SafeAveragers: Adapting DeepAvergers framework for Safe Offline Reinforcement Learning.

Aayam Shrestha, Kin-Ho Lam

[1]Oregon State University: AI 539

Abstract

We study the problem of safe offline reinforcement learning (RL), where given an offline dataset, the goal is to learn a policy that maximizes reward while satisfying safety constraints without further interaction with the environment. Here we propose SafeAveragers: A framework based on optimally and iteratively solving finitely-represented MDPs derived from a static dataset of experience. The SafeAveragers framework extends DeepAveragers with Costs DAC-MDP (DAC-MDP) to explicitly handle safety constraints and iteratively refine the derived MDP in order to prune actions that allow the optimal policy to violate the safety constraints. We also explore the theoretical aspects and study the assumptions that allow us to lower-bound the performance of DAC-MDP solutions while guaranteeing the safety constraints. Furthermore we investigate the empirical behavior of the proposed approach in a simple yet challenging scenarios of continuous grid-world domain with continuous state and action spaces.

Introduction

Reinforcement learning (RL) has demonstrated success in solving many challenging domains. However, most RL systems learn good policies only after millions of trial-and-error explorations in simulated environments. While this process of trial-and-error works for domains like video games where experiments can be repeated indefinitely, many real-world domains, such as industrial systems, cannot afford to have millions of trial-and-error experiments performed due to safety.

Learning from pre-collected data, offline reinforcement learning promises a new solution to domains where learning by exploration is infeasible or prohibitively expensive. However, by limiting learning to data collected a priori without further interaction with the environment, open problems including distribution shift in training verses deployment continues to make learning a safe and effective policy from offline data a challenge.

Safe RL has been modeled as a constrained markov decision process (CMDP), where there are hard constraints and soft constraints (Altman 1999). We focus on soft constraints, where the policy must satisfy all safety constraints in expectation throughout the whole trajectory. By introducing constraints to the learning process, the challenge becomes evaluating constraint violations accurately while maximizing reward. In regular RL, this can be accomplished by rolling out the policy in the simulated enviornmnet and evaluating constraint values via on-policy samples. In offline RL, this approach is not possible because the agent is limited to the distribution of the data set. Additionally the data set may not contain adequate coverage of important regions, thus introducing unpredictable behaviors that may be unsafe.

While there has been methods which penalize both the original unsafe actions,as well as the actions that are out of the data distribution, this has not been used in the DeepAveragers framework which potentially has computational as well as other benefits like fast adaptation in the real world.

Preliminary

Background

A Constrained Markov Decision Process (CMDP) is represented by a tuple $(\mathcal{S}, \mathcal{A}, R, C, T, \gamma, \eta)$,where $\mathcal{S} \subset \mathbb{R}^n$ is the closed and bounded state space and $\mathcal{A} \subset \mathbb{R}^m$ is the action space. Here, $R : \mathcal{S} \times \mathcal{A} \mapsto [0, \overline{R}]$ and $C : \mathcal{S} \times \mathcal{A} \mapsto [0, \overline{C}]$ denote the reward and cost function, bounded by $\overline{R}$ and $\overline{C}$. Let $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ denote the transition probability function that maps state-action pairs to a distribution over the next state and η denote the initial state distribution. And finally, let $\gamma \in [0, 1)$ denote the discount factor for future reward and cost. A policy $\pi : \mathcal{S} \mapsto a \in \mathcal{A}$ corresponds to a map from states to an action. Specifically, $\pi(s)$ denotes the action a chosen by the policy π in state s. The cumulative reward under policy π is denoted as $V_R(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right)\right]$, where $\tau = (s_0, a_0, s_1, a_1, \ldots)$ is a trajectory and $\tau \sim \pi$ means the distribution over trajectories is induced by policy π. Similarly, the cumulative cost takes the form as $V_C(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t C\left(s_t, a_t\right)\right]$

In our problem, we assume an offline dataset $\mathcal{D} = (s_i, a_i, s_i', r(s, a), c(s, a))$, generated by one or more unknown behavior policies. We note that these behavior policies may or may not have the knowledge about safety constraints and may violate them. We use π_β to represent the empirical behavior policy of the dataset, formally,

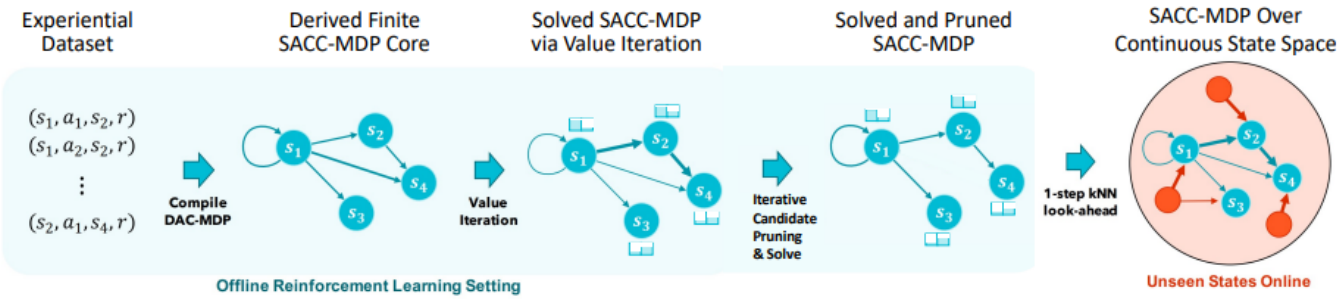

Figure 1: SACC-MDP

$\pi_\beta(a_0 \mid s_0) := \frac{\sum_{s,a \in \mathcal{D}} \mathbf{1}[s=s_0, a=a_0]}{\sum_{s \in \mathcal{D}} 1[s=s_0]}$, for all state $s_0 \in \mathcal{D}$. The goal of safe offline learning is to learn a policy π from $\mathcal{D}$ that maximizes the cumulative reward while satisfying the cumulative cost constraint, denoted as

$$\begin{aligned} \max_{\pi} \quad & V_R(\pi) \\ \text{s.t.} \quad & V_C(\pi) \leq l \end{aligned} \tag{1}$$

where l is the safe constraint limit (a known constant).

DeepAveragers with costs (DAC)- MDPs

A DAC-MDP is defined in terms of an experience dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}$ from the true MDP $\mathcal{M}$ with continuous latent state space $\mathcal{S}$ and finite action space $\mathcal{A}$. The DAC-MDP $\tilde{M} = (\mathcal{S}, \mathcal{A}, \tilde{R}, \tilde{T})$ shares the same state and action spaces as $\mathcal{M}$, but defines the reward and transition functions in terms of empirical averages over the k nearest neighbors of (s, a) in $\mathcal{D}$.

The distance metric $d(s, a, s', a')$ between pairs is the euclidean distance between their state action representation. In particular, the distance between (s, a) and a data tuple (s_i, a_i, r_i, s_i') is given by $d(s, a, s_i, a_i)$. Also, we let $kNN(s, a)$ denote the set of indices of the k nearest neighbors to (s, a) in $\mathcal{D}$, noting that the dependence on $\mathcal{D}$ and d is left implicit. Given hyperparameters k (smoothing factor) and C_p (penalty factor) we can now specify the DAC-MDP reward and transition function.

$$\tilde{R}(s, a) = \frac{1}{k} \sum_{i \in kNN(s,a)} r_i - C_p \cdot d(s, a, s_i, a_i) \tag{2}$$

$$\tilde{T}(s, a, s') = \frac{1}{k} \sum_{i \in kNN(s,a)} I[s' = s_i'] \tag{3}$$

The reward for (s, a) is simply the average reward of the nearest neighbors with a penalty for each neighbor that grows linearly with the distance to a neighbor. Thus, the farther (s, a) is to its nearest neighbor set, the less desirable its immediate reward will be. The transition function is simply the empirical distribution over destination states of the nearest neighbor set.

Importantly, even though a DAC-MDP has an infinite continuous state space, it has a special finite structure. Since the transition function $\tilde{T}$ only allows transitions to states appearing as destination states in $\mathcal{D}$. We can view $\tilde{M}$ as having a finite core set of states $\mathcal{S}_D = \{s_i' \mid (s_i, a_i, r_i, s_i') \in \mathcal{D}\}$. States in this core do not transition to non-core states and each non-core state immediately transitions to the core for any action. Hence, the value of core states is not influenced by non-core states. Further, once the core values are known, we can compute the values of any non-core state via onestep look ahead using $\tilde{T}$. Thus, we can optimally solve a DAC-MDP by solving just its finite core.

Specifically, let $\tilde{Q}$ be the optimal Q-function of $\tilde{M}$. We can compute $\tilde{Q}$ for the core states by solving the finite MDP $\tilde{M}_D = \left(\mathcal{S}_D, \mathcal{A}, \tilde{R}, \tilde{T}\right)$. We can then compute $\tilde{Q}$ for any non-core state on demand via the following one-step look-ahead expression. This allows us to compute the optimal policy of $\tilde{M}$, denoted $\tilde{\pi}$, using any solver of finite MDPs.

$$\tilde{Q}(s, a) = \frac{1}{k} \sum_{i \in kNN(s,a)} r_i + \gamma \max_a \tilde{Q}(s_i', a) - C \cdot d(s, a, s_i, a_i)$$

SACC-MDPs

From a practical perspective our approach carries out the following steps as illustrated in Figure I. (1) We start with a static experience dataset, where the states are assumed to come from a continuous latent state space. (2) Next we compile the dataset into a tabular MDP over the "core states" of the SACC-MDP (those in the dataset). This compilation uses k-nearest neighbor (kNN) queries to define the reward, transition as well as the cost functions (Equation 2) functions of the core states. (3) Next we use an adapted GPU implementation of value iteration to solve for the tabular MDP's optimal Q-function and keep track of cumulative cost-function. (4) We then iteratively update the candidate action out Finally, this tabular Q-function is used to define the O-function over the entire DAC-MDP (Equation 3). Previously unseen states at test time are assigned Q-values and in turn a policy action via kNN queries over the core states.

SACC-MDP Definition

As in DAC-MDPs a SACC-MDP is defined in terms of an experience dataset $\mathcal{D} = \{(s_i, a_i, r_i, c_i, s_i')\}$ from the true MDP $\mathcal{M}$ with continuous latent state space $\mathcal{S}$ and finite action space $\mathcal{A}$. Similar to DAC-MDPs, $\tilde{M} = (\mathcal{S}, \tilde{\mathcal{A}}, \tilde{R}, \tilde{C}, \tilde{T})$ shares the same state space as $\mathcal{M}$, however action spaces is defined differently in order to handle continuous action spaces. here $\tilde{A} = \{a : a \in \Omega(s_i) i \in \mathcal{D}\}$ where Ω is a candidate action model which specifies a list of actions for any query state. We then define the reward, costs and transition functions in terms of empirical averages over the k nearest neighbors of (s, a) in $\mathcal{D}$ as specified below.

$$\tilde{R}(s,a) = \frac{1}{k} \sum_{i \in kNN(s,a)} r_i - C_p \cdot d\left(s, a, s_i, a_i\right) \quad (4)$$

$$\tilde{C}(s,a) = \frac{1}{k} \sum_{i \in kNN(s,a)} c_i \quad (5)$$

$$\tilde{T}\left(s, a, s'\right) = \frac{1}{k} \sum_{i \in kNN(s,a)} I\left[s' = s_i'\right] \quad (6)$$

Here the distance metric $d\left(s, a, s', a'\right)$, smoothness parameter k and a kNN function $kNN(s, a)$ are defined in a similar fashion as in DAC-MDPs as described in previous section.

The reward and costs for (s, a) is simply the average reward and costs of the nearest neighbors with a penalty for each neighbor that grows linearly with the distance to a neighbor. Thus, the farther (s, a) is to its nearest neighbor set, the less desirable its immediate reward will be. The transition function is simply the empirical distribution over destination states of the nearest neighbor set.

The SACC-MDP inherits the a finite structure for representing an infinite continuous state space from DAC-MDPs. Thus, we can optimally solve a SACC-MDP by solving just its finite core. Specifically, let $\tilde{Q}$ be the optimal Q-function of $\tilde{M}$. We can compute $\tilde{Q}$ for the core states by solving the finite MDP $\tilde{M}_D = \left(\mathcal{S}_D, \mathcal{A}, \tilde{R}, \tilde{C}, \tilde{T}\right)$. We can then compute $\tilde{Q}$ for any non-core state on demand as specified by DAC-MDPs.

$$\tilde{Q}(s,a) = \frac{1}{k} \sum_{i \in kNN(s,a)} r_i + \gamma \max_a \tilde{Q}\left(s_i', a\right) - C \cdot d\left(s, a, s_i, a_i\right)$$

Candidate Pruning for SACC-MDPs

While SACC-MDPs are defined to have constraints the optimal policy given by the VI solver does not take into considerations the cost as it is simply solving for an optimal policy that maximizes the cumulative reward V_R for each state. However, we define the VI solver such that it does keep track of the cumulative constraints V_c for the optimal policy given by the VI solver. Hence we defined a prune step for SACC-MDPs that prunes any candidate actions that is allowing the optimal policy to violate the constraints specified in the problem definition. i.e. $V_C \leq l$

Algorithm 1: Iteratively Prune and Solve

Data: $SACC - MDP\ \tilde{M}$
Result: $SACC - MDP\ \tilde{M}$
$S_{start} \leftarrow \eta_{\tilde{M}}$
while $V_C(s) > l\ for\ s \in S_{start}$ **do**
  # Prune Sub Routine Start #
  $\tau = sampleTrajectory(S_{start}, \tilde{M})$
  $saPairs = \{(s,a) : s \in \tau, a \in \Omega(s)$
    $if\ \tilde{C}(s, \tilde{\pi}^*(s)) - \tilde{C}(s,a) > 0\}$
  for (s,a) in $\Omega(s)$:
    $if\ \tilde{C}(s,a) > C_{max}$:
      $\Omega(s) = \Omega(s) - a$
  # Prune Sub Routine End #

  update $\tilde{M}$ *according to updated* Ω
  solve $\tilde{M}$
end

We define a $PRUNE$ subroutine such that for the it updates a candidate action function Ω for any one of the state in the current optimal trajectory path. Here the subroutine calculates the reward advantage as well as the cost advantage of the given optimal action has over any candidate actions for the states in the optimal trajectory. We then choose an state action pair among the selected states where there is a positive cost advantage and the least amount of reward advantage. Here the solution to the pruned SACC-MDP is guaranteed to amount to an optimal policy with less cumulative cost for at least one state. Iterating this process will allow us to find an optimal policy that satisfies all the safety constraints while maximizing the cumulative reward.

Handling Continuous Action Spaces in DAC-MDPs

Essentially, the DAC-MDP framework relies on a candidate action model which outputs a list of candidate actions for each state in the core state as well as any new query state. While, we simply consider all possible actions as the candidate set of actions for discrete action spaces, this is not possible for continuous action spaces. Hence we have to define a candidate action model Ω such that it returns a set of candidate actions for any new state. Here the selection of Ω can be viewed as a part of the solution when adapting DAC-MDPs for our approach.

Experimental Setup

Lacking existing solution, we propose a custom 2D grid world environment called "watch-your-steps-grid-world". The grid world has a start state distribution, goal area, and some safety constraints attached to its action space. Here the action-space comprises of two real valued components [heading, step_size]. The agent will traverse the given step_size in the specified heading. The agent will be given a reward of 100 when it reaches the goal region. And the safety constraint is defined over a penalty function defined over the step_size, where large steps results in failure, hence we would like to put a constraint that is non-linear in the step

size. Also the agent receives reward of -1 for each step that it takes.

The environment includes different regions that will lead to different step penalties. For example a reward of -10 is given for each time the agent lands in lava.

We consider three domains. We first consider a no-lava-room where there is no lava, so the agents incentive is to go to the goal as fast as possible while adhering to the constraints. Here the agent should find the faster route as the constraints are comparatively relaxed.

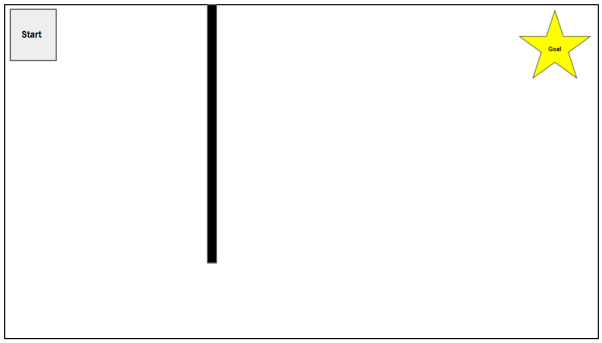

Figure 2: No lava with small wall.

Our second room shall have a bridge connecting two regions spanning a lava region. The agent has to make sure the actions that have the highest safety constraint will be within the lava region in order to maximize the reward.

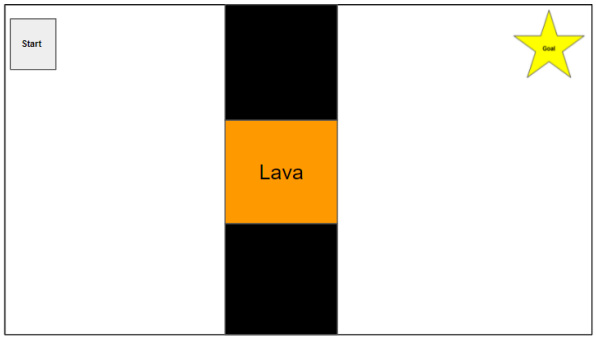

Figure 3: Short bridge of lava.

The final room shall consist of a long lava bridge where the risk of trying to cross it is not worth trying. Hence, the agent will have to not take any actions and say that the risk of taking actions is too great.

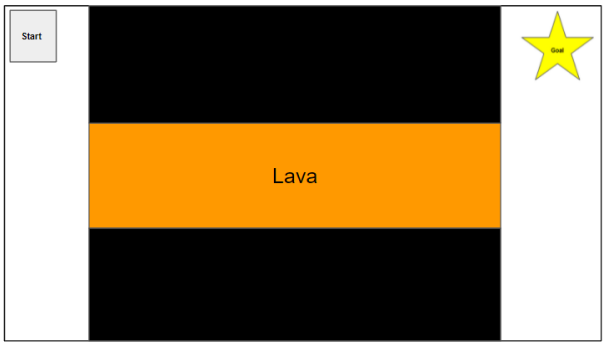

Figure 4: Long bridge of lava.

Fast adaptability may be shown when we are constantly adapting the safety function as we are learning from the data. The agent will choose larger steps in the Lava environment as soon as it finds that the safety constraints has been updated.

Action Plan / Deliverable

Feb 10: Domain Simulator Completed + Data Collected
Feb 20: Trail DAC-MDP Completed & Evaluating Results
March 7: Experiments Finalized & Report mostly complete
Final Deliverable: Final project presentation, 9th of March

References

Altman, E. 1999. *Constrained Markov decision processes*, volume 7. CRC Press.