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. Sutton and Barto (2017) found a way to unify both of them in the Q(σ) algorithm. This paper extends the Q(σ) algorithm to an online multi-step algorithm using a new kind of eligibility trace. The new eligibility trace is a weighted average between the two different kinds of eligibility traces usually used for Sarsa(λ) and Q(λ). Experiments suggest that the new algorithm outperforms the classical TD control methods Sarsa and Q-Learning as well as the one-step Q(σ) algorithm.

1 Introduction

Reinforcement Learning is a filed of machine learning addressing the problem of sequential decision making. It is usually formulated as an interaction of an agent, which can take actions, and an environment. Agent and environment interact over a number of discrete time steps t. At each time point the agent chooses an action A_t from the action space $\mathcal{A}$ based on information about 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}. The reward is a scalar numeric feedback signal which quantifies how good an action is.

The agent is thereby following a policy π which is the behavior function and maps 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. A MDP models state transitions using a state transition matrix $P^a_{ss'} = P(S_{t+1} = s'|S_t = s, A_t = a)$. Many problems can be formulated as episodes, where the interaction leads to a terminal state, which terminates the episode. There as continuing environments go on forever, a terminal state does not exist.

Usually the goal is to maximize the sum of rewards over the lifetime of the agent. The return G_t is defined by

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

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 and action. 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 estimating q_π. When representing the action values as a table we speak of tabular reinforcement learning, else we speak of approximate reinforcement learning if the action value is approximated by a function approximator. For sake of simplicity the following analysis is done for tabular reinforcement learning but can be easily extended to function approximation. Temporal-difference learning is a class of model-free methods which estimate q_π from sample transitions by a one-step look-ahead.

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

Sarsa is a temporal-difference learning algorithm which updates the Q values using updates of the form

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right). \tag{4}$$

The term $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ is called the TD target. It consists of the reward plus the value of the next state and next action. Updating values using value estimates of successor states is called bootstrapping. The term $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$ is then called the TD error. The update rule then corrects the current Q values in the direction of the error between the current estimate and the TD target, where α is the learning rate, which controls how much the values are corrected.

Sarsa is an on-policy method, the TD target uses $Q(S_{t+1}, A_{t+1})$, 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 - can be different from the target policy, 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). States and actions are sampled using a behaviour policy μ, e.g. an ϵ-greedy policy, but the TD target is computed using the greedy policy with respect to the current Q values. The update rule is therefore

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\right). \tag{5}$$

Expected Sarsa generalizes Q-Learning to arbitrary target policies. The update rule is

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t)\right). \tag{6}$$

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. Q-Learning's target policy π is the greedy policy with respect to Q:

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = \text{argmax } Q(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

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 an on-policy algorithm if the target policy is chosen to be the same as the behaviour policy.

In the following we will have a look of how to unify Expected Sarsa with Q-Learning as a special case and Sarsa.

Sutton and Barto (2017) propose a new TD control algorithm called Q(σ). The TD target of this new algorithm is a weighted mean of the Sarsa and Expected Sarsa TD targets. 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. For intermediate values of $sigma$ new algorithms are obtained.

The update rule for Q(σ) is

$$\begin{aligned} Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma(\sigma Q(S_{t+1}, A_{t+1}) + \\ (1-\sigma) \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a')) - Q(S_t, A_t)). \end{aligned} \tag{8}$$

3 Multi-step Methods and Eligibility Traces

The TD methods presented so far are one-step methods, which use only information 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 update could then be

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2}) - Q(S_t, A_t) \right). \tag{9}$$

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

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

as a TD target. The n-step versions of Q-Learning and Expected Sarsa can be obtained similarly, e.g. for Q-Learning

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \Big(R_{t+1} + \gamma R_{t+2} + \gamma^2 \max_{a''} Q(S_{t+2}, a'') - Q(S_t, A_t) \Big). \tag{11}$$

Often it is also advantageous to average different n-step returns to the so called λ-return defined by

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t. \tag{12}$$

For $\lambda = 0$ the λ-return is equal to the one-step Sarsa target, else it is a weighted average of all n-step returns.

When using a 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 be then computed, and cannot be applied to continuing problems at all. When all updates to a value function are made after 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 e.g. Sarsa and Q-Learning.

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. This combines a frequency and recency point of view. 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).

The accumulating trace uses an update of the form

$$E_{t+1}(s,a) = \begin{cases} \gamma\lambda E_t(s,a) + 1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a), & \text{otherwise.} \end{cases} \tag{13}$$

Whenever taking action A in state S 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 weighting all n-step returns equally at the other extreme, which is known as a Monte Carlo update.

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

$$E_{t+1}(s,a) = \begin{cases} 1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a), & \text{otherwise.} \end{cases} \tag{14}$$

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

$$Q(s,a) = Q(s,a) + \alpha\delta E_t(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{15}$$

with δ denoting the TD error. 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. Note that the Sarsa(λ) algorithm only approximates the offline λ-return algorithm.

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

Then the eligibility trace update is

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

This can be simplified to

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

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).

How can eligibility traces used for off-policy TD algorithms, so we can obtain a λ-version of Q-Learning and Expected Sarsa? One idea is to use the same Equations as above, ignoring that learning is off-policy, this is what is called Naive

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. ϵ-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. ϵ-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

Q(λ) in combination with Q-Learning. Another Q(λ) algorithm called 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. More generally the eligibility is weighted by the target policy's probability of the next action. The update rule is then

$$E_{t+1}(s,a) = \begin{cases} (1-\beta)(\gamma\lambda E_t(s,a)\pi(A_{t+1}|S_{t+1})) + 1, & \text{if } A_t = a, S_t = s \\ \gamma\lambda E_t(s,a)\pi(A_{t+1}|S_{t+1}), & \text{otherwise.} \end{cases} \tag{18}$$

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.

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:

$$\delta_t = R_{t+1} + \gamma(\sigma Q(S_{t+1}, A_{t+1}) + (1-\sigma)\sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a')) - Q(S_t, A_t) \tag{19}$$

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

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

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.

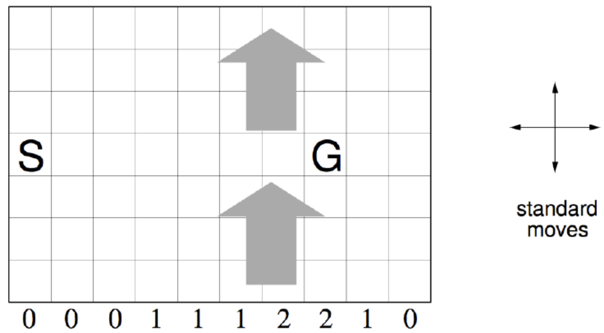

Figure 1: The windy gridworld task. The goal is to move from the start state S to the goal state G while facing an upward wind in the middle of the grid, which is denoted in the numbers below the grid. Described by Sutton and Barto (1998).

Asis et al. (2017) present an n-step version of the Q(σ) algorithm. As an n-step algorithm it has the usual disadvantage that it needs to store TD errors at each step and can only update the value function after n steps. The newly proposed algorithm, which we call Q($\sigma\lambda$) on the other side can learn online and does not need to store all TD errors. Of course σ can also be varied over time, depending on states and actions.

Pseudocode for tabular episodic Q($\sigma\lambda$) is given in Algorithm 1. Extending this to function approximation is simple, just keep one eligibility per weight of the function approximator.

4 Experiments

In this section the performance of the newly proposed Q($\sigma\lambda$) algorithm is tested on the windy gridworld task. 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 1 visualizes the gridworld.

Experiments were conducted using an ϵ-greedy behavior policy with $\epsilon = 0.1$ for different values of σ, λ and β. The learning rate α was set to 0.1. The target policy of the Expected Sarsa target was chosen to be the greedy policy with respect to Q, which is Q-Learning.

Figure 2 shows that learning occurs faster if λ is higher, so the multi-step method Q($\sigma\lambda$) dominates one-step Q(σ) for different values of σ.

Also Sarsa (Q(1)) performs initially better than Q(σ) with lower σ values, e.g. Q-Learning (Q(0)). This is not surprising because the eligibility trace will be multiplied with the target policy's probability, which is 0 when a non-greedy action occurs. So for a non-greedy action the eligibility will be updated by

$$E_{t+1}(s,a) = \gamma\lambda\sigma E_t(s,a) \quad \text{if } A_{t+1} \neq \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a), \tag{21}$$

so it will be reduced by the factor σ at each step. Q-Learning in the extreme sets the eligibility trace to 0.

Figure ... shows that intermediate values of σ give better performance than the extremes Sarsa ($\sigma = 1$) and Q-Learning ($\sigma = 0$). This confirms the results in Asis et al. (2017).

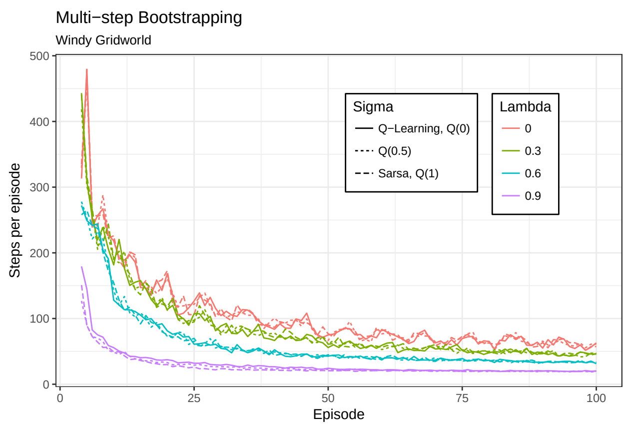

Figure 2: Windy gridworld results. Performance of Q($\sigma\lambda$) for different values of σ and λ. For higher values of λ learning is faster and Sarsa learns faster than Q-Learning. Steps per episode are averaged over 100 runs.

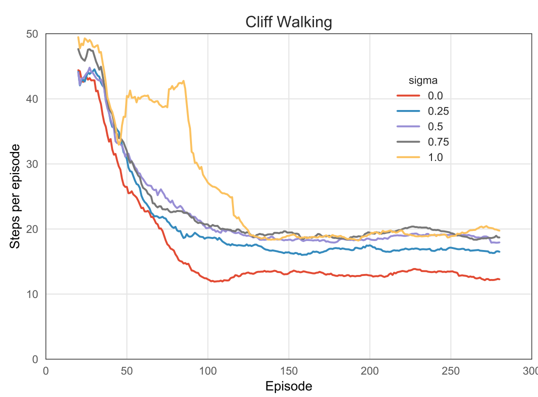

Figure 3: Cliff Walking results. Sarsa (Q(1)) learns the safe path at the top of the gridworld, while Q-Learning learns the shortest path with a risk of falling down the cliff. Interestingly intermediate values of σ are between the two extremes.

Figure ... shows the learning behaviour using replacing, accumulating traces or a mixture of both by varying the factor β.

The optimal policy in the windy gridworld task needs 15 steps to reach the goal state.

5 Conclusions

This paper has presented a new multi-step version of the Q(σ) algorithm called Q($\sigma\lambda$), which combines Sarsa and Q-Learning. It uses an eligibility update to allow online learning as in Sarsa(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. It also combines replacing and accumulating traces into one trace. Empirical results suggest that the new Q($\sigma\lambda$) algorithm outperforms classical Sarsa(λ) and Q(λ) as well as the one-step Q(σ) on different tasks.

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*

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*

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