Temporal Difference Updating without a Learning Rate

Marcus Hutter
RSISE@ANU and SML@NICTA
Canberra, ACT, 0200, Australia
marcus@hutter1.net www.hutter1.net

Shane Legg
IDSIA, Galleria 2, Manno-Lugano CH-6928, Switzerland
shane@vetta.org www.vetta.org/shane

Abstract

We derive an equation for temporal difference learning from statistical principles. Specifically, we start with the variational principle and then bootstrap to produce an updating rule for discounted state value estimates. The resulting equation is similar to the standard equation for temporal difference learning with eligibility traces, so called TD(λ), however it lacks the parameter α that specifies the learning rate. In the place of this free parameter there is now an equation for the learning rate that is specific to each state transition. We experimentally test this new learning rule against TD(λ) and find that it offers superior performance in various settings. Finally, we make some preliminary investigations into how to extend our new temporal difference algorithm to reinforcement learning. To do this we combine our update equation with both Watkins' Q(λ) and Sarsa(λ) and find that it again offers superior performance without a learning rate parameter.

1 Introduction

In the field of reinforcement learning, perhaps the most popular way to estimate the future discounted reward of states is the method of *temporal difference learning*. It is unclear who exactly introduced this first, however the first explicit version of temporal difference as a learning rule appears to be Witten [8]. The idea is as follows: The *expected future discounted reward* of a state s is,

$$\overline{V}_s := \mathbf{E}\left\{r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + \cdots | s_k = s\right\},$$

where the rewards $r_k, r_{k+1}, \ldots$ are geometrically discounted into the future by $\gamma < 1$. From this definition it follows that,

$$\overline{V}_s = \mathbf{E}\left\{r_k + \gamma \overline{V}_{s_{k+1}} | s_k = s\right\}. \tag{1}$$

Our task, at time t, is to compute an estimate V_s^t of $\overline{V}_s$ for each state s. The only information we have to base this estimate on is the current history of state transitions, $s_1, s_2, \ldots, s_t$, and the current history of observed rewards, $r_1, r_2, \ldots, r_t$. Equation (1) suggests that at time $t+1$ the value of $r_t + \gamma V_{s_{t+1}}$ provides us with information on what V_s^t should be: If it is higher than $V_{s_t}^t$ then perhaps this estimate should be increased, and vice versa. This intuition gives us the following estimation heuristic for state s_t,

$$V_{s_t}^{t+1} := V_{s_t}^t + \alpha\left(r_t + \gamma V_{s_{t+1}}^t - V_{s_t}^t\right),$$

where α is a parameter that controls the rate of learning. This type of temporal difference learning is known as TD(0).

One shortcoming of this method is that at each time step the value of only the last state s_t is updated. States before the last state are also affected by changes in the last state's value and thus these could be updated too. This is what happens with so called *temporal difference learning with eligibility traces*, where a history, or trace, is kept of which states have been recently visited. Under this method, when we update the value of a state we also go back through the trace updating the earlier states as well. Formally, for any state s its eligibility trace is computed by,

$$E_s^t := \begin{cases} \gamma\lambda E_s^{t-1} & \text{if } s \neq s_t, \\ \gamma\lambda E_s^{t-1} + 1 & \text{if } s = s_t, \end{cases}$$

where λ is used to control the rate at which the eligibility trace is discounted. The temporal difference update is then, for all states s,

$$V_s^{t+1} := V_s^t + \alpha E_s^t \left(r + \gamma V_{s_{t+1}}^t - V_{s_t}^t \right). \tag{2}$$

This more powerful version of temporal different learning is known as TD(λ) [6].

The main idea of this paper is to derive a temporal difference rule from statistical principles and compare it to the standard heuristic described above. Superficially, our work has some similarities to LSTD(λ) ([**?**] and references therein). However LSTD is concerned with finding a least-squares linear function approximation, it has not yet been developed for general γ *and* λ, and has update time quadratic in the number of features/states. On the other hand, our algorithm "exactly" coincides with TD/Q/Sarsa(λ) for finite state spaces, but with a novel learning rate derived from statistical principles. We therefore focus our comparison on TD/Q/Sarsa. For a recent survey of methods to set the learning rate see [**?**].

In *Section 2* we derive a least squares estimate for the value function. By expressing the estimate as an incremental update rule we obtain a new form of TD(λ), which we call HL(λ). In *Section 3* we compare HL(λ) to TD(λ) on a simple Markov chain. We then test it on a random Markov chain in *Section 4* and a non-stationary environment in *Section 5*. In *Section 6* we derive two new methods for policy learning based on HL(λ), and compare them to Sarsa(λ) and Watkins' Q(λ) on a simple reinforcement learning problem. *Section 7* ends the paper with a summary and some thoughts on future research directions.

2 Derivation

The *empirical future discounted reward* of a state s_k is the sum of *actual rewards* following from state s_k in time steps $k, k+1, \ldots$, where the rewards are discounted as they go into the future. Formally, the empirical value of state s_k at time k for $k = 1, ..., t$ is,

$$v_k := \sum_{u=k}^{\infty} \gamma^{u-k} r_u, \tag{3}$$

where the future rewards r_u are geometrically discounted by $\gamma < 1$. In practice the exact value of v_k is always unknown to us as it depends not only on rewards that have been already observed, but also on unknown future rewards. Note that if $s_m = s_n$ for $m \neq n$, that is, we have visited the same state twice at different times m and n, this does not imply that $v_n = v_m$ as the observed rewards following the state visit may be different each time.

Our goal is that for each state s the estimate V_s^t should be as close as possible to the true expected future discounted reward $\overline{V}_s$. Thus, for each state s we would like V_s to be close to v_k for all k such that $s = s_k$. Furthermore, in non-stationary environments we would like to discount old evidence by some parameter $\lambda \in (0, 1]$. Formally, we want to minimise the loss function,

$$L := \frac{1}{2} \sum_{k=1}^{t} \lambda^{t-k} \left(v_k - V_{s_k}^t \right)^2. \tag{4}$$

For stationary environments we may simply set $\lambda = 1$ a priori.

As we wish to minimise this loss, we take the partial derivative with respect to the value estimate of each state and set to zero,

$$\frac{\partial L}{\partial V_s^t} = -\sum_{k=1}^{t} \lambda^{t-k} \left(v_k - V_{s_k}^t \right) \delta_{s_k s} = V_s^t \sum_{k=1}^{t} \lambda^{t-k} \delta_{s_k s} - \sum_{k=1}^{t} \lambda^{t-k} \delta_{s_k s} v_k = 0,$$

where we could change $V_{s_k}^t$ into V_s^t due to the presence of the Kronecker $\delta_{s_k s}$, defined $\delta_{xy} := 1$ if $x = y$, and 0 otherwise. By defining a discounted state visit counter $N_s^t := \sum_{k=1}^t \lambda^{t-k}\delta_{s_k s}$ we get

$$V_s^t N_s^t = \sum_{k=1}^{t} \lambda^{t-k}\delta_{s_k s} v_k. \tag{5}$$

Since v_k depends on future rewards r_k, Equation (5) can not be used in its current form. Next we note that v_k has a self-consistency property with respect to the rewards. Specifically, the tail of the future discounted reward sum for each state depends on the empirical value at time t in the following way,

$$v_k = \sum_{u=k}^{t-1} \gamma^{u-k} r_u + \gamma^{t-k} v_t.$$

Substituting this into Equation (5) and exchanging the order of the double sum,

$$\begin{aligned} V_s^t N_s^t &= \sum_{u=1}^{t-1}\sum_{k=1}^{u} \lambda^{t-k}\delta_{s_k s}\gamma^{u-k} r_u + \sum_{k=1}^{t} \lambda^{t-k}\delta_{s_k s}\gamma^{t-k} v_t \\ &= \sum_{u=1}^{t-1} \lambda^{t-u} \sum_{k=1}^{u} (\lambda\gamma)^{u-k}\delta_{s_k s} r_u + \sum_{k=1}^{t} (\lambda\gamma)^{t-k}\delta_{s_k s} v_t \\ &= R_s^t + E_s^t v_t, \end{aligned}$$

where $E_s^t := \sum_{k=1}^t (\lambda\gamma)^{t-k}\delta_{s_k s}$ is the eligibility trace of state s, and $R_s^t := \sum_{u=1}^{t-1} \lambda^{t-u} E_s^u r_u$ is the discounted reward with eligibility.

E_s^t and R_s^t depend only on quantities known at time t. The only unknown quantity is v_t, which we have to replace with our current estimate of this value at time t, which is $V_{s_t}^t$. In other words, we bootstrap our estimates. This gives us,

$$V_s^t N_s^t = R_s^t + E_s^t V_{s_t}^t. \tag{6}$$

For state $s = s_t$, this simplifies to $V_{s_t}^t = R_{s_t}^t/(N_{s_t}^t - E_{s_t}^t)$. Substituting this back into Equation (6) we obtain,

$$V_s^t N_s^t = R_s^t + E_s^t \frac{R_{s_t}^t}{N_{s_t}^t - E_{s_t}^t}. \tag{7}$$

This gives us an explicit expression for our V estimates. However, from an algorithmic perspective an incremental update rule is more convenient. To derive this we make use of the relations,

$$N_s^{t+1} = \lambda N_s^t + \delta_{s_{t+1}s}, \qquad E_s^{t+1} = \lambda\gamma E_s^t + \delta_{s_{t+1}s}, \qquad R_s^{t+1} = \lambda R_s^t + \lambda E_s^t r_t,$$

with $N_s^0 = E_s^0 = R_s^0 = 0$. Inserting these into Equation (7) with t replaced by $t+1$,

$$\begin{aligned} V_s^{t+1} N_s^{t+1} &= R_s^{t+1} + E_s^{t+1} \frac{R_{s_{t+1}}^{t+1}}{N_{s_{t+1}}^{t+1} - E_{s_{t+1}}^{t+1}} \\ &= \lambda R_s^t + \lambda E_s^t r_t + E_s^{t+1} \frac{R_{s_{t+1}}^t + E_{s_{t+1}}^t r_t}{N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t}. \end{aligned}$$

By solving Equation (6) for R_s^t and substituting back in,

$$\begin{aligned} V_s^{t+1} N_s^{t+1} &= \lambda\big(V_s^t N_s^t - E_s^t V_{s_t}^t\big) + \lambda E_s^t r_t + E_s^{t+1} \frac{N_{s_{t+1}}^t V_{s_{t+1}}^t - E_{s_{t+1}}^t V_{s_t}^t + E_{s_{t+1}}^t r_t}{N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t} \\ &= \big(\lambda N_s^t + \delta_{s_{t+1}s}\big) V_s^t - \delta_{s_{t+1}s} V_s^t - \lambda E_s^t V_{s_t}^t + \lambda E_s^t r_t \\ &\quad + E_s^{t+1} \frac{N_{s_{t+1}}^t V_{s_{t+1}}^t - E_{s_{t+1}}^t V_{s_t}^t + E_{s_{t+1}}^t r_t}{N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t}. \end{aligned}$$

Dividing through by $N_s^{t+1} (= \lambda N_s^t + \delta_{s_{t+1}s})$,

$$V_s^{t+1} = V_s^t + \frac{-\delta_{s_{t+1}s} V_s^t - \lambda E_s^t V_{s_t}^t + \lambda E_s^t r_t}{\lambda N_s^t + \delta_{s_{t+1}s}}$$

$$+\frac{(\lambda\gamma E_s^t + \delta_{s_{t+1}s})(N_{s_{t+1}}^t V_{s_{t+1}}^t - E_{s_{t+1}}^t V_{s_t}^t + E_{s_{t+1}}^t r_t)}{(N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t)(\lambda N_s^t + \delta_{s_{t+1}s})}.$$

Making the first denominator the same as the second, then expanding the numerator,

$$\begin{aligned}
V_s^{t+1} = V_s^t &+ \frac{\lambda E_s^t r_t N_{s_{t+1}}^t - \lambda E_s^t V_{s_t}^t N_{s_{t+1}}^t - \delta_{s_{t+1}s} V_s^t N_{s_{t+1}}^t - \lambda\gamma E_{s_{t+1}}^t E_s^t r_t}{(N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t)(\lambda N_s^t + \delta_{s_{t+1}s})} \\
&+ \frac{\lambda\gamma E_{s_{t+1}}^t E_s^t V_{s_t}^t + \gamma E_{s_{t+1}}^t V_s^t \delta_{s_{t+1}s} + \lambda\gamma E_s^t N_{s_{t+1}}^t V_{s_{t+1}}^t - \lambda\gamma E_s^t E_{s_{t+1}}^t V_{s_t}^t}{(N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t)(\lambda N_s^t + \delta_{s_{t+1}s})} \\
&+ \frac{\lambda\gamma E_s^t E_{s_{t+1}}^t r_t + \delta_{s_{t+1}s} N_{s_{t+1}}^t V_{s_{t+1}}^t - \delta_{s_{t+1}s} E_{s_{t+1}}^t V_{s_t}^t + \delta_{s_{t+1}s} E_{s_{t+1}}^t r_t}{(N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t)(\lambda N_s^t + \delta_{s_{t+1}s})}.
\end{aligned}$$

After cancelling equal terms (keeping in mind that in every term with a Kronecker δ_{xy} factor we may assume that $x = y$ as the term is always zero otherwise), and factoring out E_s^t we obtain,

$$V_s^{t+1} = V_s^t + \frac{E_s^t \big(\lambda r_t N_{s_{t+1}}^t - \lambda V_{s_t}^t N_{s_{t+1}}^t + \gamma V_s^t \delta_{s_{t+1}s} + \lambda\gamma N_{s_{t+1}}^t V_{s_{t+1}}^t - \delta_{s_{t+1}s} V_{s_t}^t + \delta_{s_{t+1}s} r_t\big)}{(N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t)(\lambda N_s^t + \delta_{s_{t+1}s})}$$

Finally, by factoring out $\lambda N_{s_{t+1}}^t + \delta_{s_{t+1}s}$ we obtain our update rule,

$$V_s^{t+1} = V_s^t \;+\; E_s^t \,\beta_t(s, s_{t+1}) \left(r_t + \gamma V_{s_{t+1}}^t - V_{s_t}^t\right), \tag{8}$$

where the learning rate is given by,

$$\beta_t(s, s_{t+1}) \;:=\; \frac{1}{N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t} \; \frac{N_{s_{t+1}}^t}{N_s^t}. \tag{9}$$

Examining Equation (8), we find the usual update equation for temporal difference learning with eligibility traces (see Equation (2)), however the learning rate α has now been replaced by $\beta_t(s, s_{t+1})$. This learning rate was derived from statistical principles by minimising the squared loss between the estimated and true state value. In the derivation we have exploited the fact that the latter must be self-consistent and then bootstrapped to get Equation (6). This gives us an equation for the learning rate for each state transition at time t, as opposed to the standard temporal difference learning where the learning rate α is either a fixed free parameter for all transitions, or is decreased over time by some monotonically decreasing function. In either case, the learning rate is not automatic and must be experimentally tuned for good performance. The above derivation appears to theoretically solve this problem.

The first term in β_t seems to provide some type of normalisation to the learning rate, though the intuition behind this is not clear to us. The meaning of second term however can be understood as follows: N_s^t measures how often we have visited state s in the recent past. Therefore, if $N_s^t \ll N_{s_{t+1}}^t$ then state s has a value estimate based on relatively few samples, while state s_{t+1} has a value estimate based on relatively many samples. In such a situation, the second term in β_t boosts the learning rate so that V_s^{t+1} moves more aggressively towards the presumably more accurate $r_t + \gamma V_{s_{t+1}}^t$. In the opposite situation when s_{t+1} is a less visited state, we see that the reverse occurs and the learning rate is reduced in order to maintain the existing value of V_s.

3 A simple Markov process

For our first test we consider a simple Markov process with 51 states. In each step the state number is either incremented or decremented by one with equal probability, unless the system is in state 0 or 50 in which case it always transitions to state 25 in the following step. When the state transitions from 0 to 25 a reward of 1.0 is generated, and for a transition from 50 to 25 a reward of -1.0 is generated. All other transitions have a reward of 0. We set the discount value $\gamma = 0.99$ and then computed the true discounted value of each state by running a brute force Monte Carlo simulation.

We ran our algorithm 10 times on the above Markov chain and computed the root mean squared error in the value estimate across the states at each time step averaged across each run. The optimal

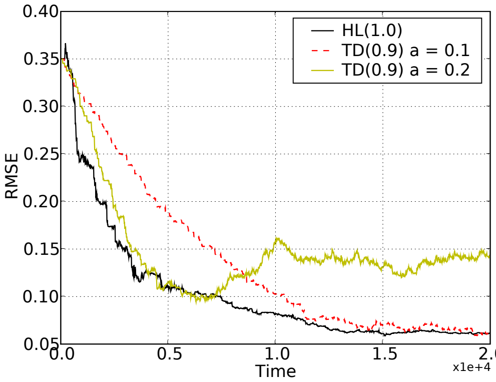

Figure 1: 51 state Markov process averaged over 10 runs. The parameter a is the learning rate α.

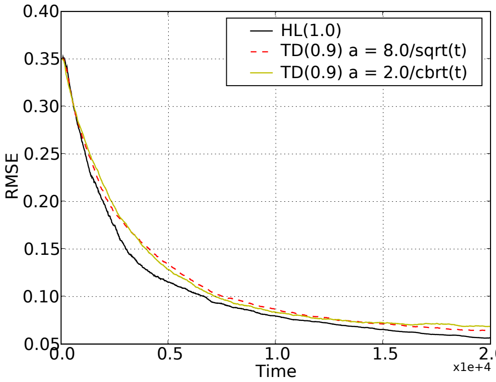

Figure 2: 51 state Markov process averaged over 300 runs.

value of λ for HL(λ) was 1.0, which was to be expected given that the environment is stationary and thus discounting old experience is not helpful.

For TD(λ) we tried various different learning rates and values of λ. We could find no settings where TD(λ) was competitive with HL(λ). If the learning rate α was set too high the system would learn as fast as HL(λ) briefly before becoming stuck. With a lower learning rate the final performance was improved, however the initial performance was now much worse than HL(λ). The results of these tests appear in Figure 1.

Similar tests were performed with larger and smaller Markov chains, and with different values of γ. HL(λ) was consistently superior to TD(λ) across these tests. One wonders whether this may be due to the fact that the implicit learning rate that HL(λ) uses is not fixed. To test this we explored the performance of a number of different learning rate functions on the 51 state Markov chain described above. We found that functions of the form $\frac{\kappa}{t}$ always performed poorly, however good performance was possible by setting κ correctly for functions of the form $\frac{\kappa}{\sqrt{t}}$ and $\frac{\kappa}{\sqrt[3]{t}}$. As the results were much closer, we averaged over 300 runs. These results appear in Figure 2.

With a variable learning rate TD(λ) is performing much better, however we were still unable to find an equation that reduced the learning rate in such a way that TD(λ) would outperform HL(λ). This is evidence that HL(λ) is adapting the learning rate optimally without the need for manual equation tuning.

4 Random Markov process

To test on a Markov process with a more complex transition structure, we created a random 50 state Markov process. We did this by creating a 50 by 50 transition matrix where each element was set to 0 with probability 0.9, and a uniformly random number in the interval $[0, 1]$ otherwise. We then scaled each row to sum to 1. Then to transition between states we interpreted the i^{th} row as a probability distribution over which state follows state i. To compute the reward associated with each transition we created a random matrix as above, but without normalising. We set $\gamma = 0.9$ and then ran a brute force Monte Carlo simulation to compute the true discounted value of each state.

The λ parameter for HL(λ) was simply set to 1.0 as the environment is stationary. For TD we experimented with a range of parameter settings and learning rate decrease functions. We found that a fixed learning rate of $\alpha = 0.2$, and a decreasing rate of $\frac{1.5}{\sqrt[3]{t}}$ performed reasonable well, but never as well as HL(λ). The results were generated by averaging over 10 runs, and are shown in Figure 3.

Although the structure of this Markov process is quite different to that used in the previous experiment, the results are again similar: HL(λ) preforms as well or better than TD(λ) from the beginning to the end of the run. Furthermore, stability in the error towards the end of the run is better with HL(λ) and no manual learning tuning was required for these performance gains.

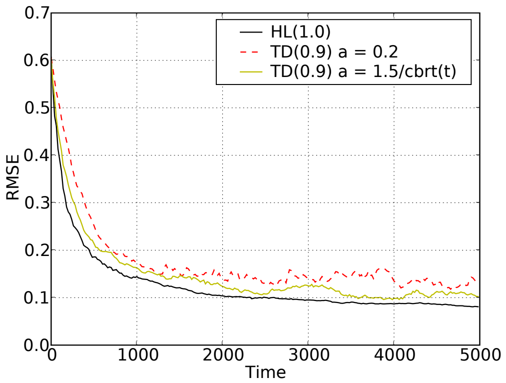

Figure 3: Random 50 state Markov process. The parameter a is the learning rate α.

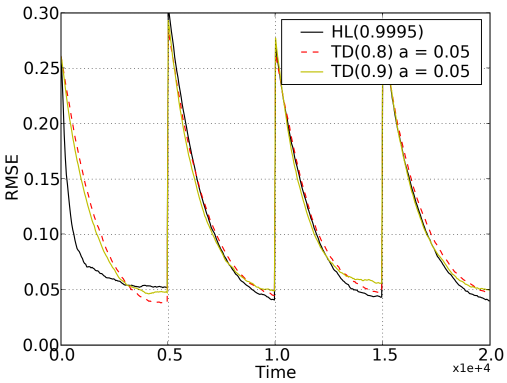

Figure 4: 21 state non-stationary Markov process.

5 Non-stationary Markov process

The λ parameter in HL(λ), introduced in Equation (4), reduces the importance of old observations when computing the state value estimates. When the environment is stationary this is not useful and so we can set $\lambda = 1.0$, however in a non-stationary environment we need to reduce this value so that the state values adapt properly to changes in the environment. The more rapidly the environment is changing, the lower we need to make λ in order to more rapidly forget old observations.

To test HL(λ) in such a setting, we used the Markov chain from Section 3, but reduced its size to 21 states to speed up convergence. We used this Markov chain for the first 5,000 time steps. At that point, we changed the reward when transitioning from the last state to middle state to from -1.0 to be 0.5. At time 10,000 we then switched back to the original Markov chain, and so on alternating between the models of the environment every 5,000 steps. At each switch, we also changed the target state values that the algorithm was trying to estimate to match the current configuration of the environment. For this experiment we set $\gamma = 0.9$.

As expected, the optimal value of λ for HL(λ) fell from 1 down to about 0.9995. This is about what we would expect given that each phase is 5,000 steps long. For TD(λ) the optimal value of λ was around 0.8 and the optimum learning rate was around 0.05. As we would expect, for both algorithms when we pushed λ above its optimal value this caused poor performance in the periods following each switch in the environment (these bad parameter settings are not shown in the results). On the other hand, setting λ too low produced initially fast adaption to each environment switch, but poor performance after that until the next environment change. To get accurate statistics we averaged over 200 runs. The results of these tests appear in Figure 4.

For some reason HL(0.9995) learns faster than TD(0.8) in the first half of the first cycle, but only equally fast at the start of each following cycle. We are not sure why this is happening. We could improve the initial speed at which HL(λ) learnt in the last three cycles by reducing λ, however that comes at a performance cost in terms of the lowest mean squared error attained at the end of each cycle. In any case, in this non-stationary situation HL(λ) again performed well.

6 Windy Gridworld

Reinforcement learning algorithms such as Watkins' Q(λ) [7] and Sarsa(λ) [4, 3] are based on temporal difference updates. This suggests that new reinforcement learning algorithms based on HL(λ) should be possible.

For our first experiment we took the standard Sarsa(λ) algorithm and modified it in the obvious way to use an HL temporal difference update. In the presentation of this algorithm we have changed notation slightly to make things more consistent with that typical in reinforcement learning. Specifically, we have dropped the t super script as this is implicit in the algorithm specification, and have

Algorithm 1 HLS(λ)

Initialise $Q(s,a) = 0$, $N(s,a) = 1$ and $E(s,a) = 0$ for all s, a
Initialise s and a
repeat
  Take action a, observed r, s'
  Choose a' by using ϵ-greedy selection on $Q(s', \cdot)$
  $\Delta \leftarrow r + \gamma Q(s', a') - Q(s,a)$
  $E(s,a) \leftarrow E(s,a) + 1$
  $N(s,a) \leftarrow N(s,a) + 1$
  for all s, a **do**
    $\beta((s,a),(s',a')) \leftarrow \frac{1}{N(s',a') - \gamma E(s',a')} \frac{N(s',a')}{N(s,a)}$
  end for
  for all s, a **do**
    $Q(s,a) \leftarrow Q(s,a) + \beta\big((s,a),(s',a')\big) E(s,a) \Delta$
    $E(s,a) \leftarrow \gamma\lambda E(s,a)$
    $N(s,a) \leftarrow \lambda N(s,a)$
  end for
  $s \leftarrow s'; a \leftarrow a'$
until end of run

defined $Q(s,a) := V_{(s,a)}$, $E(s,a) := E_{(s,a)}$ and $N(s,a) := N_{(s,a)}$. Our new reinforcement learning algorithm, which we call HLS(λ) is given in Algorithm 1. Essentially the only changes to the standard Sarsa(λ) algorithm have been to add code to compute the visit counter $N(s,a)$, add a loop to compute the β values, and replace α with β in the temporal difference update.

To test HLS(λ) against standard Sarsa(λ) we used the Windy Gridworld environment described on page 146 of [5]. This world is a grid of 7 by 10 squares that the agent can move through by going either up, down, left of right. If the agent attempts to move off the grid it simply stays where it is. The agent starts in the 4^{th} row of the 1^{st} column and receives a reward of 1 when it finds its way to the 4^{th} row of the 8^{th} column. To make things more difficult, there is a "wind" blowing the agent up 1 row in columns 4, 5, 6, and 9, and a strong wind of 2 in columns 7 and 8. This is illustrated in Figure 5. Unlike in the original version, we have set up this problem to be a continuing discounted task with an automatic transition from the goal state back to the start state.

We set $\gamma = 0.99$ and in each run computed the empirical future discounted reward at each point in time. As this value oscillated we also ran a moving average through these values with a window length of 50. Each run lasted for 50,000 time steps as this allowed us to see at what level each learning algorithm topped out. These results appear in Figure 6 and were averaged over 500 runs to get accurate statistics.

Despite putting considerable effort into tuning the parameters of Sarsa(λ), we were unable to achieve a final future discounted reward above 5.0. The settings shown on the graph represent the best final value we could achieve. In comparison HLS(λ) easily beat this result at the end of the run, while being slightly slower than Sarsa(λ) at the start. By setting $\lambda = 0.99$ we were able to achieve the same performance as Sarsa(λ) at the start of the run, however the performance at the end of the run was then only slightly better than Sarsa(λ). This combination of superior performance and fewer parameters to tune suggest that the benefits of HL(λ) carry over into the reinforcement learning setting.

Another popular reinforcement learning algorithm is Watkins' Q(λ). Similar to Sarsa(λ) above, we simply inserted the HL(λ) temporal difference update into the usual Q(λ) algorithm in the obvious way. We call this new algorithm HLQ(λ)(not shown). The test environment was exactly the same as we used with Sarsa(λ) above.

The results this time were more competitive (these results are not shown). Nevertheless, despite spending a considerable amount of time fine tuning the parameters of Q(λ), we were unable to beat HLQ(λ). As the performance advantage was relatively modest, the main benefit of HLQ(λ) was that it achieved this level of performance without having to tune a learning rate.

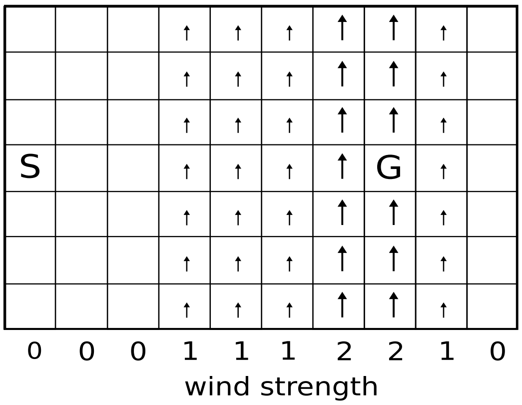

Figure 5: [Windy Gridworld] S marks the start state and G the goal state, at which the agent jumps back to S with a reward of 1.

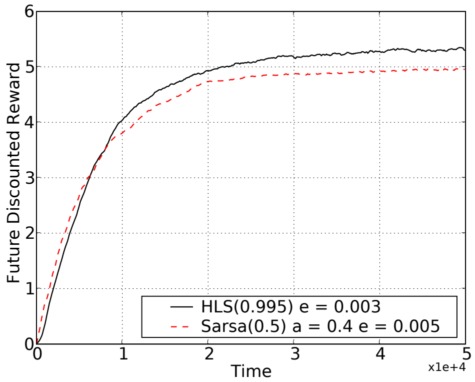

Figure 6: Sarsa(λ) vs. HLS(λ) in the Windy Gridworld. Performance averaged over 500 runs. On the graph, e represents the exploration parameter ϵ, and a the learning rate α.

7 Conclusions

We have derived a new equation for setting the learning rate in temporal difference learning with eligibility traces. The equation replaces the free learning rate parameter α, which is normally experimentally tuned by hand. In every setting tested, be it stationary Markov chains, non-stationary Markov chains or reinforcement learning, our new method produced superior results.

To further our theoretical understanding, the next step would be to try to prove that the method converges to correct estimates. This can be done for TD(λ) under certain assumptions on how the learning rate decreases over time. Hopefully, something similar can be proven for our new method. In terms of experimental results, it would be interesting to try different types of reinforcement learning problems and to more clearly identify where the ability to set the learning rate differently for different state transition pairs helps performance. It would also be good to generalise the result to episodic tasks. Finally, just as we have successfully merged HL(λ) with Sarsa(λ) and Watkins' Q(λ), we would also like to see if the same can be done with Peng's Q(λ) [2], and perhaps other reinforcement learning algorithms.

Acknowledgements

This research was funded by the Swiss NSF grant 200020-107616.

References

[1] J. Peng and R. J. Williams. Increamental multi-step Q-learning. In W. W. Cohen and H. Hirsh, editors, *Proc. of the Eleventh International Conference of Machine Learning*, pages 226–232, San Franscisco, 1994. Morgan Kaufmann.

[2] J. Peng and R. J. Williams. Increamental multi-step Q-learning. *Machine Learning*, 22:283–290, 1996.

[3] G. A. Rummery. *Problem solving with reinforcement learning*. PhD thesis, Cambridge University, 1995.

[4] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technial Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University, 1994.

[5] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. Cambridge, MA, MIT Press, 1998.

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

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

[8] I. H. Witten. An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 34:286–295, 1977.