Reward Constrained Policy Optimization

Sagae Tomoya 0756153
Department of Computer Science
National Chiao Tung University
jptom.cs07g@nctu.edu.tw

1 Introduction

In this paper They proposed the 'Reward Constrained Policy Optimization' (RCPO) algorithm. RCPO adds the constrain as a penalty signal in to the reward function. This penalty signal guides optimal solutions in a limited environment. They proved that RCPO converges almost surely, under soft assumptions, to a constraint satisfying solution. Additionally, They challenged to show better result (faster convergence and improved stability) in some simulation environment with RCPO.

The goal of Reinforcement Learning (RL) is to maximize the target function, generally it is the reward. In constrained optimization, the task is to maximize a target function $f(x)$ while satisfying an inequality constraint $g(x) \leq \alpha$. While constraints are a promising solution to ensuring a satisfying

2 Problem Formulation

2.1 Markov Decision Process (MDP)

A Markov Decision Processes $\mathcal{M}$ is defined by the tuple $(S, A, R, P, \mu, \gamma)$. Where S is the set of states, A the available actions, $R : S \times A \times S \mapsto \mathbb{R}$ is the reward function, $P : S \times A \times S \mapsto [0, 1]$ is the transition matrix, where $P(s'|s, a)$ is the probability of transitioning from state s to s' assuming action a was taken, $\mu : S \mapsto [0, 1]$ is the initial state distribution and $\gamma \in [0, 1)$ is the discount factor for future rewards. A policy $\pi : S \mapsto \Delta_A$ is a probability distribution over actions and $\pi(a|s)$ denotes the probability of taking action a at state s. For each state s, the value of following policy π is denoted by:

$$V_R^\pi(s) = \mathbb{E}^\pi[\sum_t \gamma^t r(s_t, a_t)|s_0 = s] \ .$$

An important property of the value function is that it solves the recursive Bellman equation:

$$V_R^\pi(s) = \mathbb{E}^\pi[r(s, a) + \gamma V_R^\pi(s')|s] \ .$$

The goal is then to maximize the expectation of the reward-to-go, given the initial state distribution μ:

$$\max_{\pi \in \Pi} J_R^\pi \ \text{, where} \quad J_R^\pi = \mathbb{E}^\pi_{s\sim\mu}[\sum_{t=0}^{\infty} \gamma^t r_t] = \sum_{s \in S} \mu(s) V_R^\pi(s) \ . \tag{1}$$

2.2 Constrained MDPs

A Constrained Markov Decision Process (CMDP) extends the MDP framework by introducing a penalty $c(s, a)$, a constraint $C(s_t) = F(c(s_t, a_t), ..., c(s_N, a_N))$ and a threshold $\alpha \in [0, 1]$. A constraint may be a discounted sum (similar to the reward-to-go), the average sum and more (see Altman (1999) for additional examples).

Preprint. Under review.

They denote the expectation over the constraint by:

$$J_C^\pi = \mathbb{E}_{s\sim\mu}^\pi[C(s)] \ . \tag{2}$$

The problem thus becomes:

$$\max_{\pi\in\Pi} J_R^\pi \ \texttt{, s.t.} \quad J_C^\pi \leq \alpha \ . \tag{3}$$

2.3 Parametrized Policies

In this work they consider parametrized policies, such as neural networks. The parameters of the policy are denoted by θ and a parametrized policy as π_θ. They make the following assumptions in order to ensure convergence to a constraint satisfying policy:

Assumption 1 *The value $V_R^\pi(s)$ is bounded for all policies $\pi \in \Pi$.*

Assumption 2 *Every local minima of $J_C^{\pi_\theta}$ is a feasible solution.*

Assumption 2 is the minimal requirement in order to ensure convergence, given a general constraint, of a gradient algorithm to a feasible solution. Stricter assumptions, such as convexity, may ensure convergence to the optimal solution; however, in practice constraints are non-convex and such assumptions do not hold.

2.4 Constrained Policy Optimization

When $V^{\pi(\theta)}$ and $\mathbb{E}^{\pi(\theta)}[C]$ are convex, (3) can be solved using standard convex optimization tools, which are shown to find the optimal solution. However, in practice, the value function is non-convex and the agent only observes noisy estimates; as such, convergence to the optimal solution is not ensured.

2.5 Lagrange relaxation

A common approach, called Lagrange relaxation [**?**], is to convert the CMDP into an equivalent unconstrained problem, also known as the Lagrange dual form. In this approach, in addition to the objective, a penalty term is added for infeasibility, thus making infeasible solutions sub-optimal. Given a CMDP (3), the unconstrained problem is:

$$\min_{\lambda\geq 0}\max_{\theta} L(s,\lambda,\theta) = \min_{\lambda\geq 0}\max_{\theta}\Big[V^{\pi(\theta)}(s) - \lambda\cdot(\mathbb{E}^{\pi(\theta)}[C]-\alpha)\Big] \ , \tag{4}$$

where $\lambda \geq 0$ is the Lagrange multiplier (a penalty coefficient). Notice, as λ increases, the solution to (4) converges to that of (3). This suggests a two-timescale approach: on the faster timescale, θ is found by solving (4), while on the slower timescale, λ is increased until the constraint is satisfied. The goal is to find a saddle point $(\theta^*(\lambda^*),\lambda^*)$ of (4), which satisfies:

$$L(s,\theta,\lambda^*) \leq L(s,\theta^*(\lambda^*),\lambda^*) \leq L(s,\theta^*(\lambda),\lambda), \ \ \forall\theta\in\Theta, \ \ \forall\lambda\geq 0 \tag{5}$$

Definition 1 *Denote the optimal solution to the CMDP as π^* and a near-optimal solution as π. A near-optimal solution of the CMDP is defined as a solution which satisfies the constraints, e.g. $\mathbb{E}^\pi[C]\leq\alpha$, and $V^\pi(s)\leq V^{\pi^*}(s), \forall s\in\mu$.*

they note that due to the non-convexity of the value function $V^{\pi(\theta)}$, any fixed point of (4) is by definition a near-optimal solution to the CMDP (3).

2.6 Estimating the gradient

Note that they operate in a simulation optimization setting, i.e., we have access to reward samples from the underlying MDP. Thus, the simulation based algorithm for the constrained optimization problem (3) is:

$$\lambda_{k+1} = \Gamma_\lambda[\lambda_k - \eta_1(k)\nabla_\lambda L(s,\lambda_k,\theta_k)] \ , \tag{6}$$

$$\theta_{k+1} = \Gamma_\theta[\theta_k + \eta_2(k)\nabla_\theta L(s,\lambda_k,\theta_k)] \ , \tag{7}$$

where Γ_θ is a projection operator, which keeps the iterate θ_k stable by projecting onto a compact and convex set and Γ_λ projects λ into the range $[0, \lambda_{max}]$. [1]$\nabla_\theta L$ and [2]$\nabla_\lambda L$ are derived from (4):

$$\nabla_\theta L(s, \lambda, \theta) = \nabla_\theta \log \pi(s, a; \theta) \mathbb{E}^{\pi(\theta)} \left[\sum_{t=0}^{T-1} \gamma^t r(s_t) - \lambda \cdot C | s_0 = s \right] \ , \tag{8}$$

$$\nabla_\lambda L(s, \lambda, \theta) = -(\mathbb{E}^{\pi(\theta)}[C] - \alpha) \ . \tag{9}$$

and $\eta_1(k), \eta_2(k)$ are step-sizes which ensure that the policy update is performed on a faster timescale than that of the penalty coefficient λ, the following assumption formalizes this requirement:

Assumption 3

$$\sum_{k=0}^{\infty} \eta_1(k) = \sum_{k=0}^{\infty} \eta_2(k) = \infty, \ \sum_{k=0}^{\infty} \left(\eta_1(k)^2 + \eta_2(k)^2 \right) < \infty \ \textit{and} \ \frac{\eta_1(k)}{\eta_2(k)} \to 0 \ .$$

Theorem 1 *Denote by $\Theta = \{\theta : J_C^{\pi_\theta} \leq \alpha\}$ the set of feasible solutions and the set of local-minimas of $J_{C_\gamma}^{\pi_\theta}$ as Θ_γ. Assuming that $\Theta_\gamma \subseteq \Theta$ then the 'Reward Constrained Policy Optimization' (RCPO) algorithm converges almost surely to a fixed point $(\theta^*(\lambda^*, v^*), v^*(\lambda^*), \lambda^*)$ which is a feasible solution (e.g. $\theta^* \in \Theta$).*

2.7 Reward Constrained Policy Optimization

In Section 2.4 they presented the Lagrange dual approach to constrained policy optimization. However, this approach has two major setbacks; (i) a general penalty signal does not necessarily exhibit the recursive property (i.e. the Bellman equation) required for estimation using a TD-critic (requiring the optimization process to consider the entire trajectory), (ii) in practice many penalty signals are non-convex.

To overcome this issue, they introduce the per-state penalty signal $\hat{C}(s, a)$, a memory-less signal, and the discounted penalty C_γ, a discounted sum over per-state penalty signals.

Definition 2 *They denote the per-state penalty as $\hat{C}(s, a)$, the value of the penalty function C given the single-step trajectory (s, a). And define the discounted penalty random variable C_γ as:*

$$C_\gamma = \sum_{t=0}^{T-1} \gamma^t \hat{C}(s_t, a_t) \tag{10}$$

2.8 Penalized reward functions

They introduce the penalized reward $\hat{r}$, state-action value $\hat{Q}$, and state value $\hat{V}$ functions in Definition 3.

Definition 3 *The penalized reward functions are defined as:*

$$\hat{r}(\lambda, s, a) \triangleq r(s, a) - \lambda \hat{C}(s, a) \ , \tag{11}$$

$$\hat{V}^\pi(\lambda, s) \triangleq \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} \gamma^t \hat{r}(\lambda, s_t) | s_0 = s \right] \ , \tag{12}$$

$$\hat{Q}^\pi(\lambda, s, a) \triangleq \mathbb{E}^\pi \left[\sum_{t=0}^{T-1} \gamma^t \hat{r}(\lambda, s_t, a_t) | s_0 = s, a_0 = a \right] \ . \tag{13}$$

3 Theoretical Analysis

3.1 RCPO Algorithm

Now I introduce RCPO algorithm using DQN and A2C.

[1]The formulation for $\nabla_\theta L$ is achieved using the log-likelihood trick [**?**].

[2]Note that the fixed point $\nabla_\lambda L = 0$ is when $\mathbb{E}^{\pi(\theta)}[C] = \alpha$ or $\lambda = 0$.

Algorithm 1 RCPO + DQN

The original DQN 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 replay memory D to capacity N
Initialize action-value function $Q(\cdot, \cdot; \theta)$ amd target action-value function $Q(\cdot, \cdot; \theta^-)$ with random weights
Initialize $\lambda = 0$
for episode $= 1, 2, ..., M$ **do**
    $s_0 \sim \mu$ ▷ Restart
    $t = 0$
    while $t < T_{max}$ and s_t is not a terminal state **do**
        Reset gradients $d\theta \leftarrow 0, \forall i : d\lambda_i \leftarrow 0$
        $a_t = \begin{cases} \text{random action} & , \text{ w.p. } \epsilon \\ arg\max_a Q(s_t, a_t) & , \text{ otherwise} \end{cases}$
        Perform action a_t
        Receive r_t, s_{t+1} and penalty scores $\hat{C}_t$
        Store transition $(s_t, a_t, r_t, \hat{C}_t)$ in D
        Sample minibatch of transitions $(s_j, a_j, r_j, \hat{C}_j)$ from D
        Set $y_j = \begin{cases} r_j - \lambda \cdot \hat{C}_j & , \text{ for terminal } s_{j+1} \\ r_j - \lambda \cdot \hat{C}_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^-) & , \text{ otherwise} \end{cases}$ ▷ Equation 11
        $d\theta \leftarrow d\theta + \partial (y_j - Q(s_j, a_j; \theta))^2 / \partial \theta$
        Update θ
        $t = t + 1$
    $d\lambda \leftarrow -(C - \alpha)$ ▷ Equation 9
    Update λ
    Set $\lambda = \max(\lambda, 0)$ ▷ Ensure weights are non-negative (Equation 4)

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 11
        $d\theta_p \leftarrow d\theta_p + \nabla_{\theta_p} \log \pi(a_\tau|s_\tau; \theta_p)(R - V(s_\tau; \theta_v))$
        $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_\tau; \theta_v))^2 / \partial \theta_v$
    if s_t is terminal state **then**
        $d\lambda \leftarrow -(C - \alpha)$ ▷ Equation 9
        $t \leftarrow 0$
        $s_0 \sim \mu$
    Update θ_v, θ_p and λ
    Set $\lambda = \max(\lambda, 0)$ ▷ Ensure weights are non-negative (Equation 4)

4 Conclusion

In this paper, they argued the importance of regularization in RL, which can be applied in the form of constraints. They presented the pitfalls of the standard approaches (i.e. regularization via reward shaping). Additionally, they showed that constraints provide a natural and consistent approach to regularization.

They introduced a novel actor-critic approach, called 'Reward Constrained Policy Optimization' (RCPO). RCPO approaches regularization using a multi-timescale approach; on the fast timescale an alternative, discounted, objective is estimated using a actor-critic; on the intermediate timescale the policy is learned using policy gradient methods; and on the slow timescale the penalty coefficient λ is learned by descending on the original Lagrange dual form. They validate our approach using simulations on both grid-world and robotics domains and show that RCPO converges in a stable and sample efficient manner to a constraint satisfying policy, without the need for manually defining the penalty coefficient