Unifying Q-Learning and Sarsa with Eligibility Traces

Markus Dumke
Department of Statistics
Ludwig-Maximilians-Universität München
markus.dumke@campus.lmu.de

Abstract

Temporal-difference (TD) learning is an important field in reinforcement learning. The most used TD control algorithms are probably Sarsa and Q-Learning. While Sarsa is an on-policy algorithm, Q-Learning learns off-policy. The Q(σ) algorithm (Sutton and Barto (2017)) unifies both. This paper extends the Q(σ) algorithm to an online multi-step algorithm using eligibility traces. The new proposed eligibility trace is a weighted average between the eligibility traces usually used with Sarsa and Q-Learning. Experiments suggest that the new algorithm called Q($\sigma\lambda$) outperforms the classical TD control methods Sarsa and Q-Learning as well as the n-step Q(σ) algorithm.

1 Introduction

Reinforcement Learning is a field of machine learning addressing the problem of sequential decision making. It is formulated as an interaction of an agent and an environment over a number of discrete time steps t. At each time step the agent chooses an action A_t based on the environment's state S_t. The environment takes A_t as an input and returns the next state observation S_{t+1} and reward R_{t+1}, a scalar numeric feedback signal.

The agent is thereby following a policy π, which is the behavior function mapping a state to action probabilities

$$\pi(a|s) = P(A_t = a|S_t = s). \tag{1}$$

Markov-Decision-Processes (MDP) provide the theoretical framework for reinforcement learning. An MDP models state transitions using a probability matrix $P^a_{ss'} = P(S_{t+1} = s'|S_t = s, A_t = a)$. If the interaction between agent and environment ends after a finite number of time steps we call this an episodic task else we speak of a continuing task.

In reinforcement learning the goal is to maximize the return G_t, which is the sum of discounted rewards over the lifetime of the agent,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{T-1} \gamma^k R_{t+1+k}, \tag{2}$$

where $\gamma \in [0, 1]$ is the discount factor and T is the length of the episode or infinity for a continuing task.

While rewards are short-term signals about the goodness of an action, values represent the long-term value of a state or state-action pair. The action value function $q_\pi(s,a)$ is defined as the expected return taking action a from state s and thereafter following policy π:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]. \tag{3}$$

Value-based reinforcement learning is concerned with finding the optimal action value function $q_* = \max_\pi q_\pi$. Temporal-difference learning is a class of model-free methods which estimates q_π from sample transitions and iteratively updates the estimated values using observed rewards and estimated values of successor actions. At each step an update of the following form is applied:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\, \delta_t, \tag{4}$$

where Q is an estimate of q_π, α is the learning rate (also called step size) and δ_t is the TD error, the difference between our current estimate and a newly computed target value. The current Q value is therefore shifted by the TD error multiplied with the learning rate. The following TD control algorithms can all be characterized by their TD error.

When the action values Q are represented as a table we call this tabular reinforcement learning, else we speak of approximate reinforcement learning, e.g. when using a neural network to compute the action values. For sake of simplicity the following analysis is done for tabular reinforcement learning but can be easily extended to function approximation.

2 TD control algorithms: From Sarsa to Q(σ)

Sarsa (Rummery and Niranjan (1994)) is a temporal-difference learning algorithm which samples states and actions using an epsilon-greedy policy and then updates the Q values using the following TD error

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t). \tag{5}$$

The term $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ is called the TD target and consists of the reward plus the discounted value of the next state and next action.

Sarsa is an on-policy method, i.e. the TD target consists of $Q(S_{t+1}, A_{t+1})$, where A_{t+1} is sampled using the current policy. In general the policy used to sample the state and actions - the so called behaviour-policy μ - can be different from the target policy π, which is used to compute the TD target. If behaviour and target policy are different we call this off-policy learning. An example for an off-policy TD control algorithm is the well known Q-Learning algorithm proposed by Watkins (1989). As in Sarsa states and actions are sampled using an exploratory behaviour policy, e.g. an ϵ-greedy policy, but the TD target is computed using the greedy policy with respect to the current Q values. The TD error of Q-Learning is

$$\delta_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t). \tag{6}$$

Expected Sarsa generalizes Q-Learning to arbitrary target policies. The TD error is

$$\delta_t = R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t). \tag{7}$$

The current state-action pair is updated using the expectation of all subsequent action values with respect to the action value. You can easily see that Q-Learning is just a special case of Expected Sarsa if π is the greedy policy with respect to Q:

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = \operatorname{argmax}_a Q(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Then $\sum_{a'} \pi(a'|S_{t+1})Q(S_{t+1}, a')$ is equal to $\max_{a'} Q(S_{t+1}, a')$ because all non-greedy actions will have a probability of 0 and the sum reduces to the Q value of the greedy action, which is the maximum Q value.

Of course Expected Sarsa could also be used as an on-policy algorithm if the target policy is chosen to be the same as the behaviour policy (Van Seijen et al. (2009)).

Sutton and Barto (2017) propose a new TD control algorithm called Q(σ) which unifies Sarsa and Expected Sarsa. The TD target of this new algorithm is a weighted mean of the Sarsa and Expected Sarsa TD targets, where the parameter σ controls the weighting. When $\sigma = 1$ Q(σ) is equal to Sarsa, when $\sigma = 0$ Q(σ) is equal to Expected Sarsa and when using $\sigma = 0$ and a greedy target policy Q(σ) is equal to Q-Learning. For intermediate values of $sigma$ new algorithms are obtained, which might achieve better performance (Asis et al. (2017)).

The TD error of Q(σ) is

$$\delta_t = R_{t+1} + \gamma(\sigma Q(S_{t+1}, A_{t+1}) + (1-\sigma)\sum_{a'} \pi(a'|S_{t+1})Q(S_{t+1}, a')) - Q(S_t, A_t). \tag{9}$$

3 N-step TD Methods

The TD methods presented so far are one-step methods, which use only rewards and values from the next step $t + 1$. These can be extended to look into a more distant time horizon, e.g. to incorporate data from two steps away. A two-step Sarsa TD target could then be

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2}). \tag{10}$$

Of course instead of the two-step return we could also use any n-step return defined by

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^n R_{t+n} + \gamma^{n+1} Q(S_{t+n+1}, A_{t+n+1}) \tag{11}$$

as a TD target.

N-step versions of Q-Learning and Expected Sarsa can be obtained similarly, e.g. the n-step return for Q-Learning is defined by

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^n R_{t+n} + \gamma^{n+1} \max_a Q(S_{t+n+1}, a). \tag{12}$$

Sutton and Barto (2017) and Asis et al. (2017) also propose an n-step version of Q(σ), where the n-step return is defined by

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n-1, T-1)} \delta_k^\sigma \prod_{i=t+1}^{k} \gamma \left[(1-\sigma_i)\pi(A_i|S_i) + \sigma_i\right] \tag{13}$$

where δ_t^σ is the TD error of Q(σ):

$$\delta_t^\sigma = R_{t+1} + \gamma \left(\sigma Q(S_{t+1}, A_{t+1}) + (1-\sigma) \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') \right) - Q(S_t, A_t). \tag{14}$$

N-step returns can also be averaged to the so called λ-return defined by

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t. \tag{15}$$

N-step algorithms have the advantage over one-step algorithms that they make use of data of the following n steps. A disadvantage is that n-step returns can only be computed after n steps, so a value can only be adjusted after the next n rewards have been observed. When using G_t^λ as a TD target the algorithm can only update the value function after the end of the episode, because the last n-step return can only then be computed, and cannot be applied to continuing problems at all. When all updates to a value function are made at the end of an episode, we call this an offline algorithm, in contrast to online algorithms, which update the value function immediately after every step. Examples for online algorithms are the one-step Sarsa, Expected Sarsa and Q(σ) algorithms presented so far.

Updating the value of a state towards rewards obtained at later time steps is called the forward-view. Often there exists a computationally advantageous backward-view, which assigns the current error backwards to previously visited states using eligibility traces. Using this backward view we can obtain online multi-step algorithms for the one-step algorithms presented so far.

4 Online Multi-step TD Algorithms using Eligibility Traces

An eligibility trace is a scalar numeric value for each state-action pair. Whenever a state-action pair is visited its eligibility is increased, if not, the eligibility fades away over time. State-action pairs visited often will have a higher eligibility than those visited less frequently and state-action pairs visited recently will have a higher eligibility than those visited long time ago.

Different eligibility traces are proposed in the literature. Two kinds are especially commonly used: the accumulating trace and the replacing trace (Singh and Sutton (1996)).

4.1 Combining Replacing and Accumulating Eligibility Traces

The accumulating trace uses an update of the form

$$E_{t+1}(s,a) = \begin{cases} \gamma\lambda E_t(s,a) + 1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a), & \text{otherwise.} \end{cases} \tag{16}$$

Whenever taking action A_t in state S_t the eligibility of this pair is increased by 1 and for all states and actions decreased by a factor $\gamma\lambda$, where λ controls the trade-off with a one-step TD method at one extreme and Monte Carlo at the other extreme. Higher λ values assign the current TD error back over longer time intervals than lower λ values.

The replacing trace is similar to the accumulating trace but the eligibility has an upper bound of 1. It is defined by

$$E_{t+1}(s,a) = \begin{cases} 1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a), & \text{otherwise.} \end{cases} \tag{17}$$

The replacing trace has the advantage that the eligibility cannot be greater than 1, therefore it can be more stable than the accumulating trace when episodes are very long and states are revisited frequently. Then the eligibility using accumulating traces would become very large which can lead to instability (Singh and Sutton (1996)).

The update rule using eligibility traces is then modified to

$$Q(s,a) \leftarrow Q(s,a) + \alpha \, \delta_t \, E_t(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{18}$$

The corresponding algorithm using the one-step Sarsa TD error and an update using eligibility traces is called Sarsa(λ). Though it looks like a one-step algorithm, it is in fact a multi-step algorithm, because the current TD error is assigned back to all previously visited states and actions weighted by their eligibility.

A natural idea is to combine replacing and accumulating trace into a weighted average.

Then the eligibility trace update is

$$E_{t+1}(s,a) = \begin{cases} (1-\beta)(\gamma\lambda E_t(s,a)+1)+\beta, & \text{if } A_t = a, S_t = s \\ (1-\beta)\gamma\lambda E_t(s,a) + \beta\gamma\lambda E_t(s,a), & \text{otherwise.} \end{cases} \tag{19}$$

This can be simplified to

$$E_{t+1}(s,a) = \begin{cases} (1-\beta)(\gamma\lambda E_t(s,a))+1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a), & \text{otherwise.} \end{cases} \tag{20}$$

The factor β now controls the weighting, with $\beta = 0$ being the usual accumulating trace and $\beta = 1$ being the replacing trace. In the tabular case this is equal to the so called Dutch trace proposed by van Seijen et al. (2015), where β is set to equal the learning rate α.

4.2 Eligibility Traces for off-policy Learning

How can we extend off-policy TD algorithms to online algorithms using eligibility traces, so we can obtain a multi-step online version of Q-Learning and Expected Sarsa? Different ideas have been proposed: Naive Q(λ) uses the same eligibility updates as described before, ignoring that learning is off-policy. Watkin's Q(λ) uses the same updates as long as the greedy action is chosen by the behaviour policy, but sets the Q values to 0, whenever a non-greedy action is chosen assigning credit only to state-action pairs we would actually have visited if following the target policy π and not the behaviour policy μ. More generally the eligibility is weighted by the target policy's probability of the next action. The update rule is then

$$E_{t+1}(s,a) = \begin{cases} (1-\beta)(\gamma\lambda E_t(s,a)\pi(A_{t+1}|S_{t+1}))+1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a)\pi(A_{t+1}|S_{t+1}), & \text{otherwise.} \end{cases} \tag{21}$$

Whenever an action occurs, which is unlikely in the target policy, the eligibility of all previous states is decreased sharply. If the target policy is the greedy policy, the eligibility will be set to 0 for the complete history.

The final step is to propose a new kind of eligibility trace to extend the Q(σ) algorithm to an online multi-step algorithm, which we will call Q($\sigma\lambda$).

4.3 Q($\sigma\lambda$) Algorithm

Recall that the one-step target of Q(σ) is a weighted average between the on-policy Sarsa and off-policy Expected Sarsa targets weighted by the factor σ:

$$\delta_t = R_{t+1} + \gamma(\sigma Q(S_{t+1}, A_{t+1}) + (1-\sigma)\sum_{a'} \pi(a'|S_{t+1})Q(S_{t+1}, a')) - Q(S_t, A_t) \tag{22}$$

The natural idea is then to weight the eligibility accordingly with the same factor σ. The eligibility will then be a weighted average between the on-policy eligibility used in Sarsa(λ) and the off-policy eligibility used in $Q(\lambda)$. Then the eligibility trace is updated at each step by

$$E_{t+1}(s,a) = \begin{cases} (1-\beta)(\gamma\lambda E_t(s,a)(\sigma + (1-\sigma)\pi(A_{t+1}|S_{t+1}))) + 1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a)(\sigma + (1-\sigma)\pi(A_{t+1}|S_{t+1})), & \text{otherwise.} \end{cases} \tag{23}$$

When $\sigma = 0$ the one-step target of Q(σ) is equal to the Sarsa one-step target and therefore the eligibility update reduces to the standard on-policy eligibility trace update (accumulate or replace trace according to β). When $\sigma = 1$ the one-step target of Q(σ) is equal to the Expected Sarsa target and accordingly the eligibility is weighted by the target policy's probability of the current action. For intermediate values of σ the eligibility is weighted in the same way as the TD target.

Pseudocode for tabular episodic Q($\sigma\lambda$) is given in Algorithm 1. This can be easily extended to function approximation using one eligibility per weight of the function approximator and to continuing tasks.

Of course σ or λ can also be varied over time, e.g. depending on the number of episodes, visited states or rewards.

Algorithm 1 Q($\sigma\lambda$)

```
Initialize $Q(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$
Repeat for each episode:
    $E(s,a) \leftarrow 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$
    Initialize $S_0 \neq$ terminal
    Choose $A_0$, e.g. $\epsilon$-greedy from $Q(S_0, .)$
    Loop for each step of episode
        Take action $A_t$, observe reward $R_{t+1}$ and next state $S_{t+1}$
        Choose next action $A_{t+1}$, e.g. $\epsilon$-greedy from $Q(S_{t+1}, .)$
        $\delta = R_{t+1} + \gamma\,(\sigma\,Q(S_{t+1}, A_{t+1}) + (1-\sigma)\ \sum_{a'} \pi(a'|S_{t+1})\ Q(S_{t+1}, a') - Q(S_t, A_t)$
        $E(S_t, A_t) \leftarrow (1-\beta)\ E(S_t, A_t) + 1$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,\delta\,E(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$
        $E(s,a) \leftarrow \gamma\lambda E(s,a)(\sigma + (1-\sigma)\pi(A_{t+1}|S_{t+1})) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$
        $A_t \leftarrow A_{t+1}$, $S_t \leftarrow S_{t+1}$
        If $S_t$ is terminal: Break
```

5 Experiments

In this section the performance of the newly proposed Q($\sigma\lambda$) algorithm will be tested on different tasks compared with the classical TD control algorithms Sarsa(λ), Q(λ) and Expected Sarsa(λ) as well as the n-step Q(σ) algorithm. First we will explore learning behaviour on a simple gridworld navigation task, then on a problem with a continuous state space using function approximation.

5.1 Windy Gridworld

The windy gridworld is a simple navigation task from Sutton and Barto (1998). The goal is to get from a start state to a goal state using the standard actions, moving left, right, up or down. In each column of the grid the agent is moved a certain number of columns upward. When an action would take the agent outside the grid, the agent is placed in the nearest cell inside the grid. The task is treated as an undiscounted episodic task with a reward of -1 for each transition. Figure **??** visualizes the gridworld.

5.2 Mountain Car

The Mountain Car task as described by Sutton and Barto (1998) is a classic control task with a continuous state space. The goal is to move an under-powered car as fast as possible from the valley up a mountain. The state space is characterized by the position and velocity of the car and there are three actions ("push left", "do nothing", "push right"). Figure **??** visualizes the environment. For each step the agent receives a reward of -1 until the goal state is reached and we treat this as an undiscounted task ($\gamma = 1$). Because the state space is continuous function approximation needs to be used. Therefore the action value function was approximated using tile coding, i.e. using Sutton's tile coding software version 3 with 8 tilings and `max_size` = 4096.

6 Conclusions

This paper has presented Q($\sigma\lambda$), a new online multi-step version of the Q(σ) algorithm, which combines Sarsa and Q-Learning. It uses an eligibility update to allow online learning as in Sarsa(λ) or Watkin's Q(λ). The eligibility trace proposed is a weighted average between the classical on-policy trace used for Sarsa and the off-policy trace used for Q-Learning and other off-policy algorithms. Empirical results suggest that the new Q($\sigma\lambda$) algorithm outperforms classical Sarsa(λ) and Q(λ) as well as the n-step Q(σ) on different tasks. Dynamically varying the value of σ allows to combine the good initial performance of Sarsa with the good asymptotic performance of Expected Sarsa to obtain new state of the art results.

Future research might focus on performance of Q($\sigma\lambda$) when used with non-linear function approximation and different schemes to update σ or λ over time.

References

Asis, K. D., Hernandez-Garcia, J. F., Holland, G. Z. and Sutton, R. S. (2017). Multi-step reinforcement learning: A unifying algorithm, *CoRR* **abs/1703.01327**.
URL: *http://arxiv.org/abs/1703.01327*

Rummery, G. A. and Niranjan, M. (1994). On-line q-learning using connectionist systems, *Technical report*.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces, *Machine Learning* **22**(1): 123–158.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*, 1st edn, MIT Press, Cambridge, MA, USA.

Sutton, R. S. and Barto, A. G. (2017). Reinforcement learning : An introduction. Accessed: 2017-08-01.

van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C. and Sutton, R. S. (2015). True online temporal-difference learning, *CoRR* **abs/1512.04087**.
URL: *http://arxiv.org/abs/1512.04087*

Van Seijen, H., Van Hasselt, H., Whiteson, S. and Wiering, M. (2009). A theoretical and empirical analysis of expected sarsa, *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, IEEE, pp. 177–184.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*, PhD thesis, King's College, Cambridge.