Balancing Constraints and Rewards with Meta-gradient D4PG

高辰潇

NJUAI, Nanjing University

February 22, 2021

概述

- Dan A.Calian, Daniel J.Mankowitz, Deepmind, ICLR2020.
- **创新点**：针对 CMDP 问题，在 RCDP 的基础上利用 meta-gradient 在 CMDP 不可行时自适应地调整拉格朗日乘子并进行策略优化。
- **相关领域**
  - safe RL
  - constrained MDP: [**Sutton 2018**]
  - constrained MDP 算法: CDP[**Achiam 2017**], RCDP[**Tessler 2018**]
  - meta-gradient RL: [**Xu 2018**], Self-Tuning AC[**Zahavy 2020**]

目录

Safe RL and Constrained RL

Meta-gradient RL

Balancing constraints and rewards with meta-gradient

目录

Safe RL and Constrained RL

Meta-gradient RL

Balancing constraints and rewards with meta-gradient

Safe RL and Constrained RL

- 在某些现实情况下，智能体的安全相当重要，因此不仅需要关注长期奖励的最大化，而且也许要关注避免智能体行为带来的损害。
- 两种 Safe RL 的解决思路
  - 基于优化准则：改变 RL 的优化目标、在优化问题中添加约束等
  - 修改探索过程
- Constrained RL: 根据智能体的安全性要求，在 RL 优化问题中添加约束。通常可表示为

$$\begin{aligned} \max_{\pi} \quad & J_r^{\pi} = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t] \\ s.t. \quad & J_{c_i}^{\pi} = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t c_{i,t}] \leq \beta_i \quad \text{for } i = 1, ..., K \end{aligned} \tag{1}$$

此处 $c_{i,t}$ 为与第 i 个约束相关的即时惩罚，β_i 为与第 i 个约束相关的阈值。

Constrained RL 算法

- **思路 1**: Reward Shaping，将即时奖赏设置为 $r_t - \lambda c_t$，其中 λ 手工指定。
- **思路 2**: 直接求解优化问题 (1)。当使用参数化表示时，由于原问题的目标函数和约束项均非凸，CPO[**Achiam 2017**] 为目标和约束构造出一系列放松的、凸的替代函数并基于替代函数对问题进行优化。
- **思路 3**: 求解对偶问题 (2)，转化为求解一个无约束的鞍点优化问题。RCPO[**Tessler 2018**] 在 Advantage AC 算法的基础上 λ 和 θ_p, θ_a 设置了不同的更新速率 (two timescale)，保证算法一定能够收敛到目标函数的不动点。在不同更新速率的设置下，可在内层更新中使用现有的强化学习算法。

$$\min_{\lambda \geq 0} \max_{\pi} \quad J_r^{\pi} - \lambda \cdot (J_c^{\pi} - \beta) \tag{2}$$

RCPO Framework

Algorithm 1 Template for an RCPO implementation

Input: penalty $c(\cdot)$, constraint $C(\cdot)$, threshold α, learning rates $\eta_1(k) < \eta_2(k) < \eta_3(k)$
Initialize actor parameters $\theta = \theta_0$, critic parameters $v = v_0$, Lagrange multipliers and $\lambda = 0$
for $k = 0, 1, ...$ **do**
    Initialize state $s_0 \sim \mu$
    for $t = 0, 1, ..., T-1$ **do**
        Sample action $a_t \sim \pi$, observe next state s_{t+1}, reward r_t and penalties c_t
        $\hat{R}_t = r_t - \lambda_k c_t + \gamma \hat{V}(\lambda, s_t; v_k)$ ▷ Equation 10
        Critic update: $v_{k+1} \leftarrow v_k - \eta_3(k) \left[\partial(\hat{R}_t - \hat{V}(\lambda, s_t; v_k))^2 / \partial v_k\right]$ ▷ Equation 11
        Actor update: $\theta_{k+1} \leftarrow \Gamma_\theta \left[\theta_k + \eta_2(k) \nabla_\theta \hat{V}(\lambda, s)\right]$ ▷ Equation 6
    Lagrange multiplier update: $\lambda_{k+1} \leftarrow \Gamma_\lambda \left[\lambda_k + \eta_1(k) \left(J_C^{\pi_\theta} - \alpha\right)\right]$ ▷ Equation 8
return policy parameters θ

Advantage AC RCPO

Algorithm 2 RCPO Advantage Actor Critic

The original Advantage Actor Critic algorithm is in gray, whereas our additions are highlighted in black.

Input: penalty function $C(\cdot)$, threshold α and learning rates η_1, η_2, η_3
Initialize actor $\pi(\cdot|\cdot;\theta_p)$ and critic $V(\cdot;\theta_v)$ with random weights
Initialize $\lambda = 0$, $t = 0$, $s_0 \sim \mu$ ▷ Restart
for $T = 1, 2, ..., T_{max}$ **do**
    Reset gradients $d\theta_v \leftarrow 0$, $d\theta_p \leftarrow 0$ and $\forall i : d\lambda_i \leftarrow 0$
    $t_{start} = t$
    while s_t `not terminal and` $t - t_{start} < t_{max}$ **do**
        Perform a_t according to policy $\pi(a_t|s_t;\theta_p)$
        Receive r_t, s_{t+1} and penalty score $\hat{C}_t$
        $t \leftarrow t + 1$
        $R = \begin{cases} 0 & , \text{ for terminal } s_t \\ V(s_t, \theta_v) & , \text{ otherwise} \end{cases}$
    for $\tau = t-1, t-2, ..., t_{start}$ **do**
        $R \leftarrow r_\tau - \lambda \cdot \hat{C}_\tau + \gamma R$ ▷ Equation 10
        $d\theta_p \leftarrow d\theta_p + \nabla_{\theta_p} \log \pi(a_\tau|s_\tau;\theta_p)(R - V(s_\tau;\theta_v))$
        $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_\tau;\theta_v))^2/\partial\theta_v$
    if s_t is terminal state **then**
        $d\lambda \leftarrow -(C - \alpha)$ ▷ Equation 8
        $t \leftarrow 0$
        $s_0 \sim \mu$
    Update θ_v, θ_p and λ
    Set $\lambda = \max(\lambda, 0)$ ▷ Ensure weights are non-negative (Equation 4)

图: Advantage AC RCPO

目录

Safe RL and Constrained RL

Meta-gradient RL

Balancing constraints and rewards with meta-gradient

Meta-gradient RL

- 借助 meta object，对强化学习问题中的超参数进行调整。
- 定义 η 为超参数（折扣 γ，学习率，reward shaping 中的超参数等），θ 为强化学习问题中的参数（比如 AC 算法中 critic 和 actor 的参数），已经参数 θ 的更新函数 f:

$$\theta' = \theta + f(\tau, \theta, \eta) = \theta + \alpha \frac{J(\tau, \theta, \eta)}{\partial \theta} \tag{3}$$

- 引入与 θ', η', τ' 相关的 meta object $J'(\tau', \theta', \eta')$，此时可得到 J' 对于超参数 η 的导数

$$\frac{\partial J'(\tau', \theta', \eta')}{\partial \eta} = \frac{\partial J'(\tau', \theta', \eta')}{\partial \theta'} \frac{\mathrm{d}\theta'}{\mathrm{d}\eta} \tag{4}$$

Meta-gradient RL (cont'd)

- **算法流程**
  1. 固定当前 η，使用当前策略 π_θ 采样得到轨迹 τ，使用式 (3) 更新策略参数 θ 为 θ'。
  2. 使用更新后的参数 θ' 采样新轨迹 τ'，计算 meta object $J'(\tau', \theta', \eta')$，利用式 (4) 更新超参数 η。
  3. 重复直到收敛。
- **直观理解：智能体在使用强化学习算法对参数 θ 进行学习的同时，还在基于元指标 J' 判断哪种超参数能够指导自身获得更好的性能**
- **这里使用 θ' 桥接 J' 和 η，迫使 η 通过策略改进 (θ') 的方式提升 meta object。**

Meta-gradient RL (cont'd)

- 使用前向 TD(λ) 算法，在 Atari 上进行了实验。可供控制的超参数为折扣因子 γ 和 bootstrapping parameter λ。

$$g(\tau_t) = R_{t+1} + \gamma(1-\lambda)v_\theta(S_{t+1}) + \gamma\lambda g(\tau_{t+1})$$

	η	Human starts		No-op starts	
		$\gamma = 0.99$	$\gamma = 0.995$	$\gamma = 0.99$	$\gamma = 0.995$
IMPALA	$\{\}$	144.4%	211.9%	191.8%	257.1%
Meta-gradient	$\{\lambda\}$	156.6%	214.2%	185.5%	246.5%
		$\gamma' = 0.99$	$\gamma' = 0.995$	$\gamma' = 0.99$	$\gamma' = 0.995$
Meta-gradient	$\{\gamma\}$	233.2%	267.9%	280.9%	275.5%
Meta-gradient	$\{\gamma, \lambda\}$	221.6%	292.9%	242.6%	287.6%

目录

Safe RL and Constrained RL

Meta-gradient RL

Balancing constraints and rewards with meta-gradient

Balancing constraints and rewards with meta-gradient

- 提出了 D4PG 版本的 RCPO 算法：RC-D4PG
- soft constraint meta-gradient method：MetaL
- 实验

RC-D4PG

使用的参数更新方法和 RCPO 完全相同，区别在于转变为 offline 算法。

Algorithm 3 Reward Constrained D4PG (RC-D4PG)

Input: penalty $c(\cdot)$, constraint $C(\cdot)$, threshold β, learning rates $\alpha_1 < \alpha_{\theta_a} < \alpha_{\theta_c}$, max. number of episodes M
Initialize actor parameters θ_a, critic parameters θ_c, Lagrange multipliers $\lambda = 0$
for $1 \dots M$ **do**
  Sample episode penalty J_C^π from the penalty replay buffer
  $\lambda \leftarrow [\lambda - \alpha_1 (J_C^\pi - \beta)]_+$ ▷ **Lagrange multiplier update**
  Sample batch $\langle s_t, a_t, r_t, s_{t+1}, c_t \rangle_{t=1}^T$ from ER
  $\nabla\theta_c = 0, \nabla\theta_a = 0$
  for $t = 1 \dots T$ **do**
    $\hat{R}_t = r_t - \lambda c_t + \gamma \hat{Q}(\lambda, s_{t+1}, a_{t+1} \sim \pi_{Target}(s_{t+1}); \theta_c)$
    $\nabla\theta_c \mathrel{+}= \alpha_{\theta_c} \partial (\hat{R}_t - \hat{Q}(\lambda, s_t, a_t; \theta_c))^2 / \partial \theta_c$ ▷ **Critic update**
    $\nabla\theta_a \mathrel{+}= \alpha_{\theta_a} \mathbb{E}[\nabla_a Q(s_t, a_t) \nabla_{\theta_a} \pi_{\theta_a}(s_t)|_{a_t = \pi(s_t)}]$ ▷ **Actor update**
  $\theta_c \leftarrow \theta_c - \frac{1}{T} \sum \nabla\theta_c$
  $\theta_a \leftarrow \theta_a + \frac{1}{T} \sum \nabla\theta_a$
return policy parameters θ_a

MetaL

- Motivation：在许多现实应用场景中，一方面违反约束并不会造成严重的影响，有时 Agent 只有在违反一定程度的约束时才能表现出一些良好的行为；另一方面，一组给定的约束条件往往相互矛盾导致不存在可行解，而如何合理地设置约束尚不明晰。
  RCPO 算法本质上是 hard constraint 方法，当 $J_C^\pi \geq \beta$ 时，拉格朗日乘子 λ 会不断增大而过拟合约束。因此，需要一种 soft constraint 算法，在对偶问题不可行时找到合适的乘子 λ 作为惩罚项的加权系数。
- MetaL 在 RCPO 的基本框架中为 λ 的更新添加了超参数 α_λ，即 λ 的更新率。使用 Meta-gradient 算法自适应地调整 α_λ，从而调整 λ 的变化速率以防止过拟合约束。

MetaL (cont'd)

Algorithm 1 MetaL

Input: penalty $c(\cdot)$, constraint $C(\cdot)$, threshold β, learning rates $\alpha_1, \alpha_{\theta_a}, \alpha_{\theta_c}, \alpha_\eta$, max. number of episodes M
Initialize actor, critic parameters θ_a and θ_c, Lagrange multiplier $\lambda = 0$, meta-parameter $\eta = \alpha_\lambda$
for $1 \dots M$ **do**
  Inner loss:
  Sample episode penalty J_C^π from the penalty replay buffer
  $\lambda' \leftarrow [\lambda - \alpha_1 \exp(\alpha_\lambda)\,(\beta - J_C^\pi)]_+$ ▷ **Lagrange multiplier update**
  Sample a batch with tuples $\langle s_t, a_t, r_t, c_t \rangle_{t=1}^T$ from ER and split into training/validation sets
  Accumulate and apply actor and critic updates over training batch T_{train} by:
  $\nabla\theta_c = 0, \nabla\theta_a = 0$
  for $t = 1 \dots T_{train}$ **do**
    $\hat{R}_t = r_t - \lambda' c_t + \gamma \hat{Q}(\lambda, s_{t+1}, a_{t+1} \sim \pi_T(s_{t+1}); \theta_c)$
    $\nabla\theta_c \mathrel{+}= \alpha_{\theta_c} \partial (\hat{R}_t - \hat{Q}(\lambda, s_t, a_t; \theta_c))^2 / \partial\theta_c$ ▷ **Critic update**
    $\nabla\theta_a \mathrel{+}= \alpha_{\theta_a} \mathbb{E}[\nabla_a Q(s_t, a_t) \nabla_{\theta_a} \pi_{\theta_a}(s_t)|_{a_t = \pi(s_t)}]$ ▷ **Actor update**
  $\theta_c' \leftarrow \theta_c - \frac{1}{T_{train}} \sum \nabla\theta_c$
  $\theta_a' \leftarrow \theta_a + \frac{1}{T_{train}} \sum \nabla\theta_a$
  Outer loss: Compute outer loss and meta-gradient update using validation set $T_{validate}$:
  $\alpha_\lambda' \leftarrow \alpha_\lambda - \alpha_\eta \frac{\partial J'(\theta_c'(\alpha_\lambda), \lambda'(\alpha_\lambda))}{\partial \alpha_\lambda}$ ▷ **Meta-parameter update - Theorem 1**
  $\lambda \leftarrow \lambda', \theta_a \leftarrow \theta_a', \theta_c \leftarrow \theta_c', \alpha_\lambda \leftarrow \alpha_\lambda'$
return $\theta_a, \theta_c, \lambda$

inner loss 对应 RC-D4PG 框架，区别在于 λ 的更新步长由 $\exp(\alpha_\lambda)$ 决定。

outer loss J' 对应 meta-gradient 中的 meta object

实验设置

- **在实验中**，outer loss **被经验性地设置为** critic loss，**即在** $\theta_c = \theta'_c(\alpha_\lambda), \lambda = \lambda'(\alpha_\lambda)$ **时的** L_{critic}

$$L_{\text{critic}}(\theta_c, \lambda) = \left(r(s,a) - \lambda c(s,a) + \gamma Q_{\text{target}}(s, \pi_{\text{target}}(s')) - Q_{\theta_c}(s,a)\right)^2$$

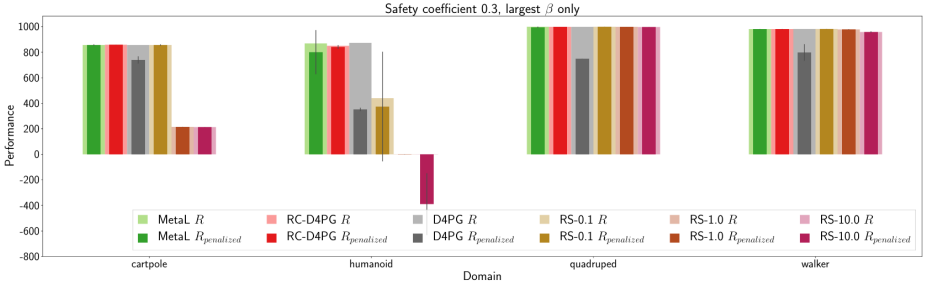
- **实验追踪两个指标**：average episode return R **和** penalized return $R_{\text{penalized}} = R - \kappa \max(0, J_C^\pi - \beta)$。**其中** κ **被设置为** 1000。
- **实验环境**：Domain, safety coefficient
- **算法**：MetaL, RC-D4PG, RS-D4PG

实验

Algorithm	$R_{penalized}$	R	$\max(0, J_C^\pi - \beta)$
D4PG	432.70 ± 11.99	**927.66**	0.49
MetaL	$\mathbf{677.93} \pm \mathbf{25.78}$	921.16	0.24
RC-D4PG	478.60 ± 89.26	648.42	**0.17**
RS-0.1	641.41 ± 26.67	906.76	0.27
RS-1.0	511.70 ± 15.50	684.30	0.17
RS-10.0	208.57 ± 61.46	385.42	0.18
RS-100.0	118.50 ± 62.54	314.93	0.20

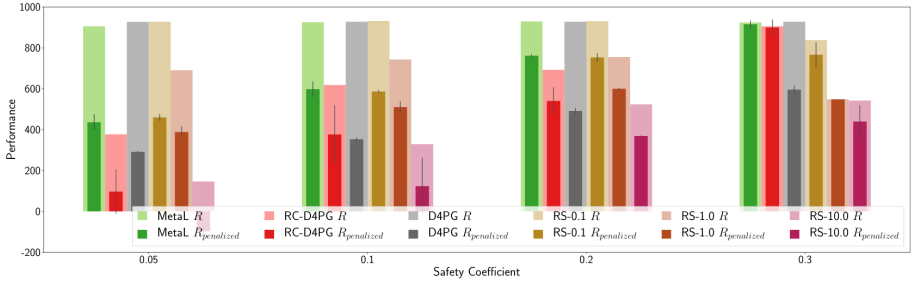
Table 1: Overall performance across domains, safety coefficients and thresholds.

实验：不同 domain 下各算法的性能



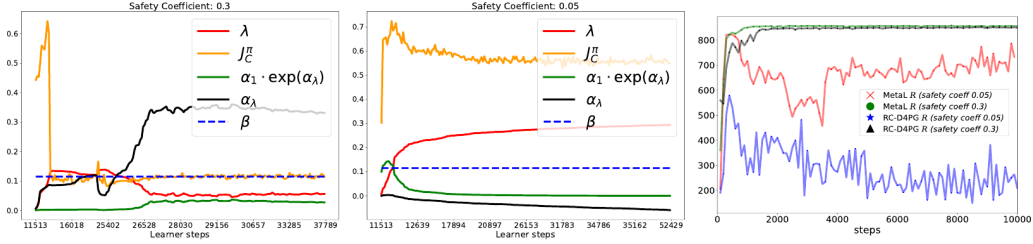
结论：Reward Shaping 不具备跨 domain 的泛化性能。在约束宽松的情况下，MetaL 能够取得与 D4PG 和 RC-D4PG 相近的性能。

实验：不同 safety coefficient 设置下各算法的性能



结论：在约束严格时，RC-D4PG 会过拟合约束，而 MetaL 因为引入了 critic loss 作为指导 α_l 更新的外部信号，因而仍能取得不错的效果。

实验：超参数 α_l 的更新行为



结论：在问题不可行（图 2）时，$\alpha_1 \cdot \exp(\alpha_\lambda) \to 0$，导致 λ 收敛于定值。

Q&A

References I

- Calian, Dan A., et al. "Balancing Constraints and Rewards with Meta-Gradient D4PG." arXiv preprint arXiv:2010.06324 (2020).
- Tessler, Chen, Daniel J. Mankowitz, and Shie Mannor. "Reward constrained policy optimization." arXiv preprint arXiv:1805.11074 (2018).
- Achiam, Joshua, et al. "Constrained policy optimization." International Conference on Machine Learning. PMLR, 2017.
- Xu, Zhongwen, Hado van Hasselt, and David Silver. "Meta-gradient reinforcement learning." arXiv preprint arXiv:1805.09801 (2018).
- Zahavy, Tom, et al. "A self-tuning actor-critic algorithm." arXiv e-prints (2020): arXiv-2002.