Machine Learning 33, 105–115 (1998)
© 1998 Kluwer Academic Publishers. Manufactured in The Netherlands.

Fast Online Q(λ)

MARCO WIERING marco@idsia.ch
JÜRGEN SCHMIDHUBER juergen@idsia.ch
IDSIA, Corso Elvezia 36, 6900 Lugano, Switzerland

Editor: Leslie Pack Kaelbling

Abstract. Q(λ)-learning uses TD(λ)-methods to accelerate Q-learning. The update complexity of previous online Q(λ) implementations based on lookup tables is bounded by the size of the state/action space. Our faster algorithm's update complexity is bounded by the number of actions. The method is based on the observation that Q-value updates may be postponed until they are needed.

Keywords: reinforcement learning, Q-learning, TD(λ), online Q(λ), lazy learning

1. Introduction

Q(λ)-learning (Watkins, 1989; Peng & Williams, 1996) is an important reinforcement learning (RL) method. It combines Q-learning (Watkins, 1989; Watkins & Dayan, 1992) and TD(λ) (Sutton, 1988; Tesauro, 1992). Q(λ) is widely used—it is generally believed to outperform simple one-step Q-learning, since it uses *single* experiences to update evaluations of *multiple* state/action pairs (SAPs) that have occurred in the past.

Online vs. offline. We distinguish *online* RL and *offline* RL. Online RL updates modifiable parameters after each visit of a state. Offline RL delays updates until after a trial is finished, that is, until a goal has been found or a time limit has been reached. Without explicit trial boundaries, offline RL does not make sense at all. But even where applicable, offline RL tends to get outperformed by online RL, which uses experience earlier and therefore more efficiently (Rummery & Niranjan, 1994). Online RL's advantage can be huge. For instance, online methods that punish actions (to prevent repetitive selection of identical actions) can discover certain environments' goal states in polynomial time (Koenig & Simmons, 1996), while offline RL requires exponential search time (Whitehead, 1992).

Previous Q(λ) implementations. To speed up Q-learning, Watkins (1989) suggested combining it with TD(λ) learning. His approach resets eligibility traces once exploratory actions are executed, while Peng and Williams' variant (1996) does not require this. Typical *online* Q(λ) implementations based on lookup tables or other local approximators such as CMACs (Albus, 1975; Sutton, 1996) or self-organizing maps (Kohonen, 1988), however, are unnecessarily time-consuming. Their update complexity depends on the values of λ and discount factor γ, and is proportional to the number of SAPs (state/action pairs) which have occurred.

The latter is bounded by the size of state/action space (and by the trial length which may be proportional to this).

Lin's *offline* Q(λ) (1993) creates an action-replay set of experiences after each trial. Cichosz' *semi-online* method (1995) combines Lin's offline method and online learning. It needs fewer updates than Peng and Williams' online Q(λ), but postpones Q-updates until several subsequent experiences are available. Hence, actions executed before the next Q-update are less informed than they could be. This may result in performance loss. For instance, suppose that the same state is visited twice in a row. If some hazardous action's Q-value does not reflect negative experience collected after the first visit then it may get selected again with higher probability than wanted.

The novel method. Previous methods either are not truly online and thus most likely require more experiences than necessary, or their updates are less efficient than they could be and thus require more computation time. Our Q(λ) variant is truly online and more efficient than others because its update complexity does not depend on the number of states. The method can also be used for speeding up tabular TD(λ). It uses "lazy learning" (introduced in memory-based learning, e.g., Atkeson, Moore, & Schaal, 1997) to postpone updates until they are needed.

Outline. Section 2 reviews Q(λ) and describes Peng and William's Q(λ)-algorithm (PW). Section 3 presents our more efficient algorithm. Section 4 concludes.

2. Q(λ)-learning

We consider finite Markov decision processes, using discrete time steps $t = 1, 2, 3, \ldots,$ a finite set of states $S = \{S_1, S_2, S_3, \ldots, S_n\}$ and a finite set of actions A. The state at time t is denoted by s_t, and $a_t = \Pi(s_t)$ the selected action, where Π represents the learner's policy mapping states to actions. The transition probability to the next state s_{t+1}, given s_t and a_t, is determined by $P_{ij}^a = P(s_{t+1} = j \mid s_t = i, a_t = a)$ for $i, j \in S$ and $a \in A$. A reward function R maps SAP $(i, a) \in S \times A$ to scalar reinforcement signals $R(i, a) \in \mathbb{R}$. The reward at time t is denoted by r_t. A discount factor $\gamma \in [0, 1]$ discounts later against immediate rewards. The controller's goal is to select actions which maximize the expected long-term cumulative discounted reinforcement, given an initial state selected according to a probability distribution over possible initial states.

Reinforcement learning. To achieve this goal, most reinforcement learning methods learn an action evaluation function or Q-function. The optimal Q-value of SAP (i, a) satisfies

$$Q^*(i, a) = R(i, a) + \gamma \sum_j P_{ij}^a V^*(j), \tag{1}$$

where $V^*(j) = \max_a Q^*(j, a)$. To learn this Q-function, RL algorithms repeatedly do: (1) Select action a_t given state s_t; (2) collect reward r_t and observe successor state s_{t+1}; and (3) update the Q-function using the latest experience (s_t, a_t, r_t, s_{t+1}).

Q-learning. Given (s_t, a_t, r_t, s_{t+1}), standard one-step Q-learning updates just a single Q-value $Q(s_t, a_t)$ as follows (Watkins, 1989):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)e'_t.$$

Here the temporal difference or TD(0)-error e'_t is given by:

$$e'_t = (r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)),$$

where the value function $V(s)$ is defined as $V(s) = \max_a Q(s, a)$, and $\alpha_k(s_t, a_t)$ is the learning rate for the kth update of SAP (s_t, a_t).

Learning rate adaptation. The learning rate $\alpha_k(s, a)$ for the kth update of SAP (s, a) should decrease over time to satisfy two conditions for stochastic iterative algorithms (Watkins & Dayan, 1992; Bertsekas & Tsitsiklis, 1996):

1. $\sum_{k=1}^{\infty} \alpha_k(s, a) = \infty$,
2. $\sum_{k=1}^{\infty} \alpha_k^2(s, a) < \infty$.

They hold for $\alpha_k(s, a) = 1/k^\beta$, where $1/2 < \beta \le 1$.

Q(λ)-learning. Q(λ) uses TD(λ)-methods (Sutton, 1988) to accelerate Q-learning. First note that Q-learning's update at time $t + 1$ may change $V(s_{t+1})$ in the definition of e'_t. Following Peng & Williams (1996) we define the TD(0)-error of $V(s_{t+1})$ as

$$e_{t+1} = (r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1})).$$

Q(λ) uses a factor $\lambda \in [0, 1]$ to discount TD-errors of future time steps:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)e^\lambda_t,$$

where the TD(λ)-error e^λ_t is defined as

$$e^\lambda_t = e'_t + \sum_{i=1}^{\infty} (\gamma\lambda)^i e_{t+i}.$$

Eligibility traces. The updates above cannot be made as long as TD-errors of future time steps are not known. We can compute them incrementally, however, by using eligibility traces (Barto, Sutton, & Anderson, 1983; Sutton, 1988). In what follows, $\eta^t(s, a)$ will denote the indicator function which returns 1 if (s, a) occurred at time t, and 0 otherwise.

Omitting the learning rate for simplicity, the increment of $Q(s, a)$ for the complete trial is:

$$\begin{aligned}\Delta Q(s,a) &= \lim_{k\to\infty}\sum_{t=1}^{k} e_t^{\lambda}\eta^t(s,a)\\ &= \lim_{k\to\infty}\sum_{t=1}^{k}\left[e'_t\eta^t(s,a) + \sum_{i=t+1}^{k}(\gamma\lambda)^{i-t}e_i\eta^t(s,a)\right]\\ &= \lim_{k\to\infty}\sum_{t=1}^{k}\left[e'_t\eta^t(s,a) + \sum_{i=1}^{t-1}(\gamma\lambda)^{t-i}e_t\eta^i(s,a)\right]\\ &= \lim_{k\to\infty}\sum_{t=1}^{k}\left[e'_t\eta^t(s,a) + e_t\sum_{i=1}^{t-1}(\gamma\lambda)^{t-i}\eta^i(s,a)\right].\end{aligned} \quad (2)$$

To simplify this we use an eligibility trace $l_t(s, a)$ for each SAP (s, a):

$$l_t(s,a) = \sum_{i=1}^{t-1}(\gamma\lambda)^{t-i}\eta^i(s,a).$$

Then the online update at time t becomes:

$$\forall(s,a)\in S\times A \text{ do}:\quad Q(s,a)\leftarrow Q(s,a)+\alpha_k(s_t,a_t)[e'_t\eta^t(s,a)+e_t l_t(s,a)].$$

Online Q(λ). We will focus on Peng and Williams' algorithm (PW) (1996), although there are other possible variants (e.g., Rummery and Niranjan, 1994). PW uses a list H of SAPs that have occurred at least once. SAPs with eligibility traces below $\epsilon \geq 0$ are removed from H. Boolean variables *visited* (s, a) are used to make sure no two SAPs in H are identical.

```
PW's Q(λ)-update($s_t, a_t, r_t, s_{t+1}$) :
(1) $e'_t \leftarrow (r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$
(2) $e_t \leftarrow (r_t + \gamma V(s_{t+1}) - V(s_t))$
(3) For each SAP $(s, a) \in H$ Do :
        (3a) $l(s, a) \leftarrow \gamma\lambda l(s, a)$
        (3b) $Q(s, a) \leftarrow Q(s, a) + \alpha_k(s_t, a_t)e_t l(s, a)$
        (3c) If ($l(s, a) < \epsilon$)
                (3c-1) $H \leftarrow H \setminus (s, a)$
                (3c-2) $visited(s, a) \leftarrow 0$
(4) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)e'_t$
(5) $l(s_t, a_t) \leftarrow l(s_t, a_t) + 1$
(6) If ($visited(s_t, a_t) = 0$)
        (6a) $visited(s_t, a_t) \leftarrow 1$
        (6b) $H \leftarrow H \cup (s_t, a_t)$
```

Comments.

1. The SARSA algorithm (Rummery & Niranjan, 1994) replaces the right-hand side in lines (1) and (2) by $(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$.
2. For replacing eligibility traces (Singh & Sutton, 1996), step (5) should be: $\forall a : l(s_t, a) \leftarrow 0; l(s_t, a_t) \leftarrow 1$.
3. Representing H by a doubly linked list and using direct pointers from each SAP to its position in H, the functions operating on H (deleting and adding elements—see lines (3c-1) and (6b)) cost $O(1)$.

Complexity. Deleting SAPs from H (step (3c-1)) once their traces fall below a certain threshold may significantly speed up the algorithm. If $\gamma\lambda$ is sufficiently small, then this will keep the number of updates per time step manageable. For large $\gamma\lambda$, PW does not work that well: it needs a sweep (sequence of SAP updates) after each time step, and the update cost for such sweeps grows with $\gamma\lambda$. Let us consider worst-case behavior, which means that each SAP occurs just once (if SAPs reoccur then the history list will grow at a slower rate). In the beginning of the trial the number of updates increases linearly until at some time step t some SAPs get deleted from H. This will happen as soon as $t \geq \log \epsilon / \log(\gamma\lambda)$. Since the number of updates is bounded from above by the number of SAPs, the total update complexity increases towards $O(|S||A|)$ per update for $\gamma\lambda \to 1$.

The space complexity of the algorithm is $O(|S||A|)$. We need to store for all SAPs: Q-values, eligibility traces, the "visited" bit variable and three pointers for managing the history list (one from the SAP to its place in the history list, and two for the doubly linked list).

3. Fast Q(λ)-learning

The main contribution of this paper is an efficient, fully online algorithm with time complexity $O(|A|)$ per update. The algorithm is designed for $\lambda\gamma > 0$—otherwise we can use simple Q-learning.

Main principle. The algorithm is based on the observation that the only Q-values needed at any given time are those for the possible actions given the current state. Hence, using "lazy learning", we can postpone updating Q-values until they are needed. Suppose some SAP (s, a) occurs at steps $t_1, t_2, t_3, \ldots$. Let us abbreviate $\eta^t = \eta^t(s, a)$, $\phi = \gamma\lambda$. First, we unfold terms of expression (1):

$$\sum_{t=1}^{k}\left[e'_t\eta^t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta^i\right]$$
$$= \sum_{t=1}^{t_1}\left[e'_t\eta^t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta^i\right] + \sum_{t=t_1+1}^{t_2}\left[e'_t\eta^t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta^i\right]$$
$$+ \sum_{t=t_2+1}^{t_3}\left[e'_t\eta^t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta^i\right] + \cdots$$

Since η^t is 1 only for $t = t_1, t_2, t_3, \ldots$ and 0 otherwise, we can rewrite this as

$$\begin{aligned}
&e'_{t_1} + e'_{t_2} + \sum_{t=t_1+1}^{t_2} e_t \phi^{t-t_1} + e'_{t_3} + \sum_{t=t_2+1}^{t_3} e_t(\phi^{t-t_1} + \phi^{t-t_2}) + \cdots \\
&\quad = e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}} \sum_{t=t_1+1}^{t_2} e_t \phi^t + e'_{t_3} + \left(\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}}\right) \sum_{t=t_2+1}^{t_3} e_t \phi^t + \cdots \\
&\quad = e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}} \left(\sum_{t=1}^{t_2} e_t \phi^t - \sum_{t=1}^{t_1} e_t \phi^t \right) + e'_{t_3} + \left(\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}}\right) \\
&\qquad \times \left(\sum_{t=1}^{t_3} e_t \phi^t - \sum_{t=1}^{t_2} e_t \phi^t \right) + \cdots
\end{aligned}$$

Defining $\Delta_t = \sum_{i=1}^{t} e_i \phi^i$, this becomes

$$e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}}(\Delta_{t_2} - \Delta_{t_1}) + e'_{t_3} + \left(\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}}\right)(\Delta_{t_3} - \Delta_{t_2}) + \cdots \tag{3}$$

This will allow for constructing an efficient online Q(λ) algorithm. We define a local trace $l'_t(s, a) = \sum_{i=1}^{t} \frac{\eta^i(s,a)}{\phi^i}$, and use (3) to write down the total update of $Q(s, a)$ during a trial:

$$\Delta Q(s, a) = \lim_{k \to \infty} \sum_{t=1}^{k} e'_t \eta^t(s, a) + l'_t(s, a)(\Delta_{t+1} - \Delta_t). \tag{4}$$

To exploit this we introduce a global variable Δ keeping track of the cumulative TD(λ)-error since the start of the trial. As long as SAP (s, a) does not occur we postpone updating $Q(s, a)$. In the update below we need to subtract that part of Δ which has already been used (see Eqs. (3) and (4)). We use for each SAP (s, a) a local variable $\delta(s, a)$ which records the value of Δ at the moment of the last update, and a local trace variable $l'(s, a)$. Then, once $Q(s, a)$ needs to be known, we update $Q(s, a)$ by adding $l'(s, a)(\Delta - \delta(s, a))$. Figure 1 illustrates that the algorithm substitutes the varying eligibility trace $l(s, a)$ by multiplying a global trace ϕ^t by the local trace $l'(s, a)$. The value of ϕ^t changes all the time, but $l'(s, a)$ does not in intervals during which (s, a) does not occur.

Algorithm overview. The algorithm relies on two procedures: the *Local update* procedure calculates exact Q-values once they are required; the *Global update* procedure updates the global variables and the current Q-value. Initially, we set the global variables $\phi^0 \leftarrow 1.0$ and $\Delta \leftarrow 0$. We also initialize the local variables $\delta(s, a) \leftarrow 0$ and $l'(s, a) \leftarrow 0$ for all SAPs.

Local updates. Q-values for all actions possible in a given state are updated before an action is selected and before a particular V-value is calculated. For each SAP (s, a) a variable $\delta(s, a)$ tracks changes since the last update:

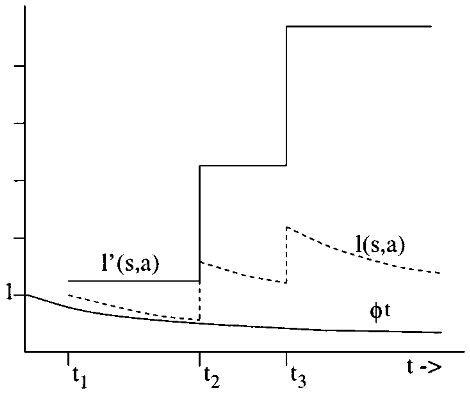

Figure 1. SAP (s, a) occurs at times $t_1, t_2, t_3, \ldots$. The standard eligibility trace $l(s, a)$ equals the product of ϕ^t and $l'(s, a)$.

```
**Local update** $(s_t, a_t)$ **:**
(1) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)(\Delta - \delta(s_t, a_t))l'(s_t, a_t)$
(2) $\delta(s_t, a_t) \leftarrow \Delta$
```

The global update procedure. After each executed action we invoke the procedure *Global update*, which consists of three basic steps: (1) To calculate $V(s_{t+1})$ (which may have changed due to the most recent experience), it calls *Local update* for the possible next SAPs; (2) it updates the global variables ϕ^t and Δ; and (3) it updates (s_t, a_t)'s Q-value and trace variable and stores the current Δ value (in *Local update*).

```
**Global update** $(s_t, a_t, r_t, s_{t+1})$ **:**
(1) $\forall a \in A$ Do
        (1a) Local update$(s_{t+1}, a)$
(2) $e'_t \leftarrow (r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$
(3) $e_t \leftarrow (r_t + \gamma V(s_{t+1}) - V(s_t))$
(4) $\phi^t \leftarrow \gamma\lambda\phi^{t-1}$
(5) $\Delta \leftarrow \Delta + e_t\phi^t$
(6) Local update$(s_t, a_t)$
(7) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)e'_t$
(8) $l'(s_t, a_t) \leftarrow l'(s_t, a_t) + 1/\phi^t$
```

For replacing eligibility traces (Singh & Sutton, 1996), step (8) should be changed as follows: $\forall a : l'(s_t, a) \leftarrow 0; l'(s_t, a_t) \leftarrow 1/\phi^t$.

Machine precision problem and solution. Adding $e_t\phi^t$ to Δ in line (5) may create a problem due to limited machine precision: for large absolute values of Δ and small ϕ^t

there may be significant rounding errors. More importantly, line (8) will quickly overflow any machine for $\gamma\lambda < 1$. The following addendum to the procedure *Global update* detects when ϕ^t falls below machine precision ϵ_m, updates all SAPs which have occurred (again we make use of a list H), and removes SAPs with $l'(s, a) < \epsilon_m$ from H. Finally, Δ and ϕ^t are reset to their initial values.

Global update : addendum

```
(9) If ($visited(s_t, a_t) = 0$)
        (9a) $H \leftarrow H \cup (s_t, a_t)$
        (9b) $visited(s_t, a_t) \leftarrow 1$
(10) If ($\phi^t < \epsilon_m$)
        (10a) Do $\forall (s, a) \in H$
                (10a-1) $Local\ update(s, a)$
                (10a-2) $l'(s, a) \leftarrow l'(s, a)\phi^t$
                (10a-3) If ($l'(s, a) < \epsilon_m$)
                        (10a-3-1) $H \leftarrow H \setminus (s, a)$
                        (10a-3-2) $visited(s, a) \leftarrow 0$
                (10a-4) $\delta(s, a) \leftarrow 0$
        (10b) $\Delta \leftarrow 0$
        (10c) $\phi^t \leftarrow 1.0$
```

Comments. Recall that *Local update* sets $\delta(s, a) \leftarrow \Delta$, and update steps depend on $\Delta - \delta(s, a)$. Thus, after having updated all SAPs in H, we can set $\Delta \leftarrow 0$ and $\delta(s, a) \leftarrow 0$. Furthermore, we can simply set $l'(s, a) \leftarrow l'(s, a)\phi^t$ and $\phi^t \leftarrow 1.0$ without affecting the expression $l'(s, a)\phi^t$ used in future updates—this just rescales the variables. Note that if $\gamma\lambda = 1$, then no sweeps through the history list will be necessary.

Complexity. The algorithm's most expensive part are the calls of *Local update*, whose total cost is $O(|A|)$. This is not bad: even simple Q-learning's action selection procedure costs $O(|A|)$ if, say, the Boltzmann rule (Thrun, 1992; Caironi & Dorigo, 1994) is used. Concerning the occasional complete sweep through SAPs still in history list H: during each sweep the traces of SAPs in H are multiplied by $l < e_m$. SAPs are deleted from H once their trace falls below e_m. In the worst case one sweep per n time steps updates $2n$ SAPs and costs $O(1)$ on average. This means that there is an additional computational burden at certain time steps, but since this happens infrequently our method's total average update complexity stays $O(|A|)$.

The space complexity of the algorithm remains $O(|S||A|)$. We need to store the following variables for all SAPs: Q-values, eligibility traces, previous delta values, the "visited" bit, and three pointers to manage the history list (one from each SAP to its place in the history list, and two for the doubly linked list). Finally, we need to store the two global variables.

Comparison to PW. Figure 2 illustrates differences between both methods for $|A| = 5$, $|S| = 1000$, and $\gamma = 1$. We plot the number of updates against time for $\lambda \in \{0.7, 0.9, 0.99\}$. The plots refer to worst-case behavior: we assume that at each time step a new SAP is added

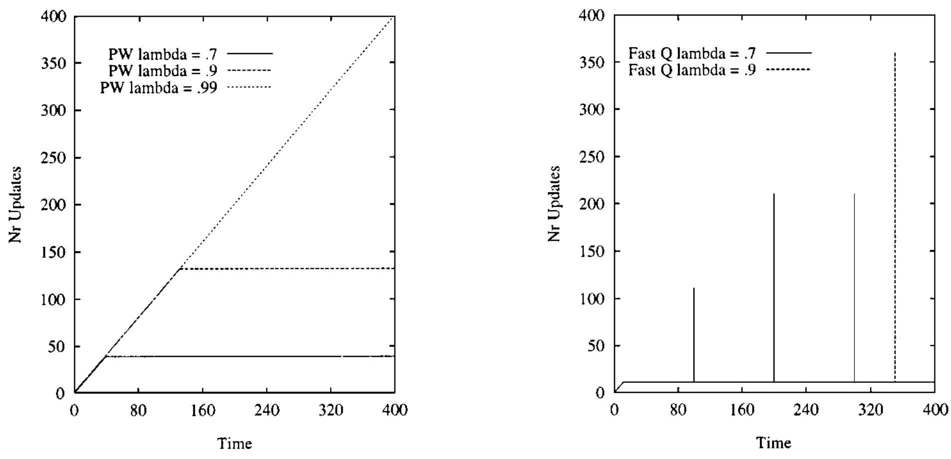

Figure 2. Number of updates plotted against time: a worst case analysis comparing our method (right) and Peng and Williams' (left) for different values of λ. The occasional spikes (right) hardly affect our method's average performance, which is very close to the horizontal line.

to the history list. The accuracy parameter ϵ (used in PW) is set to 10^{-6} (in practice, less precise values may be used, but this will not change matters much). The machine precision parameter ϵ_m is set to 10^{-16}. Plot 2(A) shows that PW's update costs increase until the history lists have reached their maximum size, although for $\lambda = 0.99$ this does not happen before time step 1375. The spikes in the fast Q(λ) plot reflect occasional full sweeps through the history list due to limited machine precision (the corresponding average number of updates, however, is very close to the value indicated by the horizontal solid line—as explained above, the spikes hardly affect the average). No sweep is necessary in fast Q(0.99)'s plot during the shown interval. Fast Q needs on average a total of 13 update steps: 5 in choose-action, 5 for calculating $V(s_{t+1})$, 1 for updating the chosen action, and 2 for taking into account the full sweeps. See Wiering & Schmidhuber (1998) for illustrative experiments with maze tasks.

Extension to function approximators. Tabular representations do not allow for generalizing from previous experiences, which is necessary in case of large state spaces. Fast Q(λ), however, can also improve matters in case of "local" function approximators (FAs) consisting of separate building blocks, such as Kohonen networks (Kohonen, 1988), CMACs (Albus, 1975; Sutton, 1996), locally weighted learning (Atkeson, Moore, & Schaal, 1996), and neural gas (Fritzke, 1994). Such FAs work as follows: there are $|S|$ possible state space "features". There are Q-values for all possible feature/action pairs, just like there are Q-values of state/action pairs in the case of tabular representation. Q-values of $I \leq |S|$ features are combined to evaluate an input (query). Here the update complexity of fast Q(λ) equals $O(I|A|)$. This results in a speed-up of $|S|/I$ in comparison to PW's method.

Our method can also be used in conjunction with arbitrary gradient-based FAs. In case of global FAs, such as monolithic neural networks, it will be useless due to $I = |S|$. Consider, however, a combination of CMACs and neural network FAs. Only few neural networks are used at any given time, hence our method will be useful: in *Global update* we simply add to each weight trace the gradient of the combined Q-values (the weighted sum of all used network outputs) with respect to the used weight. More formally: in the lookup-table case we used $l'(s, a) := l'(s, a) + 1/\phi^t$ (in line (8) of *Global update*). Now we replace this by $l'(w_i) := l'(w_i) + \frac{\partial Q(s,a)/\partial w_i}{\phi^t}$, where $l'(w_i)$ is the fast-eligibility trace of weight w_i. The speed-up equals the average fraction of weights used per time step, a measure of the inference process' degree of locality.

Multiple trials. We have described a single-trial version of our algorithm. One might be tempted to think that in case of multiple trials all SAPs in the history list need to be updated and all eligibility traces reset after each trial. This is not necessary—we may use cross-trial learning as follows:

We introduce Δ^M variables, where index M stands for the Mth trial. Let N denote the current trial number, and let variable *visited*(s, a) represent the trial number of the most recent occurrence of SAP (s, a). Now we slightly change *Local update*:

```
Local update($s_t, a_t$) :
(1) $M \leftarrow visited(s_t, a_t)$
(2) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)(\Delta^M - \delta(s_t, a_t))l'(s_t, a_t)$
(3) $\delta(s_t, a_t) \leftarrow \Delta^N$
(4) If ($M < N$)
        (4a) $l'(s_t, a_t) \leftarrow 0$
        (4b) $visited(s_t, a_t) \leftarrow N$
```

Thus we update $Q(s, a)$ using the value Δ^M of the most recent trial M during which SAP (s, a) occurred and the corresponding values of $\delta(s_t, a_t)$ and $l'(s_t, a_t)$ (computed during the same trial). In case SAP (s, a) has not occurred during the current trial we reset the eligibility trace and set *visited* (s, a) to the current trial number. In *Global update* we need to change lines (5) and (10b) by adding trial subscripts to Δ, and we need to change line (9b) in which we have to set *visited* $(s_t, a_t) \leftarrow N$. At trial end we reset ϕ^t to $\phi^0 = 1.0$, increment the trial counter N, and set $\Delta^N \leftarrow 0$. This allows for postponing certain updates until after the current trial's end.

4. Conclusion

While other Q(λ) approaches are either offline, inexact, or may suffer from average update complexity depending on the size of the state/action space, ours is fully online Q(λ) with average update complexity linear in the number of actions. Efficiently dealing with eligibility traces makes fast Q(λ) applicable to larger scale RL problems.

Acknowledgments

Thanks to Nic Schraudolph for helpful comments. This work was supported in part by SNF grant 2100-49'144.96 "Long Short-Term Memory".

References

Albus, J.S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Dynamic Systems, Measurement and Control*, *97*, 220–227.

Atkeson, C.G., Schaal, S., & Moore, A.W. (1997). Locally weighted learning. *Artificial Intelligence Review*, *11*, 11–73.

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

Bertsekas, D.P., & Tsitsiklis, J.N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Caironi, P.V.C., & Dorigo, M. (1994). *Training Q-agents* (Technical Report IRIDIA-94-14). Université Libre de Bruxelles.

Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *Journal of Artificial Intelligence Research*, *2*, 287–318.

Fritzke, B. (1994). Supervised learning with growing cell structures. In J. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems* (Vol.6, pp. 255–262). San Mateo, CA: Morgan Kaufmann.

Koenig, S., & Simmons, R.G. (1996). The effect of representation and knowledge on goal-directed exploration with reinforcement learning algorithms. *Machine Learning*, *22*, 228–250.

Kohonen, T. (1988). *Self-organization and associative memory* (2nd ed.). Springer.

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh.

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). UK: Cambridge University.

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

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

Sutton, R.S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D.S. Touretzky, M.C. Mozer, & M.E. Hasselmo (Eds.), *Advances in neural information processing systems*, (Vol. 8, pp. 1033–1045). Cambridge, MA: MIT Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. In D.S., Lippman, J.E. Moody, & D.S Touretzky (Eds.), *Advances in neural information processing systems* (Vol. 4, pp. 259–266). San Mateo, CA: Morgan Kaufmann.

Thrun, S. (1992). Efficient exploration in reinforcement learning (Technical Report CMU-CS-92-102). Carnegie-Mellon University.

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

Watkins, C.J.C.H., & Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, *8*, 279–292.

Whitehead, S. (1992). *Reinforcement learning for the adaptive control of perception and action*. Ph.D. thesis, University of Rochester.

Wiering, M.A., & Schmidhuber, J. (1998). Speeding up Q(λ)-learning. In C. Nedellec, & C. Rouveirol (Eds.), *Machine Learning: Proceedings of the Tenth European Conference*. Berlin: Springer Verlag.

Received July 9, 1997
Accepted March 4, 1998
Final Manuscript April 9, 1998