Model-Based Reinforcement Learning for Alternating Markov Games

Drew Mellor

School of Electrical Engineering and Computer Science
The University of Newcastle
Callaghan, 2308, Australia
Telephone: (+612) 4921 6034
Facsimile: (+612) 4921 6929
dmellor@cs.newcastle.edu.au

Abstract. Online training is a promising technique for training reinforcement learning agents to play strategy board games over the internet against human opponents. But the limited training experience that can be generated by playing against real humans online means that learning must be data-efficient. Data-efficiency has been achieved in other domains by augmenting reinforcement learning with a model: model-based reinforcement learning. In this paper the Minimax-MBTD algorithm is presented, which extends model-based reinforcement learning to deterministic alternating Markov games, a generalisation of two-player zero-sum strategy board games like chess and Go. By using a minimax measure of optimality the strategy learnt generalises to arbitrary opponents, unlike approaches that explicitly model specific opponents. Minimax-MBTD is applied to Tic-Tac-Toe and found to converge faster than direct reinforcement learning, but focussing planning on successors to the current state resulted in slower convergence than unfocussed random planning.

Keywords: Game playing, machine learning, planning, reinforcement learning, search.

1 Introduction

The field of reinforcement learning contains a broad group of methods for finding an optimal policy based on trial-and-error interaction with the problem domain [21]. Two classes of methods are the direct (model-free) and indirect (model-based) methods. Model-free reinforcement learning samples the problem domain directly but does not attempt to store the experience in a model. Model-based reinforcement learning, introduced by Sutton with the Dyna framework [17, 19, 20], saves the experience in a model, which it uses as another source of input into the underlying learning mechanism. The extra processing of modelled state transitions - called *planning* - accelerates convergence per interaction with the problem domain, and is particularly beneficial for systems where the cost of interacting with the environment is high relative to the computational

T.D. Gedeon and L.C.C. Fung (Eds.): AI 2003, LNAI 2903, pp. 520–531, 2003.

expense. In this paper, model-based reinforcement learning is extended to deterministic alternating Markov games, a generalisation of two-player zero-sum strategy board games like chess and Go.

Efficiency in reinforcement learning can be measured by the number of observations made from the environment (data-efficiency), or by the number of applications of the learning rule (computational-efficiency). In direct reinforcement learning, these measures reflect each other, but for model-based approaches they can be very different. In systems that can be modelled well, data-efficiency can be high as the number of observations required by the system is small. The computational efficiency though, will be equivalent to that of a direct reinforcement learning approach, since the computational requirements of the problem have not decreased - in fact model-based systems may apply the learning rule less optimally than direct approaches and can be less computationally efficient.

Strategy board games are a prime candidate for model-based methods because a model does not have to be learnt - the state transitions can be inferred from the game rules, and opponent's moves can selected by application of the minimax heuristic. Previously, data-efficiency has not been an issue when learning strategy board games, as the typical training method, self-play, generates training examples very cheaply. Self-play is an attractive method of training since it requires almost no domain specific knowledge, and because it has achieved spectacular success in the domain of backgammon [22, 4]. However, further research has shown that the nature of backgammon itself facilitates the use of self-play [14], and that the method performs poorly for deterministic games like chess [2].

A promising alternative to self-play, is to register the program on an Internet game server and train against real humans. This online training method has produced a strong chess player, KnightCap [2], (and also solves the problem of domain knowledge representation - by embodiment in human form). Interestingly, incremental training was achieved naturally, since as the program's rating improved it attracted higher caliber opponents.

Training against real opponents is more costly than training against simulated opponents however, and data-efficiency is a necessity. Whereas the backgammon programs could afford to learn from hundreds of thousands of training matches, and sometimes even millions, KnightCap had to make equivalent progress over only a few hundred matches. KnightCap was able to reduce convergence time by employing a knowledge intensive approach, particularly by initialising it's linear evaluation function approximator with good weight values. Finding a set of good initial weights is less likely to be the case for non-linear approximators, such as multi-layer perceptrons. Model-based techniques are a more general way of achieving data-efficiency.

The remainder of this paper is organised as follows. Section 2 reviews model-free methods for learning strategy board games, and ends with comments about their suitability as methods for doing planning backups in model-based approaches. Section 3 discusses some issues that arise when extending model-based methods to strategy board games, and presents a new algorithm Minimax-

MBTD. Section 4 describes some experiments where Minimax-MBTD was applied to Tic-Tac-Toe. The paper ends with some conclusions and ideas for further research.

2 Model-Free Reinforcement Learning for Strategy Board Games

The planning component of model-based reinforcement learning selects target states from the model, which are then backed up. A *backup* is an application of the underlying learning rule to a target state. Model-free methods are, in a sense, the trivial case for model-based methods when the number of planning backups is zero. This section presents two model-free algorithms that have been used to learn strategy board games, then discusses their suitability as methods for doing planning backups in model-based approaches. The first, TD(0), is based on the framework of Markov decision processes and has become very popular. The second and lesser known, Minimax-TD, is grounded in the framework of alternating Markov games.

Markov Decision Processes: A Markov Decision Process (MDP) [7] consists of a decision making agent operating within an environment. More precisely, it is a finite discrete set of states, $\mathcal{S}$, and controls, $\mathcal{A}$, a state transition function, $\mathcal{P}$, and a reward function, $\mathcal{R}$. At time t, the agent finds itself in state $s_t \in \mathcal{S}$, it chooses control $a_t \in \mathcal{A}$ and moves to state s_{t+1} with probability $\mathcal{P}^{a_t}_{s_t s_{t+1}}$, and is given some "reward" $\mathcal{R}^{a_t}_{s_t s_{t+1}}$. It is the agent's task to maximise the amount of reward it receives, that is, to find an optimal policy mapping states to controls that maximises the expected sum of discounted reward, $E(\sum_{t=0}^{\infty} \gamma^t \mathcal{R}^{a_t}_{s_t s_{t+1}})$. The discount factor, $\gamma \in [0, 1)$, ensures that the sum is bounded and also makes the agent prefer to be rewarded sooner rather than later.

The optimal value function, V^*, gives the total expected discounted reward for each state when following the optimal policy. It satisfies the following Bellman equation

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{s,s'} [\mathcal{R}^a_{s,s'} + \gamma V^*(s')] \tag{1}$$

which expresses a recursive relation between the optimal value of a state and it's successors. Once V^* is known, the optimal policy is found by choosing the control, a, that satisfies (1) given the state, s.

Temporal Difference Learning: A widely used algorithm for finding the optimal value function is TD(0), the method of temporal differences [18]. The agent maintains a table, $\hat{V}$, that stores an estimate of V^*, the optimal value function. It samples the state space according to the estimate of the optimal policy derived from $\hat{V}$, and at each time step, t, the algorithm shifts the estimate of the value function for the current state, $\hat{V}(s_t)$, to be more consistent with the

sum of the immediate reward, r_t, and the discounted estimate of the value of the next state, $\hat{V}(s_{t+1})$, as expressed in the following *temporal difference* rule

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha(r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)) \tag{2}$$

where $\alpha \in [0, 1]$ is a learning rate. It can be shown that the TD(0) algorithm converges to an optimal policy given that all states are visited infinitely often [18]. In order to ensure that all states are adequately explored, at every time step a control is chosen uniformly at random with some small probability, ϵ, the exploration threshold.

The method can be applied to sequential games by treating the opponent as part of the environment, or more precisely, as part of the state transition function $\mathcal{P}$. For each training match played, the sequence of positions for the learning player is observed, $s_1, s_2, \ldots, s_N$, where s_i is the position of the game immediately after the learning agent has made their ith move, and N is the total number moves that they played during the match. Starting with s_1, the temporal difference rule (2) is applied to each state in turn, with the exception of s_N, when $\hat{V}(s_{t+1})$ is replaced with zero. The reward function is usually determined by the win-loss relation, for example, the following reward function is commonly used for many games

$$r_t = \begin{cases} 1 & s_t \text{ wins} \\ -1 & s_t \text{ looses} \\ 0 & \text{otherwise.} \end{cases}$$

For some games, such as those that involve capturing territory, like Go and Othello, a more expressive reward can be given at the conclusion of the match based on the size of the territory captured or the piece difference between the players.

Alternating Markov Games: The domain of alternating Markov games [10], sometimes called sequential games, generalises MDPs to capture the essence of strategy board games like chess, Go and backgammon.

The domain has a discrete, finite set of states, $\mathcal{S}$, and two opponents, agent 1 and agent 2, which in each turn of the game alternately choose controls from their respective control sets, $\mathcal{A}_1, \mathcal{A}_2$. The sequence of events in one complete turn is shown by $\ldots s \overset{u \in \mathcal{A}_1}{\rightarrow} x \overset{v \in \mathcal{A}_2}{\rightarrow} s' \ldots$, a portion of a trajectory through a hypothetical match. For any turn, the probability that it ends in state s' given the initial state s, and the control choices u and v, is $\mathcal{P}^{u,v}_{s,s'} = \sum_x \mathcal{P}^u_{s,x} \mathcal{P}^v_{x,s'}$ where $\mathcal{P}^u_{s,x}$ and $\mathcal{P}^v_{x,s'}$ are the transitional probabilities for the state transitions occurring during the turn. Let the corresponding reward for each turn be $\mathcal{R}^{u,v}_{s,s'}$, which one agent attempts to maximise and the other agent attempts to minimise, resulting in diametrically opposed goals for the two agents.

Like Markov Decision Processes, the agent's aim is maximise (minimise) the expected sum of discounted reward, $E(\sum_{t=0}^{\infty} \gamma^t \mathcal{R}^{u_t,v_t}_{s_t s_{t+1}})$. But this time there is a complication, since the accrued reward depends not only on the agent's choice of controls, but the opponent's strategy too. Which opponent strategy should be

used to determine the accumulated reward? Typically, the resolution is to use the opponent that makes the agent's policy look the worst. This choice leads to a conservative measure of optimality, where strategies that consistently draw are preferred to strategies that give big wins but give big losses too. Using this "minimax" definition of optimality, the Bellman equation for alternating Markov games is given by

$$V^*(s) = \max_u \min_v \sum_{s'} \mathcal{P}^{u,v}_{s,s'}[\mathcal{R}^{u,v}_{s,s'} + \gamma V^*(s')]$$

assuming agent 1 is the maximiser (if agent 1 is the minimiser then negate the reward function $\mathcal{R}$). For deterministic games there is only one successor for every state, so (3) can be simplified by removing the summation, as follows

$$V^*(s) = \max_u \min_v [\mathcal{R}^{u,v}_{s,s'} + \gamma V^*(s')]. \tag{3}$$

A review of solution methods for alternating Markov games can be found in [10, 3].

Minimax Temporal Difference Learning: I now present Minimax-TD, an algorithm for solving deterministic alternating Markov games. Like the TD(0) algorithm, the agent maintains a table, $\hat{V}$, that stores an estimate of the optimal value function, and is used to generate it's policy. For simplicity, in this work the policy will be generated by greedily selecting the successor position with the best estimated value[1], but there is nothing to prevent the use of game searches over $\hat{V}$. The value function estimate is improved over a series of training matches as follows. Let s_t be the board position directly after the agent has made their tth move. At each turn in the game, t, the agent applies the consistency relation (3) to $\hat{V}(s_t)$, using the following backup rule

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha(r_t + \gamma \max_u \min_v \hat{V}(T(s_t, u, v)) - \hat{V}(s_t)), \tag{4}$$

where $T(s, u, v)$ is the transition function that gives the position succeeding s when the opponent selects control v then the agent chooses control u, with the proviso that u can be a *null move* if the opponent's move v has ended the match. The assumption that the game is deterministic is required to ensure that T returns a unique move. Minimax-TD has been applied to Othello [23], and generalised for Markov Games with simultaneous turns [9, 6].

Comparison of TD(0) and Minimax-TD: This section presented two model-free algorithms that have been used to learn strategy board games. The first, TD(0), forms strategies that are predictive and exploit weaknesses in the opponent's play, but the flip side is that they are biased by the training partner and may not generalise well to arbitrary opponents.

[1] Technically, I use ϵ-greedy policies, which ensure that positions estimated to be sub-optimal, perhaps incorrectly, are also occasionally visited.

The second, Minimax-TD, is based on a formalisation of strategy board games called alternating Markov games. It uses a minimax measure of optimality, resulting in conservative strategies that assume the opponent will make optimal moves irrespective of their actual behaviour. It's advantage is that it plays well against arbitrary opponents, therefore this rule will be used for the planning backups of Minimax-MBTD.

3 Model-Based Reinforcement Learning for Strategy Board Games

To extend model-based reinforcement learning to strategy board games, some further issues must be considered. Model-based reinforcement learning relies on a model of the environment to predict the state transitions. In strategy board games, the state transitions depend on the behaviour of the opponent as well as the rules of the game. The game rules are completely known to the agent, but the behaviour of the opponent usually isn't. So the first issue is: how should the opponent to be modelled?

The planning component of model-based methods involves a selection process, which chooses target states to be backed up. Careful choice of target states can increase computational efficiency. The second issue is: which game positions should be selected as backup targets?

The remainder of this section examines these two issues, and proposes answers that will form the basis for a new algorithm Minimax-MBTD.

Modelling the Opponent: An opponent model can be acquired by *model-learning*, that is, by observing the frequency of the opponent's moves for each state. For example, given a set of board positions, the opponent's decision in each position, and a feature decomposition of the game, a set of constraints can be constructed over the co-efficient vector for the features, which can be solved using linear programming techniques [5]. We do not pursue this approach because it "overfits" to the training opponent - when playing against new opponent, the model has to be re-learnt.

A more general approach would be to model the opponent as an optimal player. Since the optimal strategy is not known it must be estimated, for example by using the minimax heuristic [16]. If our evaluation function reflects the true game theoretic values, then the heuristic actually gives opponent's optimal response. Of course, at the beginning of training, the evaluation function estimate will be far from reflecting the true game theoretic values, leading to suboptimal predictions for the opponent's moves. As training continues and the evaluation function improves, the predictions should improve. The current research focuses on this approach.

Focussing Planning Backups: The simplest way to select target states for the planning component would be to choose them at random. The Dyna framework

is an example of this approach [17, 19, 20]. Unsurprisingly, random selection does not make best use of the model, and better performance has been reported for methods that employ more focussed approaches to selection, such as prioritized sweeping [11], Queue-Dyna [13] and trajectory sampling [8].

The trajectory sampling method looks forward along the trajectory given by the current estimate of the optimal policy, and selects the next k successor states. These successor states then form the target states for planning, and are backed up in reverse order, i.e. furthest away first. The heuristic behind this approach is that attention is focused on states that are likely to be encountered in the near future. The current research generalises the trajectory sampling method by focusing on *all* the k successor states, covering future possibilities more broadly. This way of selecting target states for the planning backups is inspired by suggestions in Sutton and Barto's book ([21], see Section 9.7).

The Minimax-MBTD Algorithm: I now give the Minimax-MBTD algorithm (see Figure 1). For each position encountered during training, the algorithm generates the tree of k successors to use as targets for planning (not counting transpositions and the opponent's positions), then backs them up using Minimax-TD. Within a layer, sibling successors are ordered by their evaluation function estimate so that positions with higher estimated importance are selected before positions with a lower estimate (see Figure 2). The order of backups is from the leaves towards the root, which propagates the leaf values towards the root. Note that when $k = 0$ the algorithm reduces to a model-free version of the algorithm, which is Minimax-TD.

- Let $s_1, s_2, \ldots, s_N$ be the N positions occurring during the match after each of the agent's moves. For $t = 1, 2, \ldots, N$ let r_t be the reward corresponding to position s_t.
- Let $H(s_t)$ be the game tree rooted at s_t after all transpositions are removed, and siblings are ordered s.t. if x and y are siblings and they are positions after the agent has moved and $\hat{V}(x) > \hat{V}(y)$, then x is to the left of y; else if x and y are siblings and they positions after the opponent has moved and $\hat{V}(x) < \hat{V}(y)$, then x is to the left of y.
- Let s_t^l be the lth node in $H(s_t)$, where nodes are numbered in the same order as visited by a breath-first search, the opponent's positions are not counted, and $s_t^0 = s_t$.
- For $t = 1$ to N do
  Generate $H(s_t)$ to depth(s_t^k)
  For $l = k$ downto 0 do
  $\hat{V}(s_t^l) \leftarrow \hat{V}(s_t^l) + \alpha(r_t + \gamma \max_u \min_v \hat{V}(T(s_t^l, u, v)) - \hat{V}(s_t^l))$

Fig. 1. The Minimax-MBTD algorithm

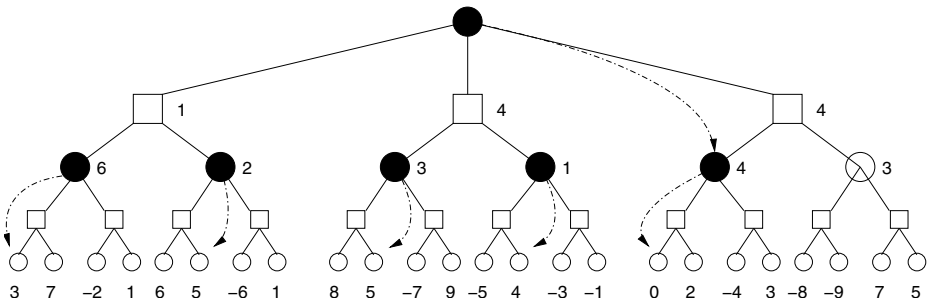

Fig. 2. A partial game tree generated by Minimax-MBTD when $k = 5$. The target positions for planning are shown as filled black circles and the backups are indicated by dashed arrows. The values next to the nodes are returned by the value function estimate, that is, they have not been propagated up the tree from the leaves like a minimax search would do

4 Experiments

In this section I apply Minimax-MBTD to Tic-Tac-Toe, also called noughts and crosses. The aim is firstly to see if the model-based approach will converge faster than direct reinforcement learning in a strategy board game domain, and secondly to see if there is an advantage to focussing the planning steps on successors to the current state. In all experiments, the learning rate, α, was set at 0.2 and annealed during training; the discount factor, γ, was set at 1 (no discounting); the exploration threshold, ϵ, was set at 5%; and the table entries were initialised to random values drawn uniformly from the interval [0.1,-0.1].

Generalisation techniques (such as neural networks) were not used because Minimax-TD is an off-policy rule. *On-policy* rules sample the problem domain according to the current estimation policy, whereas *off-policy* rules sample using a different distribution. TD(0) is an on-policy rule, and is frequently combined with neural networks when learning board games, but off-policy rules are not guaranteed to converge when used with function approximation [15], in fact simple problems exist for which they never converge [1]. Tic-Tac-Toe has 6045 unique legal positions which is easily small enough to store in memory, and to explore without using a generalisation method, so using tables avoided complications arising from the combined use of function approximation and off-policy backups.

Training and Evaluation: Training was by self-play, with the two opponents alternating as the first player every match. Every 1,000 matches a set of 100 evaluation matches were played against two fixed strategy challengers, during which learning was switched off, and no exploratory moves were made. After

each evaluation set is played, a match equity score is computed as follows

$$\text{equity}(e_i) = \frac{\text{wins}(p, e_i) - \text{wins}(c, e_i)}{\text{total}(e_i)}$$

where e_i is the ith evaluation set, $\text{total}(e_i)$ is the total number of games played during e_i - always 100 in the current experiments, $\text{wins}(x, e_i)$ is the number of wins counted during e_i for player $x \in \{p, c\}$, and p and c are the learning player and the challenger respectively.

Challengers: The first challenger is a *semi-random player*, who firstly tries to complete a winning line, then secondly to block the opponent from winning on the next turn; or if neither of these options are possible, it chooses a move from all legal candidates uniformly at random. The semi-random challenger will cover the state space broadly due to the randomness in its decision making.

The other challenger is an *heuristic player*, who selects from all the immediate legal candidate moves, that which maximises the following *max-lines* heuristic [12]. All of the eight groupings of three cells lying in a straight line are examined, and the number of friendly singlets, S_f, and doublets, D_f; and enemy singlets, S_e, and doublets, D_e, are counted. A singlet contains one marker only, a doublet contains two by the same player only. In addition, if a Tic-Tac-Toe is found a flag is set. An evaluation for the board position, s, is computed as

$$V(s) = \begin{cases} 1 & \text{Tic-Tac-Toe made} \\ -1 & D_e > 0 \\ \frac{2D_f + Sf - S_e}{6} & \text{otherwise} \end{cases}$$

If more than one move has the greatest evaluation, then one of those moves is made at random. The heuristic challenger plays a strong game, although it can be beaten as the second player by constructing a fork.

Results: The first experiment compares the model-based approach ($k > 0$) with the model-free ($k = 0$). The results of the experiment are shown in Figure 3. Against both challengers all model-based approaches converge by about 5,000 training matches, whereas for the model-free case it is closer to 10,000 matches. Interestingly, better strategies were also found as the number of planning steps increases.

To test whether focused backups are better than unfocused, the previous experiment was repeated, only this time the target positions for the planning steps were selected uniformly at random from all legal positions. The results are shown in Figure 4. In all cases convergence occurs by 1000 training matches, much faster than the focused case. In addition, the strategies learnt are also more optimal than those found using focussed planning.

The strong performance of the unfocussed approach is due to the size of the problem domain relative to number of planning backups. The state space of Tic-Tac-Toe is smaller than the total number of planning backups, therefore

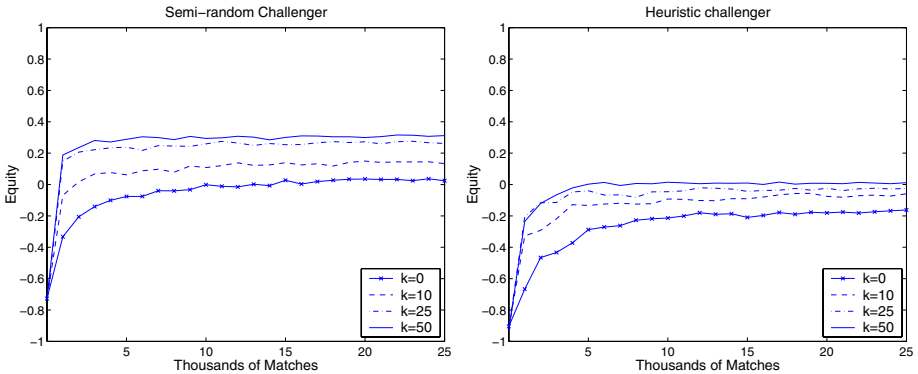

Fig. 3. Learning curves for model-based Minimax-MBTD ($k > 0$) versus model-free Minimax-TD ($k = 0$). Each curve is the average of 100 training episodes

a random distribution of states will cover it more completely than the distribution focused on successors to the current state. However, focussed approaches are likely to scale better, because the state space cannot be covered completely for larger games, and a heuristic is needed to restrict attention to the more "interesting" states.

5 Conclusions

This paper outlines a way of extending model-based reinforcement learning to strategy board games, and presents Minimax-MBTD, an algorithm based on these ideas. A key finding is that it converges faster than direct methods per observation, which is consistent with other studies of model-based methods. More interestingly, as the number of planning steps increased, the final strategy improved too. Unfocussed backups were found to perform better than focused, nevertheless focussed approaches may scale better to larger games.

Future work could concentrate on scaling up to larger games. The comparatively small state space of Tic-Tac-Toe meant that the entire value function could be stored in a lookup table, however, most games have too many states for a table based approach to work. The issue is deeper than the limit that physical memory places on the size of the evaluation function; more importantly, it is the need to recognise when new situations resemble previously encountered ones, so that best use can be made of limited training experience. Therefore, when scaling up to games with larger state spaces it is typical to use a generalisation technique, such as a neural network.

Minimax-TD is off-policy and therefore could be susceptible to the divergence frequently observed for the combination of off-policy methods and function approximation [1]. An intuitive explanation for the divergence is that approximator error is compounded by the min and max operators of Minimax-TD (or just max

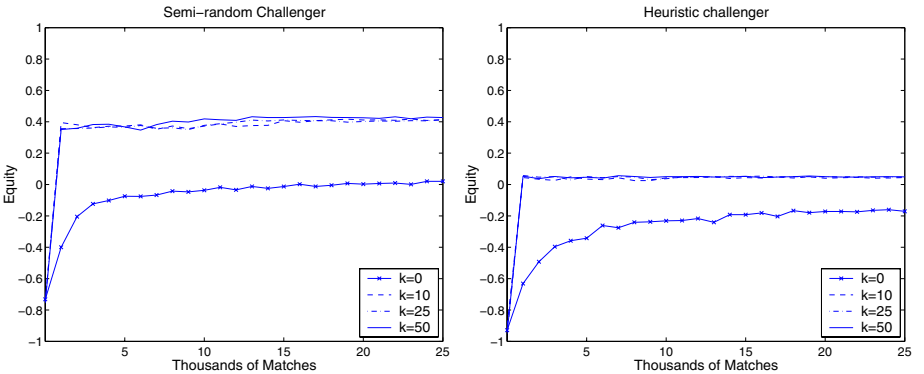

Fig. 4. Learning curves when the k target positions for planning are selected at random. Each curve is the average of 100 training episodes

in the case of Q-Learning), propagated back to the approximator in the training signal, where it causes further approximation error, and so on, ultimately leading to divergence. More optimistically, Minimax-TD has been combined with non-linear gradient descent RBF networks to train an Othello playing program [23], though my preliminary efforts at combining Minimax-TD with backpropagation MLP networks for Tic-Tac-Toe have been discouraging.

References

[1] Leemon C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995. 527, 529

[2] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. TDLeaf(λ): Combining temporal difference learning with game-tree search. *Australian Journal of Intelligent Information Processing Systems ISSN 1321-2133*, 5(1):39–43, 1998. 521

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996. 524

[4] Justin A. Boyan. Modular neural networks for learning context-dependent game strategies. Master's thesis, University of Cambridge, 1992. 521

[5] David Carmel and Shaul Markovitch. Model-based learning of interaction strategies in multiagent systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):309–332, 1998. 525

[6] Fredrik A. Dahl and Ole Martin Halck. Minimax TD-learning with neural nets in a markov game. In Ramon López de Mántaras and Enric Plaza, editors, *Proceedings of the 11th European Conference on Machine Learning*, pages 117–128. Springer-Verlag, 2000. 524

[7] Ronald A. Howard. *Dynamic Programming and Markov Processes*. The MIT Press, Cambridge, MA, 1960. 522

[8] L. Kuvayev and R. Sutton. Model-based reinforcement learning with an approximate, learned model. In *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, pages 101–105, 1996. 526

[9] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, pages 157–163, New Brunswick, NJ, 1994. Morgan Kaufmann. 524

[10] Michael L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, 1996. 523, 524

[11] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993. 526

[12] Daniel Kenneth Olson. Learning to play games from experience: An application of artificial neural networks and temporal difference learning. Master's thesis, Pacific Lutheran University, Washington, 1993. 528

[13] J. Peng and R. J. Williams. Efficient learning and planning within the dyna framework. In *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior, Hawaii*, 1993. 526

[14] Jordan B. Pollack and Alan D. Blair. Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32(1):225–240, 1998. 521

[15] Doina Precup, Richard S. Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *Proc. 18th International Conf. on Machine Learning*, pages 417–424. Morgan Kaufmann, San Francisco, CA, 2001. 527

[16] C. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(4):256–275, 1950. 525

[17] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990. 520, 526

[18] Richard S. Sutton. Learning to predict by the method of temporal differences. *Science*, 3(1):9–44, 1988. 522, 523

[19] Richard S. Sutton. DYNA, an Integrated Architecture for Learning, Planning and Reacting. In *Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures*, pages 151–155, 1991. 520, 526

[20] Richard S. Sutton. Planning by incremental dynamic programming. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 353–357. Morgan Kaufmann, 1991. 520, 526

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998. 520, 526

[22] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves masterlevel play. *Neural Computation*, 6(2):215–219, 1994. 521

[23] Taku Yoshioka, Shin Ishii, and Minoru Ito. Strategy acquisition for the game "Othello" based on reinforcement learning. *IEICE Transactions on Information and Systems E82-D*, 12:1618–1626, 1999. 524, 530