Temporal Coherence and Prediction Decay in Temporal Difference Learning

Beal, Don F.; Smith, M.C.

ISSN 1369-1961

Department of Computer Science

Technical Report No. 756

Temporal Coherence and Prediction Decay in Temporal Difference Learning

D. F. Beal & M.C. Smith

QUEEN MARY
AND WESTFIELD COLLEGE
UNIVERSITY OF LONDON

August 1998

Temporal Coherence and Prediction Decay in Temporal Difference Learning

D.F. Beal[1] *and M.C. Smith*

London, England

Abstract

This paper describes an extension of the temporal difference (TD) learning method. The standard form of the TD method has the problem that two control parameters, learning rate and temporal discount, need to be chosen appropriately. These parameters can have a major effect on performance, particularly the learning rate, which affects the stability of the process as well as the number of observations required. Our extension to the TD algorithm automatically sets and subsequently adjusts these parameters. The main performance advantage comes from the learning rate adjustment, which is based on a new concept we call temporal coherence (TC). The experiments reported here compare the TC algorithm performance with human-chosen parameters and with an earlier method for learning rate adjustment, in both a simple learning task and in a complex domain. The task domains were a random-walk state-learning task and the task of learning the relative values of pieces in a game, without any initial domain-specific knowledge. The results show that in both domains our method leads to better learning (i.e. faster and less subject to the effects of noise), than the selection of human-chosen values for the control parameters, and the comparison method.

1. Introduction

Two major parameters that control the behaviour of the temporal difference (TD) algorithm are the learning rate (or *step-size),* α, and the temporal discount parameter, λ.

The choice of these parameters can have a major effect on the efficacy of the learning algorithm, and in practical problems they are often determined somewhat arbitrarily, or else by trying a number of values and 'seeing what works' (e.g. Tesauro, 1992). Another widely used method is to use a learning rate that decreases over time, but such systems still require the selection of a suitable schedule.

Sutton and Singh (1994) describe systems for setting both α and λ, within the framework of Markov-chain models. However these methods assume relatively small numbers of distinct states, and acylic graphs, and so are not directly applicable to

[1] Department of Computer Science, Queen Mary and Westfield College, University of London, Mile End Road, London E1 4NS, England. Email: Don.Beal@dcs.qmw.ac.uk

more complex real-world problems. Jacobs (1988) presented the 'delta-bar-delta' algorithm for adjusting α during the learning process. We compared the performance of delta-bar-delta with our algorithm on two sample domains. More recently, Almeida (1998) and Schraudolph (1998) have presented other methods for α adaptation for stochastic domains and neural networks respectively.

We describe a new system which automatically adjusts α and λ . This system does not require any *a priori* knowledge about suitable values for learning rate or temporal discount parameters for a given domain. It adjusts the learning rate and temporal discount parameters according to the learning experiences themselves. We present results that show that this method is effective, and in our sample domains yielded better learning performance than our best attempt to find optimum choices of fixed α and λ , and better learning performance than delta-bar-delta.

1.1 Temporal difference learning

Temporal difference learning methods are a class of incremental learning procedures for learning predictions in multi-step prediction problems. Whereas earlier prediction learning procedures were driven by the difference between the predicted and actual outcome, TD methods are driven by the difference between temporally successive predictions (Sutton, 1988). Kaelbling et al (1996) give a survey of a wider range of reinforcement algorithms, including TD methods.

Sutton's TD process can be summarised by the following formula. Given a vector of adjustable weights, *w*, and a series of successive predictions, *P*, adjustments to the weights are determined at each timestep according to:

$$\Delta w_t = \alpha\left(P_{t+1} - P_t\right)\sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

where α is the parameter controlling the learning rate, $\nabla_w P_k$ is the partial derivative of P_k with respect to *w*, and P_t is the prediction at timestep *t*. The temporal discount parameter, λ, provides an exponentially decaying weight for more distant predictions.

The formula shows that TD learning is parameterised by α, the learning rate, and λ, the temporal discount factor. Both parameters, and especially α, can have a major effect on the speed with which the weights approach an optimum. In Sutton's paper, learning behaviour for different α and λ values in sample domains is presented, but no method for determining suitable values *a priori* is known. Learning rates too high can cause failure to reach stable values and learning rates too low can lead to orders of magnitude more observations being necessary. Methods of choosing suitable α and λ values before or during the learning are therefore advantageous. There have been several algorithms proposed for adjusting α in supervised and TD learning: ours is based on a new principle that we call temporal coherence.

2. Temporal Coherence: adjustments to learning rates

Our system of self-adjusting learning rates is based on the concept that the learning rate should be higher when there is significant learning taking place, and lower when changes to the weights are primarily due to noise. Random noise will tend to produce adjustments that cancel out as they accumulate. Adjustments making useful adaptations to the observed predictions will tend to reinforce as they accumulate. As weight values approach their optimum, prediction errors will become mainly random noise.

Motivated by these considerations, our Temporal Coherence (TC) method estimates the significance of the weight movements by the relative strength of reinforcing adjustments to total adjustments. The learning rate is set according to the proportion of reinforcing adjustments as a fraction of all adjustments. This method has the desirable property that the learning rate reduces as optimum values are approached, tending towards zero at optimum values. It has the equally desirable property of allowing the learning rate to increase if random adjustments are subsequently followed by a consistent trend.

Separate learning rates are maintained for each weight, so that weights that have become close to optimum do not fluctuate unnecessarily, and thereby add to the noise affecting predictions. The use of a separate learning rate for each weight allows for the possibility that different weights might become stable at different times during the learning process. For example, if weight A has become fairly stable after 100 updates, but weight B is still consistently rising, then it is desirable for the learning rate for weight B to be higher than that for weight A. An additional potential advantage of separate learning rates is that individual weights can be independent when new weights are added to the learning process. If new terms or nodes are added to an existing predictor, independent rates make it possible for the new weights to adjust quickly, whilst existing weights only increase their learning rates in response to perceived need.

The TC learning rates are determined by the history of *recommended* adjustments to each weight. We use the term 'recommended change' to mean the temporal difference adjustment prior to multiplication by the learning rate. This detachment of the learning rate enables the TC algorithm to respond to the underlying adjustment impulses, unaffected by its own recent choice of learning rate. It has the additional advantage that if the learning rate should reach zero, future learning rates are still free to be non-zero, and the learning does not halt.
The *recommended change* for weight w at timestep t is defined as:

$$r_{w,t} = (P_{t+1} - P_t)\sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

The change actually applied is:

$$\Delta w_t = \alpha_w r_{w,t}$$

where α_w = learning rate for weight w

For each weight we are interested in two numbers: the accumulated *net change* (the sum of the individual recommended changes), and the accumulated *absolute change* (the sum of the absolute individual recommended changes). The ratio of net change, *N*, to absolute change, *A*, allows us to measure whether the adjustments to a given weight are mainly in the same 'direction'. We take reinforcing adjustments as indicating an underlying trend, and cancelling adjustments as indicating noise from the stochastic nature of the domain (or limitations of the domain model that contains the weights).

At the end of each sequence, each weight, together with its net change and absolute change, is updated. The update formulae are:

$$w \leftarrow w + \sum_{i=1}^{t} \Delta w_i \qquad N_w \leftarrow N_w + \sum_{i=1}^{t} r_{w,i} \qquad A_w \leftarrow A_w = \sum_{i=1}^{t} \left| r_{w,i} \right|$$

N_w = accumulated net change to weight w
A_w = accumulated absolute change to weight w
$r_{w,i}$ = recommended change to weight w at timestep i

At each update, the learning rates for each weight are set according to the N:A ratio:

$$\alpha_w \leftarrow \frac{1}{v} \frac{N_w}{A_w}$$

v = number of adjustable weights

The learning rates are divided by v, since all v weights are being adjusted simultaneously, and each is assumed to contribute $1 / v$ to the total adjustment being made.

The foregoing formulae describe updating the weights and learning rates at the end of each sequence. The method may be easily amended to update more frequently (e.g. after each prediction), or less frequently (e.g. after a batch of sequences). For the domains and experiments reported in this paper, update at the end of each sequence is natural and convenient to implement.

3. Prediction Decay: determining the temporal discount parameter λ

We determine a value for the temporal discount parameter, λ, by computing a quantity ψ we call *prediction decay*. Prediction decay is a function of observed prediction values, indexed by temporal distance between them, described in more detail in appendix A. An exponential curve is fitted to the observed data, and the exponential constant, ψ, from the fitted curve is the *prediction decay*. We set $\lambda = 1$ initially, and $\lambda = \psi$ thereafter.

The use of $\lambda = \psi$ has the desirable characteristics that (i) a perfect predictor will result in $\psi = 1$, and TD(1) is an appropriate value for the limiting case as predictions approach perfection, (ii) as the prediction reliability increases, ψ increases, and it is reasonable to choose higher values of λ for TD learning as the prediction reliability improves. We make no claim that setting $\lambda=\psi$ is optimum. Our experience is that it typically performs better than human-guessed choice of a fixed λ *a priori*.[2]

The advantage of using prediction decay is that it enables TD learning to be applied effectively to domains without prior domain knowledge, and without prior experiments to determine an optimum λ. When combined with our method for adjusting learning rates, the resulting algorithm performs better than the comparison method, and better than using fixed rates, in both test domains.

4. Delta-bar-delta

The delta-bar-delta algorithm (DBD) for adapting learning rates is described by Jacobs (1988). Sutton (1992) later introduced Incremental DBD for linear tasks. The original DBD was directly applied to non-linear tasks, and hence more easily adapted to both our test domains. In common with our temporal coherence method, it maintains a separate learning rate for each weight. If the current derivative of a weight and the exponential average of the weight's previous derivatives possess the same sign, then DBD increases the learning rate for that weight by a constant, κ. If they possess opposite signs, then the learning rate for that weight is decremented by a proportion, ϕ, of its current value. The exponential average of past derivatives is calculated with θ as the base and time as the exponent. The learning rates are initialised to a suitable value, ε_0, and are then set automatically, although the meta-parameters κ, ϕ, θ and ε_0 must be supplied. To adapt DBD to TD domains, we compute a weight adjustment term, and a learning rate adjustment at each timestep, after each prediction, but we only apply the weight and learning rate adjustments at the end of each TD sequence. DBD is very sensitive to its meta-parameters and prior to our experiments we performed many test runs, exploring a large range of meta-parameter values and combinations. We used the best we found for the comparison between DBD and TC reported here. Both algorithms update the weights, and the internal meta-parameters, at the end of each sequence.

5. Test domain one: a bounded random walk

The methods described in this paper are designed to be domain independent, and should be applicable over a wide range of possible domains. We report first on a simple domain, that of a bounded random walk as presented in Sutton (1988) and Dayan (1992).

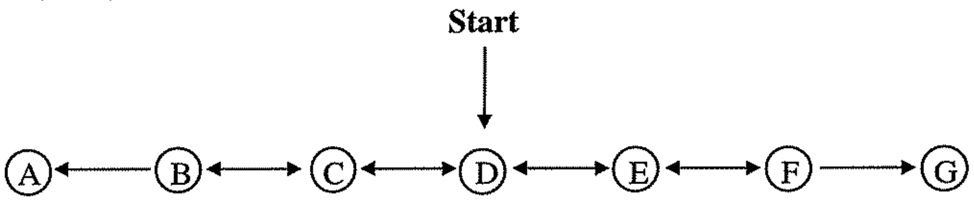

Figure 1: A bounded random walk

[2] By expending sufficient computation time to repeatedly re-run the experiments we were able to find somewhat better values for λ.

All walks begin in state D. When in states B, C, D, E, and F, there is a 50% chance of moving to the adjacent left state and a 50% chance of moving to the adjacent right state. When either end state (A or G) is reached, then the walk terminates, with a final outcome defined to be 0 in state A, and 1 in state G. This absorbing Markov process generated the random walks used in the experiments. Each sequence for the TD learning process is based on one walk. The learning task is to obtain five weights, one for each of the five internal states. These weights are estimates of the probabilities of terminating the walk at G, starting at the given internal state.

Sutton (1988) presents this task, and shows that temporal difference learning is more effective here than the widely-used supervised learning method of Least Means Squared (Widrow-Hoff), given an appropriate choice of control parameters. His experiments showed that the results achieved by TD(λ) in this domain were sensitive to the choice of both α and λ.

5.1 Results from the bounded random walk

For each experiment, a set of 1000 random walks was generated and each of the learning procedures was then applied to the same 1000 sequences. With this large number of sequences, the values towards the end of the run, averaged over recent sequences, are very close to the known theoretical value, and the weight movements are random noise. Nevertheless, to allow for variations due to different random sequences, each experiment was repeated 10 times using a different random number seed. The results from each of the 10 seeds were all very similar. Figures 2 through 5 are all derived from one particular starting seed, and are typical. Figure 6 presents results averaged from all 10 seeds.

Figures 2 and 3 show the weight movements from typical runs, using TD with two fixed learning rates, chosen to cover the range we found to be best from many runs, and a fixed λ (0.3) we chose as the most suitable for this task from results presented in Sutton (1988). The five traces on each graph show the estimated value of each of the five unknown states, after each of 1000 sequences. For this task, the true values are known, and are shown on the graphs as horizontal lines. The graphs illustrate that the higher the learning rate, the faster the weights approach the target values initially, but also illustrate that as the learning rate rises, the less stable the weight values. At high rates, it may become impracticable to extract stable weights.

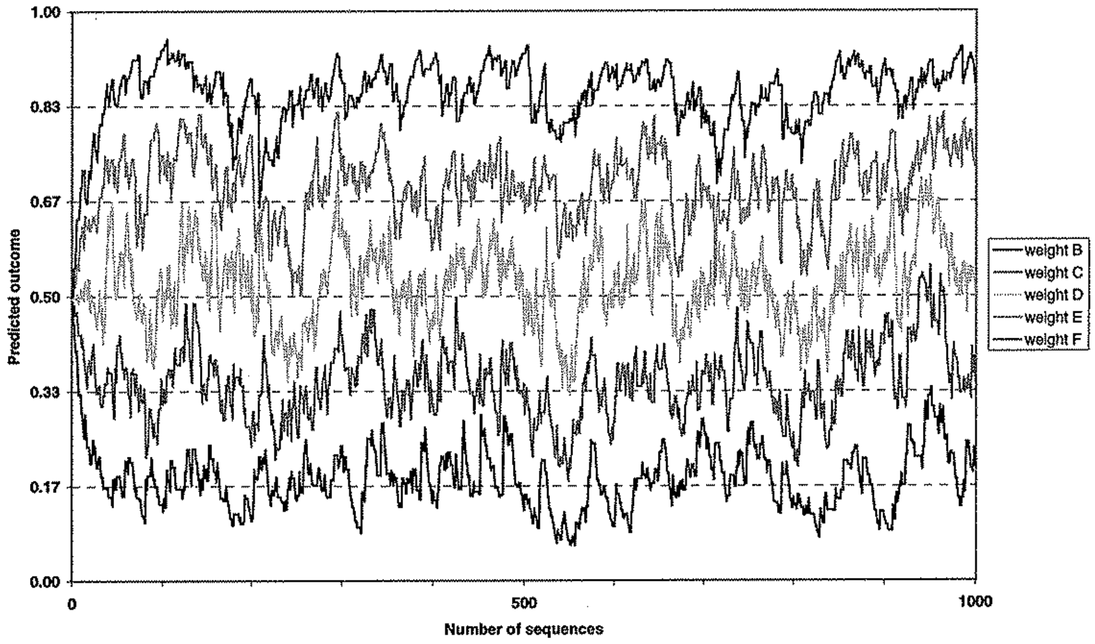

Figure 2: Weight movements from a typical run using a fixed α of 0.1

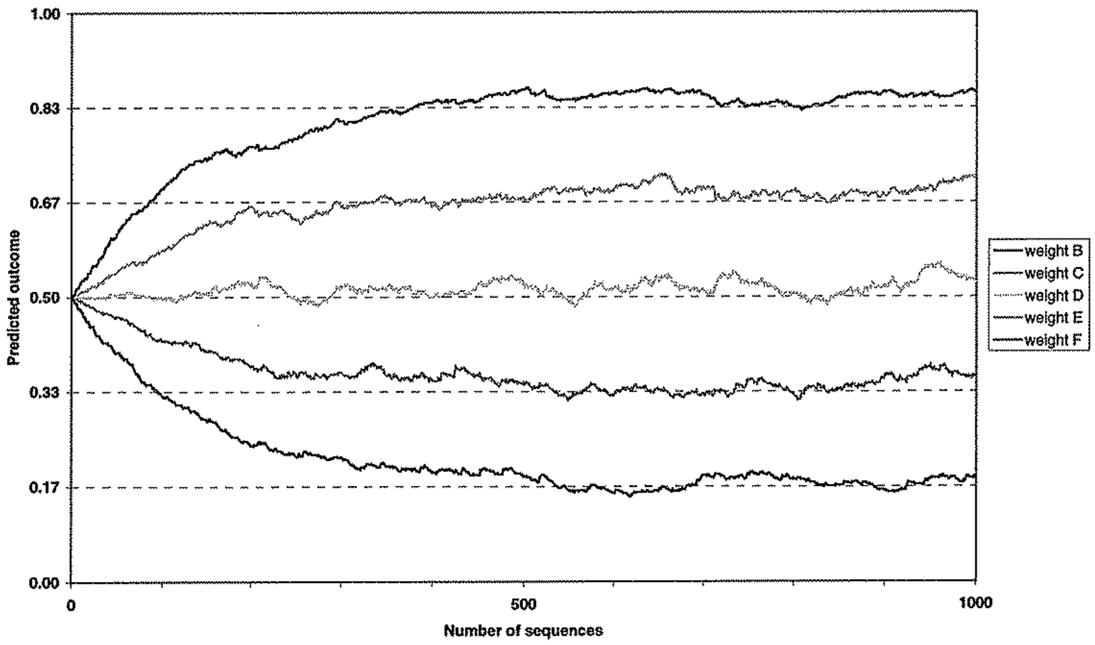

Figure 3: Weight movements from a typical run using a fixed α of 0.01

From figure 2 it can be seen that the weight adjustment made using a fixed α of 0.1 are seriously unstable, even towards the end of the run, when the average value is close to the desired value. This learning rate is too high for obtaining stable weights. On the other hand, figure 3 shows that if the learning rate is lower, the final weights are much more stable. However, the weights in figure 3 take of the order of 500 sequences to approach the right values, whereas the high learning rate of figure 2 only took around 50 sequences. This behaviour was repeated in all the runs.

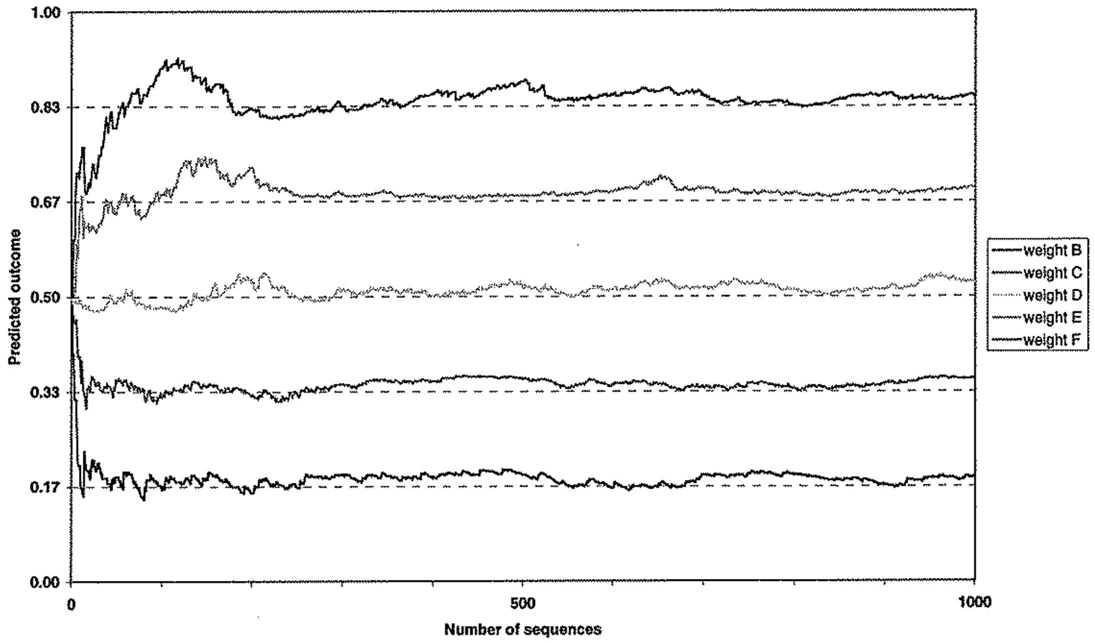

Figure 4: Weight movements from a typical run using temporal coherence

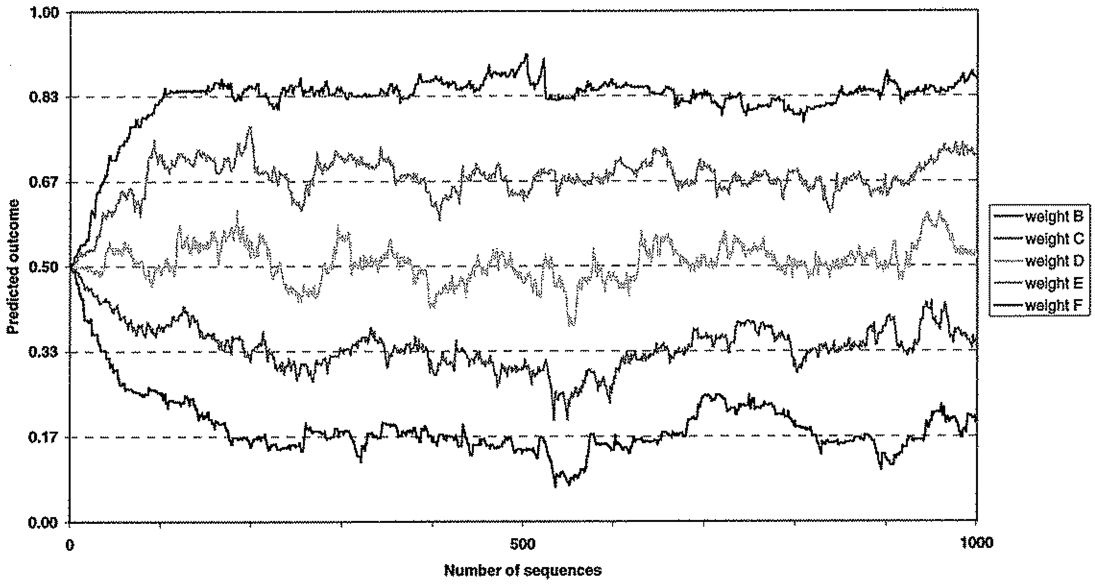

Figure 5: Weight movements from a typical run using delta-bar-delta

Figures 4 and 5 show typical results from our temporal coherence algorithm, and delta-bar-delta, respectively. The TC algorithm requires no *a priori* parameter setting. We found the delta-bar-delta algorithm to be sensitive to its meta-parameters (starting rate, adjustment step size and ratio, and exponential decay factor of weight changes). Figure 5 shows the best result from several runs we performed with different values of the meta-parameters, guided by the values used in Jacobs (1988). We used parameter settings of: $\kappa = 0.01$, $\phi = 0.333$, $\theta = 0.7$, $\varepsilon_0 = 0.03$ and λ=0.3. We tried a number of other parameter settings, none of which performed any better than the chosen set.

Figure 4 shows that TC yields fast initial movement towards the target values, and enables the weights to stay close to the target values thereafter. In figure 4, the weights approach the right values within about 50 sequences (as least as fast as the quick but unstable learning of figure 1), and become as stable as the slow learning of figure 2 within about 250 sequences (twice as fast as the slow learning rate). Figure 5 shows that delta-bar-delta did not do as well as temporal coherence.

Figures 2 through 5 are all based on the random sequences generated with one particular starting seed. We repeated the experiments with many starting seeds, and saw very similar results. The graphs presented here are typical. However, to avoid reliance on the particular sequences involved, figure 6 presents a summary of the performance of the different algorithms, showing their progress towards the values sought. Each trace in figure 6 represents the squared error, summed over all weights, after each sequence, for a given algorithm, averaged over 10 runs.

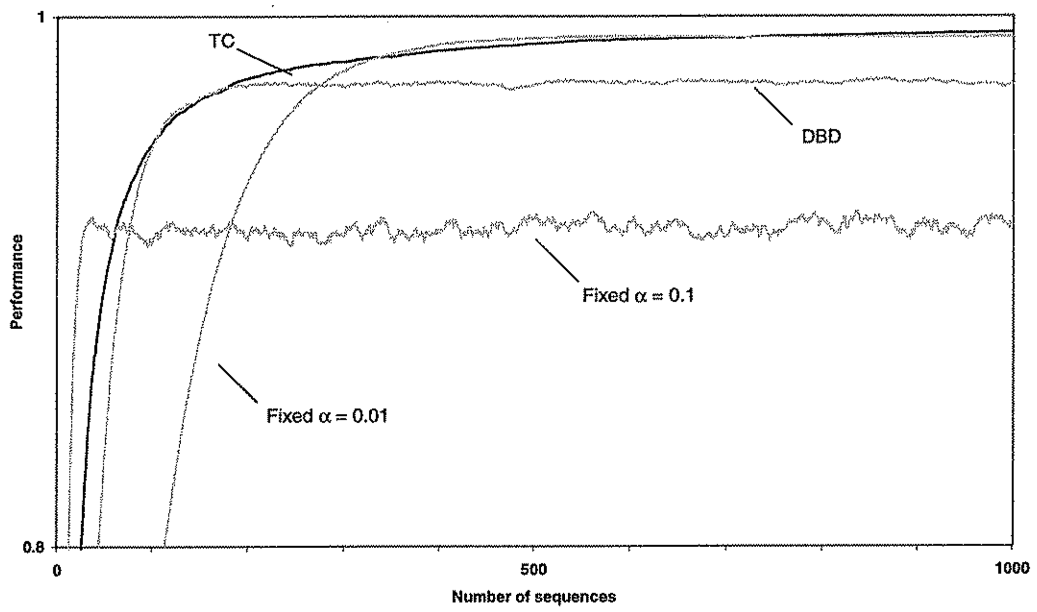

Figure 6: Performance averaged over 10 runs for various learning rate methods.[3]

Figure 6 shows that the fast fixed learning rate has the best initial scores, but the performance trace for that option shows that it never reaches accurate values. It remains unstable and well below the other traces from about sequence 50 onwards. Figure 6 shows that temporal coherence has the best overall performance. It is almost as fast as the unstable learning rate initially, achieves a closer final approach at the end

[3] For presentation convenience, the sum-of-squared error is first normalised to the range [0,1] where 1 is the error at the start of the run, and 0 is no error (achieved when the weight equals the true value), then charted as g, the inverse of error: $g = 1 - \sum_{i=1}^{5}(w_i - v_i)^2 \Big/ \sum_{i=1}^{5}(0.5 - v_i)^2$

w_i is the weight for state i, v_i is the true value for state i, 0.5 is the initial value for all weights at the start of the run.

of the runs than any of the other methods tested, and is either closer to the right values or reaches them sooner, when compared to the other methods.

6. Test domain two: learning the values of chess pieces

In addition to the simple bounded walk problem, we also tested our methods in a more complex domain. The chosen task was the learning of the values of chess pieces by a minimax search program, in the absence of any chess-related initial knowledge other than the rules of the game.

We attempted to learn suitable values for five adjustable weights (Pawn, Knight, Bishop, Rook and Queen), via a series of randomised self-play games. Learning from self-play has the important advantage that no existing expertise (human or machine) is assumed, and thus the method is transferable to domains where no existing expertise is available. Using this method it is possible to learn relative values of the pieces (Beal and Smith, 1997) that perform at least as well as those quoted in elementary chess books. The learning performance of the temporal coherence scheme was compared with the learning performance using fixed learning rates, and with delta-bar-delta.

The TD learning process is driven by the differences between successive predictions of the probability of winning during the course of playing a series of games. In this domain each temporal sequence is a set of predictions for all the positions reached in one game, each game corresponding to one sequence in the learning process. The predictions vary from 0 (loss) to 1 (win), and are determined by a search engine that uses the adjustable piece weights to evaluate game positions. The weights are updated after each game. Details of the game tree search program can be found in Appendix B.

At the start of the experiments all piece weights were initialised to one, and a series of games were played using a 5-ply search. To avoid the same games from being repeated, the move lists were randomised. This had the effect of selecting at random from all tactically equal moves, and the added benefit of ensuring a wide range of different types of position were encountered.

6.1 Converting evaluation scores into prediction probabilities

In order to make use of temporal differences, the values of positions were converted from the evaluation provided by the chess program into estimations of the probability of winning. This was done using a standard sigmoid squashing function. Thus the prediction of probability of winning for a given position is determined by:

$$P = \frac{1}{1+e^{-v}}$$

where v = the 'evaluation value' of the position.

This sigmoid function has the advantage that it has a simple derivative, thus simplifying the implementation of the TD algorithm.

$$\frac{dP}{dw} = P(1-P)$$

hence the derivative is a simple function of the prediction.

6.2 Results from the game-playing domain

To visualise the results obtained from the various methods for determining learning rates, we present graphs produced by plotting the weight movements for each of the five piece values over the course of runs consisting of 2,000 game sequences each. As the absolute values of the piece weights is unimportant compared with their *relative* values, the graphs are normalised so that the average value of the Pawn weight over the last 200 game sequences is 1. This enables comparison with the widely quoted elementary values of Pawn = 1, Knight and Bishop = 3, Rook = 5 and Queen = 9. As in the bounded walk domain, the number of sequences in each run is large enough that the values reach a quasi-stable state of random noise around a learnt value. To confirm that the apparent stability is not an artifact, each experiment was repeated 10 times, using different random number seeds.

Figure 7 shows the weights learnt by the temporal coherence system over a typical single run. In this figure we can see that the learning process is essentially completed after 500 sequences, and that the weights remain fairly stable for the remainder of the sequences. This was typical of all runs, as reflected in figure 12 later.

Figure 8 shows the weights achieved using fixed settings of $\alpha = 0.05$ and λ=0.95 over a typical single run. These settings offered a good combination of learning rate and stability from the many fixed settings that we tried. A lower learning rate produced more stable values, but at the cost of further increasing the number of sequences needed to establish an accurate set of relative values. Raising the learning rate makes the weights increasingly unstable.

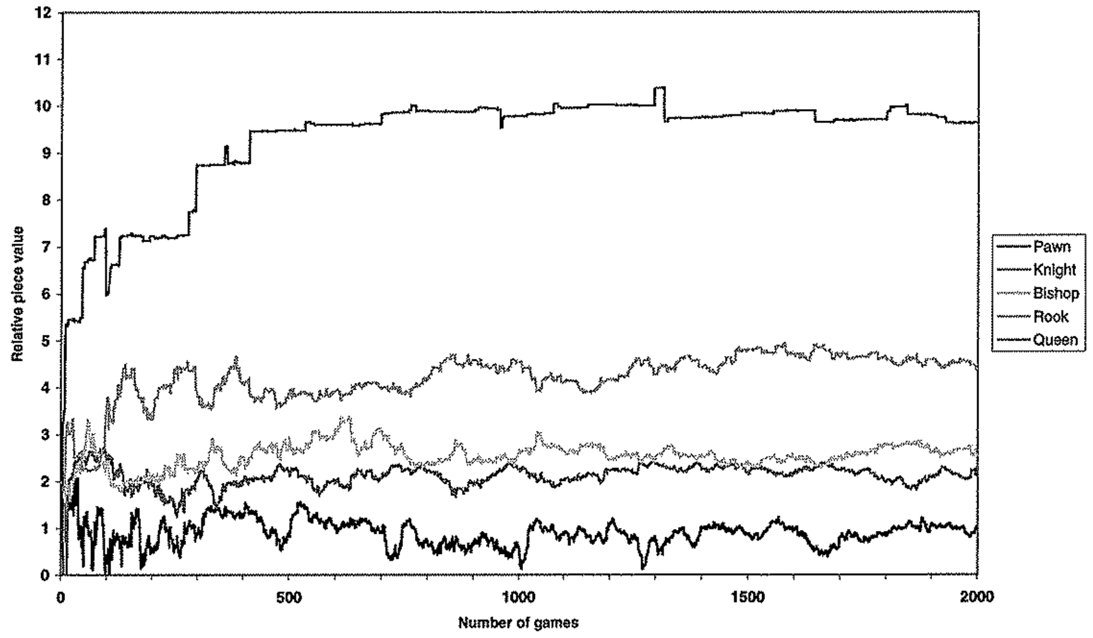

Figure 7: Weight movements from a typical single run using temporal coherence

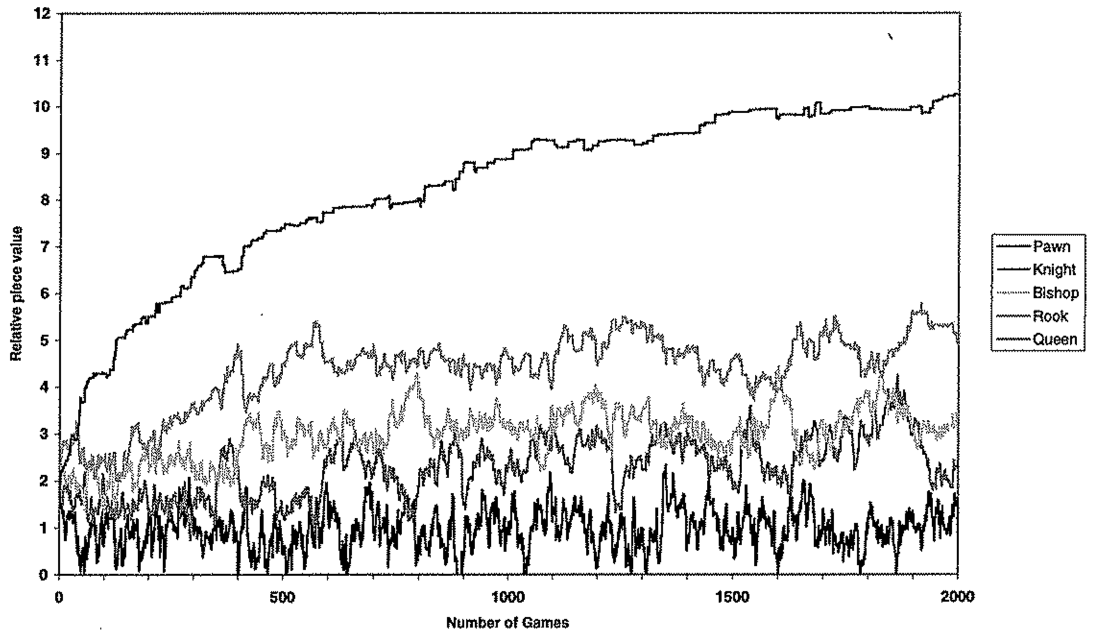

Figure 8: Weight movements from a typical single run using fixed α=0.05

Comparing figure 8 with figure 7 we can see that the piece values are unstable compared with figure 7, and the relative ordering of the pieces is not consistent over the length of the run. In addition, comparing with figure 7 we can see that the speed of learning is significantly slower in the fixed-rate run, with a significant amount of the learning occurring after 500 sequences.

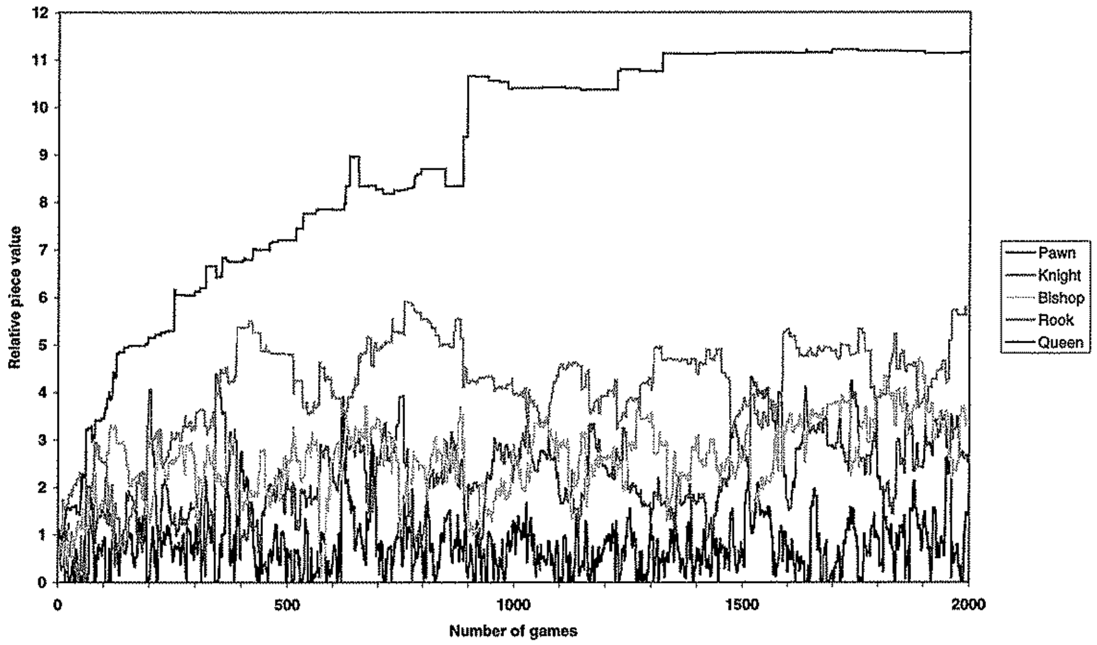

Figure 9: Weight movements from a typical single run using delta-bar-delta.

Figure 9 shows the results achieved from a typical run using delta-bar-delta. Comparing this figure with figure 7, we can see that the weights produced using DBD are much less stable than those produced by TC, and the relative ordering of the pieces is not consistent. For this domain we used meta-parameters of: $\kappa = 0.035$, $\phi = 0.333$, $\theta = 0.7$ and $\varepsilon_0 = 0.05$, guided by data presented by Jacobs (1988) and preliminary experiments in this domain. For λ, which DBD does not set, we used λ=0.95 derived from our experience with the fixed rate runs. We tried a number of other meta-parameter settings, none of which performed better than the chosen set. It is possible that a comprehensive search for a better set of meta-parameters might have improved the performance of the delta-bar-delta algorithm, but given the computational cost of a single run of 2000 sequences, we were unable to attempt a systematic search of all the meta-parameter values. Other runs with different random seeds showed similar behaviour to figure 9.

Figures 7-8 showed typical results obtained from single runs, determined by a random number seed. We repeated the experiment ten times, using ten different seeds. Figures 10 and 11 show averaged weight movements, to confirm that the characteristics seen in the single runs are consistent behaviour.

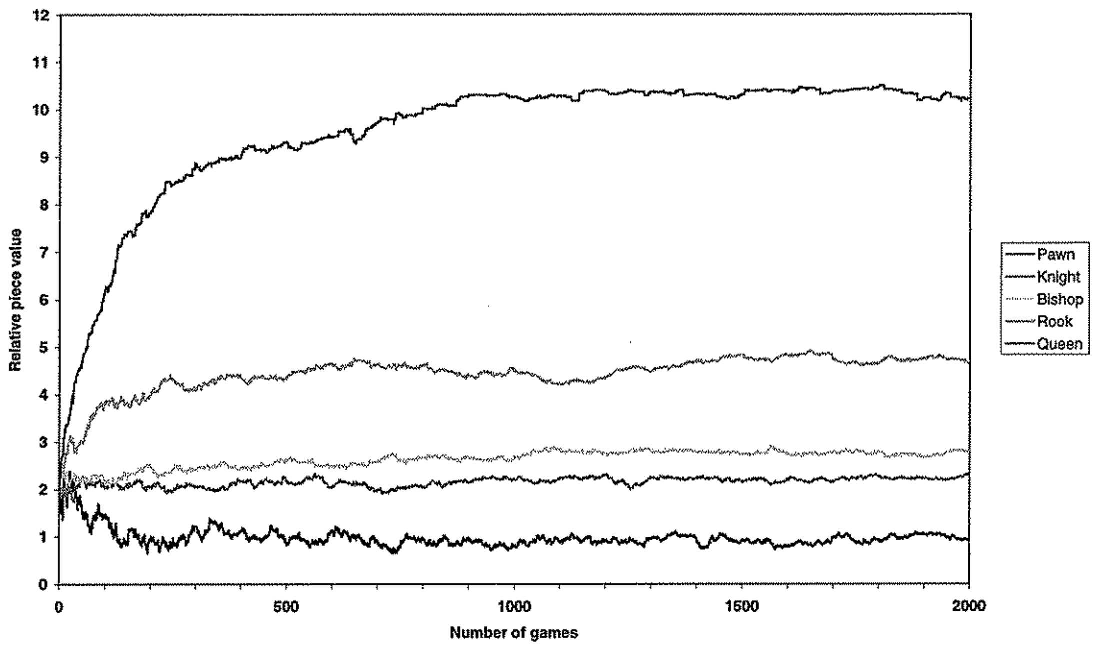

Figure 10: Average weight movements from 10 runs using temporal coherence

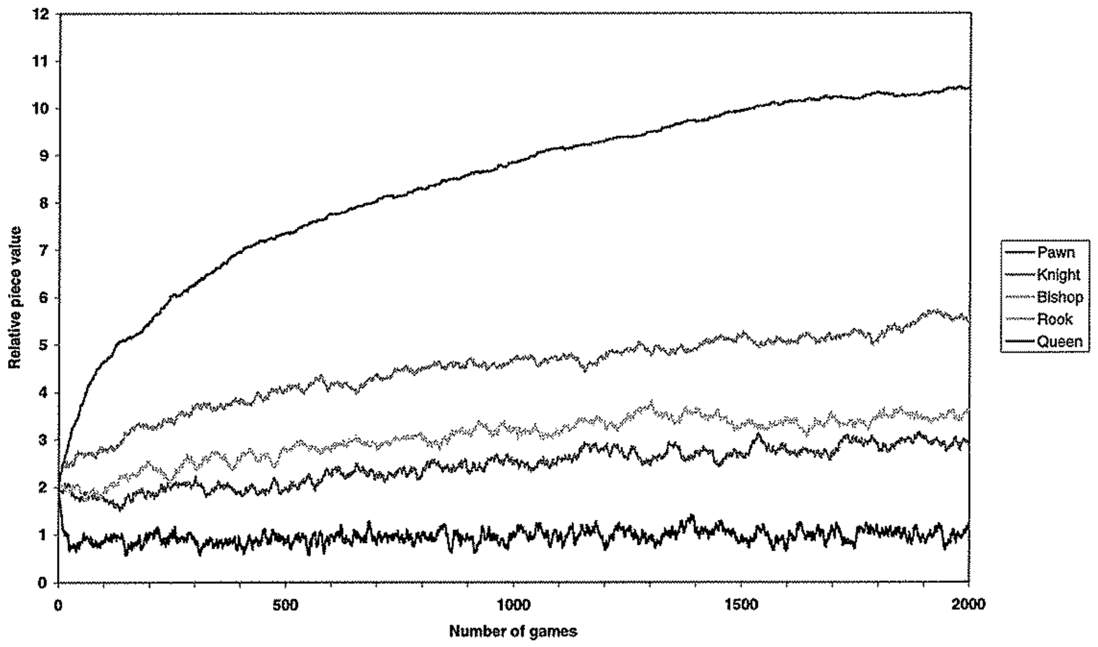

Figure 11: Average weight movements from 10 runs using a fixed α of 0.05

Figures 10 and 11 show the piece values averaged from 10 runs using TC, and a fixed learning rate of 0.05 respectively. In figure 10 all traces have approached their final values after about 900 sequences (some weights much sooner), whereas in figure 11 the traces do not approach final values until around 1500 sequences (and even then have more systematic movement to make). Thus, as in the single runs, the TC algorithm is faster to approach final values, and more stable once they are reached.

To confirm that the learnt piece values had approached 'correct' values, a match was played pitting the learnt values against the values widely quoted in elementary chess books (see appendix B). One program used the well-known values. The other used the learnt weights, as a check that the learnt values were at least as good as the standard ones. The piece values learnt by the TC runs achieved a score of 58% against the well-known values.

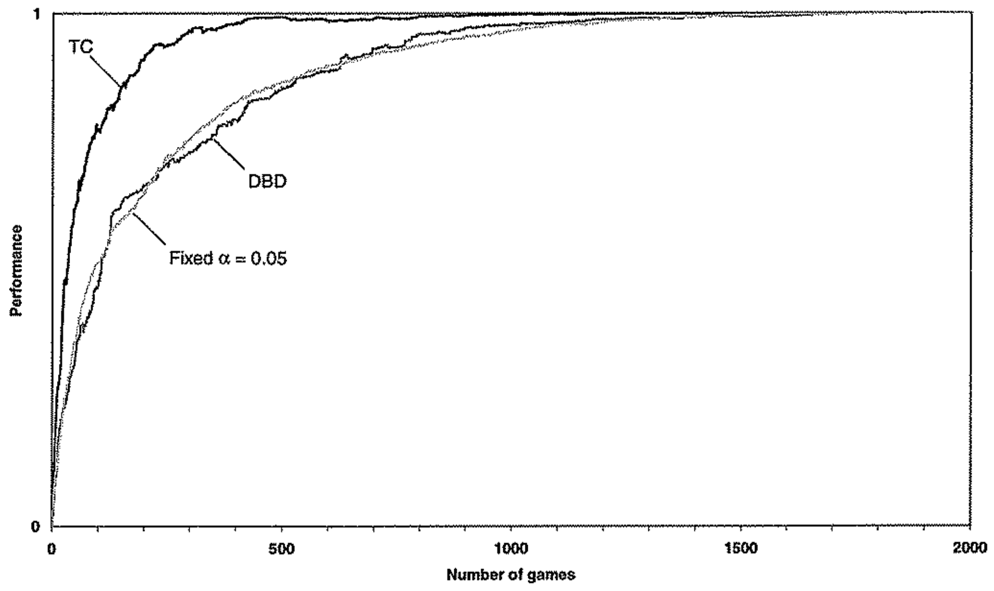

Figure 12: Performance over an average of 10 runs

Figure 12 shows the average piece values over 10 runs for the various methods, combined into a single 'performance' measure, similar to that presented for the bounded walk results. From this figure we can see that delta-bar-delta does not improve much on a carefully-chosen fixed learning rate, but that TC clearly produces faster learning.

7. Conclusions

We have described new extensions, temporal coherence, and prediction decay, to the temporal difference learning method that set the major control parameters, learning rate and temporal discount, automatically as learning proceeds. The resulting TD algorithm does not require initial settings for α and λ, and has been tested in depth on two domains.

The results from the bounded walk domain demonstrated both faster learning and more stable final values than a previous algorithm and the best of the fixed learning rates. The new methods were also successfully applied to a more complex task, that of learning relative piece values in the game of chess, without supplying any domain-specific knowledge. In this domain also, the results from the new algorithm showed faster learning and more stable final values, including those attained by the best of the fixed learning rates.

In our comparisons with the delta-bar-delta algorithm, we tried to find good parameter sets for DBD, which requires four meta parameters instead of the one control parameter, α. We tried several different (meta-) parameter sets in each domain, but were unable to find a set of parameters that improved performance over the results presented in sections 5.1 and 6.2. It is possible that a systematic search for better set of meta-parameters in each of the domains might improve performance. However, it is a major drawback for the method that it requires its meta-parameters to be tuned to the domain it is operating in. It is part of the advantage of the methods presented here that they do not require a search for good parameter values.

The experimental results have demonstrated that the temporal coherence plus prediction decay algorithm achieves three benefits: (1) removal of the need to specify parameters (2) faster learning and (3) more stable final values.

References

Almeida, Langlois, & Amaral, 1998, On-Line Step Size Adaptation, *Technical Report RT07/97 INESC*, 9 Rua Alves Redol, 1000 Lisbon, Portugal.

Beal, D. F., and Smith, M.C., 1997, Learning piece values using temporal differences. *International Computer Chess Association Journal*, ***20*** (3): 147-151.

Dayan, P., 1992. The convergence of TD(λ) for general λ. *Machine Learning*, *8*: 341-362.

Jacobs, R. A., 1988. Increased rates of convergence through learning rate adaptation. *Neural Networks*, ***1***: 295-307.

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

Marsland, T.A., 1992. Computer Chess and Search. In S. Shapiro (ed.), *Encyclopaedia of Artificial Intelligence* (2nd edition). J. Wiley & Sons.

Schraudolph, N.N., 1998, Online Local Gain Adaptation for Multi-layer Perceptrons, *Technical Report IDSIA-09-98*, IDSIA, Corso Elvezia 36, 6900 Lugano, Switzerland.

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

Sutton, R.S., 1992. Adapting bias by gradient descent: an incremental version of delta-bar-delta. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 171-176.

Sutton, R.S., and Singh, S.P., 1994. On step-size and bias in temporal-difference learning. *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pp. 91-96.

Tesauro, G., 1992. Practical issues in temporal difference learning. *Machine Learning*, **8**: 257-277.

APPENDIX A Setting the temporal discount parameter using prediction decay

Prediction decay is the average deterioration in prediction quality per timestep. A *prediction quality* function measures the correspondence between a prediction and a later prediction (or end-of-sequence outcome). The observed prediction qualities for each temporal distance are averaged. An exponential curve is then fitted to the average prediction qualities against distance (Figure 13 shows an example), and the exponential constant of that fitted curve is the prediction decay, ψ. We set the TD discount parameter λ, to1 initially, and $\lambda = \psi$ thereafter. In the experiments reported, ψ (and hence λ) were updated at the end of each sequence.

The *prediction quality* measure, $Q_d(p, p')$ we used is defined below. It is constructed as a piece-wise linear function with the following properties:

i. When the two predictions p and p', are identical, $Q_d = 1$. (The maximum Q_d is 1)
ii. As the discrepancy between p and p' increases, Q_d decreases.
iii. When one prediction is 1 and the other is 0, then $Q_d = -1$. (The minimum Q_d is -1)
iv. For any given p, the average value of Q_d for all possible values of p', such that $0 \leq p' \leq 1$, equals 0. (Thus random guessing yields a score of zero.) This property is achieved by the quadratic equations in the definition below.

We achieve these properties by defining:

$$Q_d(p,p') = \begin{Bmatrix} p \geq .5 & : & F(p,p') \\ p < .5 & : & F(1-p, 1-p') \end{Bmatrix} \qquad F(p,p') = \begin{Bmatrix} p \leq s & : & \begin{Bmatrix} r \leq x & : & 1 - r/x \\ r > x & : & -p(r-x)/(p-x) \end{Bmatrix} \\ \\ p > s & : & \begin{Bmatrix} r \leq y & : & 1 - r/y \\ r > y & : & -p(r-y)/(p-y) \end{Bmatrix} \end{Bmatrix}$$

where:

$r = |p - p'|$

$s =$ solution of $2s^2 - 5s + 1 = 0$

$x =$ solution of $2(1+p)x^2 - 4px + p = 0$

$y =$ solution of $(1+p)y^2 + \left(2 - 2p - p^2\right)y - (1-p)^2 = 0$

p is the current prediction, p' is an earlier prediction, and d refers to the temporal distance between p and p'. Predictions lie in the range [0, 1].

It is assumed that the learning occurs over the course of many multi-step sequences, in which a prediction is made at each step; and that the sequences are independent. To form a prediction pair, both predictions must lie within the same sequence.

$\overline{Q}_d$ is the average prediction quality over all prediction pairs separated by distance d observed so far. For this purpose, the terminal outcome at the end of the sequence is treated as a prediction. At every prediction, the $\overline{Q}_d$ are incrementally updated.

An example graph from our experimental results is given in Figure 13. The exponential curve is fitted to the averaged prediction qualities by minimising the mean squared error between the exponential curve and the observed $\overline{Q}_d$ values. To prevent rarely occurring distances from carrying undue weight in the overall error, the error term for each distance is weighted by the number of observed prediction pairs. Thus we seek a value of ψ which minimises:

$$\sum_{d=0}^{l} \left(\overline{Q}_d - \psi^d\right)^2 N_d$$

where $\overline{Q}_d$ is the average prediction quality for distance d, and N_d is the number of prediction pairs separated by that distance, and l is the length of the longest sequence in the observations so far. In the experiments reported here the value for ψ was obtained by simple iterative means, making small incremental changes to its value until a minimum was identified. Values for ψ (and hence λ) were updated at the end of each sequence.

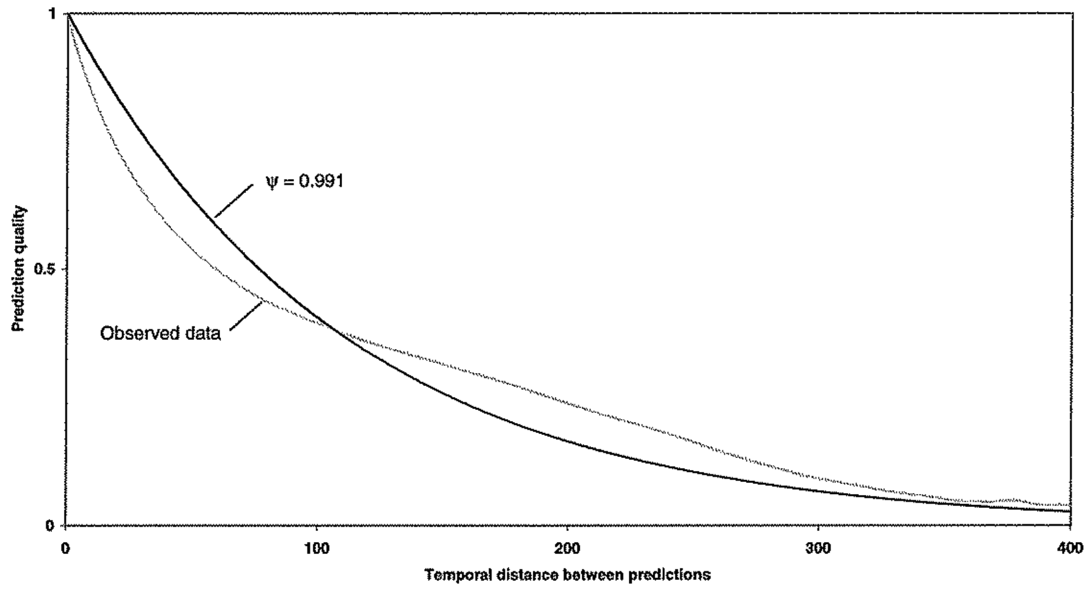

Figure 13: Fit of the prediction quality temporal decay to observed data from the game domain, at the end of a run of 2000 games.

Figure 13 shows an example of the fit of a value for ψ to the observed prediction qualities, as described above. This example is typical of the fit to the observed data in the game domain. In that domain ψ was fairly stable in the range 0.990 - 0.993 during the test runs. In the random walk domain, ψ was typically around 0.7.

APPENDIX B Domain specific details

Learning piece values in chess: Probably the first heuristic to be taught to most beginners is the value of the pieces: that knights and bishops are worth about three pawns; rooks about five pawns; and the queen about nine pawns (the king is not given a value as it cannot be captured). Thus under this scheme, it would be considered a fair exchange to trade a rook for a bishop and two pawns, or a queen for two bishops and a knight.

This simple numerical scheme provides a crude evaluation of how 'good' (i.e. likely to lead to victory) a given position is. Such a scheme (or others very similar) provides the backbone for almost all chess-playing computer programs' evaluation functions, including that of IBM's Deep Blue, although for high performance play the basic scheme is augmented with numerous, and very elaborate, additional scoring terms.

The system attempts to learn values for the pieces via a series of randomised self-play games. It does not benefit from seeing the play of a well-informed opponent against it, nor is it given games played by experts to examine. At the start of the experiments all pieces values are initialised to one, and so the first game is played entirely at random, the system not even knowing that it is good to capture opponent's pieces and preserve your own.

The same method would be directly applicable to many other two-person, perfect information games, e.g. Checkers, Shogi and Chinese Chess. As well as learning piece values, the same method could be used to optimise weights for other evaluation function terms, such as mobility, centre control etc.

The search engine: The chess-playing search engine used for the experiments was a simple, conventional one, using a full-width iteratively deepened search, with alpha-beta pruning and a captures-only quiescence search at the full-width horizon (Marsland, 1992). The search was made more efficient by the use of a transposition table. The evaluation function consisted of the material score only, with the move choice being made randomly from the materially-equal moves.

The principal variation: The effect of the full-width minimax search algorithm is to select a sequence of moves that represents best play by both sides (as defined by the evaluation function). This line of play is referred to as the *principal variation.* The evaluation score from the position at the end of the principal variation (the *principal position*) is 'backed up' to the root of the search. Thus from a given position, a minimax search returns a value that corresponds to the evaluation of the position at the end of the principal variation.

The learning process: The TD learning process is driven by the differences between successive predictions of the probability of winning during the course of playing a series of games, and the piece values updated after each game. For implementation convenience, in the experiments reported in this paper the weight adjustments were not computed incrementally, but computed and adjusted at the end of each game.

During the course of each individual game, after each move a record was created of the value returned by the search, and the corresponding principal position. At the end of each game, these values are converted into prediction probabilities via the squashing function, and adjustments made to the weights according to the differences between successive predictions, and the differences in piece counts of the corresponding principal positions.

Match details: The piece value weights learnt using TC were tested by playing a match between two identical search engines, one using the widely quoted standard values (Pawn=1, Knight=3, Bishop=3, Rook=5, Queen=9), the other using the newly learnt values. To avoid fluctuations in the weights due to noise from the stochastic nature of the domain, the piece values used in the match were calculated by averaging over last 10% of games in the 10 TC runs.

Games that ended in mate were scored as 1 point for the winning side. Games that ended in a draw according to the laws of chess (stalemate, repetition, insufficient material) were scored as 1/2 point for both sides. Games that were unfinished after 400 ply (200 moves each) were scored as win for one side only if both programs' evaluations agreed one side was ahead on material, otherwise the game was scored as a draw.

The weights achieved by the temporal coherence runs won in a match against the widely quoted standard piece values. A match of consisting of 5000 games was played, of which the TC weights won 2658, drew 483, and lost 1859, giving a match score of 58%.