Temporal Coherence and Prediction Decay in TD Learning

Don F. Beal and Martin C. Smith
Department of Computer Science
Queen Mary and Westfield College
University of London
Mile End Road
London E1 4NS
UK

Abstract

This paper describes improvements to the temporal difference TD(λ) learning method. The standard form of the TD(λ) 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 parameter, which affects the stability of the process as well as the number of observations required. Our extension to the TD(λ) algorithm automatically sets and subsequently adjusts these parameters. The learning rate adjustment is based on a new concept we call temporal coherence (TC). The experiments reported here compare the extended TD(λ) algorithm performance with human-chosen parameters and with an earlier method for learning rate adjustment, in a complex game domain. The learning task was that of learning the relative values of pieces, without any initial domain-specific knowledge, and from self-play only. The results show that the improved 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 a comparison method.

1 Introduction

Two major parameters that control the behaviour of Sutton's [1988] temporal difference algorithm TD(A.) 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 more complex real-world problems. Jacobs [1988] presented the [4]delta-bar-delta' algorithm for adjusting α during the learning process. We compared the performance of delta-bar-delta with our algorithm on our sample domain. 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 these parameters according to the learning experiences themselves. We present results that show that this method is effective, and in our sample domain 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.

2. 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's TD(λ) algorithm can be summarised by the following formula. Given a vector of adjustable weights, w, and a scries of successive predictions, *P,* weight adjustments are determined at each timestep according to:

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

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(λ) 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.

3. Temporal Coherence: adjusting α

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. 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_i at timestep t is defined as:

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

The actual change made to weight w_i after game is:

$$\Delta w_i = c\, \alpha_i \sum_{t=1}^{end-1} r_{i,t}$$

where α_i is the individual learning rate for weight w_i and c is a learning rate for the whole process.

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). The individual learning rate, α_i for each weight w_i is set to be the ratio of net recommended change to absolute recommended change:

$$\alpha_i = \frac{|N_i|}{A_i}$$

with the following definitions and update rules:

$$N_i \leftarrow N_i + \sum_{t=1}^{end-1} r_{i,t}$$

$$A_i \leftarrow A_i + \sum_{t=1}^{end-1} |r_{i,t}|$$

$r_{i,t}$ = recommended change for weight w_i at prediction t
$P_1..P_{end-1}$ are predictions, P_{end} is the final outcome

The operational order is that changes to w_i are made first, using the previous values of N_i, *A,* and α_i , then N_i, A_i and α_i arc updated. The parameter *c* has to be chosen, but this does not demand a choice between fast learning and eventual stability, since it can be set high initially, and the α_i then provide automatic adjustment during the learning process. All the α_i are initialised to 1 at the start of the learning process.

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 experiments reported in this paper, update at the end of each game sequence is natural and convenient to implement.

4. Prediction Decay: determining 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}*

The advantage of using prediction decay is that it enables $\mathbf{TD(\lambda)}$ to be applied effectively to domains without prior domain knowledge, and without prior experiments to determine an effective 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.

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 1 shows an example), and the exponential constant of that fitted curve is the prediction decay, ψ. We set the TD discount parameter λ, to 1 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_y the average value of Q_d for all possible values of p', such that $0 \le p' \le 1$, equals 0. (Thus random guessing yields a score of zero.) This property is achieved by the quadratic equations in the definition below.

We achieve all these properties by defining:

$$Q_d(p,p') = \left\{ \begin{array}{lll} p \ge .5 & : & F(p,p') \\ p < .5 & : & F(1-p,1-p') \end{array} \right\}$$

$$F(p,p') = \left\{ \begin{array}{lll} p \le s & : & \left\{ \begin{array}{ll} r \le x & 1 - r/x \\ r > x & -p(r-x)/(p-x) \end{array} \right\} \\ p > s & : & \left\{ \begin{array}{ll} r \le y & 1 - r/y \\ r > y & -p(r-y)/(p-y) \end{array} \right\} \end{array} \right\}$$

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 $(1+p)y^2 + (2-2p-p^2)y - (1-p)^2 = 0$

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.

$\bar{Q}_d$ 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 $\bar{Q}_d$ are incrementally updated.

An example graph from our experimental results is given in Figure 1. This example is typical of the fit to the observed data in the test domain. The exponential curve is fitted to the average prediction quality by minimising the mean squared error between the exponential curve and the observed $\bar{Q}_d$ values. ψ was fairly stable in the range 0.990 - 0.993 during the test runs.

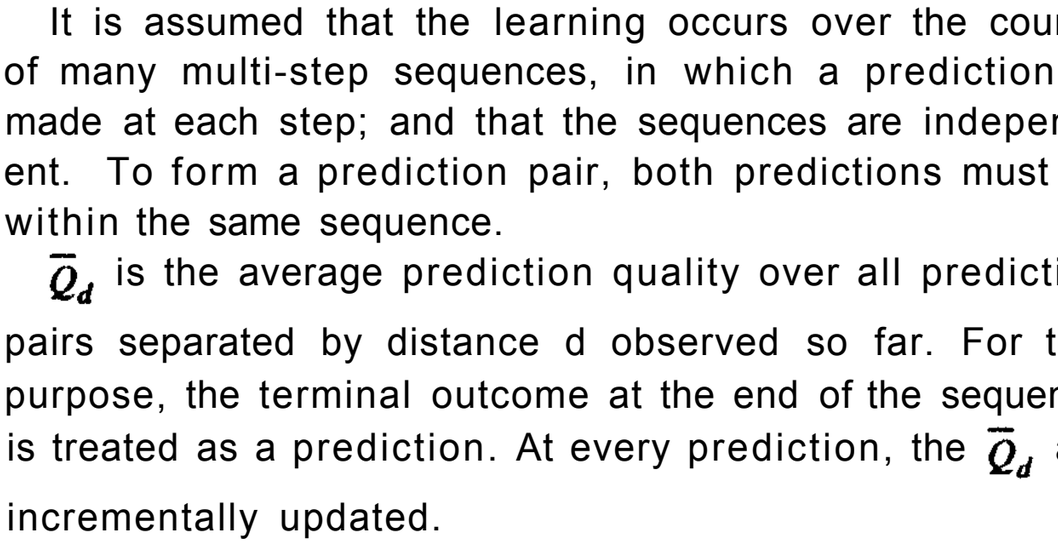

Figure 1: Fit of the prediction quality temporal decay to observed data from the game domain, after 2000 games.

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:

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

where $\bar{Q}_d$ is the average prediction quality for distance d_y 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.

[1] By expending sufficient computation time to repeatedly rerun the experiments we found somewhat better values for λ.

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.

5. 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 our test domain. 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-parametcrs 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.

6. Learning in a complex domain

We tested our methods in a complex game 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. Beal and Smith 11997; 1998J show that, using this method, it is possible to learn relative values of the pieces 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.

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 Evaluation predictions

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 giver, position is determined by:

$$P(v) = \frac{1}{1+e^{-v}} \qquad \text{where} \quad v = \sum_i w_i f_i$$

where v is the 'evaluation value', and/, is the piece count differential for piece type i at the given position.

This sigmoid function has the advantage that it has a simple derivative:

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

6.2 Results

To visualise the results obtained from the various methods for determining learning rates, we present graphs produced by plotting weights for each of the five piece values over the course of runs consisting of 2,000 game sequences each. Beal and Smith [1997) show that this method is capable of learning relative piece values that compare favourably with the widely quoted elementary values of Pawn = 1, Knight and Bishop = 3, Rook = 5 and Queen = 9. 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 artefact, each experiment was repeated 10 times, using different random number seeds.

Figure 2 shows the average weights achieved using fixed settings of a = 0.05 and X = 0.95 over a series of 10 runs. 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 3 shows the average weights produced by the delta-bar-delta algorithm over 10 runs. For this domain

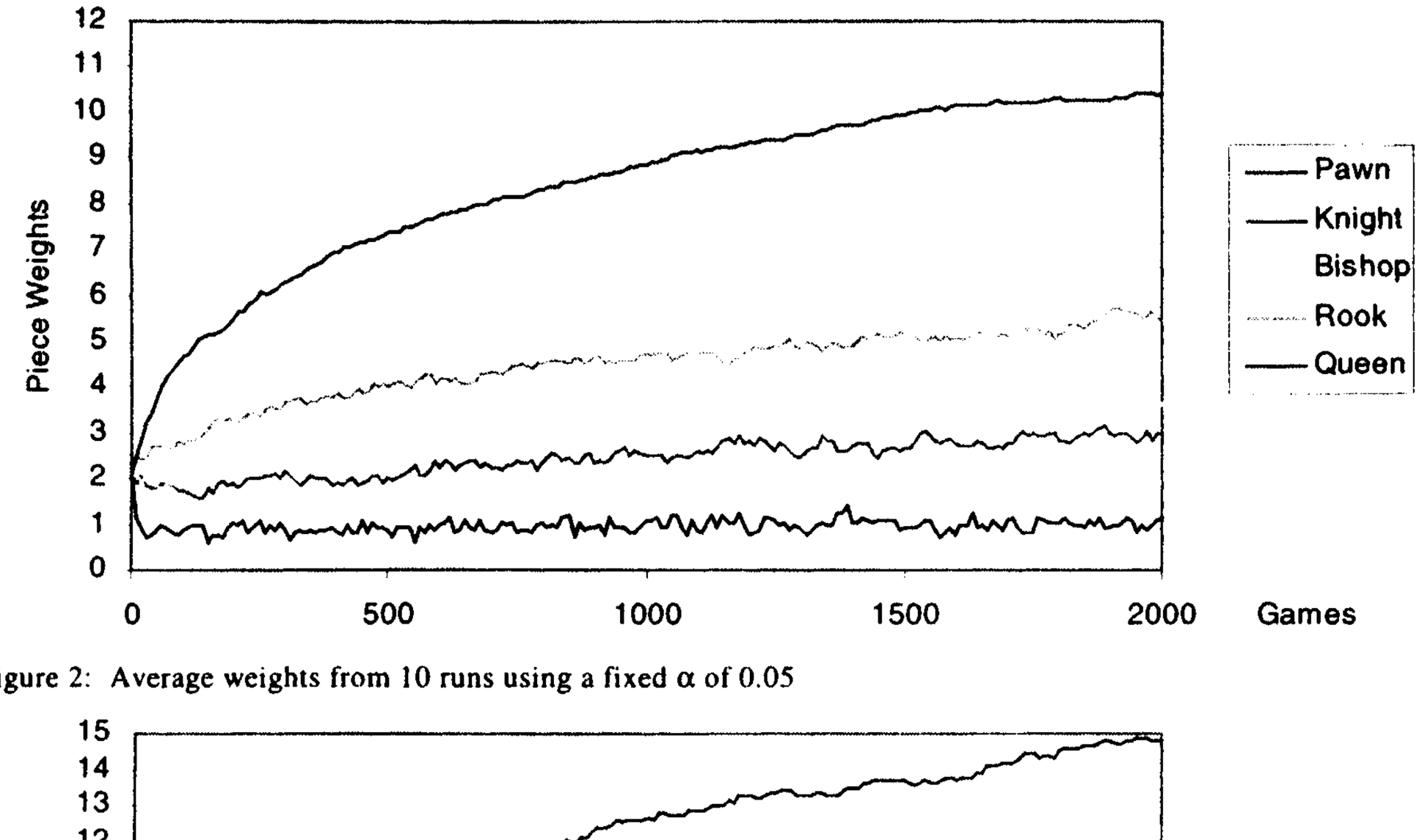

Figure 2: Average weights from 10 runs using a fixed α of 0.05

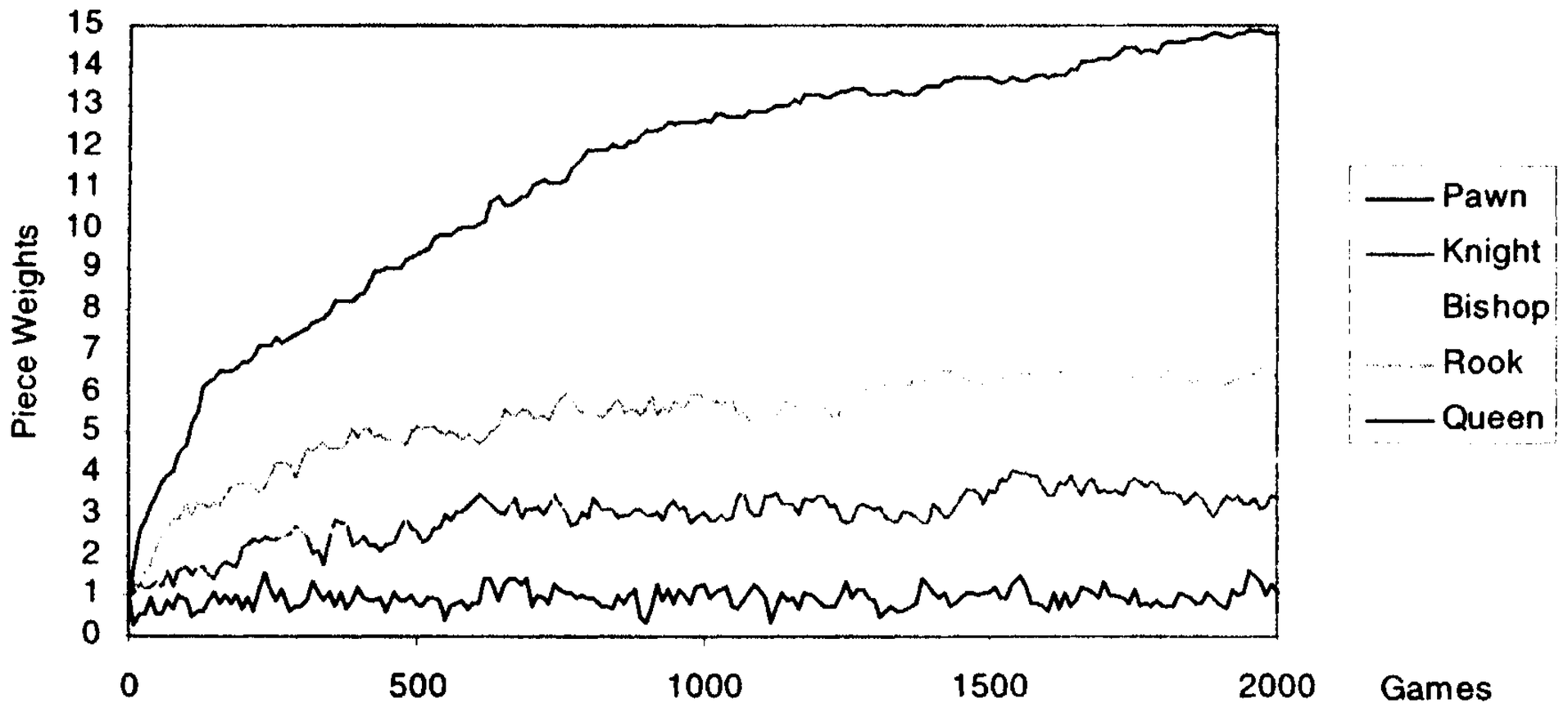

Figure 3: Average weights from 10 runs using delta-bar-delta

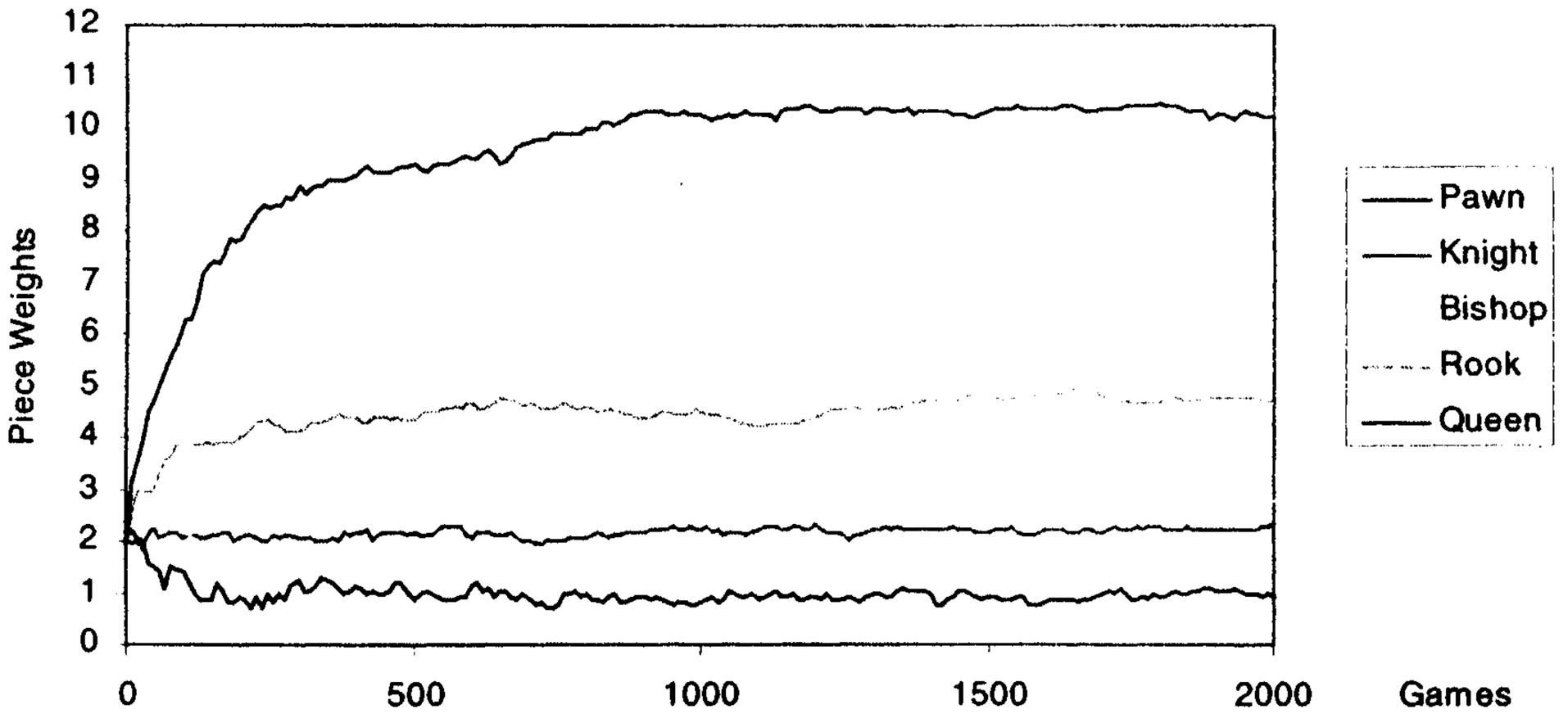

Figure 4: Average weights from 10 runs using temporal coherence

we used meta-parametcrs of K = 0.035, ϕ = 0.333, θ = 0.7 and ε_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.

Figure 4 shows the average weights obtained using temporal coherence. It can be seen from the figure that all traces have approached their final values after about 900 sequences (some weights much sooner). Comparing with figures 2 and 3 it can be seen that the TC algorithm is faster to approach final values, and more stable once they are reached. In addition, the traces in figure 4 are smoother than in figures 2 and 3, representing less variation due to noise in the individual runs.

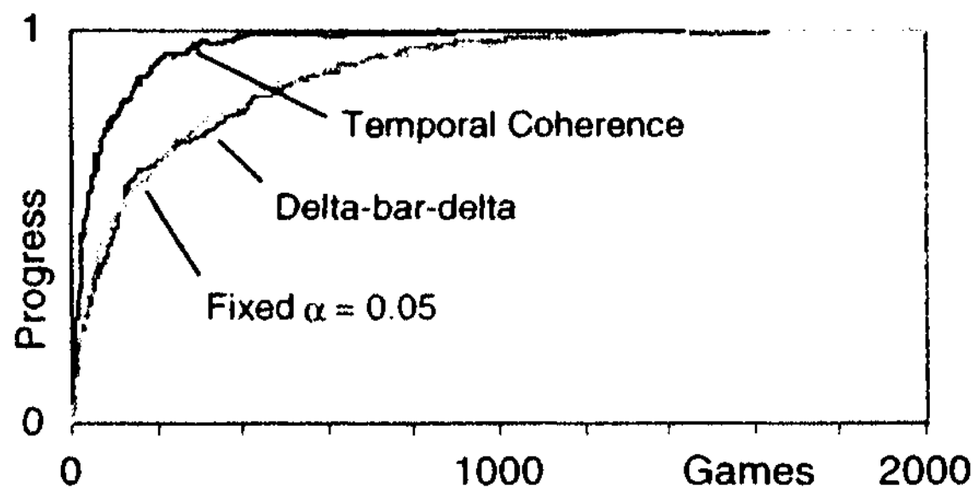

Figure 5: Progress over an average of 10 runs

Figure 5 shows the average piece values over 10 runs for the various methods, combined into a single term measuring progress toward the values achieved at the ends of the runs. From this figure we can see that delta-bar-delta does not improve much on a carefully-chosen fixed learning rate, and that temporal coherence clearly produces faster learning. The TC and fixed-a final weights were not significantly different.

To confirm that the learning process had produced satisfactory values, a match was played pitting the learnt values against the values widely quoted in elementary chess books Q=9, R=5, N/B=3, P=I. One program used those values, the other used the weights learnt using temporal coherence, as a check that the learnt values were at least as good as the standard ones. In a match of 2,000 games the TC values achieved a score of 58% against the standard values (won= 1,119 lost=781 drawn=100).

7. Conclusions

We have described two new extensions, temporal coherence, and prediction decay, to the temporal difference learning method that set and adjust the major control parameters, learning rate and temporal discount, automatically as learning proceeds. The resulting TD algorithm has been tested in depth on a complex domain.

The results demonstrated both faster learning and more stable final values than a previous algorithm and the best of the fixed learning rates. The test domain was one in which values were learnt without supplying any domain-specific knowledge. We also tested the TC algorithm in a bounded walk domain [Sutton, 1998J, and found similar advantages.

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-parametcrs 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. The methods presented here do not require a search for good parameter values.

The experimental results demonstrated that the temporal coherence plus prediction decay algorithm achieved three benefits: (1) automatic setting and adjustment of 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.

Beal, D.F., and Smith, M.C., 1998, Temporal Difference Learning for Heuristic Domains. *JCIS'98 Proceedings Vol(I), Oct 23-28, 1998:* 431-434.

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

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,* 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,* 91-96.

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