Formatting Instructions for RLC Submissions

Anonymous authors
Paper under double-blind review

Abstract

Hello

1 Introduction

In reinforcement learning, ...

2 Problem Setting

Reinforcement learning is often modeled as a Markov decision process (MDP) given by the tuple $M = (\mathcal{S}, \mathcal{A}, P, R)$. In this model, an agent and environment interact over a sequence of time steps t. At each time step, the agent receives a state $S_t \in \mathcal{S}$ from the environment, where $\mathcal{S}$ denotes the set of all possible states. The agent uses the information given by the state to select and action A_t from the set of possible actions $\mathcal{A}$. Based on the state of the environment and the agents behavior, i.e. action, the agent receives a scalar reward $R_t = R(S_t, A_t)$ and transitions to the next state $S_{t+1} \in \mathcal{S}$ according to the state-transition probability $P(s'|s, a) = P(S_{t+1} = s'|S_t = s, A_t = a)$ for each $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$.

The behavior of an agent is given by a policy $\pi(a|s)$, which is a probability distribution over actions given a state. The agent's goal is to learn the optimal policy, $\pi^\star$, which is the policy that maximizes the expected discounted return

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+1} + \cdots = \sum_{k=1}^{T-t} \gamma^{k-1} R_{t+k-1}$$

either for a discounted factor $\gamma \in [0, 1)$ when the task is continuing, $T = \infty$, or $\gamma \in [0, 1]$ and $T < \infty$ in episodic task.

Through the process of policy iteration Sutton & Barto (2018), Monte-Carlo algorithms strive to maximize the expected return by computing value-functions that estimate the expected future returns. The state-value function is quantifies the agent's expected return starting in state s and following policy π, i.e. $v^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$. When learning to control in RL, we often want to estimate the action-value function, which is the agent's expected return starting in state s, taking action a and then following policy π, i.e.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \,|\, S_t = s, A_t = a].$$

In many RL problems of interest, the state-space $\mathcal{S}$ is prohibitively large and function approximation is needed in order to enable sample-efficient learning. For clarity, the algorithmic ideas in this paper are initial presented as linear solution methods, but can be extended to arbitrary function approximation schemes. Given a set of linear features $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and a trajectory $S_1, A_1, R_1, \ldots, R_T, S_{T+1}$ collected following policy π in M, the action values can be approximated by solving the least-squares problem

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \sum_t \left(\phi(S_t, A_t)^\top \theta - G_t\right)^2$$

and letting $q_\pi(s,a) \approx \phi(s,a)^\top \hat{\theta}$. With good features and enough data, the minimizer of the least-squares problem is a good approximation of the action-value function.

2.1 Monte-Carlo Policy Iteration

Our main objective in *policy iteration* is to maximize the expected return from observations collected via interacted with the environment, ideally using as few updates as possible. Monte-Carlo policy iteration works by first performing policy evaluation to learn the action-values using the returns and then acting greedily with respected to the learned action-values. Monte-Carlo policy iteration can be summarized as

1. (Policy evaluation): Collect $N \in \mathbb{N}$ trajectories $S_1, A_1, R_1, \ldots, R_T, S_{T+1}$ following policy π in MDP M. Then solve the least squares problem

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(\phi(S_{t,i}, A_{t,i})^\top \theta - G_{t,i}\right)^2$$

   where $S_{t,i}$ corresponds to the t-th state in the i-th trajectory for $i \in \{1, \ldots, N\}$ and $t \in \{1, \ldots, T\}$.

2. (Greedify): Take the new policy to be the policy that is greedy with respect to the approximate action-values, i.e. $\pi(s) = \arg\max_{a \in \mathcal{A}} \phi(s,a)^\top \hat{\theta} \approx \arg\max q^{\pi'}(s,a)$ where π' is the previous policy.

In this work, we propose further discretizing, or subsampling, the trajectory in order to learn with as few updates as possible. Zhang et al. (2024) establishes the advantage of updating using only a subset of the tuples in the trajectory when performing Monte-Carlo value estimation. In their work, they show, both theoretically and empirically, that a simple uniform discretization over the trajectory, i.e. updating using every M-th tuple in the trajectory, leads to significantly better value estimation when the number of updates is fixed. In the next section we will introduce both a uniform scheme for discretizing the Monte-Carlo updates and an adaptive scheme for discretizing the updates.

3 Algorithm

In this section, we detail both a uniform and adaptive method for discretizing the trajectories when performing updates in Monte-Carlo policy iteration.

Proposition 3.1. *Given tolerance $\tau \geq 0$ and a real-valued list xs of size N, Algorithm 1 returns an approximate sum Q that satisfies $|Q - sum(xs)| \leq \tau$.*

Proof. This proof will follow by induction. Assume that $N < 3$. Then Algorithm 1 returns $Q = \text{sum}(xs)$. Therefore $|Q - \text{sum}(xs)| = |\text{sum}(xs) - \text{sum}(xs)| = 0 \leq \tau$ for all $\tau \geq 0$.

Now assume that for a fixed $N > 2$ and $\tau \geq 0$ it holds that $|Q - \text{sum}(xs)| \leq \tau$. □

References

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

Zichen Vincent Zhang, Johannes Kirschner, Junxi Zhang, Francesco Zanini, Alex Ayoub, Masood Dehghan, and Dale Schuurmans. Managing temporal resolution in continuous value estimation: A fundamental trade-off. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Algorithm 1 ADAPTIVE

Input: A list of real numbers xs, a list of integers idx, dictionary $idxes$ and tolerance $\tau \geq 0$
$N = \text{length}(xs)$.
if $N \geq 2$ **then**
  $idxes[idx[0]] = 1$ and $idxes[idx[-1]] = 1$
  $Q = N \cdot (xs[0] + xs[-1])/2$.
else
  $idxes[idx[0]] = 1$
  return $xs[0]$
end if
$\varepsilon = |Q - \text{sum}(xs)|$
if $\varepsilon \geq \tau$ **then**
  $c = \lfloor N/2 \rfloor$.
  $Q, idxes = \text{ADAPTIVE}(xs[:c], idx[:c], \tau/2, idxes) + \text{ADAPTIVE}(xs[c:], idx[c:], \tau/2, idxes)$
end if
return $Q, idxes$.

Algorithm 2 UNIFORM

Input: Integers N, M.
$idxes = []$, $idx = 0$ and $i = 0$
while $idx < N - 1$ **do**
  $idx = i * M$
  $idxes$.append$(i * M)$
  $i = i + 1$
end while
return $idxes$.

A Experimental Details

Algorithm 3 MONTE-CARLO POLICY ITERATION

Input: Action-value features $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$, number of evaluation trajectories N, a tolerance $\tau \geq 0$ and uniform spacing integer M and dictionary of update points pivots $= \{\}$.
Initialize the action value weights $\theta_0 \in \mathbb{R}^d$ arbitrarily
for $i = 1, 2, \ldots$ **do**
  $\pi_i(s) = \arg\max_{a \in \mathcal{A}} \phi(s, a)^\top \theta_{i-1}$
  for $j = 1, 2, \ldots, N$ **do**
    Collect trajectories $S_{1,i,j}, A_{1,i,j}, R_{1,i,j}, \ldots, R_{T,i,j}, S_{T+1,i,j}$ using policy π_i
    if uniform **then**
      pivots$[i, j]$ = UNIFORM(T, M)
    else if adaptive **then**
      temp, pivots$[i, j]$ = ADAPTIVE$([R_{1:T,i,j}], \{1, 2, \ldots, T\}, \{\}, \tau])$
    else
      pivots$[i, j] = \{1, 2, \ldots, T\}$
    end if
  end for
  Update $\theta_i = \arg\min_{\theta \in \mathbb{R}^d} \sum_{i,j} \sum_{t \in \text{pivots}[i,j]} \left(\phi(S_{t,i,j}, A_{t,i,j}) - G_{t,i,j}\right)^2$
end for