CS238 Project Report: Safe Autonomous Driving Through a Crosswalk

Xiyuan Chen
xycshine@stanford.edu

Siyun Li
lisiyun@stanford.edu

Xi Yan
xiyan@stanford.edu

Abstract

Navigating safely and efficiently in stochastic environments remains a crucial challenge in the field of autonomous driving. In this work, we explore the problem of safe driving through a crosswalk for autonomous vehicles. We introduce the Crosswalk Driving problem where the vehicle must drive to a destination while obeying reliable safety constraints. We formulate our problem as a Markov Decision Process (MDP) and propose two environments: Simple Crosswalk and Jaywalker Crosswalk. We solve our MDP using several model-free reinforcement learning approaches and provide analysis on our results. Our code for the simulation environment and the RL algorithms used is available at `https://github.com/yanxi0830/CS238CrosswalkDriving`.

1 Introduction

In recent years, we have seen rapid advancements in the field of autonomous driving. Yet, safe driving in all driving scenarios remains a difficult and open problem. Developing an autonomous driving (AD) system involves taking many aspects into consideration, ranging from vehicle design to control, perception, planning, coordination, and human interaction [1]. One of the main challenges for autonomous vehicles is the frequent interactions between the ego vehicle, pedestrians, and other vehicles whose behavior inherently involve some degree of randomness.

In our work, we consider the problem of safe driving through a crosswalk for an autonomous vehicle. Specifically, the objective of the ego vehicle is to learn a sequence of optimal operation decisions (i.e. acceleration) which allows it to reach the destination with a target speed as soon as possible and avoid collision with any unforeseen pedestrians along its way. Our work can be broken down into two parts. First, we formulate our problem as a Markov Decision Process (MDP) to complement the decision-making process for autonomous vehicles. Second, we consider several model-free reinforcement learning approaches including Q-Learning, Q(λ), Sarsa, and Sarsa(λ) to solve our MDP model.

2 Related Work

There is a vast number of works in the literature on decision making for autonomous driving. [2] presents a comprehensive review into the recent developments and research on collision avoidance system and adaptive cruise control (ACC) for autonomous vehicles. Most recently, [3], [4] explores the effectiveness of deep reinforcement learning algorithms on a taxonomy of autonomous driving tasks and intersection problems. Related to our work, Wei et al [5] introduce a point-based MDP for single-lane autonomous driving control under uncertainties. They incorporate uncertainties including vehicle's behavior and sensor inputs with the objective to better avoid unsafe behaviors. Bai et al [6] tackle a similar problem of driving near pedestrians and incorporate a POMDP online planning approach. In this work, we investigate MDP formulation on the autonomous driving scenario involving a single-lane street with a crosswalk where the ego vehicle aims to safely interact

with unforeseen pedestrians. In contrast to [6], we approach our problem using several model-free reinforcement learning algorithms.

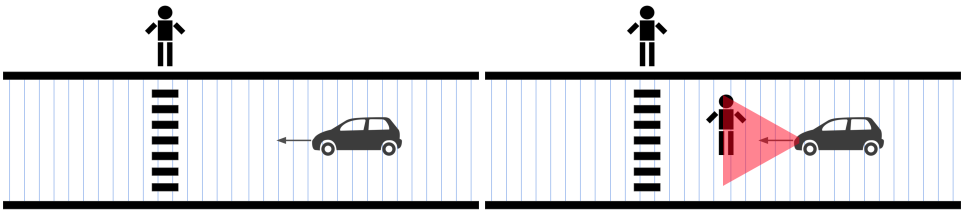
3 Problem Formulation

In this section, we introduce the Crosswalk Driving problem. We assume a one-dimensional layout, where the ego vehicle is driving along a straight one-lane street, with a crosswalk at a preset position unknown to the agent. The goal for the vehicle is to reach the destination located past the crosswalk at the end of the street with a certain speed as soon as possible, while obeying the traffic law and avoiding collision with any pedestrians. We consider two scenarios: Simple Crosswalk and Jaywalker Crosswalk.

Simple Crosswalk. In the Simple Crosswalk scenario, we assume that there is a pedestrian standing still at the crosswalk, which is hand-coded to a fixed position when initializing the simulation environment. In addition to the objective of reaching the destination at the end of the road with the targeted speed as soon as possible, the vehicle must yield to the pedestrian(i.e. slow down to satisfy the preset speed limit) when it approaches the crosswalk.

Jaywalker Crosswalk. In the Jaywalker Crosswalk scenario, we extend the Simple Crosswalk case and incorporate the uncertainty from potential jaywalking pedestrians who may appear in front of the ego vehicle at any location on its way. The vehicle is equipped with a sensor which gives 3 different signals based on whether it detects a jaywalking pedestrian near the ego vehicle: Safe, Warning, Danger. The vehicle is expected to decrease its speed to a preset safe speed whenever the sensor outputs a Warning signal, and should stop whenever there is a Danger signal.

We formulate our problem as a Markov Decision Process (MDP) with $\{\mathcal{S}, \mathcal{A}, T, R, \gamma\}$. At each step t, the agent observes a state s_t, and performs an action a_t according to a policy π, leading to a new state s_{t+1} given by the transition function $T(s'|s, a)$, and a corresponding reward r_t given by the reward function $R(s, a)$. The rest of Section 3 describes our setup for $\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}$ and γ in detail.



(a) (b)

Figure 1: Visualization of our problem setting. **(a)** Simple Crosswalk. **(b)** Jaywalker Crosswalk. The sensor generates a Danger signal on detecting a pedestrian.

3.1 State Space

For the Simple Crosswalk scenario, the state space is composed of two distinct features: the vehicle's position and velocity. As the vehicle drives along a straight street with a single lane, we use scalar values x_t to encode its position and v_t for its velocity at step t. For the Jaywalker Crosswalk scenario, we introduce a third feature to the state space: the jaywalk signal given by the sensor l_t.

All three features are discretized in this project. For the Simple Crosswalk scenario, we also investigate two state spaces with different scales to compare the learning speed of the various methods described in Section 4. The state spaces used in this project are summarized in Table 1.

In each scenario, the initial state is at position 0 with velocity 3. The terminal state is reached when the vehicle has passed the crosswalk and reached the destination at the end of the road.

Scenario	# Positions	# Velocity	# Sensor Signal	State Space Size $\|\mathcal{S}\|$
Simple Crosswalk (small)	21	5	N/A	**105**
Simple Crosswalk (large)	101	21	N/A	**2121**
Jaywalker Crosswalk	21	5	3	**315**

Table 1: State Spaces in Different Driving Scenarios

3.2 Action Space

The only action we consider in this project is the throttling/braking control. At each step t, the agent can perform an action a_t from a discrete action space. Action 0 means that the vehicle will maintain its current speed. Actions 1 and 2 represent small and large accelerations while actions 3 and 4 represent small and large decelerations. To prevent the agent from reaching negative speed after deceleration, we clip the minimum speed to 0. We also limit the maximum speed to some fixed value to avoid dangerously high speed.

3.3 Transition Function

The transition model $T(s_{t+1}|s_t, a_t)$ for the ego vehicle is deterministic. For action a_t, we map it to different values of accelerations Δv_t shown in Table 2.

For each step, we update the vehicle's velocity and position using the following rules:

$$v_{t+1} = v_t + \Delta v_t \tag{1}$$

$$x_{t+1} = x_t + v_{t+1} \tag{2}$$

Action a_t	0	1	2	3	4
Acceleration Δv_t	0	1	2	-1	-2

Table 2: Mapping Between Action and Acceleration

3.4 Reward Function

To achieve our safe driving goal, we consider four components when designing our reward function $R(s, a)$, each with a specific objective.

(1) $R_{\text{crosswalk}}$ encodes the objective that the vehicle should slow down when approaching the crosswalk where a pedestrian is standing still. If the vehicle exceeds the speed limit $v_{max\_crosswalk}$ when it passes the crosswalk, we add a penalty of -40.

(2) R_{goal} is the termination reward the vehicle receives when it reaches the destination x_{goal}. If the vehicle reaches the goal position with the targeted velocity v_{goal}, it is given a reward of $+40$. Otherwise, it receives a penalty of -40.

(3) R_{step} is the step reward that encourages the vehicle to minimize the number of steps required for reaching the destination. For every state along the vehicle's trajectory (except the terminal state), the vehicle receives a penalty of -3.

The reward function for the Simple Crosswalk scenario is given by Eq. 3.

$$R_{\text{simple}}(s, a) = R_{\text{crosswalk}} + R_{\text{goal}} + R_{\text{step}} \tag{3}$$

(4) R_{sensor} is the additional sensor reward for the Jaywalker Crosswalk scenario to incorporate sensor signals. It enforces the constraint that vehicle should slow down whenever the sensor generates a Warning signal and stop on observing a Danger signal. A penalty of -20 is added if the speed of vehicle exceeds $v_{max\_warning}$ with a Warning signal, and a penalty of -40 is added if the vehicle is still moving ($v_t > 0$) with a Danger signal. In addition, a penalty of -5 is added if the vehicle accelerates with a Warning or Danger signal.

The reward function for Jaywalker Crosswalk scenario is given by Eq. 4.

$$R_{\text{jaywalker}}(s,a) = R_{\text{crosswalk}} + R_{\text{goal}} + R_{\text{step}} + R_{\text{sensor}} \tag{4}$$

3.5 The Discount Factor

The discount factor γ for an MDP has value between 0 and 1. Lower value of the discount factor encourages the agent to prioritize the immediate rewards than the future ones. In this project, we set γ to 0.95 in all reinforcement learning algorithms.

4 Methods & Implementation Details

This section summarizes the different model-free RL algorithms we used to approach the problem.

4.1 Off-Policy: Q-Learning

Q-learning is a model-free reinforcement learning algorithm which incrementally estimates the action value function $Q(s,a)$. We initialize the action value function to 0 for all state-action pairs. Then, given the current state s_t, action a_t and next state s_{t+1}, Q-learning updates the corresponding action value function as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)] \tag{5}$$

In order to make sure the action value function converges, we allow the agent to explore by applying the ϵ-greedy strategy in the training process. Specifically, the strategy chooses a random action with probability ϵ. Otherwise, it chooses the greedy action which achieves the maximum action value function for the current state. Since most states are not explored at the beginning, there is much higher uncertainty to start with. Therefore, we set a relatively high ϵ at initialization and decay it over time. In this way, the agent is encouraged to explore the environment early in the learning process and take full advantage of what it has learnt as the policy converges.

Finally, the policy π generated by Q-learning can be expressed as follows using the updated action value function Q:

$$\pi(s) = \operatorname*{argmax}_a Q(s,a) \tag{6}$$

Q-learning has four hyper-parameters during its learning process.

- **The learning rate** α. Its value is between 0 and 1. Higher values of α generates larger update steps for $Q(s,a)$, which leads to faster learning (i.e. time to converge) but could potentially lead to sub-optimal policy.
- **The training episode** eps. The number of rollouts performed in the training process.
- **The horizon** h. The maximum number of steps for each rollout. This should increase as the state space grows.

Additionally, the ϵ-greedy strategy involves two hyperparameters.

- **The probability to explore** ϵ. Higher value of ϵ encourages the agent to explore rather than exploit.
- **The exploration decaying factor** β. Value is set between 0 and 1. Each time the agent takes a random action, ϵ is decayed by a factor of β such that the probability for the agent to take a random action decreases over time.

The Q-learning and ϵ-greedy strategy is described in Algorithm 2 and Algorithm 1 respectively.

4.2 On-Policy: Sarsa

Sarsa is an on-policy reinforcement learning method. It is different from Q-learning in that the maximum reward for the next state s_{t+1} is not necessarily used for updating the Q-value. Instead,

it chooses a new action a_{t+1} following the same policy that determined the current action a_t. The update to $Q(s, a)$ is computed as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{7}$$

This update is done after taking action a_t from a state s_t, transiting to a new state s_{t+1} and taking a new action a_{t+1}. As each update requires knowledge of $s_t, a_t, r_t, s_{t+1}, a_{t+1}$, this gives the algorithm the name Sarsa.

Same as Q-Learning, Sarsa has multiple hyper-parameters during its learning process, namely **the learning rate** α, **the training episodes** eps and **the horizon** h. In order to guarantee convergence of the action-value function, we also adopt ϵ-greedy as did for Q-Learning in Section 4.1. The final learned policy is extracted by taking the actions with maximum action value according to Eq. 6. The Sarsa algorithm is described in Algorithm 3.

4.3 Eligibility Trace: Q(λ) and Sarsa(λ)

Eligibility traces facilitate the learning process of Q-learning and Sarsa by propagating the reward backward to all state-action pairs that lead to the source of the reward. It maintains a visit count $N(s, a)$ with counts decaying exponentially using a factor of λ. The exponentially decaying visit counts are used in the propagation such that the states closer to the reward are assigned larger action values. The versions of Q(λ) and Sarsa(λ) implemented for this problem are described in Algorithms 4 and 5.

For each step over the horizon, we increment $N(s_t, a_t)$ by 1 if action a_t is taken at state s_t. We then compute the temporal difference update.

For Q(λ), the temporal difference update is computed as:

$$\delta \leftarrow r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \tag{8}$$

For Sarsa(λ), we have

$$\delta \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{9}$$

We then do a full update to the action value function $Q(s, a)$ for each $s \in S$ and $a \in A$ according to the corresponding visit counts.

$$Q(s, a) \leftarrow Q(s, a) + \alpha\delta N(s, a) \tag{10}$$

Finally, the visit counts are decayed using both the discount factor γ and the decay factor λ.

$$N(s, a) \leftarrow \gamma\lambda N(s, a) \tag{11}$$

Similar to Q-learning and Sarsa, the learnt policy is extracted from the Q matrix using Eq. 6.

Apart from hyper-parameters used in Q-learning and Sarsa, Q(λ) and Sarsa(λ) each have an extra hyper-parameter.

- **The eligibility trace decay parameter** λ. The value of λ is set between 0 and 1. Larger λ enables the reward to be propagated to states further away. Q-learning and Sarsa can be considered special cases where λ is equal to 0.

Hyperparameter settings used in our experiments can be found in our code repo at `https://github.com/yanxi0830/CS238CrosswalkDriving`.

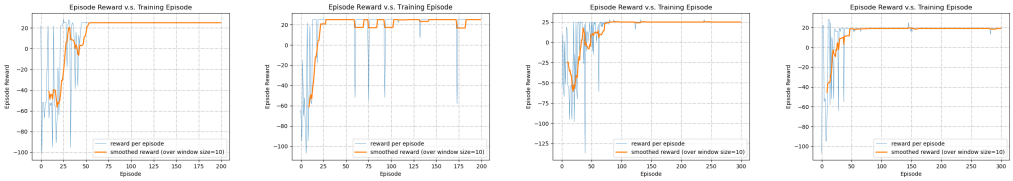
5 Results

We plot the total undiscounted reward per episode over training episodes for the algorithms in Section 4. Figures 2, 3, and 4 show the results for each of the three scenarios described in Section 3.

All four model-free algorithms converge in all three scenarios. We also observe the common trend that adding eligibility traces allows both Q-learning and Sarsa to converge faster. This is especially evident in Simple Crosswalk (large), where Q(λ) and Sarsa(λ) converges in significantly fewer training episodes compared to Q-Learning and Sarsa, respectively. This is expected as the eligibility

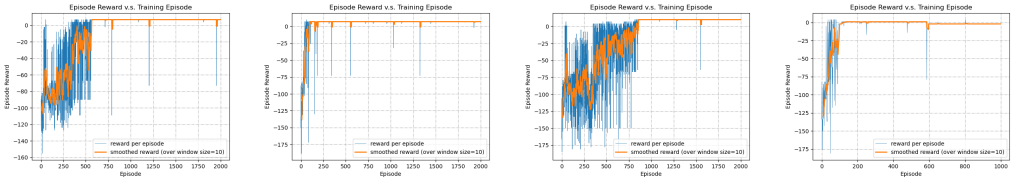
traces help propagate reward backward to the visited states and thus speed up learning. While R_{step} and R_{sensor} are rather distributed in our environment, the eligibility traces enable the vehicle to learn much faster about the sparser rewards including R_{goal} and $R_{crosswalk}$, which has pronounced impacts in the scenario with large state space. However, adding eligibility traces slows down the update procedure in each training episode and also decreases the reward value at convergence in our experiments. Moreover, we notice that Q-learning seems to converge faster than Sarsa, and so does Q(λ) compared to Sarsa(λ).

We also investigate how adding the sensor signal feature to enlarge the state space affects the learning process. While Simple Crosswalk (small) and Jaywalker Crosswalk shares the same number of discretized position and velocity values, the additional feature of sensor signal makes all algorithms take more episode to converge on the Jaywalker Crosswalk environment compared to Simple Crosswalk (small). This alludes to the fact that tabular-based RL methods suffer from the curse of dimension. In a real-world driving scenario where the observation state space is of much higher dimension, the learning process of the tabular-based methods we considered could take exponentially longer to converge. Further, these methods are not applicable off-the shelf with continuous state spaces. As a future work, we would like to explore the use of deep-learning based RL methods such as deep Q-networks (DQN) to scale up the learning process.



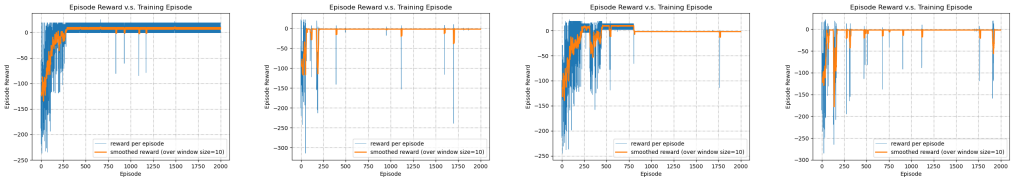
(a) (b) (c) (d)

Figure 2: Plot of total episode reward over each training episode for Simple Crosswalk (small) **(a)** Q-Learning **(b)** Q(λ) **(c)** Sarsa **(d)** Sarsa(λ)



(a) (b) (c) (d)

Figure 3: Plot of total episode reward over each training episode for Simple Crosswalk (large) **(a)** Q-Learning **(b)** Q(λ) **(c)** Sarsa **(d)** Sarsa(λ)



(a) (b) (c) (d)

Figure 4: Plot of total episode reward over each training episode for Jaywalker Crosswalk **(a)** Q-Learning **(b)** Q(λ) **(c)** Sarsa **(d)** Sarsa(λ)

Lastly, we evaluate our learned policy's performance on a test simulation against the baseline policy with randomly sampled actions. We use 2 evaluation metrics: total reward and overspeeding rate. For total reward, we compute the total undiscounted reward along the trajectory of the vehicle

following the policy learned with each algorithm. Table 3 reports the average total undiscounted reward over 5 different random runs for each algorithm. For overspeeding, we consider the vehicle to be overspeeding whenever it exceeds the speed limit driving past the crosswalk $v_{\text{max\_crosswalk}}$ or exceeds $v_{\text{max\_warning}}$ with a Warning sensor signal or has nonzero velocity with a Danger sensor signal. Table 4 reports the overspeeding rate over 5 different random runs for each algorithm. Note that all algorithms significantly outperform the random policy with higher total reward and lower overspeeding rate.

Scenario	**Random**	**Q-Learning**	**Q(λ)**	**Sarsa**	**Sarsa(λ)**
Simp.Cross. (small)	-48.6	26.2	26.2	27.4	21.4
Simp.Cross. (large)	-129.8	6.4	6.4	6.4	2.8
Jaywalker.Cross.	-130.0	6.4	-2.0	7.2	-3.0

Table 3: Average total undiscounted reward over 5 random runs in the three scenarios for all algorithms.

Scenario	**Random**	**Q-Learning**	**Q(λ)**	**Sarsa**	**Sarsa(λ)**
Simp.Cross. (small)	0.8	0.0	0.0	0.0	0.0
Simp.Cross. (large)	0.8	0.0	0.0	0.0	0.0
Jaywalker.Cross.	1.0	0.0	0.0	0.2	0.2

Table 4: Overspeeding Rate (%) over 5 random runs in the three scenarios for all algorithms.

6 Conclusion

In this work, we tackle the problem of safe driving through a crosswalk for autonomous vehicles. We introduce the Simple Crosswalk and Jaywalker Crosswalk simulation environments, and formulate the problem as an MDP. Besides, we incorporate the vehicle's velocity, position and sensor signals into our state space and discretize the action space for throttling/braking control. Moreover, We encodes safety and mobility objectives into the reward function. Lastly, we solve our MDP model using various model-free RL algorithms including Q-Learning, Q(λ), Sarsa, and Sarsa(λ) and provide benchmarking results for each of the algorithms.

There are many extensions to our problem that we want to explore for future work. First, we could incorporate more complex environments and vehicle dynamics into the simulator. For example, a two-dimensional case where there are multiple lanes in the street, or a multi-agent setting involving interacting with other vehicles. We can also explore deep-learning based RL algorithms to solve scenarios with larger and continuous state and action spaces more efficiently.

7 Contribution

All team members discussed and formulated the MDP model together. Xi implemented the simulation environment, and ran experiments and analysis for all algorithms. Xiyuan implemented the Q-learning and Q(λ) algorithms and complemented the analysis. Siyun implemented the Sarsa and Sarsa(λ) algorithms and helped with the environment setup. All team members contributed to the write-up.

References

[1] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018. [Online]. Available: https://doi.org/10.1146/annurev-control-060117-105157

[2] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 4, no. 3, pp. 143–153, 2003.

[3] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," 2020.

[4] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," 2018.

[5] J. Wei, J. M. Dolan, J. M. Snider, and B. Litkouhi, "A point-based mdp for robust single-lane autonomous driving behavior under uncertainties," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2586–2592.

[6] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online pomdp planning for autonomous driving in a crowd," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 454–460.

A Algorithms

Algorithm 1 ϵ-greedy(s)

r = random()
if r < ϵ **then**
    $\epsilon \leftarrow \beta\epsilon$
    return random()
else
    return $greedy\ action(s)$
end if

Algorithm 2 Q-Learning

Initialization: $\forall s \in \mathcal{S}, a \in \mathcal{A}, Q(s,a) = 0$
for $episode = 1, 2, \ldots, eps$ **do**
    s = initial state
    for $time\ step = 1, 2, \ldots, h$ **do**
        a = ϵ-greedy(s)
        r, s', $done$ = step(a)
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
        if $done$ **then**
            break
        end if
    end for
end for
return Q

Algorithm 3 Sarsa

Initialize $Q(s,a) = 0, \forall s \in S, a \in A(s)$
for $episode = 1, 2, \ldots$ **do**
    a = ϵ-greedy(s)
    for $timestep = 1, 2, \ldots$ **do**
        Take action a, observe r, s'
        a' = ϵ-greedy(s')
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$; $a \leftarrow a'$
    end for
end for

Algorithm 4 Q(λ)

Initialization: $\forall s \in \mathcal{S}, a \in \mathcal{A}, Q(s,a) = 0$
for $episode = 1, 2, \ldots, eps$ **do**
    s = initial state
    for $time\ step = 1, 2, \ldots, h$ **do**
        a = ϵ-greedy(s)
        r, s', $done$ = step(a)
        $\delta \leftarrow r + \gamma \max_{a'} Q(s', a') - Q(s, a)$
        $N(s,a) \leftarrow N(s,a) + 1$
        for $s \in \mathcal{S}, a \in \mathcal{A}$ **do**
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta N(s,a)$
            $N(s,a) \leftarrow \gamma\lambda N(s,a)$
        end for
        $s \leftarrow s'$
        if $done$ **then**
            break
        end if
    end for
end for
return Q

Algorithm 5 Sarsa(λ)

Initialize $Q(s,a) = 0, \forall s \in S, a \in A(s)$
Initialize $N(s,a) = 0, \forall s \in S, a \in A(s)$
for $episode = 1, 2, \ldots$ **do**
    a = ϵ-greedy(s)
    for $timestep = 1, 2, \ldots$ **do**
        Take action a, observe r, s'
        a' = ϵ-greedy(s')
        $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
        $N(s,a) \leftarrow N(s,a) + 1$
        for all s, a **do**
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta N(s,a)$
            $N(s,a) \leftarrow \gamma\lambda N(s,a)$
        end for
    end for
    $s \leftarrow s'$; $a \leftarrow a'$
end for