

Tabular True On-Line SARSA(λ)

Tech Report

Juan-Antonio Fernández-Madrigal*

Systems Engineering and Automation Dpt.
University of Málaga (Spain)
January 18, 2023

1 Introduction

In this document we deduce a tabular algorithm for True On-Line SARSA(λ) from the non-tabular, linear approximation in [1]. True On-Line SARSA(λ) boosts the performance of a classical SARSA algorithm by using the same *eligibility traces* technique of TD(λ) and thus doing multiple backups at each iteration; however, TOLSL guarantees that these backups are true equivalent of the forward model (looking at the future) on which the eligibility traces (looking at the past) are based, while SARSA is only an approximation to that.

In any of these algorithms, the proportion of change to be made at iteration k in the value of $Q(s, a)$ corresponding to the backup is stored and it decays at each further iteration if state s is not visited; initially, these *eligibility traces* are set to 0 for all states and actions.

2 The linear case

The True On-Line SARSA(λ) algorithm presented here for the tabular case has been deduced from the one at [1] (p. 307). There, the algorithm is presented in its form for a general linear approximation of the Q function; concretely, they do:

*This work has been funded by the research project *GARTHIM. Remote Reinforcement Learning for Mobile Robots Connected to Internet* (code UMA18-FEDERJA-113), granted by the Junta de Andalucía regional government through EU ERDF.

$$Q(s, a, \mathbf{w}) \simeq \mathbf{w}^T \cdot \mathbf{x}(s, a) \quad (1)$$

where $\mathbf{w}$ is a $m \times 1$ vector that maps linearly the also $m \times 1$ vector $\mathbf{x}(s, a)$ of *features* into the approximate value function $Q(s, a)$. The mapping $\mathbf{w}$ is updated at each iteration to improve that approximation, while the features must be defined previously for all given pairs (s, a) . The idea is that a finite number of features (m) suffices to represent faithfully enough the actual Q function, at least at the most interesting (s, a) pairs.

Also, in the linear case the *eligibility traces* are defined as an $m \times 1$ vector $\mathbf{z}$.

3 The tabular case

In a tabular formulation of the algorithm, we pursue that at every step the values stored in the Q matrix coincide with the ones that a continuous linear approximation would yield at the given (s, a) pairs. The simplest way of accomplishing that is to assimilate the tabular case to a *state aggregation* formulation of the continuous action-value function, i.e., each discrete state/action pair will be identified by one of the features $\mathbf{x}$ of the linear case being 1 and the rest 0. If both learning modes (discrete and continuous) are to produce the same results, the values of the Q matrix in the tabular case should coincide with the values of the weights $\mathbf{w}$ in the linear formulation; learning will be equivalent in both (assuming the same sequence of experiences) if that coincidence occurs at every step.

We can formalize this equivalence, based on a tabular setting with S states and A actions, and with $m = S \times A$, as follows:

$$\begin{aligned} \mathbf{w}^T &= \{q^{1,1}, q^{1,2}, \dots, q^{1,A}, q^{2,1}, \dots, q^{2,A}, \dots, q^{S,1}, q^{S,2}, \dots, q^{S,A}\} \\ \mathbf{x}^T(s, a) &= \{0, 0, 0, \dots, 0, \underbrace{1}_{\text{at position } (s-1) \cdot A + a + 1, \text{ i.e., at } q^{s,a}}, 0, \dots, 0\} \end{aligned} \quad (2)$$

where $q^{i,j}$ are the value stored in $Q(i, j)$ in the tabular case. With this definition, state aggregation is achieved: $Q(s, a, \mathbf{w}) = \mathbf{w}^T \cdot \mathbf{x}(s, a) = q^{s,a}$.

3.1 Calculating δ_k

δ_k is the change to produce in the value $Q_k(s_k, a_k)$ at the pair (s_k, a_k) during iteration k , where s_k is the state of the agent at the beginning of that iteration and a_k the action selected to do then.

If, after doing the action, the agent gets at state s'_k , obtains reward r_k , and selects a'_k as the next action to do, the update on $Q(s_k, a_k)$ should be, in the linear approximation algorithm:

$$\delta_k = r_k + \gamma Q'_k - Q_k$$

$$\begin{aligned} \text{with } Q_k &= \mathbf{w}^T \mathbf{x}(s_k, a_k) = q_k^{s_k, a_k} \\ Q'_k &= \mathbf{w}^T \mathbf{x}(s'_k, a'_k) = q_k^{s'_k, a'_k} \end{aligned} \tag{3}$$

therefore:

$$\delta_k = r_k + \gamma q_k^{s'_k, a'_k} - q_k^{s_k, a_k}$$

Recall that $q_k^{i,j}$ is the value that was stored in the (i, j) entry of the Q matrix at the end of the previous iteration in the case of a tabular SARSA.

3.2 Updating traces

After δ_k , the linear approximation algorithm updates all eligibility traces at once. In the linear case this is done with:

$$\underbrace{\mathbf{z}_{k+1}}_{m \times 1} = \underbrace{\lambda \gamma \mathbf{z}_k}_{m \times 1} + \left(1 - \alpha \gamma \lambda \underbrace{\mathbf{z}^T \mathbf{x}(s_k, a_k)}_{1 \times 1} \right) \underbrace{\mathbf{x}(s_k, a_k)}_{m \times 1} \tag{4}$$

where (s_k, a_k) is the state and action at the beginning of iteration k .

In our tabular mapping (recall section 3, eq. 2), $m = S \times A$, and thus each element in $\mathbf{z}$ corresponds to the eligibility trace $e(s, a)$ of a given pair (s, a) . Therefore, we can update the trace of any pair (i, j) as follows, because $\mathbf{x}(s, a)$ has 1 only in the position corresponding to (s, a) and 0 everywhere else:

$$e_{k+1}(i, j) = \begin{cases} \lambda\gamma e_k(s_k, a_k) + (1 - \alpha\gamma\lambda e_k(s_k, a_k)) \cdot 1 & \text{if } (i, j) = (s_k, a_k) \\ \lambda\gamma e_k(i, j) & \text{otherwise} \end{cases}$$

Rearranging terms in the first case: (5)

$$e_{k+1}(i, j) = \begin{cases} \lambda\gamma(1 - \alpha)e_k(s_k, a_k) + 1 & \text{if } (i, j) = (s_k, a_k) \\ \lambda\gamma e_k(i, j) & \text{otherwise} \end{cases}$$

3.3 Updating the Q matrix

In the linear case, the Q function is approximated through the $\mathbf{w}$ vector, that is updated this way:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha \cdot (\delta_k + Q_k - Q_k^{old}) \cdot \mathbf{z}_{k+1} - \alpha \cdot (Q_k - Q_k^{old}) \cdot \mathbf{x}(s_k, a_k) \quad (6)$$

Here, Q_k^{old} is a scalar that is updated at iteration $k - 1$ as $Q_k^{old} = Q'_k = q_{k-1}^{s'_{k-1}, a'_{k-1}}$ (initially, $Q_1^{old} = 0$). Also, in our case $Q_k = q_k^{s_k, a_k}$.

Substituting the values for the tabular case, and recalling that the $\mathbf{w}$ vector has $m = S \times A$ entries, each one corresponding to a pair (s, a) , we can compute the new value of each entry of our Q matrix this way:

$$q_{k+1}^{i,j} = \begin{cases} q_k^{s_k, a_k} + \alpha \cdot (\delta_k + q_k^{s_k, a_k} - Q_k^{old}) \cdot e_{k+1}(s_k, a_k) - \alpha \cdot (q_k^{s_k, a_k} - Q_k^{old}) \cdot 1 & (i, j) = (s_k, a_k) \\ q_k^{i,j} + \alpha \cdot (\delta_k + q_k^{s_k, a_k} - Q_k^{old}) \cdot e_{k+1}(i, j) & \text{otherwise} \end{cases} \quad (7)$$

Substituting Q_k^{old} , the concrete iterations to use in that term are revealed:

$$q_{k+1}^{i,j} = \begin{cases} q_k^{s_k, a_k} + \alpha \cdot \left(\delta_k + q_k^{s_k, a_k} - q_{k-1}^{s'_{k-1}, a'_{k-1}} \right) \cdot e_{k+1}(s_k, a_k) - \alpha \cdot \left(q_k^{s_k, a_k} - q_{k-1}^{s'_{k-1}, a'_{k-1}} \right) & (i, j) = (s_k, a_k) \\ q_k^{i,j} + \alpha \cdot \left(\delta_k + q_k^{s_k, a_k} - q_{k-1}^{s'_{k-1}, a'_{k-1}} \right) \cdot e_{k+1}(i, j) & \text{otherwise} \end{cases} \quad (8)$$

3.4 Complete algorithm

In order to get a practical algorithm, we add a limit to the number of eligibility traces updated at each iteration, as explained in [2]: basically, we keep a FIFO queue of past states of maximum size n ; those states in the queue are the only ones to be considered for the traces backup. Thus, a trace must have a minimum threshold τ for being considered in the backup. The relation between n and τ is explained in that paper: $n < \lfloor \frac{Ln(\tau)}{Ln(\gamma\lambda)} \rfloor$.

In episodic learning, the TOSL parameters (queue of past states, eligibility traces, etc.) should be reset at the end of each episode as long as that ending state is not the initial state of the next episode; otherwise, the TOSL algorithm is misled.

Also, in [2] the following parameters are advised for a diversity of robotic tasks: $\lambda = 0.9$, $\alpha = 0.1$, $\gamma = 0.9$, $\tau = 0.01$, $n > 21.85$, and a softmax / Boltzmann action selection strategy with $T = 1$.

Algorithm 1 Tabular True On-Line SARSA(λ) with limited traces

```
1: function TAB_TOLSL_step( $k, F_k, e_k, Q_k^M, Q_k^{old}, s_k, a_k, r_k, s'_k, a'_k; \gamma, \alpha, \lambda, \tau, n, S, A$ )  
  
   ( $a_k$  has been executed,  $r_k$  and  $s'_k$  obtained, and  $a'_k$  chosen already;  $Q^M$  is the  $Q$   
   matrix)  
  
2:   if  $k = 1$  then ▷  $k$ : iteration within episode, from 1  
3:      $F_k \leftarrow \emptyset$  ▷ Clear FIFO buffer of past states  
4:      $e_k \leftarrow \mathbf{0}_{S \times A}$  ▷  $e$  is the eligibility traces matrix  
5:      $Q_k^{old} \leftarrow 0$   
6:   end if  
  
7:    $F_{k+1} \leftarrow F_k$   
8:   if  $s_k \in F_{k+1}$  then (at most, one appearance of  $s_k$  can be found)  
9:     Extract  $s_k$  from  $F_{k+1}$   
10:  end if  
11:  Push  $s_k$  into  $F_{k+1}$  while keeping  $size(F_{k+1}) \leq n$  (erasing oldest element if not)  
  
12:   $Q_{k+1}^M \leftarrow Q_k^M$   
13:   $e_{k+1} \leftarrow e_k$   
14:   $\delta_k \leftarrow r_k + \gamma Q_k^M(s'_k, a'_k) - Q_k^M(s_k, a_k)$   
15:   $Q_k^{diff} \leftarrow Q_k^M(s_k, a_k) - Q_k^{old}$   
16:  for all  $i \in F_{k+1}$  do  
17:    for all  $j \in [1, A]$  do  
18:      if  $(i, j) \neq (s_k, a_k)$  then  
19:        if  $e_k(i, j) > \tau$  then  
20:           $e_{k+1}(i, j) \leftarrow \gamma \lambda \cdot e_k(i, j)$   
21:           $Q_{k+1}^M(i, j) \leftarrow Q_k^M(i, j) + \alpha \cdot (\delta_k + Q_k^{diff}) \cdot e_{k+1}(i, j)$   
22:        else  
23:           $e_{k+1}(i, j) \leftarrow 0$   
24:        end if  
25:      else  
26:         $e_{k+1}(s_k, a_k) \leftarrow \gamma \lambda (1 - \alpha) \cdot e_k(s_k, a_k) + 1$   
27:         $Q_{k+1}^M(s_k, a_k) \leftarrow Q_k^M(s_k, a_k) + \alpha \cdot (\delta_k + Q_k^{diff}) \cdot e_{k+1}(s_k, a_k) - \alpha \cdot Q_k^{diff}$   
28:      end if  
29:    end for  
30:  end for  
  
31:   $Q_{k+1}^{old} \leftarrow Q_k^M(s'_k, a'_k)$   
32:  return  $F_{k+1}, e_{k+1}, Q_{k+1}^M, Q_{k+1}^{old}$  ( $s_{k+1} \leftarrow s'_k, a_{k+1} \leftarrow a'_k$ )  
  
33: end function
```

References

- [1] R.S. Sutton, A.G. Barton, *Reinforcement Learning. An Introduction*, 2nd ed., The MIT press, available at <http://incompleteideas.net/book/the-book-2nd.html>. Retrieved on June 29, 2021.
- [2] A. Martínez-Tenor, J.A. Fernández-Madrigal, A. Cruz-Martín, J. González-Jiménez, *Towards a Common Implementation of Reinforcement Learning for Multiple Robotic Tasks*, Expert Systems With Applications, vol. 100, pp. 246-259, 2018.