Estimator Variance in Reinforcement Learning: Theoretical Problems and Practical Solutions

Mark D. Pendrith and Malcolm R.K. Ryan
School of Computer Science and Engineering
The University of New South Wales
Sydney 2052 Australia

E-mail: {pendrith,malcolmr}@cse.unsw.edu.au

UNSW-CSE-TR-9703 — 13 MARCH 1997

Communicated by Claude Sammut

Abstract

In reinforcement learning, as in many on-line search techniques, a large number of estimation parameters (e.g. Q-value estimates for 1-step Q-learning) are maintained and dynamically updated as information comes to hand during the learning process. Excessive variance of these estimators can be problematic, resulting in uneven or unstable learning, or even making effective learning impossible. Estimator variance is usually managed only indirectly, by selecting global learning algorithm parameters (e.g. λ for $TD(\lambda)$ based methods) that are a compromise between an acceptable level of estimator perturbation and other desirable system attributes, such as reduced estimator bias. In this paper, we argue that this approach may not always be adequate, particularly for noisy and non-Markovian domains, and present a direct approach to managing estimator variance, the new ccBeta algorithm. Empirical results in an autonomous robotics domain are also presented showing improved performance using the ccBeta method.

1 Introduction

Many domains of interest in AI are too large to be searched exhaustively in reasonable time. One approach has been to employ on-line search techniques, such as *reinforcement learning* (RL). In RL, as in many on-line search techniques, a large number of estimation parameters (e.g. Q-value estimates for 1-step Q-learning) are maintained and dynamically updated as information comes to hand during the learning process. Excessive variance of these estimators during the learning process can be problematic, resulting in uneven or unstable learning, or even making effective learning impossible.

Normally, estimator variance is managed only indirectly, by selecting global learning algorithm parameters (e.g. λ for $TD(\lambda)$ based methods) that trade-off the level of estimator perturbation against other system attributes, such as estimator bias or rate of adaptation. In this paper, we give reasons why this approach may sometimes run into problems, particularly for noisy and non-Markovian domains, and present a direct approach to managing estimator variance, the ccBeta algorithm.

2 RL as On-line Dynamic Programming

RL has been characterised as an form of asynchronous, on-line form of Dynamic Programming (DP) (Watkins, 1989; Sutton, Barto, & Williams, 1992). This characterisation works well if the domain is well-modelled by a *Markov Decision Process* (MDP) (Puterman, 1994).

Formally, an MDP can be described as a quintuple $\langle S, A, \sigma, T, \rho \rangle$. S is the set of process states, which may include a special terminal state. From any non-terminal state, an action can be selected from the set of actions A, although not all actions may be available from all states.

At time-step $t = 0$ the process starts at random in one of the states according to a starting probability distribution function σ. At any time-step $t \geq 0$ the process is in exactly one state, and selecting an action from within state s_t results in a transition to a successor state s_{t+1} according to a transition probability function T. Also, an immediate scalar payoff (or reward) r_t is received, the expected value of which is determined by ρ, a (possibly stochastic) mapping of $S \times A$ into the reals. Once started, the process continues indefinitely or until a terminal state is reached.

By a *Markov* process, we mean that both the state transition probability function T and the payoff expectation ρ is dependent only upon the action selected and the current state. In particular, the history of states/actions/rewards of the process leading to the current state does not influence T or ρ. We will also consider non-Markov Decision Processes (NMDPs) which are formally identical except that the Markov assumption is relaxed; for a variety of reasons NMDPs often better model complex real-world RL domains than do MDPs (Pendrith & Ryan, 1996).

Generally when faced with a decision process the problem is to discover a mapping $S \to A$ (or *policy*) that maximises the expected *total future discounted* *reward* $R_\gamma = \sum_{t=0}^{\infty} \gamma^t r_t$ for some *discount factor* $\gamma \in [0,1]$. If $\gamma = 1$, then the total future discounted reward is just the *total reward* $R = \sum_{t=0}^{\infty} r_t$ where all future rewards are weighted equally; otherwise, rewards received sooner are weighted more heavily than those received later.

2.1 Q-learning as on-line value iteration

If an RL method like *1-step Q-learning* (QL) (Watkins, 1989) is used to find the optimal policy for an MDP, the method resembles an asynchronous, on-line form of the DP *value iteration* method. QL can be viewed as a relaxation method that successively approximates the so-called *Q-values* of the process, the value $Q^\pi(s_t, a_t)$ being the expected value of the return by taking a action a_t from state s_t and following a policy π from that point on.

We note that if an RL agent has access to Q-values for an optimal policy for the system it is controlling, it is easy to act optimally without planning; simply selecting the action from each state with the highest Q-value will suffice. The QL algorithm has been shown to converge, under suitable conditions[1], to just these Q-values. The algorithm is briefly recounted below.

At each step, QL updates an entry in its table of Q-value estimates according to the following rule (presented here in a "delta rule" form):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \beta \Delta_t \tag{1}$$

where β is a step-size parameter, and

$$\Delta_t = \mathbf{r}_t^{(1)} - Q(s_t, a_t) \tag{2}$$

where $\mathbf{r}_t^{(1)}$ is the 1-step *corrected truncated return* (CTR):

$$\mathbf{r}_t^{(1)} = r_t + \gamma \max_a Q(s_{t+1}, a) \tag{3}$$

The 1-step CTR is a special case of the n-step CTR. Using Watkins' (1989) notation

$$\mathbf{r}_t^{(n)} = \mathbf{r}_t^{[n]} + \gamma^n \max_a Q(s_{t+n}, a) \tag{4}$$

where $\mathbf{r}_t^{[n]}$ is the simple *uncorrected* n-step truncated return (UTR)

$$\mathbf{r}_t^{[n]} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} \tag{5}$$

As $n \to \infty$, both $\mathbf{r}_t^{[n]}$ and $\mathbf{r}_t^{(n)}$ approach the *infinite horizon* or *actual return*

$$\mathbf{r}_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{6}$$

[1]Perhaps most fundamentally the Markov assumption must hold; it is known (e.g. Pendrith & Ryan, 1996) that the optimality properties of QL do *not* generalise to non-Markov systems. The original proof of QL convergence and optimality properties can be found in an expanded form in (Watkins & Dayan, 1992).

as a limiting case.

We note that if in (2) $\mathbf{r}_t^{(1)}$ were replaced with the actual return, then this would form the update rule for a Monte Carlo estimation procedure. However, rather than waiting for the completed actual return, QL instead employs a "virtual return" that is an estimate of the actual return. This makes the estimation process resemble an on-line value iteration method. One view of the n-step CTR is that it bridges at its two extremes value iteration and Monte Carlo methods. One could also make the observation that such a Monte Carlo method would be an on-line, asynchronous analog of *policy iteration*, another important DP technique.

The central conceptual importance of the CTR to RL techniques is that virtually all RL algorithms estimate the value function of the state/action pairs of the system using either single or multi-step CTRs directly, as in the case of QL or the C-Trace algorithm (Pendrith & Ryan, 1996), or as returns that are equivalent to weighted sums of varying length n-step CTRs, such as the $TD(\lambda)$ return (Sutton, 1988; Watkins, 1989).

3 CTR Bias and Variance

For RL in Markovian domains, the choice of length of CTR is usually viewed as a trade-off between bias and variance of the sample returns to the estimation parameters, and hence of the estimation parameters themselves (e.g. Watkins 1989).

The idea is that shorter CTRs should exhibit less variance but more bias than longer CTRs. The increased bias will be due to the increased weight in the return values of estimators that will, in general, be inaccurate while learning is still taking place. The expected reduction in estimator variance is due to the fact that for a UTR the variance of the return will be strictly non-decreasing as n increases.

Applying this reasoning uncritically to CTRs is problematic, however. In the case of CTRs, we note that initial estimator inaccuracies that are responsible for return bias may also result in high return variance in the early stages of learning. Thus, in the early stages of learning, shorter CTRs may actually result in the worst of both worlds – high bias *and* high variance.

By way of illustration, consider the simple MDP depicted in Figure 1. The expected variance of the sampled returns for action 0 from state A will be arbitrarily high depending upon the difference between the initial estimator values for actions from states B and C. In this case, the estimator for $\langle A, 0 \rangle$ would experience both high bias and high variance if 1-step CTRs were to be used. On the other hand, using CTRs of length 2 or greater would result in unbiased estimator returns with zero variance at all stages of learning for this MDP.

In general, as estimators globally converge to their correct values, the variance of an n-step CTR for an MDP will become dominated by the variance in the terms comprising the UTR component of the return value, and so the

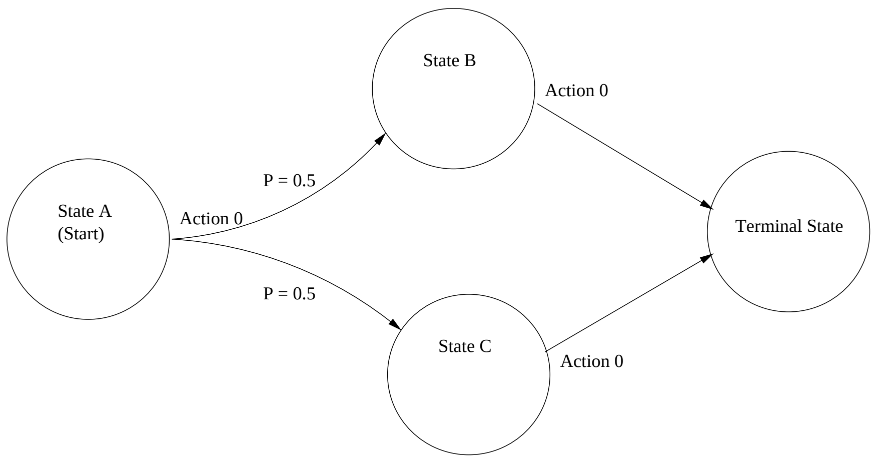

Figure 1: A 4-state/1 action MDP. State A is the starting state, states B and C are equiprobable successor states after taking action 0. Actions from B and C immediately lead to termination. The immediate reward at each step is zero. If 1-step CTRs are used, the variance as well as bias of the estimator returns for State A/action 0 depends on the difference of the initial estimator values for states B and C. On the other hand, if CTRs of length 2 or greater are used, the estimator returns will be unbiased and have zero variance.

relation

$$i < j \Rightarrow var[\mathbf{r}_t^{(i)}] \leq var[\mathbf{r}_t^{(j)}] \tag{7}$$

will be true in the limit. However, the point we wish to make here is that a bias/variance tradeoff involving n for CTRs is not as clear cut as may be often assumed, particularly in the early stages of learning, or at any stage of learning if the domain is noisy,[2] even if the domain is Markovian.

3.1 CTR Bias and Variance in NMDPs

Perhaps more importantly, if the domain is not Markovian, then the relation expressed in (7) is not guaranteed to hold for *any* stage of the learning. To demonstrate this possibly surprising fact, we consider the simple 3-state non-Markov Decision Process (NMDP) depicted in Figure 2.

For this NMDP, the expected immediate reward from taking the action 0 in state B depends upon which action was taken from state A. Suppose the (deterministic) reward function ρ is as follows:

$$\begin{array}{ll} \rho(A,0) = 0 & \\ \rho(A,1) = 0 & \\ \rho(B,0) = 0 & \text{(if the action from state A was 0)} \\ \rho(B,0) = 1 & \text{(if the action from state A was 1)} \end{array}$$

If Monte Carlo returns are used (or, equivalently in this case, n-step CTRs where $n > 1$), the estimator returns for state/action pairs $\langle A, 0\rangle$ and $\langle A, 1\rangle$ will

[2]The argument here is that for noisy domains, estimator bias is continually being reintroduced, taking the process "backwards" towards the conditions of early learning.

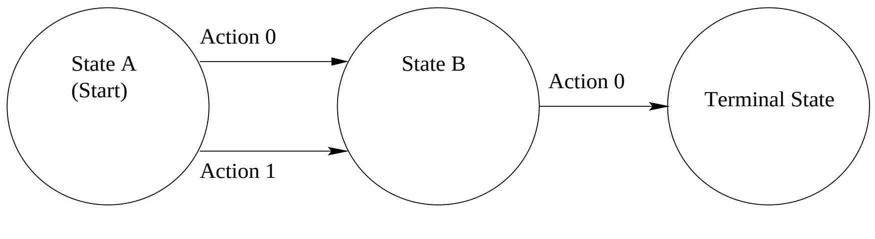

Figure 2: A 3-state NMDP, with two available actions from starting state A, and one available from the successor state B. The action from state B immediately leads to termination and a reward; the decision process is non-Markov because the reward depends on what action was previously selected from state A.

exhibit zero variance at all stages of learning, and the corresponding estimators should rapidly converge to their correct values of 0 and 1 respectively.

On the other hand, if 1-step CTRs are used, the variance of the $\langle A, 0\rangle$ and $\langle A, 1\rangle$ estimators will be non-zero while the variance of the estimator for $\langle B, 0\rangle$ is non-zero. The estimator for $\langle B, 0\rangle$ will exhibit non-zero variance as long as both actions continue to be tried from state A, which would normally be for all stages of learning for the sake of active exploration. Finally, note that the variance for $\langle B, 0\rangle$ will the same in this case for all n-step CTRs $n \geq 1$. Hence, the overall estimator variance for this NMDP is *strictly greater at all stages of learning* for 1-step CTRs than for any n-step CTRs $n > 1$.

In previously published work studying RL in noisy and non-Markovian domains (Pendrith & Ryan, 1996), excessive estimator variance appeared to be causing problems for 1-step QL in domains where using Monte Carlo style returns improved matters. These unexpected experimental results did not (and still do not) fit well with the "folk wisdom" concerning estimator bias and variance in RL. We present these analyses firstly as a tentative partial explanation for the unexpected results in these experiments.

Secondly, the foregoing analysis is intended to provide some theoretical motivation for an entirely different approach to managing estimator variance in RL, in which attention is shifted away from CTR length and is instead focused on the step-size parameter β. In the next part of this paper, we discuss a new algorithm (perhaps more accurately a family of algorithms) we call ccBeta, which results from taking such an approach.

4 β: Variance versus Adaptability

In RL, finding a good value for the step-size parameter β for a particular algorithm for a particular domain is usually in a trial-and-error process for the experimenter, which can be time consuming. The resulting choice is usually a trade-off between fast adaptation (large β) and low estimator variance (small β). In RL, and in adaptive parameter estimation systems generally, there emerges a natural tension between the issues of convergence and adaptability.

Stochastic convergence theory (Kushner & Clark, 1978) suggests that a *reducing* β series (such as $\beta_i = 1/i$) with the properties

$$\sum_{i=1}^{\infty} \beta_i = \infty, \quad \text{and} \quad \sum_{i=1}^{\infty} \beta_i^2 < \infty \tag{8}$$

may be used to adjust an estimator's value for successive returns; this will guarantee in-limit convergence under suitable conditions. However, this is in general not a suitable strategy for use non-stationary environments i.e. environments in which the schedule of payoffs may vary over time. While convergence properties for stationary environments are good using this technique, re-adaptation to changing environmental conditions can be far too slow.

In practice, a constant value for the β series is usually chosen. This has the the advantage of being constantly sensitive to environment changes, but has poorer convergence properties, particularly in noisy or stochastic environments. It appears the relatively poor convergence properties of a constant β series can lead to instabilities in learning in some situations, making an effective trade-off between learning rate and variance difficult.

A method for automatically varying the step-size β parameter by a simple on-line statistical analysis of the estimate error is presented here. The resulting β series will be neither constant nor strictly decreasing, but will vary as conditions indicate.

4.1 The ccBeta Algorithm

At each update step for the parameter estimate, we assume we are using a "delta-rule" or on-line LMS style update rule along the lines of

$$\begin{aligned} \Delta_i &\leftarrow z_i - Q_{i-1} \\ Q_i &\leftarrow Q_{i-1} + \beta_i \Delta_i \end{aligned}$$

where z_i is the i^{th} returned value in the series we are trying to estimate and β_i is the i^{th} value of the step-size schedule series used. Q_i is the i^{th} estimate of this series, and Δ_i is the i^{th} value of the error series.

The idea behind ccBeta is quite straightforward. If the series of estimate errors for a parameter is positively auto-correlated, this indicates a persistent over- or under-estimation of the underlying value to be estimated is occurring, and suggests β should be increased to facilitate rapid adaptation. If, on the other hand, the estimate errors are serially uncorrelated, then this may be taken as an indication that there is no systematic error occurring, and β can be safely decreased to minimise variance while these conditions exist.

So, for each parameter we are trying to estimate, we keep a separate set of autocorrelation statistics for its error series as follows, where cc_i is derived as an exponentially weighted autocorrelation coefficient:

$$sum\_square\_err_i \leftarrow K.sum\_square\_err_{i-1} + \Delta_i^2 \tag{9}$$

$$sum\_product_i \leftarrow K.sum\_product_{i-1} + \Delta_i.\Delta_{i-1} \tag{10}$$

$$cc_i \leftarrow \frac{sum\_product_i}{\sqrt{sum\_square\_err_i.sum\_square\_err_{i-1}}} \tag{11}$$

At the start of learning, the *sum_square_err* and *sum_product* variables are initialised to zero; but this potentially leads to a divide-by-zero problem on the RHS of (11). We explicitly check for this situation, and when detected, cc_i is set to 1.[3]

We note that if in (9) and (10) the exponential decay parameter $K \in [0, 1]$ is less than 1, two desirable properties emerge: firstly, the values of the *sum_square_err* and *sum_product* series are finitely bounded, and secondly the correlation coefficients are biased with respect to recency. While the first property is convenient for practical implementation considerations with regards to possible floating point representation overflow conditions etc., the second property is essential for effective adaptive behaviour in non-stationary environments. Setting K to a value of 0.9 has been found to be effective in all domains tested so far; experimentally this does not seem to be a particularly sensitive parameter.

It is also possible to derive an autocorrelation coefficient not of the error series directly, but instead of the sign of the the error series, i.e. replacing the Δ_i and Δ_{i-1} terms in (9) and (10) with $sgn(\Delta_i)$ and $sgn(\Delta_{i-1})$. This variant may prove to be generally more robust in very noisy environments.

In such a situation an error series may be so noisy that, even if the error signs are consistent, a good linear regression is not possible, and so β will be small even when there is evidence of persistent over- or under-estimation. This approach proved to be successful when applied to the extremely noisy real robot domain described in the next section. Based on our results to date, this version could be recommended as a good "general purpose" version of ccBeta.

Once an autocorrelation coefficient is derived, β_i is set as follows:

```
if ($cc_i > 0$)
        $\beta_i \leftarrow cc_i * MAX\_BETA$
else    $\beta_i \leftarrow 0$

if ($\beta_i < MIN\_BETA$)
        $\beta_i \leftarrow MIN\_BETA$
```

First, we note that in the above pseudo-code, negative and zero autocorrelations are treated the same for the purposes of weighting β_i. A strongly negative autocorrelation indicates alternating error signs, suggesting fluctuations around a mean value. Variance minimisation is also desirable in this situation, motivating a small β_i.

On the other hand, a strongly positive cc_i results from a series of estimation errors of the same sign, indicating a persistent over- or under-estimation, suggesting a large β_i is appropriate to rapidly adapt to changed conditions.

Setting the scaling parameters MIN_BETA to 0.01 and MAX_BETA to 1.0 has been found to be effective, and these values are used in the experiments that follow. Although values of 0 and 1 respectively might be more "natural", as this corresponds to the automatic scaling of the correlation coefficient, in

[3]This may seem arbitrary, but the reasoning is simply that if you have exactly one sample from a population to work with, the best estimate you can make for the mean of that population is the value of that sample.

Simulation experiment 1 (noisy zero fn)		
	T.S.E.	Std. Dev.
ccBeta	17.95	0.042
beta = 0.1	10.42	0.032
beta = 0.2	22.30	0.047
beta = 0.3	35.72	0.060
beta = 0.4	50.85	0.071
reducing beta	0.17	0.004

Table 1: Results for simulation experiment 1.

practice a small non-zero MIN_BETA value was observed to have the effect of making an estimate series less discontinuous (although whether this offers any real advantages has not been fully determined at this stage.)

Finally, we note that *prima facie* it would be reasonable to use cc_i^2 rather than cc_i to weight β_t. Arguably, cc_i^2 is the more natural choice, since from statistical theory it is the square of the correlation coefficient that indicates the proportion of the variance that can be attributed to the change in a correlated variable's value.

Both the cc_i and cc_i^2 variations have been tried, and while in simulations marginally better results both in terms of total square error (TSE) and variance were obtained by using cc_i^2, a corresponding practical advantage was not evident when applied to the robot experiments.

5 Experimental Results

In all the experiments that follow we use the same variant of ccBeta; this has a K parameter of 0.9, and uses $sgn(\Delta)$ normalisation to calculate cc_i. For weighting β_i, we use cc_i rather than cc_i^2, and have MIN_BETA and MAX_BETA set to 0.01 and 1.0 respectively.

5.1 Simulation experiment 1

In this section, we describe some simulation studies comparing the variance and re-adaptation sensitivity characteristics of the ccBeta method for generating a β step-size series against standard regimes of fixed β and reducing β, where $\beta_i = 1/i$.

The learning system for these experiments was a single-unit on-line LMS estimator which was set up to track an input signal for 10,000 time steps. In the first experiment, the signal was stochastic but with stationary mean: a zero function perturbed by uniform random noise in the range $[-0.25, +0.25]$. The purpose of this experiment was to assess asymptotic convergence properties, in particular estimator error and variance.

As can be seen from Table 1, the reducing beta schedule of $\beta_i = 1/i$ was superior to fixed beta and ccBeta in terms of both total square error (TSE) and estimator variance for this experiment. As we would expect, variance

Simulation experiment 2 (noisy step fn)		
	T.S.E.	Steps to Crossing
ccBeta	21.60	10
beta = 0.1	16.19	35
beta = 0.2	25.84	15
beta = 0.3	38.56	9
beta = 0.4	53.35	8
reducing beta	395.73	>9,600

Table 2: Results for simulation experiment 2.

(normalised to standard deviation in the tabled results) increased directly with the magnitude of beta for the fixed beta series. In this experiment ccBeta performed at a level between fixed beta set at 0.1 and 0.2.

5.2 Simulation experiment 2

In the second experiment, a non-stationary stochastic signal was used to assess re-adaption performance. The signal was identical to that used in the first experiment, except that after the first 400 time steps the mean changed from 0 to 1.0, resulting in a noisy step function. The adaption response for the various beta series over 50 time steps around the time of the change in mean are plotted in figures 3 and 4.

To get an index of responsiveness to the changed mean using the different beta series, we have measured the number of time steps from the time the mean level was changed to the time the estimator values first crossed the new mean level, i.e. when the estimator first reaches a value ≥ 1.

As can be seen from Table 2, the reducing beta schedule of $\beta_i = 1/i$ was far worse than than for either fixed beta or ccBeta in terms of TSE and re-adaptation performance ("Steps to Crossing", column 2). Indeed, the extreme sluggishness of the reducing beta series was such that the estimator level had risen to only about 0.95 after a further 9,560 time steps past the end of the time-step window shown in figure 3. The relatively very high TSE for the reducing beta series was also almost entirely due to this very long re-adaptation time. The inherent unsuitability of such a regime for learning in a non-stationary environment is clearly illustrated in this experiment. Despite having "nice" theoretical properties, it represents an impractical extreme in the choice between good in-limit convergence properties and adaptability.

In the figure 3 plots, the trade-off of responsiveness versus estimator variance for a fixed beta series is clearly visible. We note however that the re-adaptation response curve for ccBeta (figure 4) resembles that of the higher values of fixed beta, while its TSE (table 2) corresponds to lower values, which gives some indication the algorithm is working as intended.

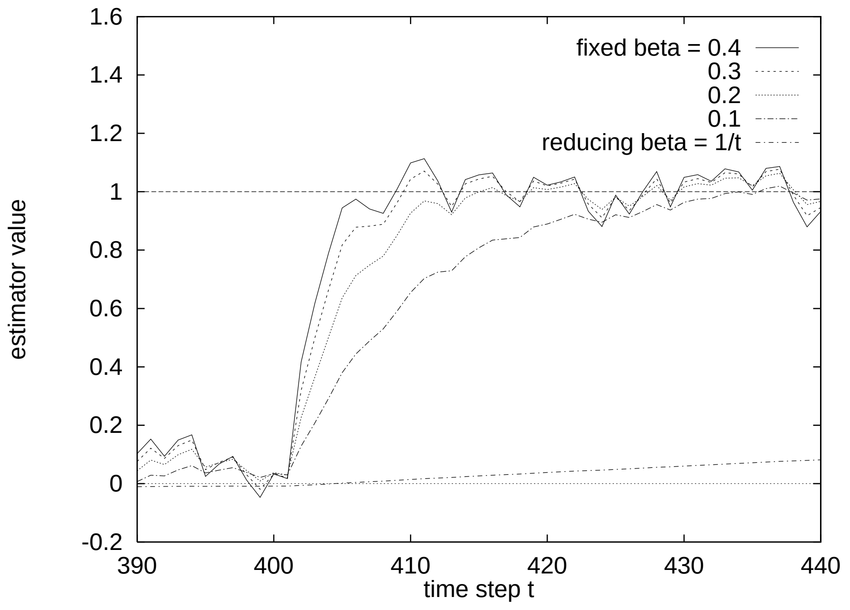

Figure 3: Fixed beta and reducing beta step response plots.

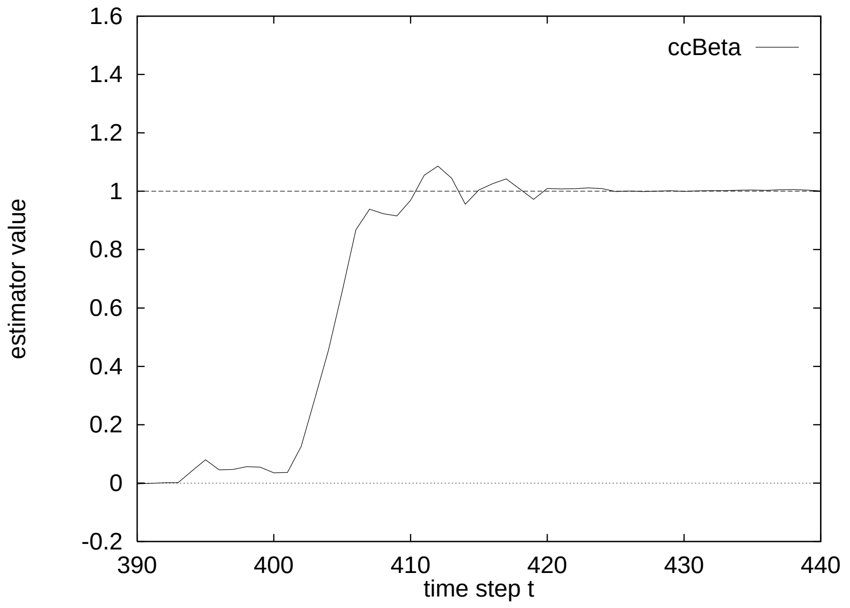

Figure 4: ccBeta step response plot.

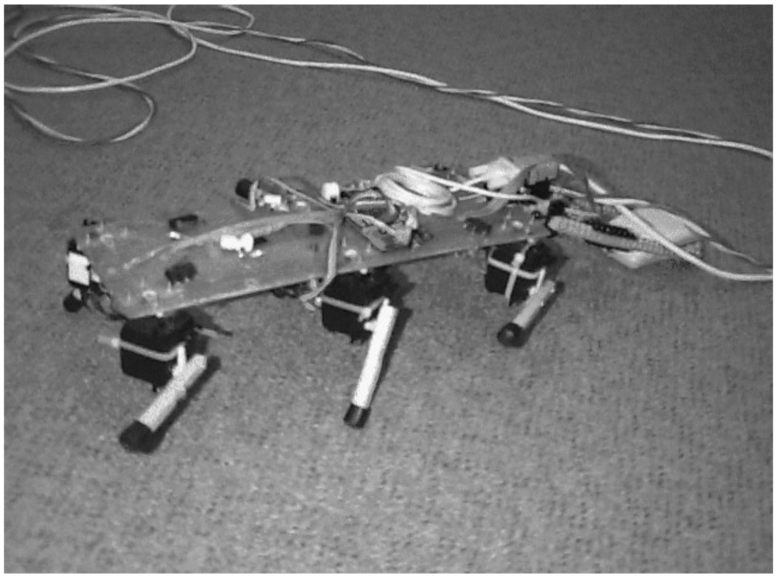

Figure 5: The robot learning to walk.

5.3 Experiment 3: Learning to Walk

For the next set of experiments we have scaled up to a real robot-learning problem: the gait coordination of a six-legged insectoid walking robot.

The robot (affectionately referred to as “Stumpy” in the UNSW AI lab) is faced with the problem of learning to walk with a forward motion, minimising backward and lateral movements. In this domain, ccBeta was compared to using hand tuned fixed beta step-size constants for two different versions of the C-Trace RL algorithm.

Stumpy is, by robotics standards, built to an inexpensive design. Each leg has two degrees of freedom, powered by two hobbyist servo motors (see Figure 5). A 68HC11 Miniboard converts instructions from a 486 PC sent via a RS-232-C serial port connection into the pulses that fire the servo motors. The primary motion sensor is a cradle-mounted PC mouse dragged along behind the robot. This provides for very noisy sensory input, as might be appreciated. The quality of the signal has been found to vary quite markedly with the surface the robot is traversing.

As well as being noisy, this domain was non-Markovian by virtue of the compact but coarse discretized state-space representation. This compact representation[4] meant learning was fast, but favoured an RL algorithm that did not rely heavily on the Markov assumption; in earlier work (Pendrith & Ryan, 1996) C-Trace had been shown to be well-suited for this domain.

The robot was given a set of primitive “reflexes” in the spirit of Rodney Brooks’ “subsumption” architecture (Brooks, 1991). A leg that is triggered will incrementally move to lift up if on the ground and forward if already lifted.

[4]1024 “boxes” in 6 dimensions: alpha and beta motor positions for each leg group (4 continuous variables each discretized into 4 ranges), plus 2 boolean variables indicating the triggered or untriggered state of each leg group at the last control action.

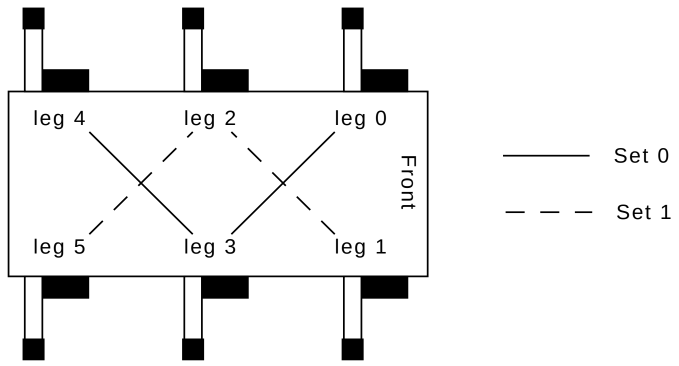

Figure 6: Grouping the legs into two tripods.

A leg that does not receive activation will tend to drop down if lifted and backwards if already on the ground. In this way a basic stepping motion was encoded in the robots "reflexes".

The legs were grouped to move in two groups of three to form two tripods (Figure 6). The learning problem was to discover an efficient walking policy by triggering or not triggering each of the tripod groups from each state. Thus the action set to choose from in each discretized state consisted of four possibilities:

- Trigger both groups of legs
- Trigger group A only
- Trigger group B only
- Do not trigger either group

The robot received positive reinforcement for forward motion as detected by the PC mouse, and negative reinforcement for backward and lateral movements.

Although quite a restricted learning problem, interesting non-trivial behaviours and strategies have been seen to emerge.

5.4 The RL algorithms

As mentioned earlier, C-Trace is an RL algorithm that uses multi-step CTRs to estimate the state/action value function. While one C-Trace variant, *multiple-visit C-Trace*, has been described in earlier work (Pendrith & Ryan, 1996) in application to this domain, the other, *first-visit C-Trace*, is a variant that has not been previously described (refer to figure 7 for pseudo-code).

It is easiest understand the difference between multiple-visit and first-visit C-Trace, in terms of the difference between a multiple-visit and a first-visit Monte Carlo algorithm. Briefly, a first-visit Monte Carlo algorithm will selectively ignore some returns for the purposes of learning in order to get a more truly independent sample set. In Singh & Sutton (1996), first-visit versus multiple-visit returns are discussed in conjunction with with a new "first-visit" version of the $TD(\lambda)$ algorithm, and significant improvements in learning performance are reported for a variety of domains using the new algorithm.

```
while not terminal state do

    get current state $s$
    select action $a$                          (* stochastic action selection *)

    if non-policy action selected then
        for all $(i,j)$ such that $VisitCount_{ij} > 0$ do
            $c \leftarrow \max_m Q_{im}$               (* truncated return correction *)
            $k \leftarrow GlobalClock - StepCount_{ij}$
            $\rho_{ij} \leftarrow \rho_{ij} + c\gamma^k$
            $Q_{ij} \leftarrow (1-\beta)Q_{ij} + \beta\rho_{ij}$    (* apply CTR update *)
            $VisitCount_{ij} \leftarrow 0$            (* zero traces *)
        endfor
    endif

    if $VisitCount_{sa} = 0$ then              (* first visit *)
        $\rho_{sa} \leftarrow 0$
        $VisitCount_{sa} \leftarrow 1$
        $StepCount_{sa} \leftarrow GlobalClock$        (* “time-stamp” visit *)
    endif

    take action $a$

    if reinforcement signal $r$ received then
        for all $(i,j)$ such that $VisitCount_{ij} > 0$ do
            $k \leftarrow GlobalClock - StepCount_{ij}$
            $\rho_{ij} \leftarrow \rho_{ij} + r\gamma^k$
        endfor
    endif

    $GlobalClock \leftarrow GlobalClock + 1$

endwhile

for all $(s,a)$ such that $VisitCount_{sa} > 0$ do   (* terminal state reached *)
    $Q_{sa} \leftarrow (1-\beta)Q_{sa} + \beta\rho_{sa}$            (* terminal update rule *)
endfor
```

Figure 7: Pseudo-code for first-visit C-Trace. The variables ρ, *StepCount* and *VisitCount* are kept separately for each state/action pair, as they are associated with a particular Q-value. The subscripts identify to which state/action pair the variable belongs. This C-Trace variant is “first-visit” in the sense that returns corresponding to subsequent visits of state/action pairs are ignored if a return is currently being traced. This approach makes the returns a more truly independent sample set. Once a trace is cleared, either by a CTR update or a terminal update, a state/action pair is once again eligible for another first-visit sample return.

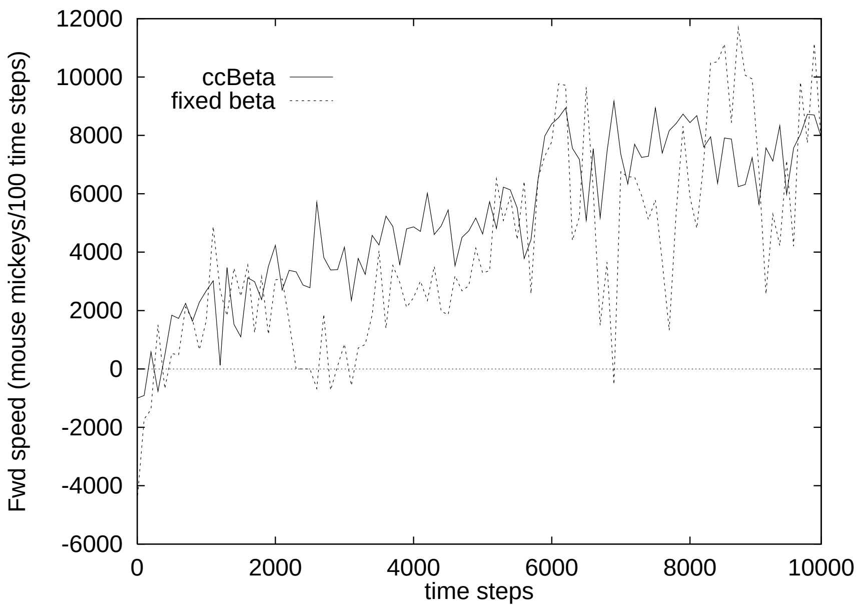

Figure 8: The robot experiment using RL algorithm 1 (multiple-visit C-Trace).

For these reasons, a first-visit version of C-Trace seemed likely to be particularly well-suited for a ccBeta implementation, as the "purer" sampling methodology for the returns should (in theory) enhance the sensitivity of the on-line statistical tests.

5.5 Discussion of results

The average forward walking speed over the first hour of learning for two versions of C-Trace using both ccBeta and fixed step-size parameters are presented in the plots in figures 8 and 9.

In the case of multiple-visit C-Trace (figure 8), we notice immediately that the learning performance is much more stable using ccBeta than with the fixed beta series. This shows up as obviously reduced variance in the average forward speed; significantly, the overall learning rate doesn't seem to have been adversely affected, which is encouraging. It would not be unreasonable to expect some trade-off between learning stability and raw learning rate, but such a trade-off is not apparent in these results.

Interestingly, the effects of estimator variance seem to manifest themselves in a subtly different way in the first-visit C-Trace experiments (figure 9). We notice that first-visit C-Trace even without ccBeta seems to have had a marked effect on reducing the step-to-step variance in performance as seen in multiple-visit C-Trace. This is very interesting in itself, and calls for further theoretical and experimental investigation.[5]

[5]At this point, we will make the following brief observations on this effect: a) The reduction in variance theoretically makes sense inasmuch as the variance of the sum of several random variables is equal to the sum of the variances of the variables if the variables are not correlated, but will be greater than than this if they are positively correlated. In multiple-visit returns,

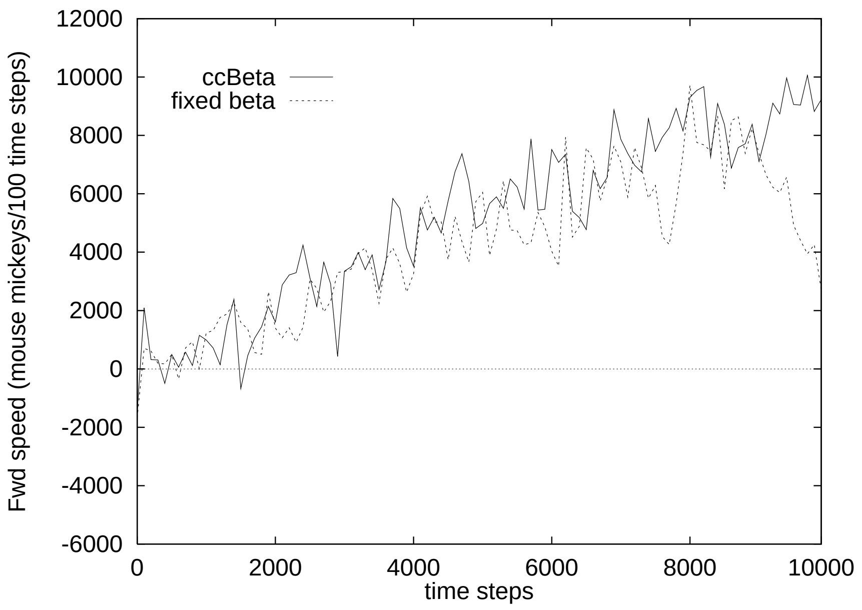

Figure 9: The robot experiment using RL algorithm 2 (first-visit C-Trace).

However, we also notice that at around time-step 8000 in the first-visit fixed-beta plot there is the start of a sharp decline in performance, where the learning seems to have become suddenly unstable.

These results were averaged over several (n = 5) runs for each plot, so this appears to be a real effect. If so, it is conspicuous by its absence in the first-visit ccBeta plot: ccBeta would appear to have effectively rectified the problem.

Overall, the combination of first-visit C-Trace and ccBeta seems to be the winning combination for these experiments, which is encouragingly in agreement with prediction.

6 Conclusions

Excessive estimator variance during on-line learning can be a problem, resulting in learning instabilities of the sort we have seen in the experiments described here.

In RL, estimator variance is traditionally dealt with only indirectly via the general process of tuning various learning parameters, which can be a time consuming trial-and-error process. Additionally, theoretical results presented here indicate some of the pervasive ideas regarding the trade-offs involved in the tuning process need to be critically examined. In particular, the relationship between CTR length and estimator variance needs reassessment, particularly in the case of learning in a non-Markov domain.

the positive correlation between returns is what prevents them being statistically independent. b) This raises the interesting possibility that the observed improved performance of "replacing traces" owes as much if not more to a reduction in estimator variance than to reduced estimator bias, which is the explanation proposed by (Singh & Sutton, 1996).

The ccBeta algorithm has been presented as a practical example of an alternative approach to managing estimator variance in RL. This algorithm has been designed to actively minimise estimator variance while avoiding the degradation in re-adaptation response times characteristic of passive methods. ccBeta has been shown to perform well both in simulation and in real-world learning domains.

A possible advantage of this algorithm is that since it is not particularly closely tied in its design or assumptions to RL algorithms, Markov or otherwise, it may turn out be usable with a fairly broad class of on-line search methods. Basically, any method that uses some form of "delta rule" for parameter updates might potentially benefit from using a ccBeta-style approach to managing on-line estimator variance.

7 Acknowledgements

The authors are indebted to Paul Wong for his assistance in performing many of the robot experiments.

References

Brooks, R. (1991). Intelligence without reason. In *Proceedings of the* 12^{th} *International Joint Conference on Artificial Intelligence*, pp. 569–595.

Kushner, H., & Clark, D. (1978). *Stochastic approximation methods for constrained and unconstrained systems.* New York: Springer-Verlag.

Pendrith, M., & Ryan, M. (1996). Actual return reinforcement learning versus Temporal Differences: Some theoretical and experimental results. In L.Saitta (Ed.), *Machine Learning: Proc. of the Thirteenth Int. Conf.* Morgan Kaufmann.

Puterman, M. (1994). *Markov decision processes : discrete stochastic dynamic programming.* New York: John Wiley & Sons.

Singh, S., & Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *To Appear In: Machine Learning.*

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

Sutton, R., Barto, A., & Williams, R. (1992). Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, *12*(2), 19–22.

Watkins, C. (1989). *Learning from Delayed Rewards.* Ph.D. Thesis, King's College, Cambridge.

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