Formatting Instructions for RLC Submissions

Anonymous authors
Paper under double-blind review

Abstract

Hello

1 Introduction

In reinforcement learning, ...

2 Preliminaries

Reinforcement learning is often modeled as a Markov decision process (MDP), which is characterized by the tuple $M = (\mathcal{S}, \mathcal{A}, P, \gamma, R)$. In this model, an agent and environment interact over a sequence of time steps denoted as 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. the 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 actual objective of the learner is to maximize 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 for $\gamma \in [0, 1]$ and $T < \infty$, i.e., in an episodic task. The latter return is influenced by the actions taken by the agent, thus its behavior defined through a policy $\pi(a|s)$, which is a probability distribution over actions given a state. Then the agent's goal is to learn the optimal policy $\pi^\star$, which is the policy that maximizes the expected return.

Through the process of policy iteration Sutton & Barto (2018), Monte-Carlo algorithms strive to maximize the expected return by approximating the value-function, which is indeed defined as its expectation. The state-value function quantifies the agent's expected return starting in state s and following policy π, i.e. $v^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$. Given that the task in RL is to face the control problem going beyond prediction, 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. Note that the algorithmic ideas presented in this paper will be given as linear solution methods, however their applicability is broader as they 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

$$\hat{\theta} = \operatorname*{arg\,min}_{\theta \in \mathbb{R}^d} \sum_{t=1}^{T} \left(\phi(S_t, A_t)^\top \theta - G_t\right)^2$$

and assuming that the feature yield a good linear representation of the action-value function, it holds that $q_\pi(s,a) \approx \phi(s,a)^\top \hat{\theta}$.

2.1 Monte-Carlo Policy Iteration

The main objective in *policy iteration* is to maximize the expected return by selecting the policy that maximizes the approximated action-value function, which is learnt from observations collected via interaction with the environment. Given that the number of point in a trajectory negatively affects computational time of the procedure and memory usage, ideally this should be done using as few samples 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.

2.2 Reducing the Number of Updates in Monte-Carlo Policy Iteration

The main question this paper aims to answer is if the number of samples from a trajectory used to approximate the action-value function can be reduced by means of an adaptive algorithm. Usually the system is assumed to have a fixed discretization step, and current algorithms makes use of all the samples they are given. However it has been recently established by Zhang et al. (2024) that using only a subset of the tuples of the trajectory yields a better result 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 this work, we also propose further discretizing, or subsampling, the trajectory in order to learn with as few updates as possible. By simply considering a coarse discretization we might lose too much information about the behaviour of the agent, while keeping the whole trajectory would lead to a computationally expensive algorithm. We will show that adapting the discretization step allows the algorithm to use less samples while still producing a good approximation of the action-value function, and consequently achieving a good policy. 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. In Algorithm 3, we detail the variant of Monte-Carlo policy iteration that is trained using either a uniform or adaptive discretization scheme. Given a function that approximates the action-values, Algorithm 3 runs the ε-greedy policy induced by the current approximate action-value function and collects N trajectories of, possibly varying, length T_j, where $j \in \{1, 2, \ldots, N\}$. If Algorithm 2 is used to discretize the updates, then Algorithm 3 performs updates with using every M-th state-action-return tuple in the trajectory, i.e. for $t \in \{1, M, 2M, \ldots, T_j\}$ we use tuples (S_t, A_t, G_t) in computing the updated parameter θ_i.

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 $i = 1$
$idxes$.append(1)
while $idx < N$ **do**
  $idx = i * M$
  $idxes$.append$(i * M)$
  $i = i + 1$
end while
$idxes$.append(N)
return $idxes$.

The choice of M in the uniform d

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.

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, a dictionary of update points pivots $= \{\}$ and exploration parameter $\varepsilon \in [0, 1]$.
Initialize the action value weights $\theta_0 \in \mathbb{R}^d$ arbitrarily
for $i = 1, 2, \ldots$ **do**
  $\pi_i(s) = \arg\max_{a \in \mathcal{A}} \phi(s, a)^\top \theta_{i-1}$ with probability $1 - \varepsilon$, else $\pi_s \sim \text{Uniform}(\mathcal{A})$.
  for $j = 1, 2, \ldots, N$ **do**
    Collect trajectories $S_{1,i,j}, A_{1,i,j}, R_{1,i,j}, \ldots, R_{T_j,i,j}, S_{T_j+1,i,j}$ using policy π_i
    if uniform **then**
      pivots$[i, j]$ = UNIFORM(T_j, M)
    else if adaptive **then**
      temp, pivots$[i, j]$ = ADAPTIVE$([R_{1:T_j,i,j}], \{1, 2, \ldots, T_j\}, \{\}, \tau])$
    else
      pivots$[i, j] = \{1, 2, \ldots, T_j\}$
    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