Univ Gustave Eiffel / Cosys

Reinforcement Learning and Optimal Control

Advanced topics in RL

Nadir Farhi
chargé de recherche, UGE - Cosys/Grettia

ENSIA - 21 April 2025

Outline

1. Review & Summary
2. Advanced policy-based :
    - TRPO $\rightarrow$ PPO
    - DDPG

RL Context & optimization

- Marokov Decision Process with states *s* and actions *a*

RL Context & optimization

- Marokov Decision Process with states s and actions a
- optimization in time $\Rightarrow$ Optimal control

RL Context & optimization

- Marokov Decision Process with states s and actions a
- optimization in time $\Rightarrow$ Optimal control
- Under uncertainty $\Rightarrow$ Stochasticity

$$\text{Criterion :} \quad \max_{\pi} \mathbb{E}_{\pi} \sum_{k=0}^{\infty} \gamma^{k} R_{t+k+1}(s_k, a_k)$$

RL Context & optimization

- Marokov Decision Process with states s and actions a
- optimization in time $\Rightarrow$ Optimal control
- Under uncertainty $\Rightarrow$ Stochasticity

$$\text{Criterion :} \quad \max_{\pi} \mathbb{E}_{\pi} \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}(s_k, a_k)$$

$$\text{The dynamics :} \quad p(s', r \mid s, a) \quad \text{known (given)}$$

RL Context & optimization

- Marokov Decision Process with states s and actions a
- optimization in time $\Rightarrow$ Optimal control
- Under uncertainty $\Rightarrow$ Stochasticity

$$\text{Criterion :} \quad \max_{\pi} \mathbb{E}_\pi \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}(s_k, a_k)$$

$$\text{The dynamics :} \quad p(s', r \mid s, a) \quad \text{known (given)}$$

$$\pi \text{ : the policy : prob. distr. on } \mathcal{A}$$

RL Context & optimization

- Marokov Decision Process with states s and actions a
- optimization in time $\Rightarrow$ Optimal control
- Under uncertainty $\Rightarrow$ Stochasticity

$$\text{Criterion :} \quad \max_{\pi} \mathbb{E}_{\pi} \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}(s_k, a_k)$$

$$\text{The dynamics :} \quad p(s', r \mid s, a) \quad \text{known (given)}$$

$$\pi \text{ : the policy : prob. distr. on } \mathcal{A}$$

- Known dynamics (a priori) $\Rightarrow$ Dynamic programming

RL Context & optimization

- Marokov Decision Process with states s and actions a
- optimization in time $\Rightarrow$ Optimal control
- Under uncertainty $\Rightarrow$ Stochasticity

$$\text{Criterion :} \quad \max_{\pi} \mathbb{E}_{\pi} \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}(s_k, a_k)$$

$$\text{The dynamics :} \quad p(s', r \mid s, a) \quad \text{known (given)}$$

$$\pi \text{ : the policy : prob. distr. on } \mathcal{A}$$

- Known dynamics (a priori) $\Rightarrow$ Dynamic programming
- Unknown dynamics $\Rightarrow$ Monte carlo sampling (simulation)

RL Context & optimization

- Marokov Decision Process with states s and actions a
- optimization in time $\Rightarrow$ Optimal control
- Under uncertainty $\Rightarrow$ Stochasticity

$$\text{Criterion :} \quad \max_{\pi} \mathbb{E}_{\pi} \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}(s_k, a_k)$$

$$\text{The dynamics :} \quad p(s', r \mid s, a) \quad \text{known (given)}$$

$$\pi \text{ : the policy : prob. distr. on } \mathcal{A}$$

- Known dynamics (a priori) $\Rightarrow$ Dynamic programming
- Unknown dynamics $\Rightarrow$ Monte carlo sampling (simulation)
- RL : combines ideas of DP and MC

Dynamic programming (Review)

- Action-value function :

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right].$$

Dynamic programming (Review)

- Action-value function :

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right].$$

- Bellman equation :

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right], \end{aligned}$$

Dynamic programming (Review)

- Action-value function :

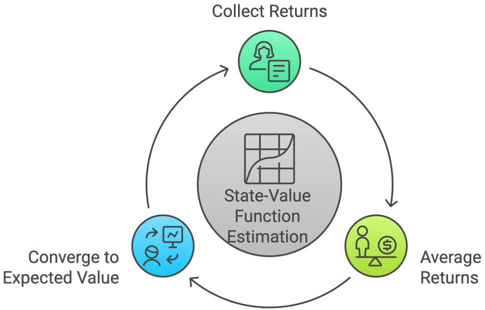
$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right].$$

- Bellman equation :

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right], \end{aligned}$$

1. Policy evaluation (Prediction)
2. Policy improvement
3. Policy iteration & GPI
4. Value iteration

Monte Carlo (Review)



- Agent interaction
- Experience collection
- No complete knowledge needed

Q-learning : Off-policy TD Control (Watkins, 1989)

Off-policy learning considers two policies :

- Target policy : learned policy $\rightarrow$ the optimal policy.
- Behavior policy : exploratory policy used to generate behavior.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Vanilla Algorithm (2015)

Introduce NN + Solve Moving targets issue + Solve Successively correlated data

Vanilla DQN algorithm

Initialize step $t = 0$;
Initialize the online Q-network Q with random weights Θ_0;
Initialize the target $\hat{Q}$-network $\hat{Q}$ with weights $\Theta_0^- = \Theta_0$;
Initialize the replay memory buffer D to capacity N
with $Nmin$ random transitions $(s, \text{rand } a \in A, s', r')$;
for episode e = 1:E **do**
  Initialize sequence, observe initial state $s_t = \phi(x_t)$;
  while s_t not terminal **do**
    With probability ϵ select a random action $a_t \in A$
    otherwise select action $a_t = arg\max_a Q_{\pi,t}(s_t, a, \Theta_t)$;
    Execute a_t in emulator ε and observe reward r_{t+1} and next state $s_{t+1} = \phi(x_{t+1})$;
    Store transition $(s_t, a_t, s_{t+1}, r_{t+1})$ in D;
    Sample random batch $U(D)$ of M transitions $(s, a, s', r')^{(m)}$ in D;
    for m = 1:M **do**
      Set TD error $\delta_t^{(m)} = r'^{(m)} + \gamma \cdot \max_{a'} Q_{\pi,t}(s'^{(m)}, a', \Theta_t^-) - Q_{\pi,t}(s^{(m)}, a^{(m)}, \Theta_t)$;
    end for
    Perform a gradient descent step on MSE loss $L_t(\delta_t)$ w.r.t. Θ_t, with Δ_t, α;
    if $t \equiv 0 \pmod{C}$ **then**
      Reset target $\hat{Q}$-network $\hat{Q}$ = online Q-network Q, with $\Theta_t^- = \Theta_t$;
    end if
    Decay ϵ with linear decay;
    Increment step $t = t + 1$;
  end while
end for

RL Policy-based methods (or Policy Gradient Methods)

Two familes :

- Reinforce (or VPG) $\rightarrow$ TRPO $\rightarrow$ PPO

RL Policy-based methods (or Policy Gradient Methods)

Two familes :

- Reinforce (or VPG) $\rightarrow$ TRPO $\rightarrow$ PPO
- DPG $\rightarrow$ DDPG $\rightarrow$ TD3 (Twin Delayed DDPG)

RL Policy-based methods (or Policy Gradient Methods)

Two familes :

- Reinforce (or VPG) $\rightarrow$ TRPO $\rightarrow$ PPO
- DPG $\rightarrow$ DDPG $\rightarrow$ TD3 (Twin Delayed DDPG)

+ Hybrid approaches :

- Actor critic $\rightarrow$ SAC (Soft AC), etc.

REINFORCE (Willams, 1992)

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\quad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

- The update increases the parameter vector proportional to the return. (move most in the directions that favor actions that yield the highest return).

REINFORCE (Willams, 1992)

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

- The update increases the parameter vector proportional to the return.
  (move most in the directions that favor actions that yield the highest return).
- and inversely proportional to the action probability.
  (otherwise actions that are selected frequently are at an advantage).

Recent version of REINFORCE (Open AI)

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
9: **end for**

TRPO - Trust Region Policy Optimization (extends Reinforce)

- TRPO extends REINFORCE.

TRPO - Trust Region Policy Optimization (extends Reinforce)

- TRPO extends REINFORCE.
- Two main problems with REINFORCE :

TRPO - Trust Region Policy Optimization (extends Reinforce)

- TRPO extends REINFORCE.
- Two main problems with REINFORCE :
  1. Past trajectories are not rused.

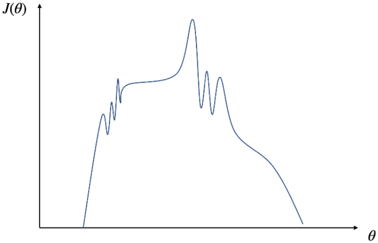
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

TRPO - Trust Region Policy Optimization (extends Reinforce)

- TRPO extends REINFORCE.
- Two main problems with REINFORCE :
  1. Past trajectories are not rused.

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

  2. Instable convergence : Difficule to find appropriate α for all the training process :



TRPO - Trust Region Policy Optimization

Abstract

- REINFORCE keeps new and old policies close in parameter space

TRPO - Trust Region Policy Optimization

Abstract

- REINFORCE keeps new and old policies close in parameter space
- But they can have large differences in performance !

TRPO - Trust Region Policy Optimization

Abstract

- REINFORCE keeps new and old policies close in parameter space
- But they can have large differences in performance !
- So, it is not good to use large step sizes with REINFORCE

TRPO - Trust Region Policy Optimization

Abstract

- REINFORCE keeps new and old policies close in parameter space
- But they can have large differences in performance !
- So, it is not good to use large step sizes with REINFORCE

———

- TRPO updates policies by taking the largest step possible to improve performance

TRPO - Trust Region Policy Optimization

Abstract

- REINFORCE keeps new and old policies close in parameter space
- But they can have large differences in performance !
- So, it is not good to use large step sizes with REINFORCE

- TRPO updates policies by taking the largest step possible to improve performance
- while satisfying a special constraint on how close the new and old policies

TRPO - Trust Region Policy Optimization

Abstract

- REINFORCE keeps new and old policies close in parameter space
- But they can have large differences in performance !
- So, it is not good to use large step sizes with REINFORCE

———

- TRPO updates policies by taking the largest step possible to improve performance
- while satisfying a special constraint on how close the new and old policies
- The constraint is expressed in terms of KL-Divergence (Kullback-Leibler)

  (a kind of measure of distance between probability distributions).

TRPO - Trust Region Policy Optimization

Quick Facts

- TRPO is an on-policy algorithm

TRPO - Trust Region Policy Optimization

Quick Facts

- TRPO is an on-policy algorithm
- TRPO can be used for environments with discrete or continuous action spaces

TRPO - Trust Region Policy Optimization

Quick Facts

- TRPO is an on-policy algorithm
- TRPO can be used for environments with discrete or continuous action spaces
- TRPO uses the hessien (2nd derivatives) in addition to the gradient.

TRPO - Trust Region Policy Optimization

Quick Facts

- TRPO is an on-policy algorithm
- TRPO can be used for environments with discrete or continuous action spaces
- TRPO uses the hessien (2nd derivatives) in addition to the gradient.
- Implementation of TRPO supports parallelization

TRPO - Trust Region Policy Optimization

Theory behind

- π_θ : a policy with parameters θ

TRPO - Trust Region Policy Optimization

Theory behind

- π_θ : a policy with parameters θ
- The theoretical TRPO update is :

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta)$$
$$\text{s.t. } \bar{D}_{KL}(\theta || \theta_k) \leq \delta$$

TRPO - Trust Region Policy Optimization

Theory behind

- π_θ : a policy with parameters θ
- The theoretical TRPO update is :

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta) \quad \text{s.t. } \bar{D}_{KL}(\theta || \theta_k) \leq \delta$$

where $\mathcal{L}(\theta_k, \theta)$ is the surrogate advantage, a measure of how π_θ performs relative to old π_{θ_k} using data from π_{θ_k}.

$$\mathcal{L}(\theta_k, \theta) = \underset{s,a \sim \pi_{\theta_k}}{\mathrm{E}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a) \right]$$

TRPO - Trust Region Policy Optimization

Theory behind

- π_θ : a policy with parameters θ
- The theoretical TRPO update is :

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta) \quad \text{s.t. } \bar{D}_{KL}(\theta || \theta_k) \leq \delta$$

where $\mathcal{L}(\theta_k, \theta)$ is the surrogate advantage, a measure of how π_θ performs relative to old π_{θ_k} using data from π_{θ_k}.

$$\mathcal{L}(\theta_k, \theta) = \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a) \right]$$

and $\bar{D}_{KL}(\theta \,||\, \theta_k)$ is an average Kullback-Leibler (KL)-divergence, or relative entropy, between policies across states visited by the old policy :

$$\bar{D}_{KL}(\theta || \theta_k) = \mathop{\mathrm{E}}_{s\sim\pi_{\theta_k}} \left[D_{KL}\left(\pi_\theta(\cdot|s) || \pi_{\theta_k}(\cdot|s) \right) \right]$$

TRPO - Trust Region Policy Optimization

KL Divergence (relative entropy) :

$$KL(P \mid\mid Q) := \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

TRPO - Trust Region Policy Optimization

KL Divergence (relative entropy) :

$$KL(P \mid\mid Q) := \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

$$= \sum_x P(x) \log P(x) - \sum_x P(x) \log Q(x)$$

TRPO - Trust Region Policy Optimization

KL Divergence (relative entropy) :

$$KL(P \,||\, Q) := \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

$$= \sum_x P(x) \log P(x) - \sum_x P(x) \log Q(x)$$

- $\sum_x P(x) \log P(x)$: The entropy $H(x)$ of $P(x)$.
- $\sum_x P(x) \log Q(x)$: The cross-entropy between P and Q.

TRPO - Trust Region Policy Optimization

Taylor expand the objective and constraint to leading order around :

$$\mathcal{L}_{\theta_{\text{old}}}(\theta) \approx \overbrace{\mathcal{L}_{\theta_{\text{old}}}(\theta_{\text{old}})}^{0} + \nabla_\theta \mathcal{L}_{\theta_{\text{old}}}(\theta)|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}}) + \dots$$

$$\begin{aligned}\bar{D}_{KL}(\theta \| \theta_{\text{old}}) \approx & \overbrace{\bar{D}_{KL}(\theta_k \| \theta_{\text{old}})}^{0} + \overbrace{\nabla_\theta \bar{D}_{KL}(\theta \| \theta_{\text{old}})|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}})}^{0} \\ & + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla^2_\theta \bar{D}_{KL}(\theta \| \theta_{\text{old}})|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}}) + \dots\end{aligned}$$

TRPO - Trust Region Policy Optimization

Taylor expand the objective and constraint to leading order around :

$$\mathcal{L}_{\theta_{\text{old}}}(\theta) \approx \overbrace{\mathcal{L}_{\theta_{\text{old}}}(\theta_{\text{old}})}^{0} + \nabla_\theta \mathcal{L}_{\theta_{\text{old}}}(\theta)|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}}) + \dots$$

$$\begin{aligned}\bar{D}_{KL}(\theta\|\theta_{\text{old}}) \approx\ & \overbrace{\bar{D}_{KL}(\theta_k\|\theta_{\text{old}})}^{0} + \overbrace{\nabla_\theta \bar{D}_{KL}(\theta\|\theta_{\text{old}})\,|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}})}^{0} \\ & + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \bar{D}_{KL}(\theta\|\theta_{\text{old}})\,|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}}) + \dots\end{aligned}$$

$$\begin{aligned}\theta_{k+1} = \arg\max_\theta\ & g^T(\theta - \theta_k) \\ \text{s.t. } & \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \le \delta.\end{aligned}$$

TRPO - Trust Region Policy Optimization

Taylor expand the objective and constraint to leading order around :

$$\mathcal{L}_{\theta_{\text{old}}}(\theta) \approx \overbrace{\mathcal{L}_{\theta_{\text{old}}}(\theta_{\text{old}})}^{0} + \nabla_\theta \mathcal{L}_{\theta_{\text{old}}}(\theta)|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}}) + \dots$$

$$\begin{aligned}\bar{D}_{KL}(\theta \| \theta_{\text{old}}) \approx{} & \overbrace{\bar{D}_{KL}(\theta_k \| \theta_{\text{old}})}^{0} + \overbrace{\nabla_\theta \bar{D}_{KL}(\theta \| \theta_{\text{old}})|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}})}^{0} \\ & + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \bar{D}_{KL}(\theta \| \theta_{\text{old}})|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}}) + \dots\end{aligned}$$

$$\begin{aligned}\theta_{k+1} = \arg\max_\theta \; & g^T(\theta - \theta_k) \\ \text{s.t. } & \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \le \delta.\end{aligned}$$

$$g \doteq \nabla_\theta \mathcal{L}_{\theta_{\text{old}}}(\theta)\big|_{\theta_{\text{old}}} \qquad H \doteq \nabla_\theta^2 \bar{D}_{KL}(\theta \| \theta_{\text{old}})\big|_{\theta_{\text{old}}}$$

TRPO - Trust Region Policy Optimization

Taylor expand the objective and constraint to leading order around :

$$\mathcal{L}_{\theta_{\text{old}}}(\theta) \approx \overbrace{\mathcal{L}_{\theta_{\text{old}}}(\theta_{\text{old}})}^{0} + \nabla_\theta \mathcal{L}_{\theta_{\text{old}}}(\theta)|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}}) + \dots$$

$$\begin{aligned}\bar{D}_{KL}(\theta\|\theta_{\text{old}}) \approx\ & \overbrace{\bar{D}_{KL}(\theta_k\|\theta_{\text{old}})}^{0} + \overbrace{\nabla_\theta \bar{D}_{KL}(\theta\|\theta_{\text{old}})|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}})}^{0} \\ & + \frac{1}{2}(\theta - \theta_{\text{old}})^T \nabla_\theta^2 \bar{D}_{KL}(\theta\|\theta_{\text{old}})|_{\theta_{\text{old}}}(\theta - \theta_{\text{old}}) + \dots\end{aligned}$$

$$\begin{aligned}\theta_{k+1} = \arg\max_\theta\ & g^T(\theta - \theta_k) \\ \text{s.t. } & \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \leq \delta.\end{aligned}$$

$$g \doteq \nabla_\theta \mathcal{L}_{\theta_{\text{old}}}(\theta)\big|_{\theta_{\text{old}}} \qquad H \doteq \nabla_\theta^2 \bar{D}_{KL}(\theta\|\theta_{\text{old}})\big|_{\theta_{\text{old}}}$$

TRPO - Trust Region Policy Optimization

The approximate problem is analytically solved (Eg. Lagrangian duality) :

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g.$$

TRPO - Trust Region Policy Optimization

The approximate problem is analytically solved (Eg. Lagrangian duality) :

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g.$$

Then we add a backtracking line search :

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g,$$

TRPO - Trust Region Policy Optimization

The approximate problem is analytically solved (Eg. Lagrangian duality) :

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g.$$

Then we add a backtracking line search :

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g,$$

- α^j is the backtracking coefficient
- j is the smallest nonnegative integer such that $\pi_{\theta_{k+1}}$ satisfies the KL constraint and produces a positive surrogate advantage.

TRPO - Trust Region Policy Optimization

- Finally, instead of computing and storing H^{-1}, which is expensive.

TRPO - Trust Region Policy Optimization

- Finally, instead of computing and storing H^{-1}, which is expensive.
- TRPO uses the conjugate gradient algorithm to solve $Hx = g$ for $x = H^{-1}g$.

TRPO - Trust Region Policy Optimization

- Finally, instead of computing and storing H^{-1}, which is expensive.
- TRPO uses the conjugate gradient algorithm to solve $Hx = g$ for $x = H^{-1}g$.
- Just set up a symbolic operation to calculate :

$$Hx = \nabla_\theta \left(\left(\nabla_\theta \bar{D}_{KL}(\theta || \theta_k) \right)^T x \right)$$

TRPO - Trust Region Policy Optimization

Algorithm 1 Trust Region Policy Optimization

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0

2: Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K

3: **for** $k = 0, 1, 2, ...$ **do**

4: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

5: Compute rewards-to-go $\hat{R}_t$.

6: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

7: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

8: Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.

9: Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, ...K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

10: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

11: **end for**

PPO - Proximal Policy Optimization

- PPO extends REINFORCE as TRPO

PPO - Proximal Policy Optimization

- PPO extends REINFORCE as TRPO
- Main question :

- How can we take the biggest possible improvement step on a policy,
- using the data we currently have,
- without stepping so far that we accidentally cause performance collapse ?

PPO - Proximal Policy Optimization

- PPO extends REINFORCE as TRPO
- Main question :
  - How can we take the biggest possible improvement step on a policy,
  - using the data we currently have,
  - without stepping so far that we accidentally cause performance collapse ?
- TRPO solves this problem with a complex second-order method (hessien)

PPO - Proximal Policy Optimization

- PPO extends REINFORCE as TRPO
- Main question :
  - How can we take the biggest possible improvement step on a policy,
  - using the data we currently have,
  - without stepping so far that we accidentally cause performance collapse ?
- TRPO solves this problem with a complex second-order method (hessien)
- PPO is a family of first-order methods that use other technics

PPO - Proximal Policy Optimization

- PPO extends REINFORCE as TRPO
- Main question :
  - How can we take the biggest possible improvement step on a policy,
  - using the data we currently have,
  - without stepping so far that we accidentally cause performance collapse ?
- TRPO solves this problem with a complex second-order method (hessien)
- PPO is a family of first-order methods that use other technics
- PPO methods are significantly simpler to implement

PPO - Proximal Policy Optimization

- PPO extends REINFORCE as TRPO
- Main question :
  - How can we take the biggest possible improvement step on a policy,
  - using the data we currently have,
  - without stepping so far that we accidentally cause performance collapse ?
- TRPO solves this problem with a complex second-order method (hessien)
- PPO is a family of first-order methods that use other technics
- PPO methods are significantly simpler to implement
- PPO empirically seem to perform at least as well as TRPO.

PPO - Proximal Policy Optimization

Two variants :

PPO-Penalty :

- Approximately solves a KL-constrained update like TRPO

PPO - Proximal Policy Optimization

Two variants :

PPO-Penalty :

- Approximately solves a KL-constrained update like TRPO
- But penalizes the KL-divergence in the objective function instead of making it a hard constraint

PPO - Proximal Policy Optimization

Two variants :

PPO-Penalty :

- Approximately solves a KL-constrained update like TRPO
- But penalizes the KL-divergence in the objective function
  instead of making it a hard constraint
- and automatically adjusts the penalty coefficient over the course of training.

PPO - Proximal Policy Optimization

Two variants :

PPO-Penalty :

- Approximately solves a KL-constrained update like TRPO
- But penalizes the KL-divergence in the objective function
  instead of making it a hard constraint
- and automatically adjusts the penalty coefficient over the course of training.

PPO-Clip :

- Doesn't have a KL-divergence term in the objective

PPO - Proximal Policy Optimization

Two variants :

PPO-Penalty :

- Approximately solves a KL-constrained update like TRPO
- But penalizes the KL-divergence in the objective function
  instead of making it a hard constraint
- and automatically adjusts the penalty coefficient over the course of training.

PPO-Clip :

- Doesn't have a KL-divergence term in the objective
- and doesn't have a constraint at all

PPO - Proximal Policy Optimization

Two variants :

PPO-Penalty :

- Approximately solves a KL-constrained update like TRPO
- But penalizes the KL-divergence in the objective function instead of making it a hard constraint
- and automatically adjusts the penalty coefficient over the course of training.

PPO-Clip :

- Doesn't have a KL-divergence term in the objective
- and doesn't have a constraint at all
- Instead, it relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

PPO - Proximal Policy Optimization

Theory behind PPO-Clip :

- PPO-Clip takes multiple steps of SGD to maximize the objective :

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L(s,a,\theta_k,\theta) \right]$$

  where

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_k}}(s,a) \right)$$

PPO - Proximal Policy Optimization

Simplified version, implemeted in OpenAI :

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s, a))\right)$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

PPO - Proximal Policy Optimization

Simplified version, implemeted in OpenAI :

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right)$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

- Positive Advantage : $\pi(a \mid s) \nearrow \quad \Rightarrow \quad A(s,a) \nearrow$:

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s,a)$$

PPO - Proximal Policy Optimization

Simplified version, implemeted in OpenAI :

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right)$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

- Positive Advantage : $\pi(a \mid s) \nearrow \quad \Rightarrow \quad A(s,a) \nearrow$:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s,a)$$

- Negative advantage : $\pi(a \mid s) \nearrow \quad \Rightarrow \quad A(s,a) \searrow$:

$$L(s,a,\theta_k,\theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s,a)$$

PPO - Proximal Policy Optimization

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

  Compute rewards-to-go $\hat{R}_t$.

  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.

  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.

end for

DDPG - Deep Deterministic Policy Gradient

DDPG

- For Deterministic policies

DDPG - Deep Deterministic Policy Gradient

DDPG

- For Deterministic policies
- It is an off-policy algorithm.

DDPG - Deep Deterministic Policy Gradient

DDPG

- For Deterministic policies
- It is an off-policy algorithm.
- only for environments with continuous action spaces.

DDPG - Deep Deterministic Policy Gradient

DDPG

- For Deterministic policies
- It is an off-policy algorithm.
- only for environments with continuous action spaces.
- A kind of deep Q-learning for continuous action spaces.

DDPG - Deep Deterministic Policy Gradient

DDPG

- For Deterministic policies
- It is an off-policy algorithm.
- only for environments with continuous action spaces.
- A kind of deep Q-learning for continuous action spaces.
- Implementation of DDPG does not support parallelization.

DDPG - Deep Deterministic Policy Gradient

DDPG

- For Deterministic policies
- It is an off-policy algorithm.
- only for environments with continuous action spaces.
- A kind of deep Q-learning for continuous action spaces.
- Implementation of DDPG does not support parallelization.

Abstract

- DDPG learns a Q-function and a policy

DDPG - Deep Deterministic Policy Gradient

DDPG

- For Deterministic policies
- It is an off-policy algorithm.
- only for environments with continuous action spaces.
- A kind of deep Q-learning for continuous action spaces.
- Implementation of DDPG does not support parallelization.

Abstract

- DDPG learns a Q-function and a policy
- It uses off-policy data and Bellman eq. to learn Q-function,

DDPG - Deep Deterministic Policy Gradient

DDPG

- For Deterministic policies
- It is an off-policy algorithm.
- only for environments with continuous action spaces.
- A kind of deep Q-learning for continuous action spaces.
- Implementation of DDPG does not support parallelization.

Abstract

- DDPG learns a Q-function and a policy
- It uses off-policy data and Bellman eq. to learn Q-function,
- and uses the Q-function to learn the policy.

DDPG - Deep Deterministic Policy Gradient

How to calculate the arg max :

$$a^*(s) = arg \max_a Q^*(s, a)$$

DDPG - Deep Deterministic Policy Gradient

How to calculate the arg max :

$$a^*(s) = arg \max_a Q^*(s, a)$$

- Discrete action space : by enumeration through the Q table.

DDPG - Deep Deterministic Policy Gradient

How to calculate the arg max :

$$a^*(s) = arg \max_a Q^*(s, a)$$

- Discrete action space : by enumeration through the Q table.
- Continuous action space : ? ? ? ?

DDPG - Deep Deterministic Policy Gradient

How to calculate the arg max :

$$a^*(s) = arg \max_a Q^*(s, a)$$

- Discrete action space : by enumeration through the Q table.
- Continuous action space : ? ? ? ?
- Using a normal optimization algorithm is expensive

DDPG - Deep Deterministic Policy Gradient

How to calculate the arg max :

$$a^*(s) = arg \max_a Q^*(s, a)$$

- Discrete action space : by enumeration through the Q table.
- Continuous action space : ? ? ? ?
- Using a normal optimization algorithm is expensive
- since it is run every time the agent wants to take an action.

DDPG - Deep Deterministic Policy Gradient

- $Q^*(s, a)$ is presumed to be differentiable with respect to the a

DDPG - Deep Deterministic Policy Gradient

- $Q^*(s, a)$ is presumed to be differentiable with respect to the a
- We use gradient-based learning rule for a policy $\mu(s)$

DDPG - Deep Deterministic Policy Gradient

- $Q^*(s, a)$ is presumed to be differentiable with respect to the a
- We use gradient-based learning rule for a policy $\mu(s)$
- We approximate $\max_a Q(s, a) \approx Q(s, \mu(s))$.

DDPG - Deep Deterministic Policy Gradient

Q-Learning side of DDPG :
Minimizing MSBE loss with stochastic gradient descent :

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s')) \right) \right)^2 \right]$$

where $\mu_{\theta_{\text{targ}}}$ is the target policy.

DDPG - Deep Deterministic Policy Gradient

Policy Learning Side of DDPG :

- Learn a deterministic policy $\mu_\theta(s)$ maximizing $Q_\phi(s, a)$.

DDPG - Deep Deterministic Policy Gradient

Policy Learning Side of DDPG :

- Learn a deterministic policy $\mu_\theta(s)$ maximizing $Q_\phi(s,a)$.
- Perform gradient ascent (w.r.t. policy parameters only) to solve :

$$\max_\theta \mathop{\mathrm{E}}_{s\sim\mathcal{D}} \left[Q_\phi(s, \mu_\theta(s))\right].$$

The Q-function parameters are treated as constants here.

DDPG - Deep Deterministic Policy Gradient

Policy Learning Side of DDPG :

- Learn a deterministic policy $\mu_\theta(s)$ maximizing $Q_\phi(s, a)$.
- Perform gradient ascent (w.r.t. policy parameters only) to solve :

$$\max_\theta \mathop{\mathrm{E}}_{s \sim \mathcal{D}} \left[Q_\phi(s, \mu_\theta(s)) \right].$$

  The Q-function parameters are treated as constants here.
- To enhance exploration, a noise is added to actions at training.

DDPG - Deep Deterministic Policy Gradient

Policy Learning Side of DDPG :

- Learn a deterministic policy $\mu_\theta(s)$ maximizing $Q_\phi(s, a)$.
- Perform gradient ascent (w.r.t. policy parameters only) to solve :

$$\max_\theta \mathop{\mathrm{E}}_{s\sim\mathcal{D}} \left[Q_\phi(s, \mu_\theta(s)) \right].$$

  The Q-function parameters are treated as constants here.
- To enhance exploration, a noise is added to actions at training.
- Uncorrelated, mean-zero Gaussian noise works perfectly well.

DDPG - Deep Deterministic Policy Gradient

Policy Learning Side of DDPG :

- Learn a deterministic policy $\mu_\theta(s)$ maximizing $Q_\phi(s, a)$.
- Perform gradient ascent (w.r.t. policy parameters only) to solve :

$$\max_\theta \mathop{\mathrm{E}}_{s \sim \mathcal{D}} \left[Q_\phi(s, \mu_\theta(s)) \right].$$

  The Q-function parameters are treated as constants here.
- To enhance exploration, a noise is added to actions at training.
- Uncorrelated, mean-zero Gaussian noise works perfectly well.
- Reduce the scale of the noise over the course of training.

DDPG - Deep Deterministic Policy Gradient

Polyak averaging :

- DQN : target network is copied every fixed-number of steps

DDPG - Deep Deterministic Policy Gradient

Polyak averaging :

- DQN : target network is copied every fixed-number of steps
- DDPG : target network is updated once per main network update :

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi,$$

where $0 \leq \rho \leq 1$, usually close to 1.

DDPG - Deep Deterministic Policy Gradient

Polyak averaging :

- DQN : target network is copied every fixed-number of steps
- DDPG : target network is updated once per main network update :

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi,$$

where $0 \leq \rho \leq 1$, usually close to 1.

DDPG - Deep Deterministic Policy Gradient

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$

repeat

  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

  If s' is terminal, reset environment state.

  if it's time to update **then**

    for however many updates **do**

      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta \end{aligned}$$

    end for

  end if

until convergence

Thank you !

TD3 - Twin Delayed DDPG

Main ideas

- Clipped Double-Q Learning. TD3 learns two Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

TD3 - Twin Delayed DDPG

Main ideas

- Clipped Double-Q Learning. TD3 learns two Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.
- Delayed Policy Updates. TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.

TD3 - Twin Delayed DDPG

Main ideas

- Clipped Double-Q Learning. TD3 learns two Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.
- Delayed Policy Updates. TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.
- Target Policy Smoothing. TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

SAC - Soft Actor Critic

- SAC : entropy regularization.

SAC - Soft Actor Critic

- SAC : entropy regularization.
- Maximize a trade-off between expected return and entropy, a measure of randomness in the policy.

SAC - Soft Actor Critic

- SAC : entropy regularization.
- Maximize a trade-off between expected return and entropy, a measure of randomness in the policy.
- This has a close connection to the exploration-exploitation trade-off :

SAC - Soft Actor Critic

- SAC : entropy regularization.
- Maximize a trade-off between expected return and entropy, a measure of randomness in the policy.
- This has a close connection to the exploration-exploitation trade-off :
- Increasing entropy results in more exploration, which can accelerate learning later on.

SAC - Soft Actor Critic

- SAC : entropy regularization.
- Maximize a trade-off between expected return and entropy, a measure of randomness in the policy.
- This has a close connection to the exploration-exploitation trade-off :
- Increasing entropy results in more exploration, which can accelerate learning later on.
- It can also prevent the policy from prematurely converging to a bad local optimum.