Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)

Two Novel On-policy Reinforcement Learning Algorithms based on TD(λ)-methods

Marco A. Wiering and Hado van Hasselt
Intelligent Systems Group
Department of Information and Computing Sciences, Utrecht University
Padualaan 14, 3508TB Utrecht, The Netherlands
Tel: +31 30 2539209, Fax: +31 30 2513791, Email: {marco,hado}@cs.uu.nl

***Abstract*— This paper describes two novel on-policy reinforcement learning algorithms, named QV(λ)-learning and the actor critic learning automaton (ACLA). Both algorithms learn a state value-function using TD(λ)-methods. The difference between the algorithms is that QV-learning uses the learned value function and a form of Q-learning to learn Q-values, whereas ACLA uses the value function and a learning automaton-like update rule to update the actor. We describe several possible advantages of these methods compared to other value-function-based reinforcement learning algorithms such as Q-learning, Sarsa, and conventional Actor-Critic methods. Experiments are performed on (1) small, (2) large, (3) partially observable, and (4) dynamic maze problems with tabular and neural network value-function representations, and on the mountain car problem. The overall results show that the two novel algorithms can outperform previously known reinforcement learning algorithms.**

I. Introduction

Reinforcement learning algorithms [11], [3] are very suitable for learning to control an agent by letting it interact with an environment. Currently, there are three well-known model-free value-function-based reinforcement learning (RL) algorithms that use the discounted future reward criterium; Q-learning [12], Sarsa [6], [10], and Actor-Critic methods [1], [11], [4]. Alternatively, a number of policy search and policy gradient algorithms have been proposed, but we will not describe these here. This paper introduces two new value-function-based RL algorithms, named QV-learning and the actor critic learning automaton (ACLA). Similar to Actor-Critic methods and in contrast to Q-learning and Sarsa, QV-learning and ACLA keep track of two functions.[1] Both algorithms learn the state value-function (V-function) with temporal difference (TD) learning [9], and use this estimated state value-function to learn a policy. QV-learning learns the state-action (Q) values using a form of Q-learning, and ACLA uses a learning automaton-like update rule [5] to learn preference values of actions. The new algorithms are also enhanced by eligibility traces [9] by learning the values of the state value-function using TD(λ) methods. In the experiments, we compare QV(λ)-learning and ACLA(λ) to Q(λ)-learning, a conventional Actor-Critic method, and Sarsa(λ). We will not compare the new algorithms to model-based algorithms, since these cannot directly work with continuous input spaces and non-linear function approximators such as neural networks. Furthermore, we will also not do experiments with batch algorithms that can decrease the number of experiences at the expense of more computation time per experience, although it is straightforward to extend the new algorithms to their batch versions. In the experiments, we first use a small maze and compare the algorithms using tabular and neural network representations. Then, we use a larger maze and compare the algorithms using tabular representations. We also perform experiments with a partially observable maze and a dynamic maze. We conclude the experiments with the mountain car problem.

Outline. Section II describes previous reinforcement learning algorithms. Section III describes the new reinforcement learning algorithms. Then, Section IV describes the results of a number of experiments with tabular and neural network representations. Section V discusses the obtained results, and Section VI concludes this paper.

II. Reinforcement Learning

Reinforcement learning algorithms are able to let an agent learn from its experiences generated by its interaction with an environment. We assume an underlying Markov decision process (MDP) which does not have to be known by the agent. A finite MDP is defined as; (1) The state-space $S = \{s^1, s^2, \ldots, s^n\}$, where $s_t \in S$ denotes the state of the system at time t; (2) A set of actions available to the agent in each state $A(s)$, where $a_t \in A(s_t)$ denotes the action executed by the agent at time t; (3) A transition function $T(s, a, s')$ mapping state-action pairs s, a to a probability distribution over successor states s'; (4) A reward function $R(s, a, s')$ which denotes the average reward obtained when the agent makes a transition from state s to state s' using action a, where r_t denotes the (possibly stochastic) reward obtained at time t; (5) A discount factor $0 \leq \gamma < 1$ that values later rewards less compared to immediate rewards.

A. Value-functions and Dynamic Programming

In optimal control or reinforcement learning (RL), we are interested in computing or learning the optimal policy for mapping states to actions. The optimal policy is defined as the policy that receives the highest possible cumulative discounted

[1]Note that Q-learning and Sarsa only learn state-action values, the value of a state can be derived from the different state-action values of actions applicable in that state.

rewards in its future from all states. In order to learn the optimal policy, value-function-based RL [11] estimates value-functions using past experiences of the agent. The value of a state $V^\pi(s)$ is the expected cumulative discounted future reward when the agent starts in state s and follows policy π:

$$V^\pi(s) = E(\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi)$$

An optimal policy π^* is a policy that has the largest state-value in all states: $\forall \pi \quad \forall s \quad V^{\pi^*}(s) \geq V^\pi(s)$. In many cases reinforcement learning algorithms used for learning to control an agent also use a Q-function for evaluating state-action pairs. Here $Q^\pi(s,a)$ is defined as the expected cumulative discounted future reward if the agent is in state s, executes action a, and follows policy π afterwards:

$$Q^\pi(s,a) = E(\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi)$$

If the optimal Q-function Q^* is known, the agent can select optimal actions by selecting the action with the largest value in a state: $\pi^*(s) = \arg\max_a Q^*(s,a)$. Furthermore the optimal value of a state corresponds to the highest action value in that state according to the optimal Q-function: $V^*(s) = \max_a Q^*(s,a)$. There exists a recursive equation known as the Bellman optimality equation [2] that relates a state-action value of the optimal value-function to other optimal state-values that can be reached from that state using a single transition:

$$Q^*(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V^*(s'))$$

This equation has led to several dynamic programming (DP) methods for solving known MDPs [2]. One of the most used DP algorithms is value iteration that uses the Bellman equation as an update:

$$Q^{k+1}(s,a) := \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V^k(s'))$$

Where $V^k(s) = \max_a Q^k(s,a)$. In each step the Q-function looks ahead one step using this recursive update rule. It can be shown that $\lim_{k\to\infty} Q^k = Q^*$, when starting from an arbitrary Q^0 containing only finite values.

B. Reinforcement Learning Algorithms

Although dynamic programming algorithms can be efficiently used for computing optimal solutions for particular MDPs, they have problems for more practical applicability; (1) The MDP should be known a-priori; (2) For large state-spaces the computational time would become very large; (3) They cannot be directly used for continuous state-action spaces. Reinforcement learning algorithms can cope with these problems: the MDP does not need to be known a-priori, all that is required is that the agent can interact with an environment. Furthermore, for large or continuous state-spaces, RL algorithms can be combined with function approximators for learning the value-functions. Then, the agent does not have to visit all states, but can generalize from experiences and concentrate on parts of the state-space where learned policies lead into.

Q-learning. A famous algorithm for learning a Q-function is Q-learning [12], [13]. Q-learning makes an update after an experience (s_t, a_t, r_t, s_{t+1}) as follows:

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t))$$

Where $0 \leq \alpha \leq 1$ is the learning rate. Q-learning is an off-policy reinforcement learning algorithm [11], which means that the agent learns about the optimal value-function while following another behavioral policy that includes exploration steps. This has as advantage that it does not matter how much exploration is used, as long as the agent visits all state-action pairs an infinite number of times, tabular Q-learning (with appropriate learning rate adaptation) will converge to the optimal Q-function [13]. A disadvantage of Q-learning is that it can diverge when combined with function approximators. Another possible disadvantage is that off-policy algorithms do not modify the behavior of the agent to better deal with the exploration/exploitation dilemma [8].

Sarsa. Instead of Q-learning, we can also use the on-policy algorithm Sarsa [6], [10] for learning Q-values. Sarsa makes the following update after an experience $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$:

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha(r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t))$$

Tabular Sarsa converges in the limit to the optimal policy under proper learning rate annealing if the exploration policy is GLIE (greedy in the limit with infinite exploration), which means that the agent should always explore, but stop exploring after an infinite number of steps [8].

Actor-Critic. Another on-policy algorithm is the Actor-Critic (AC) method. In contrary to Q-learning and Sarsa, AC methods keep track of two functions; a Critic that evaluates states and an Actor that maps states to a preference value for each action. A number of Actor-Critic methods have been proposed [1], [4], [11]. Here we will use the Actor-Critic method described in [11]. After an experience (s_t, a_t, r_t, s_{t+1}) AC makes a TD-update to the Critic's value-function V:

$$V(s_t) := V(s_t) + \beta(r_t + \gamma V(s_{t+1}) - V(s_t))$$

where β is the learning rate. AC updates the Actor with values $P(s_t,a_t)$ as follows:

$$P(s_t,a_t) := P(s_t,a_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

where α is the learning rate for the Actor. The P-values should be seen as preference values and not as exact Q-values. Consider a bandit problem with one state and two actions. Both actions lead to an immediate deterministic reward of 1. When one action is selected a number of times in a row or the initial learning rate is 1, the state or V-value and the P-value for this action converge rapidly to 1. Afterwards the P-value of the other action can never increase anymore using AC and will not converge to the underlying Q-value of 1. A number of Actor-Critic methods have still been proved to

converge to the optimal policy and state value-function for tabular representations [4].

III. QV(λ)-LEARNING AND ACLA(λ)

We will now describe the two new on-policy reinforcement learning algorithms. QV(λ)-learning works by keeping track of both the Q- and V-functions. In QV-learning the state value-function V is learned with TD(λ)-methods [9]. This is similar to Actor-Critic methods. The new idea is that the Q-values simply learn from the V-values using the one-step Q-learning algorithm. In contrary to AC these learned values can be seen as actual Q-values and not as preference values.

QV-learning. The updates after an experience (s_t, a_t, r_t, s_{t+1}) of QV-learning are the following:

$$V(s_t) := V(s_t) + \beta(r_t + \gamma V(s_{t+1}) - V(s_t))$$

and

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

Note that the V-value used in this second update rule is learned by QV-learning and not defined in terms of Q-values. There is a strong resemblance with the Actor-Critic method; the only difference is the second learning rule where $V(s_t)$ is replaced by $Q(s_t, a_t)$ in QV-learning.

QV(λ)-learning. The updates after an experience (s_t, a_t, r_t, s_{t+1}) of QV(λ)-learning are the following:

$$\forall s: \quad V(s) := V(s) + \beta\delta_t l_t(s)$$

Where the eligibility traces $l_t(s)$ for all states are updated by:

$$l_t(s) := \gamma\lambda l_{t-1}(s) + \eta_t(s)$$

where $\eta_t(s)$ is the indicator function which returns 1 if state s occurred at time t ($s = s_t$), and 0 otherwise, and δ_t is defined as:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Furthermore, the Q-values are updated again with a form of the Q-learning rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

QV-learning is an on-policy algorithm, since the value-function is learned by TD-learning that uses experiences generated by the behavioural policy that includes exploration steps.

Actor Critic Learning Automaton. ACLA learns a state value-function in the same way as QV-learning, but ACLA uses a learning automaton-like update rule [5] for changing the policy mapping states to probabilities (or preferences) for actions. The updates after an experience (s_t, a_t, r_t, s_{t+1}) of ACLA are the following:

$$V(s_t) := V(s_t) + \beta(r_t + \gamma V(s_{t+1}) - V(s_t))$$

and, now we use an update rule that examines whether the last performed action was good (in which case the state-value was increased) or not. We do this with the following update rule:

$$\begin{aligned}
\text{If } \ \delta_t \geq 0 \qquad & \Delta P(s_t, a_t) = \alpha(1 - P(s_t, a_t)) \quad \text{and} \\
& \forall a \neq a_t \quad \Delta P(s_t, a) = \alpha(0 - P(s_t, a)) \\
\text{Else} \qquad & \Delta P(s_t, a_t) = \alpha(0 - P(s_t, a_t)) \quad \text{and} \\
& \forall a \neq a_t \quad \Delta P(s_t, a) = \alpha(\tfrac{P(s_t, a)}{\sum_{b \neq a_t} P(s_t, b)} - P(s_t, a))
\end{aligned}$$

After which we add $\Delta P(s_t, a)$ to $P(s_t, a)$. For ACLA we used some additional rules to ensure the targets are always between 0 and 1, independent of the initialization. This is done by using 1 if the target is larger than 1, and 0 if the target is smaller than 0. If the denominator ≤ 0, all targets in the last part of the update rule get the value $\frac{1}{|A|-1}$ where $|A|$ is the number of actions. The update in case of $\delta_t < 0$ is chosen to increase the preference of actions which are good more than actions that are considered worse. Above is the ACLA$-$ algorithm, we also extended ACLA$-$ to ACLA+ which can make multiple updates relying on the size of $\delta_t = \gamma V(s_{t+1}) + r_t - V(s_t)$. This algorithm keeps track of the whole state space's variance using the following update rule:

$$var = var + \mu(\delta_t^2 - var)$$

with μ a step-size parameter set to 0.001 and var is initialized to 10 in our experiments. The variance var is used to compute the number of times the ACLA-update above is made. This number of times equals: $\lceil \frac{|\delta_t|}{\sqrt{var}} \rceil$. Note that only ACLA+ makes use of this multiple updating technique, ACLA$-$ does not. We simply use ACLA to denote both algorithms. ACLA is similar to a conventional actor-critic system in which the update using δ_t is replaced by an update rule using the sign of δ_t. ACLA trains the actor to output a 1 for the best action and a zero for worse actions, which leads to a different training algorithm of the mapping between states and actions than the usual state-action value-function. This can also make it easier to take into account supervised examples of states and their optimal actions. For ACLA(λ), the algorithm stays the same except that the state values are learned using TD(λ).

Although ACLA can perform very well on particular problems, we remark that for particular highly stochastic environments, ACLA can converge to a suboptimal final policy. The reason is that essentially ACLA uses the sign of δ_t to make an update, instead of δ_t itself as conventional actor-critic methods. We have analysed that the only function on δ_t that can be used is in fact a linear function for converging in stochastic environments to an optimal policy. We will show this with an example. Suppose that there is a single state s and two actions. Action a_1 receives a reward of +100 with probability 10% and a reward of -10 with probability 90%. Action a_2 receives a reward of -100 with probability 10% and a reward of +10 with probability 90%. Thus, the values of action a_1 and a_2 are +1 and -1 respectively. Since the value of the state will be between -1 and 1 after some learning, action a_1 will have the sign of δ_t positive in 10% of the cases and negative in 90% of the cases. The opposite holds for action a_2. Therefore, action a_2 will be reinforced more often and will be finally chosen by the policy. Note that this example

uses a very strange probability distribution which may happen rarely in RL problems. If there is stochasticity, but the expected values of δ_t for the actions are ordered in the same way as the expected value of the probability of having $\delta_t > 0$, then ACLA still converges to an optimal policy. This trivially holds for a deterministic environment.

Comparison with previous algorithms. It is known that better convergence guarantees exist for on-policy methods when combined with function approximators [11], and therefore QV-learning and ACLA might work better than Q-learning. An advantage is also that compared to Sarsa, QV-learning and ACLA are less sensitive to exploration actions. Sarsa can make a large update if an action occurs with low probability that has been tried few times and has a very large negative value. QV-learning and ACLA learn the state's value and are therefore less vulnerable for exploration. A last possible advantage is that the V-function is updated with all experiences, whereas the Q-function has to be updated for a specific action. This may cause the V-function to learn faster than a Q-function. Because the state value-function receives more updates, this may also cause faster bootstrapping of the policy. Although many of these advantages are in principle shared by Actor-Critic methods, the experiments have to indicate whether Actor-Critic performs better or worse than the two new algorithms. A disadvantage of these algorithms is an additional learning parameter.

IV. Experiments

We performed seven experiments with different maze tasks and the mountain car problem to compare QV-learning and ACLA to other value-function-based RL algorithms. In the first two experiments, the RL algorithms are combined with tabular and neural network representations and are compared on a small maze task. In the third and fourth experiment, the RL algorithms are tested on a much larger maze using tabular representations and ϵ-greedy and Boltzmann exploration. In the fifth and sixth experiments we use a partially observable maze and a dynamic maze respectively and neural networks as function approximators. Finally, we perform experiments with neural networks on the mountain car problem.

A. Small Maze Experiment

We compare QV(λ)-learning and ACLA(λ) to naive Q(λ) [11], Sarsa(λ), and AC(λ). AC(λ) uses eligibility traces for both the Actor and the Critic [11]. We performed experiments with Sutton's Dyna maze shown in figure 1. This simple maze consists of 9×6 states and there are four actions; north, east, south, and west. We kept the maze small, since we also want to use neural networks in the experiments and wanted to prevent too much computational cost. The goal is to arrive at the goal state G as soon as possible starting from the starting state denoted by S under the influence of stochastic (noisy) actions.

Experimental set-up. The reward for arriving at the goal is 100. When the agent bumps against a wall or border of the environment it stays still and receives a reward of -2. For other steps the agent receives a reward of -0.1. A trial is finished after the agent hit the goal or 1000 actions have been performed. The random replacement (noise) in action execution is 20%. We use λ values of 0.0, 0.6, and 0.9. We use ϵ-greedy exploration with fixed ϵ = 10%.

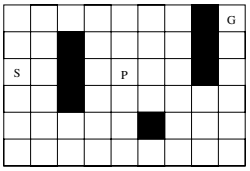

Fig. 1. Sutton's Dyna maze. The starting position is indicated by S and the goal position is indicated by G. In the partially observable maze of the fifth experiment the goal position is P and the starting position is arbitrary.

TABLE I

Tabular learning rates α/β (and discount factors) for the algorithms given different values for λ.

λ	QV	ACLA	Q	Sarsa	AC
0.0	0.15/0.15 (0.98)	0.1/0.05 (0.98)	0.25 (0.9)	0.2 (0.9)	0.15/0.05 (0.98)
0.6	0.15/0.10 (0.98)	0.15/0.04 (0.98)	0.15 (0.9)	0.2 (0.9)	0.15/0.03 (0.98)
0.9	0.15/0.04 (0.9)	0.1/0.03 (0.9)	0.1 (0.9)	0.1 (0.9)	0.1/0.03 (0.9)

1) Results for a Tabular Representation.: We performed experiments consisting of 50,000 learning steps and averaged the results of 500 simulations. For evaluation we measured after each 2,500 steps the average reward intake during that period. We first performed simulations to find the best learning rates and discount factors for the different values of λ for the different RL algorithms. We used the learning rates shown in Table I. Some algorithms use two learning rates (α and β).

TABLE II

Final results for a tabular representation with different values for λ. Results are averages of 500 simulations.

Method	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$	Rank
QV	**4.58 ± 0.16**	**4.52 ± 0.23**	**4.48 ± 0.25**	1
ACLA+	4.54 ± 0.19	4.42 ± 0.27	**4.47 ± 0.25**	2/3/4
Q	4.53 ± 0.22	4.42 ± 0.26	4.16 ± 0.35	2/3/4
Sarsa	4.52 ± 0.23	4.43 ± 0.33	4.26 ± 0.31	2/3/4
AC	4.29 ± 0.29	4.00 ± 0.35	3.75 ± 0.61	5

In Table II we show average results and standard deviations of 500 simulations of the final reward intake during the last 2500 learning-steps. The best final results for this small maze are obtained with $\lambda = 0.0$. We also see that QV-learning has the best final performance for this problem and significantly ($p < 0.01$) outperforms all other algorithms. Finally, we note that for high λ values, QV(λ)-learning and ACLA(λ) perform much better than the other algorithms. The Rank of each algorithm is computed with the student t-test using the results of the best value for λ.

In Table III we show average results and standard deviations of 500 simulations of the total summed reward (adding all 20 average reward intakes after each 2,500 steps) during the entire trial lasting 50,000 learning-steps. This evaluation measure shows the overall performance and the learning rate with which good solutions are obtained. The table shows

TABLE III

TOTAL SUMMED RESULTS FOR A TABULAR REPRESENTATION WITH DIFFERENT VALUES FOR λ. RESULTS ARE AVERAGES OF 500 SIMULATIONS.

Method	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$	Rank
QV	85.4 ± 3.1	85.2 ± 4.2	83.4 ± 4.6	2
ACLA+	**86.1 ± 4.9**	**86.2 ± 4.7**	**85.1 ± 6.2**	1
Q	81.6 ± 4.8	81.5 ± 5.1	76.0 ± 5.2	4
Sarsa	82.0 ± 4.6	83.2 ± 5.9	78.5 ± 5.5	3
AC	78.1 ± 5.7	76.7 ± 6.8	69.5 ± 13.0	5

that ACLA+ has the best overall performance and learns significantly faster for all values for λ. QV-learning comes as second best and Actor-Critic performs worst.

2) Results for Neural Networks.: We also performed experiments with neural networks as function approximators. As input-vector we used 54 inputs that indicate whether the agent is in that location. The state and actions use separate neural networks consisting of 20 hidden units and no skip-weights or input-output connections (which would allow for a tabular solution). This experiment was primarily conducted to see the difference in learning behavior between a tabular representation and the use of a neural network. We let the algorithms run for 100,000 learning steps and measured performance after each 5,000 steps. The experimental results are averages of 100 simulations.

TABLE IV

NEURAL NETWORK LEARNING RATES α/β (AND DISCOUNT FACTORS) FOR THE ALGORITHMS GIVEN DIFFERENT VALUES FOR λ.

λ	QV	ACLA	Q	Sarsa	AC
0.0	0.04/0.04 (0.9)	0.1/0.02 (0.98)	0.04 (0.9)	0.04 (0.9)	0.02/0.02 (0.98)
0.6	0.03/0.03 (0.9)	0.1/0.01 (0.98)	0.03 (0.9)	0.03 (0.9)	0.02/0.03 (0.9)
0.9	0.03/0.01 (0.9)	0.1/0.01 (0.9)	0.005 (0.9)	0.005 (0.9)	0.01/0.02 (0.9)

Parameters. We first performed experiments to set the best learning rates and discount factors. We used the learning rates shown in Table IV. For this problem we used ACLA− instead of ACLA+ since it performed slightly better.

Table V shows the final results for the last 5,000 learning steps. The table shows that ACLA− performs best in general, although for $\lambda = 0.9$ QV-learning is the best algorithm. Actor-Critic again performs worst of all algorithms.

TABLE V

FINAL RESULTS (AVERAGE REWARD FOR LAST 5,000 STEPS) WITH NEURAL NETWORKS. RESULTS ARE AVERAGES OF 100 SIMULATIONS.

Algorithm	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$	Rank
QV	4.65 ± 0.08	4.59 ± 0.10	**4.63 ± 0.10**	3/4
ACLA−	**4.71 ± 0.08**	**4.71 ± 0.07**	4.49 ± 0.13	1
Q	4.67 ± 0.08	4.45 ± 0.5	4.48 ± 0.96	2/3/4
Sarsa	4.68 ± 0.08	4.57 ± 0.5	4.37 ± 1.12	2/3
AC	3.65 ± 1.9	3.32 ± 2.2	2.9 ± 2.2	5

In Table VI we show average results and standard deviations of 100 simulations of the total summed reward (adding all 20 average reward intakes after each 5,000 steps) during the entire trial lasting 100,000 learning-steps. The table shows that ACLA− has the best overall performance and therefore learns fastest, but QV-learning performs best for $\lambda = 0.9$.

TABLE VI

NEURAL NETWORK TOTAL SUMMED RESULTS FOR 100,000 LEARNING STEPS WITH AVERAGE REWARD COMPUTATION AFTER EACH 5,000 STEPS. RESULTS ARE AVERAGES OF 100 SIMULATIONS.

Algorithm	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$	Rank
QV	81.4 ± 3.1	82.9 ± 1.9	**82.1 ± 2.4**	2
ACLA−	**84.0 ± 2.1**	**84.6 ± 0.8**	77.5 ± 2.6	1
Q	81.7 ± 2.9	70.1 ± 14.5	57.3 ± 17.8	3
Sarsa	77.4 ± 3.3	61.9 ± 18.1	49.6 ± 20.0	4
AC	56.4 ± 31.0	54.0 ± 36.3	33.7 ± 32.7	5

B. A Larger Maze Environment

We compare QV(λ)-learning and ACLA(λ) to naive Q(λ), Sarsa(λ), and AC(λ) with tabular representations on a larger maze shown in Fig. 2. The goal is again to arrive at the goal state G as soon as possible starting from the starting state denoted by S under the influence of stochastic (noisy) actions.

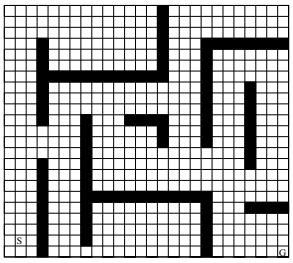

Fig. 2. The larger maze. The starting position is indicated by S and the goal position is indicated by G.

Experimental set-up. The reward function is the same as before. A trial is finished if the agent hit the goal or 10,000 actions have been performed. The random replacement in action execution is 20%. We use λ values of 0.0, 0.6, 0.9.

1) ϵ-greedy exploration: We performed experiments consisting of 400,000 learning steps with a tabular representation and ϵ-greedy exploration with $\epsilon = 0.1$. We averaged the results of 50 simulations. For evaluation we measured after each 20,000 steps the average reward intake during that period. We first performed simulations to find the best learning rates and discount factors for the different values of λ for the different RL algorithms. We used the learning rates shown in Table VII.

TABLE VII

TABULAR LEARNING RATES α/β (AND DISCOUNT FACTORS) FOR THE ALGORITHMS GIVEN DIFFERENT VALUES FOR λ.

λ	QV	ACLA	Q	Sarsa	AC
0.0	0.3/0.3 (0.99)	0.05/0.2 (0.99)	0.35 (0.98)	0.35 (0.97)	0.15/0.15 (0.98)
0.6	0.3/0.1 (0.99)	0.15/0.1 (0.99)	0.15 (0.97)	0.2 (0.99)	0.1/0.1 (0.99)
0.9	0.25/0.04 (0.99)	0.1/0.06 (0.99)	0.05 (0.96)	0.1 (0.995)	0.04/0.04 (0.99)

In Table VIII we show average results and standard deviations of 50 simulations of the final reward intake during the last 20,000 learning-steps. ACLA+ significantly outperforms the other algorithms, and Sarsa and Actor-Critic perform worst. Note that (naive) Q(λ)-learning fails completely for $\lambda = 0.9$.

In Table IX we show average results and standard deviations of 50 simulations of the total summed reward (adding all 20

TABLE VIII

FINAL RESULTS (AVERAGE REWARD FOR LAST 20,000 STEPS) ON THE LARGE MAZE FOR A TABULAR REPRESENTATION WITH DIFFERENT VALUES FOR λ. RESULTS ARE AVERAGES OF 50 SIMULATIONS.

Method	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$	Rank
QV	1.15 ± 0.09	**1.20 ± 0.05**	1.19 ± 0.05	2/3
ACLA+	**1.24 ± 0.02**	**1.20 ± 0.02**	**1.22 ± 0.02**	1
Q	1.19 ± 0.04	1.17 ± 0.05	0.27 ± 0.30	2/3
Sarsa	1.12 ± 0.06	1.13 ± 0.03	0.95 ± 0.10	5
AC	1.15 ± 0.03	1.09 ± 0.04	0.88 ± 0.19	4

average reward intakes after each 20,000 steps) during the entire trial lasting 400,000 learning-steps. It clearly indicates that ACLA+ learns fastest and has the best overall performance.

TABLE IX

TOTAL SUMMED RESULTS FOR A TABULAR REPRESENTATION WITH DIFFERENT VALUES FOR λ. RESULTS ARE AVERAGES OF 50 SIMULATIONS.

Method	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$	Rank
QV	17.2 ± 0.5	18.7 ± 0.4	18.9 ± 0.7	2
ACLA+	**19.7 ± 0.8**	**21.3 ± 0.3**	**21.6 ± 0.6**	1
Q	17.1 ± 0.6	15.0 ± 1.7	-0.5 ± 2.1	4/5
Sarsa	14.5 ± 1.0	17.0 ± 0.7	13.7 ± 0.8	4/5
AC	17.5 ± 0.6	18.1 ± 1.0	10.8 ± 3.2	3

2) Boltzmann exploration: We also performed experiments with the large maze using Boltzmann exploration. Again a trial consists of 400,000 learning steps with a tabular representation. We averaged the results of 100 simulations. For evaluation we measured after each 20,000 steps the average reward intake during that period. We first performed simulations to find the best learning rates, discount factors, and greediness (inverse of the temperature) used in the Boltzmann exploration rule. We use a fixed value for the greediness: schemes for increasing the greediness online did not improve results. We used different values of λ (0.0 and 0.9). The learning rates are shown in Table X.

TABLE X

TABULAR LEARNING RATES α/β (AND DISCOUNT FACTORS, AND GREEDINESS) FOR THE ALGORITHMS GIVEN DIFFERENT VALUES FOR λ.

λ	QV	ACLA	Q	Sarsa	AC
0.0	0.3/0.3 (0.99, 9)	0.02/0.25 (0.99, 13)	0.3 (0.98, 9)	0.3 (0.98, 9)	0.02/0.4 (0.99, 10)
0.9	0.2/0.02 (0.99, 8)	0.02/0.05 (0.99, 9)	0.07 (0.99, 8)	0.1 (0.99, 8)	0.005/0.25 (0.99, 9)

In Table XI we show average results and standard deviations of 100 simulations of the final reward intake during the last 20,000 steps. It shows that using Boltzmann exploration ACLA performs best, but is closely followed by AC.

In Table XII we show average results and standard deviations of 100 simulations of the total summed reward. It clearly indicates that ACLA+ learns fastest and has the best overall performance.

C. A Partially Observable Maze

In this experiment we will use Markov localization and neural networks to solve a partially observable Markov decision process in case the model of the environment is known. We use

TABLE XI

FINAL RESULTS (AVERAGE REWARD FOR LAST 20,000 STEPS) FOR A TABULAR REPRESENTATION WITH BOLTZMANN EXPLORATION WITH DIFFERENT VALUES FOR λ. RESULTS ARE AVERAGES OF 100 SIMULATIONS.

Method	$\lambda = 0.0$	$\lambda = 0.9$	Rank
QV	1.24 ± 0.18	0.74 ± 0.25	3/4/5
ACLA+	**1.41 ± 0.02**	**1.38 ± 0.03**	1
Q	1.23 ± 0.20	0.67 ± 0.17	3/4/5
Sarsa	1.21 ± 0.20	0.66 ± 0.19	3/4/5
AC	1.39 ± 0.02	1.15 ± 0.36	2

TABLE XII

TOTAL SUMMED RESULTS FOR A TABULAR REPRESENTATION WITH BOLTZMANN EXPLORATION WITH DIFFERENT VALUES FOR λ. RESULTS ARE AVERAGES OF 100 SIMULATIONS.

Method	$\lambda = 0.0$	$\lambda = 0.9$	Rank
QV	18.1 ± 2.8	10.9 ± 4.0	2/3
ACLA+	**22.5 ± 0.8**	**22.8 ± 1.7**	1
Q	13.7 ± 2.6	8.1 ± 3.1	4/5
Sarsa	13.5 ± 2.6	9.9 ± 3.3	4/5
AC	18.1 ± 1.7	16.9 ± 6.6	2/3

Markov localization to track the beliefstate (or probability distribution over the states) given an action and observation after each time-step. This beliefstate is then the input for the neural network. We used 20 hidden neurons in our experiments, and the maze shown in Fig. 1 with the goal indicated by P and each state can be the starting state. The initial beliefstate is a uniform distribution where only states that are not obstacles get assigned a belief. After each action a_t the beliefstate $b_t(s)$ is updated with the observation o_{t+1}:

$$b_{t+1}(s) = \eta P(o_{t+1}|s) \sum_{s'} T(s', a_t, s) b_t(s')$$

Where η is some normalization factor. The observations are whether there is a wall to the north, east, south, and west. Thus, there are 16 possible observations. We use 20% noise in the action execution and 10% noise for observing each independent wall (or empty cell) at the sides. That means that a correct observation is observed with probability 0.9^4 = 66%. Note that we use a model of the environment to be able to compute the beliefstate, and the model is based on the uncertainties in the transition and observation functions.

We performed experiments consisting of 100,000 learning steps with a neural network representation and ϵ-greedy exploration with $\epsilon = 0.1$. We averaged the results of 100 simulations. For evaluation we measured after each 5,000 steps the average reward intake during that period. We first performed simulations to find the best learning rates and discount factors for the different values of λ for the different RL algorithms. The learning rates are shown in Table XIII.

In Table XIV we show average results and standard deviations of 100 simulations of the final reward intake during the last 5,000 learning-steps. We observe that AC outperforms the other algorithms. ACLA seems to suffer from the high stochasticity in the partially obervable maze due to the noise in the observations.

TABLE XIII

NEURAL NETWORK LEARNING RATES α/β (AND DISCOUNT FACTORS) FOR THE ALGORITHMS GIVEN DIFFERENT VALUES FOR λ.

λ	QV	ACLA	Q	Sarsa	AC
0.0	0.015/0.015 (0.98)	0.035/0.005 (0.99)	0.01 (0.95)	0.03 (0.99)	0.015/0.02 (0.95)
0.6	0.005/0.005 (0.98)	0.015/0.005 (0.9)	0.005 (0.95)	0.005 (0.95)	0.01/0.02 (0.99)
0.9	0.005/0.01 (0.95)	0.015/0.005 (0.95)	0.005 (0.9)	0.01 (0.95)	0.005/0.01 (0.9)

TABLE XIV

FINAL RESULTS (AVERAGE REWARD FOR LAST 5,000 STEPS) FOR A NEURAL NETWORK REPRESENTATION ON THE PARTIALLY OBSERVABLE MAZE. RESULTS ARE AVERAGES OF 100 SIMULATIONS.

Method	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$	Rank
QV	8.80 $\pm$ 0.33	8.90 $\pm$ 0.31	8.69 $\pm$ 0.36	2/3/4
ACLA-	8.18 $\pm$ 0.29	7.94 $\pm$ 0.30	7.47 $\pm$ 0.44	5
Q	8.83 $\pm$ 0.25	8.87 $\pm$ 0.31	8.79 $\pm$ 0.31	2/3/4
Sarsa	8.62 $\pm$ 0.34	8.87 $\pm$ 0.29	8.64 $\pm$ 0.53	2/3/4
AC	8.90 $\pm$ 0.31	**9.02 $\pm$ 0.33**	8.13 $\pm$ 1.14	1

In Table XV we show average results and standard deviations of 100 simulations of the total summed reward (adding all 20 average reward intakes after each 5,000 steps) during the entire trial lasting 100,000 learning-steps.

TABLE XV

TOTAL SUMMED RESULTS FOR A NEURAL NETWORK REPRESENTATION WITH DIFFERENT VALUES FOR λ. RESULTS ARE AVERAGES OF 100 SIMULATIONS.

Method	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$	Rank
QV	**138.7 $\pm$ 4.9**	98.6 $\pm$ 12.6	94.6 $\pm$ 11.5	1/2/3
ACLA-	131.3 $\pm$ 5.0	100.7 $\pm$ 4.3	88.3 $\pm$ 13.4	4
Q	115.6 $\pm$ 11.7	104.6 $\pm$ 16.6	97.8 $\pm$ 18.1	5
Sarsa	**137.0 $\pm$ 6.2**	99.6 $\pm$ 11.6	**110.9 $\pm$ 17.1**	1/2/3
AC	129.5 $\pm$ 10.7	**137.9 $\pm$ 13.1**	93.9 $\pm$ 25.0	1/2/3

D. Solving a Dynamic Maze

We also compared QV-learning and ACLA to Sarsa, Q-learning and AC on a dynamic maze in which each trial there are several obstacles at random locations. In order to deal with this task the agent uses a neural network that receives as inputs whether a particular state-cell contains an obstacle (1) or not (0). The agent cannot go through obstacles or push them away. At the start of each new trial there are between 4 and 7 obstacles generated at random positions and it is made sure that a path to the goal exists from the fixed starting location S. A specific instance of this maze is shown in Fig. 3. Since there are many instances of this maze, essentially the neural network has to learn the knowledge of a path planner. Since preliminary experiments indicated that the best results were obtained with $\lambda = 0$, we do not show results with the use of eligibility traces.

Parameters. We used ϵ-greedy exploration with a fixed $\epsilon = 0.1$. The reward function is the same as before. We used 20% noise in action execution. A simulation lasts for 3,000,000 learning steps and we measure performance after each 150,000 steps. A trial ends after 1000 actions or when the goal is hit. Results are averages of 50 simulations. We used feedforward neural networks with 60 sigmoidal hidden units.

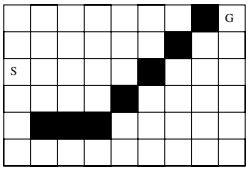

Fig. 3. The dynamic 9×6 maze. The starting position is denoted by S and the goal position is indicated by G. The obstacles indicated in black are dynamically generated at the start of each new trial.

The best found learning rates are shown in Table XVI, but note that it was difficult to do many experiments for finding optimal learning rates.

TABLE XVI

NEURAL NETWORK LEARNING RATES α/β (AND DISCOUNT FACTORS) FOR THE ALGORITHMS FOR THE DYNAMIC MAZE.

λ	QV	ACLA+	Q	Sarsa	AC
0.0	0.005/0.005 (0.9)	0.1/0.005 (0.98)	0.008 (0.9)	0.008 (0.9)	0.005/0.005 (0.9)

Table XVII shows the final and total performance of the different algorithms. Q-learning outperforms the other algorithms on this problem: it reaches the best final performance and also has the best overall learning performance.

TABLE XVII

FINAL RESULTS (AVERAGE REWARD FOR LAST 150,000 STEPS) AND TOTAL SUMMED RESULTS FOR A NEURAL NETWORK REPRESENTATION ON THE DYNAMIC MAZE. RESULTS ARE AVERAGES OF 50 SIMULATIONS.

Method	Final	Rank	Total	Rank
QV	6.05 $\pm$ 0.18	2	103.6 $\pm$ 2.2	2
ACLA+	5.72 $\pm$ 0.18	4	98.6 $\pm$ 1.5	3/4
Q	**6.22 $\pm$ 0.15**	1	**108.7 $\pm$ 2.6**	1
Sarsa	5.91 $\pm$ 0.18	3	97.6 $\pm$ 2.9	3/4
AC	1.86 $\pm$ 0.63	5	27.0 $\pm$ 6.7	5

E. Mountain Car Experiments

We finally compare the RL algorithms on the mountain car problem, see [7] for a description of this problem. We use neural networks as representations of the value functions and used 20 sigmoidal hidden units. There are 2 inputs (velocity and position) and 3 actions (right, left, no-action). The reward function emits -0.1 on every step and 0 if the goal is hit. The maximum number of actions in a trial is set to 1000. The number of trials in an experiment is set to 76,000. The results are averages of 30 simulations. The best found learning rates are shown in Table XVIII, we only report results for $\lambda = 0.0$, because it gave the best results.

TABLE XVIII

NEURAL NETWORK LEARNING RATES α/β (AND DISCOUNT FACTORS) FOR THE ALGORITHMS FOR THE MOUNTAIN CAR PROBLEM.

λ	QV	ACLA+	Q	Sarsa	AC
0.0	0.04/0.013 (0.99)	0.07/0.047 (0.99)	0.06 (0.99)	0.063 (0.99)	0.02/0.007 (0.99)

Table XIX shows the results. Sarsa significantly outperforms the other methods on this problem. It also learns very fast: its overall learning performance is better than the final performance of all other algorithms.

TABLE XIX

FINAL RESULTS (AVERAGE NUMBER OF STEPS TO REACH THE GOAL FOR LAST 4,000 TRIALS) AND TOTAL AVERAGE NUMBER OF STEPS TO REACH THE GOAL FOR A NEURAL NETWORK REPRESENTATION ON THE MOUNTAIN CAR PROBLEM. RESULTS ARE AVERAGES OF 30 SIMULATIONS.

Method	Final	Rank	Total	Rank
QV	147 ± 3	2/3/4	153 ± 3	2/3
ACLA-	144 ± 10	2/3/4	153 ± 5	2/3
Q	153 ± 3	5	157 ± 2	4/5
Sarsa	126 ± 1	1	131 ± 3	1
AC	145 ± 6	2/3/4	158 ± 9	4/5

V. DISCUSSION

The RL algorithms were compared on many different problems. If we put the ranks of all seven experiments in tables for the final performance of the learning controller, we get the overall results shown in Table XX. This table shows that the overall results of QV-learning and ACLA are better than the overall final results obtained with the other three algorithms.

TABLE XX

THE RANKS OF THE DIFFERENT ALGORITHMS WHEN WE LOOK AT FINAL PERFORMANCE OF THE LEARNED CONTROLLERS.

Experiment	QV	ACLA	Q	Sarsa	AC
1 (Tab.)	1	2/3/4	2/3/4	2/3/4	5
2 (NN)	3/4	1	2/3/4	2/3	5
3 (Tab.)	2/3	1	2/3	5	4
4 (Tab.)	3/4/5	1	3/4/5	3/4/5	2
5 (NN)	2/3/4	5	2/3/4	2/3/4	1
6 (NN)	2	4	1	3	5
7 (NN)	2/3/4	2/3/4	5	1	2/3/4
Total	19	18	21.5	21.5	25

The overall learning performance of the different algorithms where the performances are measured during a whole learning trial are shown in Table XXI. QV-learning and ACLA have the best overall performance and therefore are the fastest RL algorithms on the tested problem.

TABLE XXI

THE RANKS OF THE DIFFERENT ALGORITHMS WHEN WE LOOK AT TOTAL LEARNING PERFORMANCE DURING A COMPLETE TRIAL.

Experiment	QV	ACLA	Q	Sarsa	AC
1 (Tab.)	2	1	4	3	5
2 (NN)	2	1	3	4	5
3 (Tab.)	2	1	4/5	4/5	3
4 (Tab.)	2/3	1	4/5	4/5	2/3
5 (NN)	1/2/3	4	5	1/2/3	1/2/3
6 (NN)	2	3/4	1	3/4	5
7 (NN)	2/3	2/3	4/5	1	4/5
Total	15	14	26.5	22.5	27

The results of independent experiments have also shown some interesting results. Boltzmann exploration seems to be a good option for ACLA and AC, since these methods learn preference values for actions and therefore select actions which are clearly optimal in all cases. AC may perform better with eligibility traces, if we only use the traces for the critic and not also for the actor. We did not try that possibility. In most cases eligibility traces did not improve results, although learning speed improved in the large maze. A strange problem for using eligibility traces is the dynamic maze, where the obstacles that remain stationary during a trial get very large traces compared to the position of the agent. Therefore, for this problem eligibility traces failed.

VI. CONCLUSION

We introduced two new value-function based reinforcement learning algorithms, ACLA and QV(λ)-learning, which are based on TD(λ) methods for learning a state value-function, and another update rule to learn either Q-values or preference values for selecting actions. The new algorithms have some advantages compared to Sarsa and Q-learning, and one of them is that the state value-function is updated more often than a state-action value function, which can cause faster bootstrapping of the policy. Another advantage is that these algorithms use TD(λ)-methods and therefore are less sensitive to exploration actions and work better with eligibility traces than the other methods. The experiments showed that ACLA and QV-learning in general learn fastest and reach the best final performance, although the results differ a lot for different experiments. In future work we want to use ensembles of RL algorithms and let the agent discover which algorithm works best for a specific environment.

REFERENCES

[1] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[2] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[4] V.R. Konda and V. Borkar. Actor-critic type learning algorithms for Markov decision processes. *SIAM Journal on Control and Optimization*, 38(1):94–123, 1999.

[5] K. S. Narendra and M. A. L. Thathatchar. Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 4:323–334, 1974.

[6] G.A. Rummery and M. Niranjan. On-line Q-learning using connectionist sytems. Technical Report CUED/F-INFENG-TR 166, Cambridge University, UK, 1994.

[7] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[8] S.P. Singh, T. Jaakkola, M.L. Littman, and C. Szepesvari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[9] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[10] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1038–1045. MIT Press, Cambridge MA, 1996.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA, A Bradford Book, 1998.

[12] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

[13] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.