QV(λ)-learning: A New On-policy Reinforcement Learning Algorithm

Marco A. Wiering MARCO@CS.UU.NL
Intelligent Systems Group, Institute of Information and Computing Sciences, Utrecht University

1. Introduction

Reinforcement learning algorithms (Sutton & Barto, 1998; Kaelbling et al., 1996) are very suitable for learning to control an agent by letting it interact with an environment. Currently, there are three well-known value function based reinforcement learning (RL) algorithms that use the discounted future reward criterium; Q-learning (Watkins, 1989), Sarsa (Rummery & Niranjan, 1994; Sutton, 1996), and Actor-Critic (AC) methods (Sutton & Barto, 1998; Konda & Tsitsiklis, 2003). Alternatively, a number of policy search and policy gradient algorithms have been proposed. This paper introduces a new value function based RL algorithm, named QV-learning. Similar to Actor-Critic methods and in contrast to Q-learning and Sarsa, QV-learning keeps track of two value functions.[1] QV-learning learns the state value function with temporal difference learning, and uses this estimated state value function to learn the state-action values with Q-learning. QV-learning is also enhanced by eligibility traces (Sutton, 1988) to learn the values of the state value function using TD(λ) methods, whereas one-step Q-learning is still used to update the Q-function. In contrast to Actor-Critic methods that also learn two separate functions for the Actor and the Critic, QV-learning learns the real underlying Q-function and not preference values for the different actions.

2. QV(λ)-learning

QV(λ)-learning works by keeping track of both the Q- and V-functions. In QV-learning, the state-value function V is trained with normal TD(λ)-methods (Sutton, 1988). The new thing is that the Q-values simply learn from the V-values using the one-step Q-learning algorithm. The V-function might converge faster to optimal values than the Q-function since it does not consider the actions and is updated more often. Therefore, using QV-learning, the Q-values can be easily learned and compared by the way that an action in a state leads to different successor states. The updates after an experience (s_t, a_t, r_t, s_{t+1}) of QV(λ)-learning are the following:

$$V(s) := V(s) + \alpha \delta_t e_t(s)$$

Where the eligibility traces are updated by:

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

Where $\eta_t(s)$ is the indicator function which returns 1 if state s occurred at time 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 using the Q-learning rule:

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

Note that the V-value used in this update is learned by QV-learning and not defined in terms of Q-values. We first execute the update rule for the Q-values before using the TD(λ) rule to update the V-values. QV-learning is an on-policy algorithm, since the state value function is learned using the behavioural policy that includes exploration steps. In this way, QV-learning is quite similar to Sarsa, and for convergence it also needs that the exploration policy is greedy in the limit of infinite exploration (GLIE) and that the learning rate follows the usual stochastic approximation conditions (Singh et al., 2000). We left out the convergence proof for QV(0)-learning due to space limiations, but will present it at the workshop.

3. Experiments

We compare QV(λ)-learning to naive Q(λ), Peng and Williams' Q(λ)-learning (Peng & Williams, 1996) which we call PWQ, Sarsa(λ), and AC(λ). We performed experiments with Sutton's Dyna maze (Sutton, 1990). 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 used neural networks in the experiments and wanted to prevent

[1] Note that Q-learning and Sarsa only learn state-action values, the value of a state is defined by the different state-action values of actions applicable in that state.

too much computational cost. The goal is to arrive at the goal state G as soon as possible starting from another randomly chosen free location. The randomness (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 $\epsilon = 20\%$.

Results for Tabular Representations. We performed experiments consisting of 50,000 steps and averaged the results of 500 simulations. For evaluating we use the greedy policy and let it run 20 times 1000 steps and the average is kept as result. We first performed some smaller simulations to find the best learning rates for the different RL algorithms.

In Table 1 we show average results and standard deviations of 500 test-simulations after 50,000 steps. For higher λ values, QV(λ)-learning performs significantly better ($p < 0.02$) than the other algorithms.

Algorithm	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$
QV	9.33K	**9.33K ± 0.13K**	**9.26K ± 0.22K**
Q	9.35K	9.28K ± 0.14K	8.90K ± 1.1K
PWQ	9.35K	9.29K ± 0.33K	8.80K ± 1.5K
SARSA	9.35K	9.28K ± 0.30K	9.05K ± 0.78K
AC	9.30K	9.16K ± 0.27K	9.08K ± 0.87K

Table 1. Results for a tabular representation with different values for λ.

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. We used 20 hidden units and no skip-weights or input-output connections (which would allow for a tabular solution). We evaluated the greedy policy after each 5,000 steps during the first 50,000 steps and summed these evalations to compute the initial learning performance for all algorithms. We average over 100 simulations.

Table 2 shows the results of the summed values of the first 10 experiments (after each 5,000 steps) during the first 50,000 steps. We can see that QV-learning learns significantly faster ($p < 0.0001$) than the other algorithms.

Algorithm	$\lambda = 0.0$	$\lambda = 0.6$	$\lambda = 0.9$
QV	**49.5K ± 9.9K**	**51.4K ± 11.2K**	**52.2K**
Q	37.9K ± 13.9K	27.7K ± 16.0K	20.9K
PWQ	37.9K ± 13.9K	38.3K ± 13.3K	34.4K
SARSA	30.0K ± 14.0K	27.3K ± 11.7K	29.6K
AC	24.8K ± 16.8K	35.9K ± 23.9K	34.7K

Table 2. Neural network total results for first 50,000 steps with tests after each 5,000 steps.

4. Conclusion

We introduced a new value function based reinforcement learning algorithm, QV(λ)-learning, which is a hybrid algorithm combining Q-learning and TD(λ)-methods. QV(λ)-learning is an on-policy RL algorithm that learns Q-values and separate V-values. It has some possible advantages compared to the most commonly used algorithms Q-learning and Sarsa. Its main advantage compared to Q-learning is that QV-learning is an on-policy algorithm and therefore has better convergence guarantees when combined with function approximators compared to Q-learning. Compared to Sarsa, QV-learning is less sensitive to exploration steps. The experiments showed that QV(λ)-learning learns faster than the other algorithms when neural networks are used to approximate the value functions. Also when high values of λ are used, QV(λ)-learning outperformed the other algorithms.

References

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Konda, V., & Tsitsiklis, J. (2003). Actor-critic algorithms. *SIAM Journal on Control and Optimization*, *42(4)*, 1143–1166.

Peng, J., & Williams, R. (1996). Incremental multi-step Q-learning. *Machine Learning*, *22*, 283–290.

Rummery, G., & Niranjan, M. (1994). *On-line Q-learning using connectionist sytems* (Technical Report CUED/F-INFENG-TR 166). Cambridge University, UK.

Singh, S., Jaakkola, T., Littman, M., & Szepesvari, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, *38*, 287–308.

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

Sutton, R. S. (1990). Integrated architectures for learning, planning and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216 – 224).

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8* (pp. 1038–1045). MIT Press, Cambridge MA.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* The MIT press, Cambridge MA, A Bradford Book.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* Doctoral dissertation, King's College, Cambridge, England.