Algorithms in reinforcement learning

Pawel Wocjan*

November 29, 2018

Abstract

This is a brief overview of the most important algorithms in reinfocement learning

1 Markov decision processes

1.1 The agent-environment interface

The agent and environment interact at each of a sequence of discrete time steps, $t = 01, 2, \ldots$. At each time step t, the agent receives some representation of the environement's *state*, $S_t \in \mathcal{S}$, and on that basis selects an *action* $A_t \in \mathcal{A}(s)$. One time step later, in part as a consequence of its action, the agent receives a numerical reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}. The MDP and agent together thereby give rise to a sequence or *trajectory* that begins like this:

$$S_0, A_0, R_1, S_1, A_1, S_2, A_2, R_3, \ldots$$

2 Dynamic programming

3 Monte Carlo learning

3.1 Monte Carlo prediction

4 Temporal difference learning

4.1 TD prediction

Both TD and Monte Carlo methods use experience to solve the prediction problem. Given some experience following a policy π, both methods update their estimate V of v_π for the nonterminal states S_t occuring in that experience. Roughly speaking, Monte Carlo methods wait until the return following the visit is known, then use that

*`wocjan@cs.ucf.edu`, Department of Computer Science, University of Central Florida, USA

return as a target for $V(S_t)$. A simple every-visit Monte Carlo method suitable for nonstationary environments is

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]\,, \tag{1}$$

where G_t is the actual return following time t, and α is a constant step-size parameter.

Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S_t)$ (only then is G_t known), TD methods need to wait only until the next step. At time $t+1$ they immediately form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The simplest TD method makes the update

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2}$$

immediately on transition to S_{t+1} and receiving R_{t+1}. In effect, the target for the Monte Carlo update is G_t, whereas the target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$. This TD method is called TD(0), or one-step TD. It is a special case of the TD(λ) and n-step TD methods that described in this note as well.

Algorithm 1 Tabular TD(0) for estimating v_π

Input: the policy to π to be evaluated
Algorithm parameter: step size α
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal)$=0

for each episode **do**
  Initialize S
  for each step of episode **do**
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
    if S is terminal **then** break
  end for
end for

4.2 Sarsa: on-policy TD control

In Subsection 4.1 we considered transitions from state to state and learned the values of states. Now we consider transitions from state-action pair to state-action pair, and learn the values of state-action pairs. We use the update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]\,. \tag{3}$$

which is done after every transition from a non-terminal state S_t. ???

Algorithm 2 Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

for each episode **do**
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  for each step of episode **do**
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
    if S is terminal **then** break
  end for
end for

4.3 Expected Sarsa

Consider the learning algorithm that is just like Q-learning except that instead of the maximum over next state-action pairs it uses the expected value, taking into account how likely each action is under the current policy. That is, consider the algorithm with the update rule

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)\Big] \\ &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big], \end{aligned}$$

but that otherwise follows the schema of Q-learning.

4.4 Q-learning: off-policy TD control

One of this early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning, defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]\,. \tag{4}$$

In this case, the learned action-value function, Q, directly approximates q_*, the optimal action-value function, independent of the policy being followed. The policy still has an effect in that it determines which state-action pairs are visited an updated. However, all that is required for the correct convergence is that all pairs continue to be updated.

4.5 Double Q-learning

All previously discussed control algorithms involve maximization in the construction of their target policies. In these algorithms, a maximum over estimated values is used implicitly as an estimate of the maximum value, which can lead to a significant positive

Algorithm 3 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, except that $Q(terminal, \cdot) = 0$

```
for each episode do
    Initialize S
    for each step of the episode do
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        Q(S, A) ← Q(S, A) + α[R + γ max_a Q(S', a) − Q(S, A)]
        S ← S'
        if S is terminal then break
    end for
end for
```

bias. To see why, consider a single state s where there are many actions a whose true values $q(s, a)$, are all zero but whose estimated values $Q(s, a)$, are uncertain and thus distributed some above and some below zero. The maximum of these values is zero, but the maximum of the estimates is positive, a positive bias. We call this maximization bias.

Algorithm 4 Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

```
for each episode do
    Initialize S
    for each step of episode do
        Choose A from S using the policy ε-greedy in Q1 + Q2
        Take action A, observe R, S'
        With probability 0.5
          Q1(S, A) ← Q1(S, A) + α[R + γQ2(S', argmax_a Q1(S', a)) − Q1(S, A)]
        else
          Q2(S, A) ← Q2(S, A) + α[R + γQ1(S', argmax_a Q2(S', a)) − Q2(S, A)]
        S ← S'
        if S terminal then break
    end for
end for
```

References