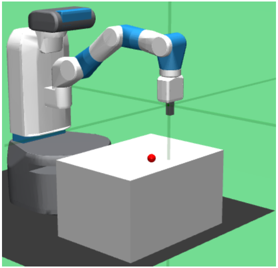
Reinforcement Learning

Alexandru Pascariu, Davide Panarella,
Federica Azzalini, Lorenzo Bonin

September 24, 2021

Problem statement

- Environment
  - 7 DOF open-chain stationary robot
- Task
  - **Fetch and reach**: the robot moves its end-effector to a certain goal position



Tools

- **OpenAi Gym**: Toolkit for simulated robotics environments
- **Mujoco**: Physics engine for model-based control tasks
- **Stable-baselines3**: a set of implementations of RL algorithms [1]

Environment

FetchReach-v1

- **Model-free**, **full observability**
- **Observations**
  - 10 values describing Cartesian position, linear velocity, fingers opening, fingers opening/closing velocity of the gripper
  - 6 values describing target position, achieved goal position
- **Actions**
  - 4 values
  - Desired movement of the gripper + desired distance between the 2 fingers
- **Reward**: 0 if the the goal is achieved (within a tolerance of 5 cm), -1 otherwise.

DDPG-Background

Consider a standard reinforcement learning setup

- Actions $a_t \in R^N$, action space $\mathcal{A} = R^N$
- History of observation, action pairs $s_t = (x_1, a_1, ..., a_{t-1}, x_t)$
  - assume fully-observable so $s_t = x_t$
- Policy $\pi : \mathcal{S} \mapsto \mathcal{P}(\mathcal{A})$
- Environment modeled as Markov Decision Process
  - initial state distribution $p(s_1)$
  - transition dynamics $p(s_{t+1}|s_t, a_t)$
- Discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) \mapsto \gamma \in [0, 1]$
- Goal learn a π which maximizes the expected return from the start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$
- Discounted state visitation distribution for a policy: ρ^{π}

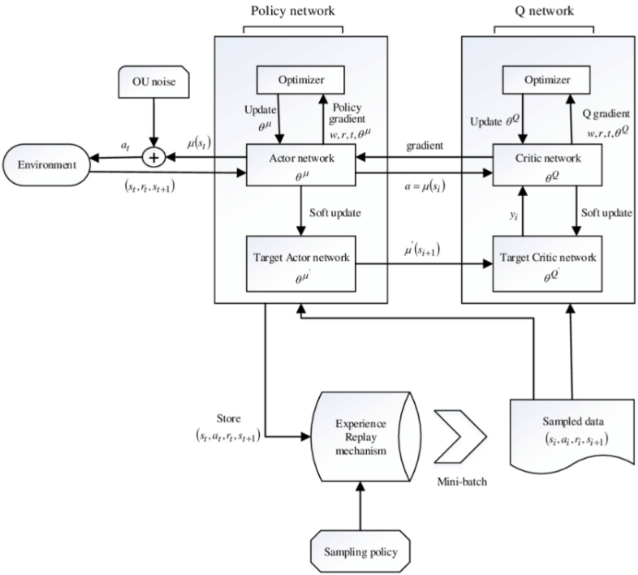
DDPG-Background

- Action-value function: Bellman equation
  $\mathcal{Q}^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[\mathcal{Q}^{\pi}(s_{t+1}, a_{t+1})]]$
- With deterministic policy $\mu : \mathcal{S} \to \mathcal{A}$
  $\mathcal{Q}^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma \mathcal{Q}^{\mu}(s_{t+1}, \mu(s_{t+1}))]$
- Minimizing the loss $L(\theta^{\mathcal{Q}}) = \mathbb{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E}[(\mathcal{Q}(s_t, a_t|\theta^{\mathcal{Q}}) - y_t)^2]$
  where $y_t = r(s_t, a_t) + \gamma \mathcal{Q}(s_{t+1}, \mu(s_{t+1})|\theta^{\mathcal{Q}})$

DDPG-Implementation

- Actor-Critic approach
- Neural network as function approximators
- Replay buffer and minibatch
- Target networks
- Batch normalization
- Noise

DDPG



Policy network
Q network
OU noise
Optimizer
Optimizer
Update θ^μ
Policy gradient w,r,t,θ^μ
Update θ^Q
Q gradient w,r,t,θ^Q
a_t
$\mu(s_t)$
Environment
Actor network θ^μ
gradient
Critic network θ^Q
(s_t, r_t, s_{t+1})
$a = \mu(s_i)$
Soft update
y_i
Soft update
Target Actor network $\theta^{\mu'}$
$\mu'(s_{i+1})$
Target Critic network $\theta^{Q'}$
Store (s_t, a_t, r_t, s_{t+1})
Experience Replay mechanism
Mini-batch
Sampled data (s_i, a_i, r_i, s_{i+1})
Sampling policy

DDPG-Pseudocode I

Algorithm 1: DDPG Algorithm[2]

Randomly initialize critic network $\mathcal{Q}$(s,a$|\theta^{\mathcal{Q}}$) and actor $\mu(s|\theta^{\mu})$ with weights $\theta^{\mathcal{Q}}$ and θ^{μ}.
Initialize target network $\mathcal{Q}$'and μ' with weights $\theta^{\mathcal{Q}'} \leftarrow \theta^{\mathcal{Q}}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay buffer R
for episode = 1 , M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$ according to the current policy and exloration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from R

DDPG-Pseudocode II

Set $y_i = r_i + \gamma \mathcal{Q}'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{\mathcal{Q}'})$
Update critic by minimizing the loss $L = \frac{1}{N}\sum_i (y_i - \mathcal{Q}(s_i, a_i|\theta^{\mathcal{Q}}))^2$
Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a \mathcal{Q}(s, a|\theta^{\mathcal{Q}})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\theta^{\mathcal{Q}'} \leftarrow \tau\theta^{\mathcal{Q}} + (1-\tau)\theta^{\mathcal{Q}'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

end for
end for

Twin Delayed DDPG (TD3)

Quick facts

- Successor of DDPG algorithm
- Model free, Off-policy algorithm
- Only for continuous action spaces
- Combination of Deep Double Q-learning, Policy Gradient, Actor Critic

Twin Delayed DDPG (TD3)

main features

- **Clipped Double Q-learning**: A trick for reducing overestimation bias

  Variant of Double Q-learning

  $\rightarrow$ pair of critic networks estimate the value function and the minimum between the two is chosen for the target update

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

Twin Delayed DDPG (TD3)

main features

- **Target Policy Smoothing**: To address the problem of the policy overfitting to sharp peaks in the value estimate

  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon,\ \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$

- **'Delayed' Policy and Target Updates**: Policy updates are delayed so that the value network becomes more stable before it updates the policy network

Twin Delayed DDPG I

Pseudocode

Algorithm 2: TD3 Algorithm[3]

Initialize critic networks $Q_{\theta 1}$, $Q_{\theta 2}$, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer B
for t = 1 to T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$
  Observe reward r and new state s'
  Store transition tuple (s, a, r, s') from B

Sample mini-bach of N transitions (s, a, r, s') from B
$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
$y \leftarrow r + \gamma min_{i=1,2} Q'_{\theta_i}(s', \tilde{a})$
Update Critics $\theta_i \leftarrow argmin_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
if t mod d **then**
  Update ϕ by the deterministic policy gradient:
  $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_{\phi(s)}} \nabla_\phi \pi_\phi(s)$
  Update target networks:
  $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
  $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
end if
end for

Soft Actor Critic (SAC)

Main features

- Stochastic policy $\pi(a|s)$
- Both for discrete and continuous spaces
- 3 main ingredients:
  - **Actor-critic architecture** with separate policy and value function networks
  - **Off-policy** formulation
  - **Entropy maximization**
    - $H(P) = \underset{x \sim P}{\mathrm{E}}[-\log P(x)]$
    - $\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_t \mathrm{E}_{(s_t, a_t) \sim \rho_\pi} [r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))]$
    - **Temperature parameter** α (exploration/exploitation trade-off)
- **Sample efficient** learning + **robustness**

- **Soft policy evaluation** $\leftrightarrows$ **Soft policy improvement**
- Soft policy evaluation
  - Bellman operator $\mathcal{T}^{\pi}Q(s_t, a_t) \triangleq r(s_t, a_t) + \gamma \mathrm{E}_{s_{t+1}\sim p}[V(s_{t+1})]$ where $V(s_t) = \mathrm{E}_{a_t \sim \pi}[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t)]$
- Soft policy improvement
  - $\pi_{\text{new}} = \underset{\pi' \in \prod}{\text{argmin}} \mathrm{D}_{\text{KL}}\left(\pi'(\cdot|s_t) \middle\| \frac{\exp(\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s_t, \cdot))}{\mathcal{Z}^{\pi_{\text{old}}}(s_t)}\right)$
- **Convergence** to the optimal maximum entropy policy among the policy in $\prod$ (in the tabular case)

Soft Actor Critic (SAC)

How it works-1

- Soft policy iteration + **Function approximation** for both π and Q
- Alternate between optimizing both networks with stochastic gradient descent
- Learning of two Q-functions (clipped double-Q trick)
- Learning of Q: minimize the **soft Bellman** residual

$$J_Q(\theta) = E_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}(Q_\theta(s_t,a_t) - (r(s_t,a_t) + \gamma E_{s_{t+1}\sim p}[V_{\bar{\theta}}(s_{t+1})]))^2\right]$$

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\psi(s_t,a_t)(Q_\theta(s_t,a_t) - (r(s_t,a_t) - \gamma(Q_{\bar{\theta}}(s_{t+1},a_{t+1}) - \alpha\log(\pi_\phi(a_{t+1},s_{t+1}))))$$

where $Q_{\bar{\theta}}$ is the **target soft Q-function**

Soft Actor Critic (SAC)

How it works-2

- Reparameterization trick: $a_t = f_\phi(\epsilon_t; s_t)$, where ϵ is a noise vector sampled from some distribution
- Learning of π: minimize the expected **KL-divergence**
  $J_\pi(\phi) = E_{s_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}}[\alpha \log \pi_\phi(f_\phi(\epsilon_t; s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))]$
  $\hat{\nabla}_\phi J_\pi(\phi) =$
  $\nabla_\phi log \pi_\phi(a_t|s_t) + (\nabla_{a_t} log \pi_\phi(a_t|s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\phi f_\phi(\epsilon_t; s_t)$
- Automatic tuning of α
  - Constant α would need to be tuned for each task
  - Minimization of
    $J(\alpha) = E_{a_t \sim \pi_t}[-\alpha \log \pi_t(a_t|s_t) - \alpha \bar{\mathcal{H}}]$
    where $\bar{\mathcal{H}}$ is the minimum expected entropy
    (*slightly different from the actual implementation*)

Soft Actor Critic (SAC) I

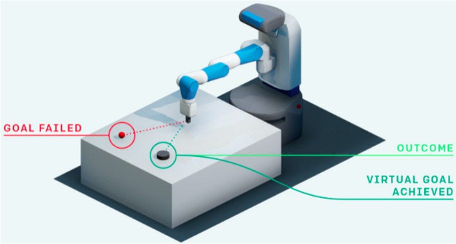
Pseudocode

Algorithm 3: SAC Algorithm [4]

input θ_1, θ_2, ϕ
    $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2, \mathcal{D} \leftarrow \emptyset$
    for each iteration **do**
        for each environment step **do**
            $a_t \sim \pi_\phi(a_t|s_t)$
            $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
            $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
        end for
        for each gradient step **do**
            $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
            $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
            $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$
            $\bar{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\bar{\theta}_i$ for $i \in \{1, 2\}$
        end for
    end for
output θ_1, θ_2, ϕ

Hindsight Experience Replay (HER)

- **Idea**: learn from failure!
- Goal decided in hindsight
- Way better with sparse rewards than with dense ones
- Typically for off-policy algorithms (however, integrations between HER and on-policy algorithms do exist)



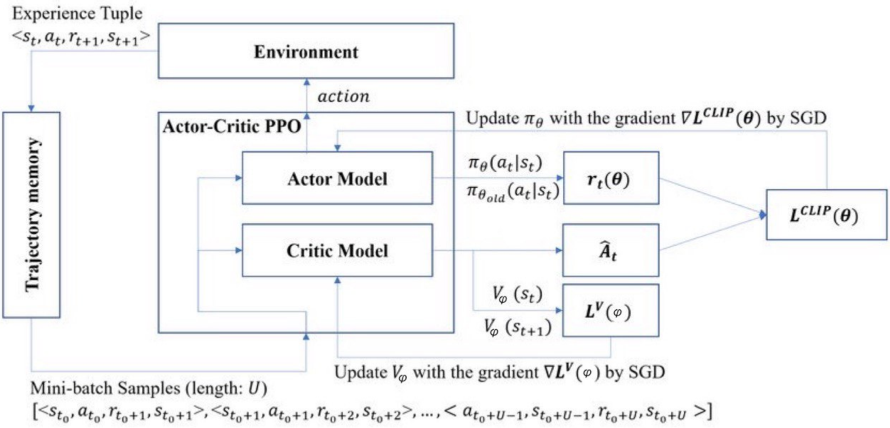
Proximal Policy Optimization

Quick facts

- **On-Policy** algorithm
  - Policy gradient method
  - Sampling data $\leftrightarrows$ optimize objective function
- Optimize the stochastic policy avoiding large updates
  - Constraint embedded in the objective function
- Actor-Critic architecture

Proximal Policy Optimization

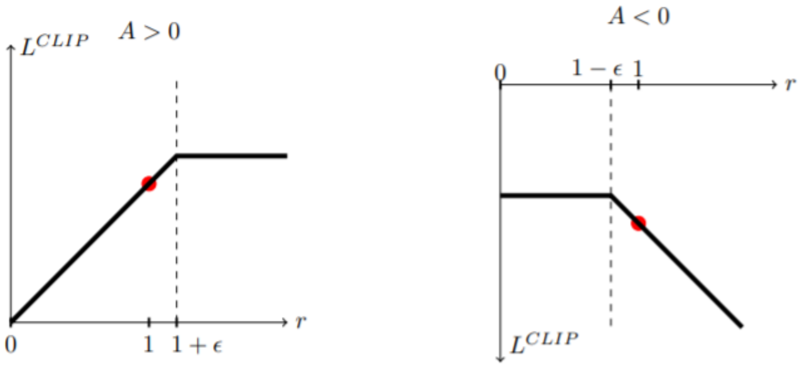
Actor-Critic scheme



Proximal Policy Optimization

Objective function

- Probability ratio : $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$
- Normal policy gradient objective function : $L(\theta) = \hat{\mathbb{E}}_t(r_t(\theta)\,\hat{A}_t)$
  - Advantage estimate : $\hat{A}_t = R_t - V_t$
- PPO objective : $L_{clip}(\theta) = \hat{\mathbb{E}}_t(\min(L(\theta), clip(r_t(\theta), \epsilon)\,\hat{A}_t))$
  - $clip(K, \epsilon) = \begin{cases} (1+\epsilon)K \;\; if \;\; K > 0 \\ (1 - \epsilon)K \;\; if \;\; K < 0 \end{cases}$

- 

$A > 0$
L^{CLIP}
0
1
$1+\epsilon$
r
$A < 0$
0
$1-\epsilon$
1
r
L^{CLIP}

Algorithm 4: PPO Algorithm[5]

initialize policy and value function parameters θ_0, ϕ_0.
for each iteration 1,2,... **do**
  for actor=1,2,..N **do**
    Run policy $\pi_{\theta old}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1,\ldots,\hat{A}_T$ based on $V_{\phi_{old}}$
  end for
  $\theta_{new} = \underset{\theta}{\operatorname{argmax}}(\mathbb{E}_{actor}(L_{clip}(\theta))$ via SGA
  $\phi_{new} = \underset{\phi}{\operatorname{argmin}}(\frac{1}{NT} \sum_{actor=1}^{N} \sum_{t=1}^{T} (V_\phi(S_t) - R_t)^2)$ via SGD
  $\theta_{old} = \theta_{new}$
end for

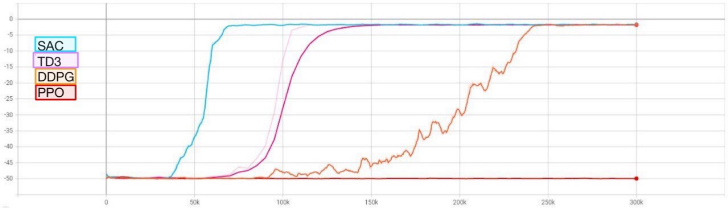


Figure: rew-mean

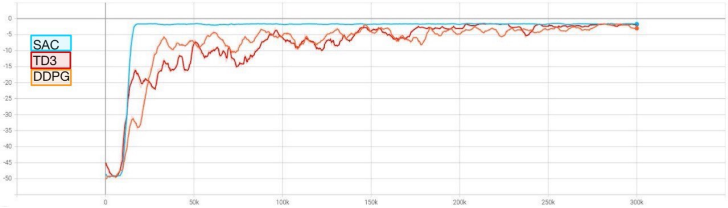


Figure: rew-mean-her

Hyperparameter tuning

- **Optuna** : Automatic hyperparameter optimization software framework[6]
- DDPG, TD3, SAC: hyperparameters from https://github.com/araffin/rl-baselines-zoo/blob/master/hyperparams/her.ymlL125
- PPO: tuned ourselves

Plots and results

Tuned parameters

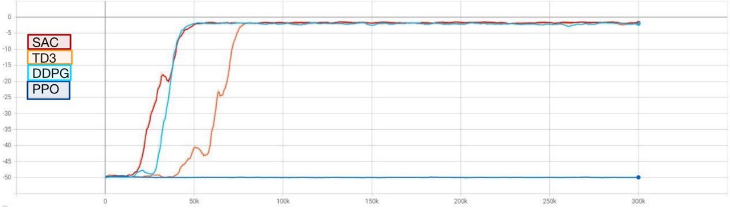


Figure: rew-mean

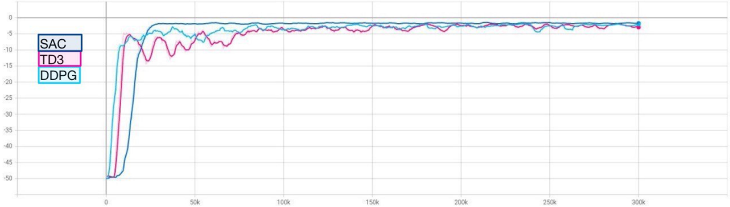


Figure: rew-mean-her

Conclusions

- Off-policy algorithms outperform on-policy algorithms
  - On policy-learning suffers from poor sample efficiency
- DDPG is able to achieve very good performances but it is heavily dependent on its hyperparameters and tends to overestimate bias
- TD3 clearly improves the DDPG robustness to hyperparameters settings
- SAC has very good performances and is pretty independent of hyperparameters tuning
- HER always improves learning speed

We thank you for your kind attention

References

Antonin Raffin et al. *Stable Baselines3*. URL: https://stable-baselines3.readthedocs.io/en/master/index.html.

Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI].

Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications*. 2019. arXiv: 1812.05905 [cs.LG].

John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.