N} \sum_{i=1}^{N} \left((y_i - Q_{w_1}(s_i, a_i))^2 + (y_i - Q_{w_2}(s_i, a_i))^2 \right)$
11: Sample action $\tilde{a} \sim \pi_\theta(\tilde{a}|s)$ (using reparameterization trick) and compute log-probability $\log \pi_\theta(\tilde{a}|s)$.
12: Update actor network π_θ by maximizing the objective:

$$J_\pi(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\min_{j=1,2} Q_{w_j}(s_i, \tilde{a}_i) - \alpha \log \pi_\theta(\tilde{a}_i|s_i) \right)$$

13: Update α by minimizing the loss: $L_\alpha = \frac{1}{N} \sum_{i=1}^{N} (-\log \alpha (\log \pi_\theta(\tilde{a}_i|s_i) + \mathcal{H}))$
14: Update the target networks using soft updates ($\tau \ll 1$):

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

15: **end for**

FetchReach

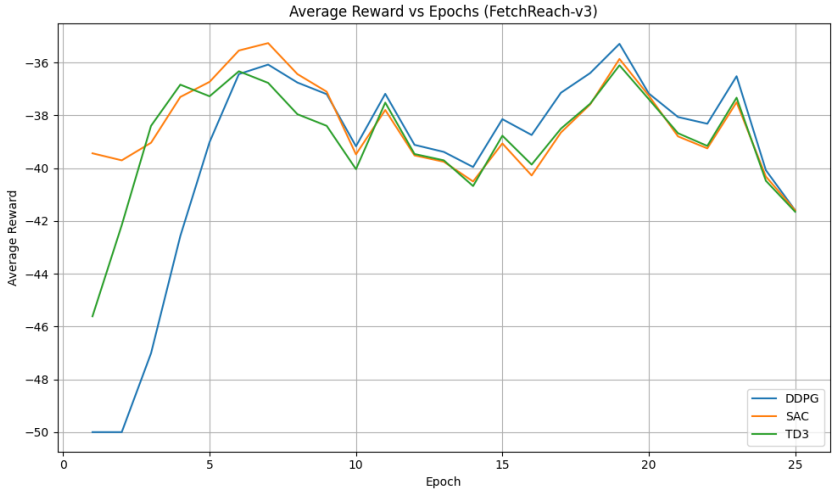


Figure: The average reward accumulated by the agent at each epoch during evaluation.

FetchReach

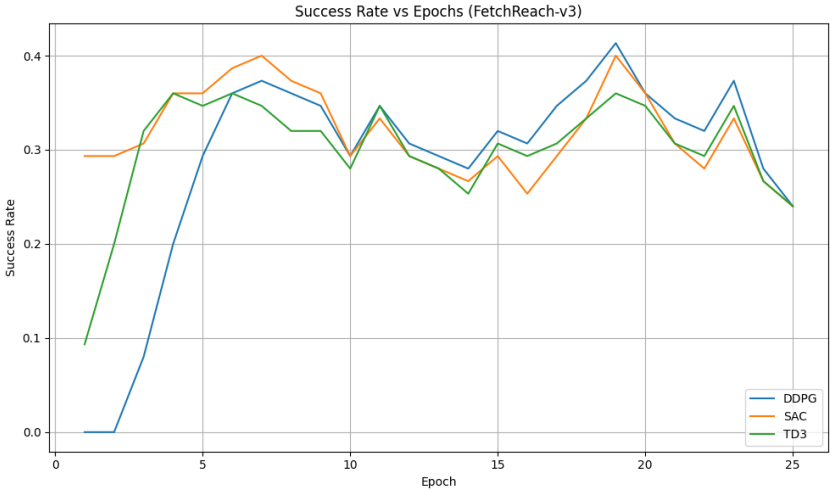


Figure: The success rate of the agent at each epoch during evaluation.

FetchReach

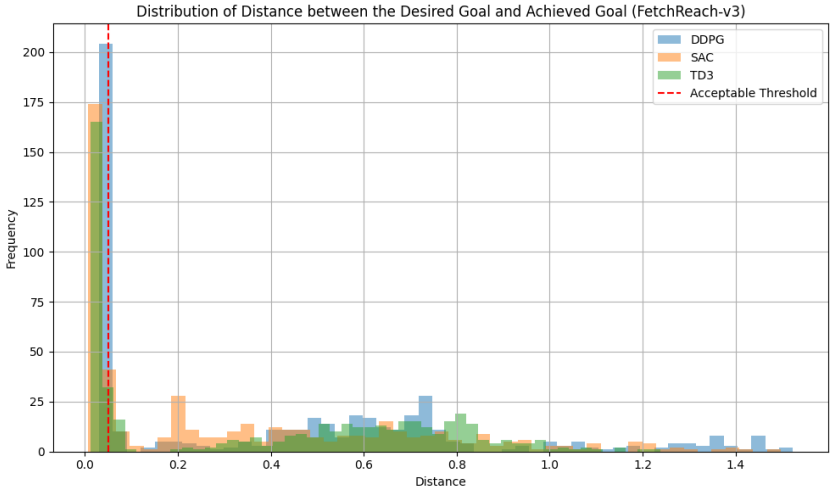


Figure: The distribution of magnitude of error made by the agent after training.

FetchPush

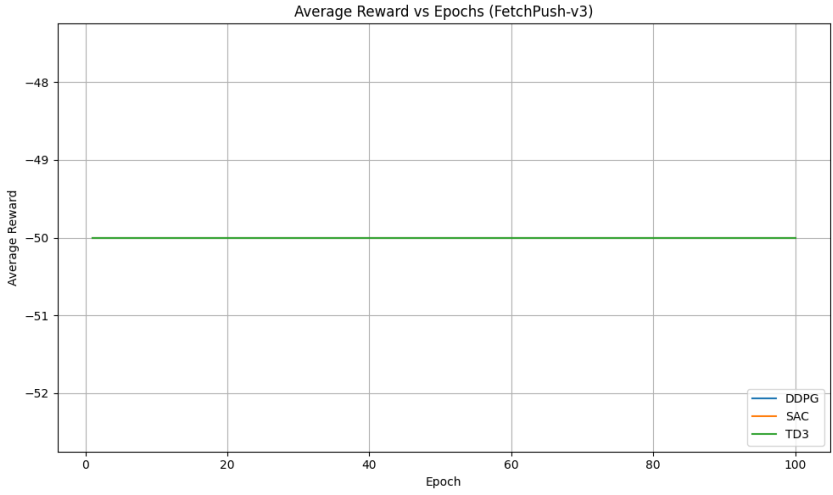
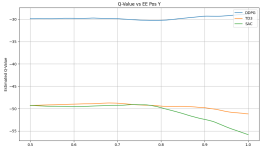
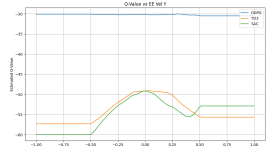


Figure: The average reward accumulated by the agent at each epoch during evaluation.

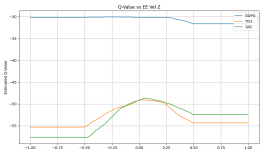
Value Function



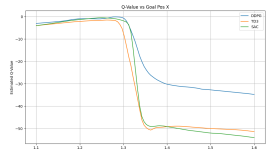
(a) End Effector Position (y component)



(b) End Effector Velocity (y component)



(c) End Effector Velocity (z component)

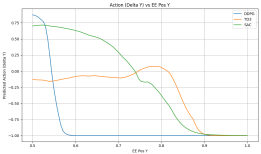


(d) Goal Position (x component)

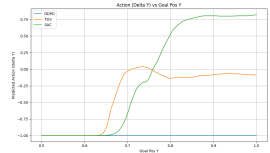
Figure: Variation in Q-Value Estimates[1] along different observation dimensions. Initial EE Position = $[1.34, 0.75, 0.53]$, Goal Position = $[1.40, 0.75, 0.60]$, Velocities assumed to be zero.

[1] Note that the True Q-Values cannot be found since the environment does not allow changing the initial or current state. The only way of manipulating the state is by performing actions.

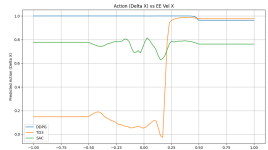
Policy Function



(a) End Effector Position (y component)



(b) Goal Position (y component)



(c) End Effector Velocity (x component)

Figure: Variation in Learned Policy (in the same spatial dimension) along different observation dimensions. Initial EE Position $= [1.34, 0.75, 0.53]$, Goal Position $= [1.40, 0.75, 0.60]$, Velocities assumed to be zero.

Simulation

Simulation of the agent trained on the FetchReach environment can be found at https://drive.google.com/file/d/13NJmlQaMdkiZFKCaScmR6k1ZMuZ1bMqM/.

Conclusion

- Q-Value Overestimation was observed with DDPG, as noted in literature.
- SAC led to better estimates in both value and policy, while TD3 resulted in only better value estimates.
- The networks tend to learn sigmoid like curves over quadratic like curves, contrary to our expectation.
- Transfer learning can be beneficial for training on the FetchPush environment. The agent can learn to utilize the existing knowledge, of moving to the initial position of the block, for the stated objective of displacing the block.
- Alternatively, auxillary rewards, such as for moving the gripper to the initial position of the block, can be helpful for learning the 'first steps'.