Minimax TD-Learning with Neural Nets in a Markov Game

Fredrik A. Dahl and Ole Martin Halck

Norwegian Defence Research Establishment (FFI)
P.O. Box 25, NO-2027 Kjeller, Norway
{Fredrik-A.Dahl, Ole-Martin.Halck}@ffi.no

Abstract. A minimax version of temporal difference learning (minimax TD-learning) is given, similar to minimax Q-learning. The algorithm is used to train a neural net to play Campaign, a two-player zero-sum game with imperfect information of the Markov game class. Two different evaluation criteria for evaluating game-playing agents are used, and their relation to game theory is shown. Also practical aspects of linear programming and fictitious play used for solving matrix games are discussed.

1 Introduction

An important challenge to artificial intelligence (AI) in general, and machine learning in particular, is the development of agents that handle uncertainty in a rational way. This is particularly true when the uncertainty is connected with the behavior of other agents.

Game theory is the branch of mathematics that deals with these problems, and indeed games have always been an important arena for testing and developing AI. However, almost all of this effort has gone into deterministic games like chess, go, othello and checkers. Although these are complex problem domains, uncertainty is not their major challenge.

With the successful application of temporal difference learning, as defined by Sutton [1], to the dice game of backgammon by Tesauro [2], random games were included as a standard testing ground. But even backgammon features perfect information, which implies that both players always have the same information about the state of the game.

We believe that imperfect information games, like poker, are more challenging and also more relevant to real world applications. Imperfect information introduces uncertainty about the opponent's current and previous states and actions, uncertainty that he cannot quantify as probabilities because he does not know his opponent's strategy. In games like chess and backgammon deception and bluff has little relevance, because a player's state and actions are revealed immediately, but with imperfect information these are important concepts.

As Koller and Pfeffer [3] and Halck and Dahl [4] have observed, imperfect information games have received very little attention from AI researchers. Some recent exceptions are given by Billings et al [5] and Littman [6].

R. López de Mántaras, E. Plaza (Eds.): ECML 2000, LNAI 1810, pp. 117-128, 2000.

The present article deals with minimax TD-learning, a value-based reinforcement learning algorithm that is suitable for a subset of two-player zero-sum games called Markov games. The set of Markov games contains some, but not all, imperfect information games, and represents a natural extension of the set of perfect information games. The algorithm is tested on a military air campaign game using a neural net.

The article is structured as follows. Section 2 covers some elementary game theory. Section 3 gives two evaluation criteria that we use for evaluating the performance of game-playing agents. In Section 4 the game Campaign, which will serve as testing-ground for our algorithm, is defined. Section 5 gives the definition of our reinforcement learning algorithm. Section 6 presents implementation and experimental results, and Section 7 concludes the article.

2 Game Theory

We now give a brief introduction to some elementary game-theoretic concepts. The theory we use is well covered by e.g. Luce and Raiffa [7].

A *game* is a decision problem with two or more decision-makers, called *players*. Each player evaluates the possible game outcomes according to some *payoff* (or *utility*) function, and attempts to maximize the expected payoff of the outcome. In this article we restrict our attention to *two-player zero-sum games*, where the two players have opposite payoffs, and therefore have no incentive to co-operate. We denote the players *Blue* and *Red*, and see the game from Blue's point of view, so that the payoff is evaluated by Blue. The zero-sum property implies that Red's payoff is equal to Blue's negated. Note that constant-sum games, where Blue's and Red's payoffs add to a fixed constant c for all outcomes, can trivially be transformed to zero-sum games by subtracting $c/2$ from all payoff values.

A *pure strategy* for a player is a deterministic plan that dictates all decisions the player may face in the course of a game. A *mixed*, or *randomized*, strategy is a probability distribution over a set of pure strategies.

Under mild conditions (e.g. finite sets of pure strategies) a two-player zero-sum game has a *value*. This is a real number v such that Blue has a (possibly mixed) strategy that guarantees the expected payoff to be no less than v, and Red has a strategy that guarantees it to be no more than v. A pair of strategies for each side that has this property is called a *minimax solution* of the game. These strategies are in equilibrium, as no side can profit from deviating unilaterally, and therefore minimax play is considered optimal for both sides.

Games of *perfect information* are an important subclass of two-player zero-sum games containing games like chess and backgammon. With perfect information both players know the state of the game at each decision point, and the turn of the players alternate. In perfect information games there exist minimax solutions that consist of pure strategies, so no randomization is required. In *imperfect information games* like two-player poker, however, minimax play will often require mixed strategies. Any experienced poker player knows that deterministic play is vulnerable. Randomization is best seen as a defensive maneuver protecting against an intelligent opponent who

may predict your behavior. In chess this plays little part, because your actions are revealed to the opponent immediately.

2.1 Matrix Games

A special class, or rather representation, of two-player zero-sum games is *matrix games*. In a matrix game both players have a finite set of pure strategies, and for each combination of Blue and Red strategy there is an instant real valued reward. The players make their moves simultaneously. If Blue's strategies are numbered from 1 to m, and Red's are numbered from 1 to n, the game can be represented by an $m \times n$ matrix $\mathbf{M}$ whose entry m_{ij} equals Blue's payoff if Blue and Red use strategies i and j respectively. Any finite two-player zero-sum game can be transformed to matrix form by enumerating the strategies. If the game is stochastic, the matrix entries will be expected payoffs given Blue and Red strategy. However, if the game has sequences of several decisions, the dimensions of the matrix will grow exponentially, making the matrix representation impractical to produce.

It has long been known that the problem of finding a minimax solution to a matrix game is equivalent to solving a linear programming problem (LP), see Strang [8]. Efficient algorithms exist for LP, such as the simplex procedure. We will return to this in the implementation section.

3 Evaluation Criteria

Our goal is to use machine learning techniques to develop agents that play two-player zero-sum games well. To quantify success of our agents, we need to define evaluation criteria. This is not quite as straightforward as one might believe, because game outcomes are not in general transitive. Even if agent A beats agent B every time, and B beats C consistently, it may well be that C beats A all the time. The obvious example of this is the well-known game scissors-paper-rock. Therefore, one cannot develop a strength measure that ranks a pool of agents consistently so that stronger agents beat weaker ones. Instead we seek to develop evaluation criteria that conform to game theory. These criteria have previously been published in Halck and Dahl [4].

3.1 *Geq*

Our strictest evaluation criterion is called *equity against globally optimizing opponent*, abbreviated *Geq*. The *Geq* of an agent is the minimum of the player's expected outcome, taken over the set of all possible opponents. The *Geq* is less than or equal to the game's value, with equality if and only if the agent is a minimax solution.

Let P_1 and P_2 be agents, and let P be the randomized agent that uses P_1 with probability p, $0 < p < 1$, and P_2 with probability $1 - p$. (Of course, P_1 and P_2 may also

contain randomization, and this is assumed independent of P's randomization between P_1 and P_2.) Then

$$Geq(P) \geq p \cdot Geq(P_1) + (1-p) \cdot Geq(P_2). \quad (\mathbf{1})$$

This is easily seen by observing that the most dangerous opponent of P_1 may be different from that of P_2. Inequality (1) shows that mixing strategies with similar *Geq* is beneficial according to the *Geq* measure, particularly if the component players have different weaknesses.

Mixing of strategies is most important in games of imperfect information, where this is required for minimax play, but even in games of perfect information it will often improve the *Geq*. Consider chess, and imagine a version of IBM's Deep Blue that plays deterministically. As long as there is even just a single way of tricking the program, its *Geq* would be zero. On the other hand, a completely random agent would do better, getting a positive *Geq*, as there is no way of beating it with probability 1.

3.2 *Peq*

Our second performance measure is *equity against perfect opponent*, abbreviated *Peq*. The *Peq* of an agent is its expected outcome against a minimax-playing opponent. Note that minimax solutions are not in general unique, so there may actually be a family of related *Peq* measures to choose from. In the following we assume that one of them is fixed.

For all agents $Peq \geq Geq$, as the minimax agent is included in the set of opponents that the *Geq* calculations minimize over. The *Peq* measure also has the game's value as its maximum, and a minimax-playing agent achieves this. But this is not a sufficient condition for minimax play. Consider again our agent P as the mixture of agents P_1 and P_2. The *Peq* measure satisfies the following equation:

$$Peq(P) = p \cdot Peq(P_1) + (1-p) \cdot Peq(P_2). \quad (\mathbf{2})$$

This property follows directly from the linearity of the expected value. Equation (2) tells us that according to the *Peq* measure there is nothing to be gained by randomizing. This makes sense, because randomization is a defensive measure taken only to ensure that the opponent does not adjust to the agent's weaknesses. When playing against a static opponent, even a perfect one, there is no need for randomization. Equation (2) implies that *Peq* only measures an agent's ability to find strategies that may be a component of some minimax solution.

This touches a somewhat confusing aspect of the minimax solution concept. Like we stated in the game theory section, a pair of agents that both play a minimax solution is in equilibrium, as neither can gain by deviating. However, the equilibrium is not very coercive, because one agent does not have anything to gain from randomizing as long as the other agent does. As we have just seen, all it takes to secure the value against a minimax-playing opponent is any deterministic strategy that may be a randomized component of a minimax solution.

This can be illustrated with an example from poker. In some poker situations it is correct (in the minimax sense) for a player to bluff with a given probability, and for

the opponent to call with a different probability. But the optimal bluffing probability is exactly the one that makes calling and folding equally strong for the opponent. And similarly, if the opponent calls with his optimal probability, it makes no difference if the first player bluffs all the time or not at all.

4 Campaign

In this article we will describe and explore an algorithm that is defined for *Markov games*, which is a proper subclass of two-player zero-sum games. Markov games include some, but not all, games with imperfect information. We have developed our own game, called Campaign, which features imperfect information, as testing ground for agents. Rather than burdening the reader with formal definitions, we present Campaign as an example of a Markov game, and describe general Markov games informally afterwards. Campaign was first defined and analyzed in Dahl and Halck [9].

4.1 Rules

Both players start the game with five *units* and zero *accumulated profit*. There are five consecutive *stages*, and at each stage both players simultaneously allocate their available units between three *roles*: *defense* (*D*), *profit* (*P*) and *attack* (*A*). A unit allocated to *P* increases the player's accumulated profit by one point. Each unit allocated to *D neutralizes* two opponent attacking units. Each unit allocated to *A*, and not neutralized, destroys one opponent unit for the remaining stages of the game. Before each stage the players receive information about both side's accumulated profit and number of remaining units. After the last stage the *score* for each player is calculated as the sum of accumulated profit and number of remaining units. The player with the higher score wins, and with equality the game is a draw. Margin of victory is irrelevant. If both players evaluate a draw as "half a win", the game is zero sum. We assign the payoffs 0, 0.5 and 1 to losing, drawing and winning, respectively, which technically makes the game constant-sum. The rules are symmetric for Blue and Red, and the value is clearly 0.5 for both. Campaign has imperfect information due to the simultaneity of the player's actions at each stage.

The military interpretation of the game is an air campaign. Obviously, a model with so few degrees of freedom can not represent a real campaign situation accurately, but it does capture essential elements of campaigning. After Campaign was developed, we discovered that it was in fact very similar to "The Tactical Air Game" developed by Berkovitz [10]. This may indicate that Campaign is a somewhat canonical air combat model.

Define the game's *state* as a four-tuple (b, r, p, n), with b being the number of remaining Blue units, r the number of Red units, p Blue's lead in accumulated profit points and n the number of rounds left. The initial state of the game is then (5,5,0,5). Note that it is sufficient to represent the difference in the players' accumulated profit, as accumulated profit only affects evaluation of the final outcome, and not the dynamics of the game. Our state representation using Blue's lead in accumulated

profit introduces some asymmetry in an otherwise symmetric game, but this is of no relevance. For that matter, both players may regard themselves as Blue, in which case a state seen as (a,b,c,d) for one player, would be perceived as $(b,a,-c,d)$ to the other. An allocation is represented as a three-tuple (D, P, A) of natural numbers summing to the side's number of remaining units. A sample game is given in Table 1.

Table 1. A sample game of Campaign

Stage	State	Blue action	Red action
1	(5,5,0,5)	(2,2,1)	(2,3,0)
2	(5,5,-1,4)	(1,4,0)	(2,3,0)
3	(5,5,0,3)	(2,2,1)	(0,0,5)
4	(4,4,2,2)	(1,2,1)	(1,0,3)
5	(3,4,4,1)	(0,3,0)	(0,4,0)
Final state:	(3,4,3,0)		

Blue wins the game, as his final lead in accumulated profit (3) is larger than his deficit in remaining units $(4-3=1)$.

4.2 Solution

Because perfect information is available to the players before each stage, earlier states visited in the game can be disregarded. It is this property that makes the game solvable in practical terms with a computer. As each state contains perfect information, it can be viewed as the starting point of a separate game, and therefore has a value, by game theory. This fact we can use to decompose the game further by seeing a state as a separate game that ends after both players have made their choice. The outcome of this game is the value of the next state reached. At each state both players have at most 21 legal allocations, so with our decomposition a game state's value is defined as the value of a matrix game with at most 21 pure strategies for both sides, with matrix entries being values of succeeding game states. One can say that our solution strategy combines dynamic programming and solution of matrix games. First all games associated with states having one stage left are solved using linear programming. (Again we refer to the implementation section for a discussion of solution algorithms for matrix games.) These have matrix entries defined by the rules of the game. Then states with two stages remaining are resolved, using linear programming on the game matrices resulting from the previous calculations, and so on.

To shed some light on what the game solution looks like, we will give some examples of states with solutions. For all states $(b, r, p, 1)$, that is, the last stage of the game, allocating all units to profit is a minimax solution. Other states have far more complex solutions. Figure 1 shows the solutions for three different states, in which solutions are unique. In each one both players have all five units remaining, and their complete state descriptions are (5,5,0,5), (5,5,0,3) and (5,5,2,2). Superficially these states appear similar, but as the figure shows, their solutions are very different. Note in particular that the allocation of one unit to defense, three to profit and one to attack is not given positive probability for the first two states, and it is in general rarely a

good allocation. But in the special case of Blue leading with two points, with all units intact and two stages to go, it is the only allocation that forces a win.

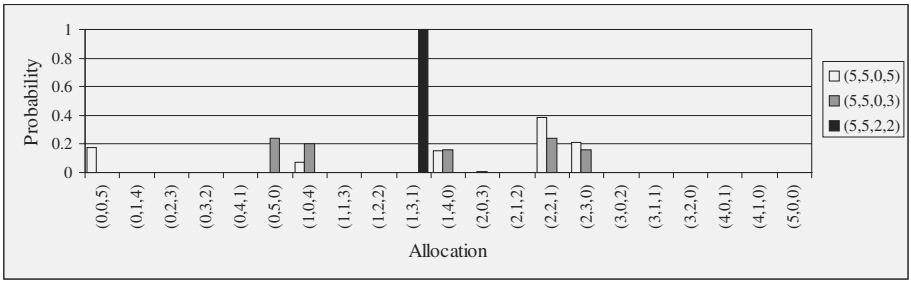

Fig. 1. Examples of solutions for some states

These examples show that apparently similar game states may have very different solutions. Therefore the game should pose a serious challenge to machine learning techniques.

4.3 Markov Games

We mentioned above that Campaign belongs to the game class called Markov games (see e.g. [6]). Markov games have the same general structure as Campaign with the players making a sequence of simultaneous decisions. The "Markov" term is taken from stochastic process theory, and loosely means that the future of a game at a given state is independent of its past, indicating that the state contains all relevant information concerning the history of the game.

There are three general features that Markov games may have that Campaign does not have. Firstly, the game may return to states that have previously been visited, creating cycles. Secondly, there may be payoffs associated with all state-action combinations, not just the terminal game states. The combination of these effects, cycles and payoffs of non-terminal states, opens the possibility of unlimited payoff accumulations, and this is usually prevented by some discounting factor that decreases exponentially with time. Thirdly, there may be randomness in the rules of the game, so that a triple (blue-action, state, red-action) is associated with a probability distribution over the set of states, rather than just a single state.

Markov games extend the class of perfect information games into the area of imperfect information. Note that perfect information games are included in Markov games by collapsing the decision set of the player that is not on turn to a single action. Therefore the simultaneous decisions trivially include sequential alternating decisions. Markov games can also be seen as a generalization of Markov decision problems (MDP), as an MDP is a "game" where the opponent's options are collapsed completely.

5 Minimax TD-Learning

Littman [6] defines minimax Q-learning for Markov games, which is similar to Q-learning in MDP, and uses this successfully for a simple soccer game. If the agent one is training knows the rules of the game (called *complete information* in the game theory language), a simpler learning rule can be used, that only estimates values of states, and this is what we do in this article. We are not aware of this algorithm being published previously, and give it the natural name of *minimax TD-learning*. We do not claim that this is an important new concept, more of a modification of minimax Q-learning. Minimax TD-learning is in fact even more similar to standard TD-learning for MDP than minimax Q-learning is to Q-learning. We assume the reader is familiar with TD-learning. Barto et al [11] gives an overview of TD-learning and other machine learning algorithms related to dynamic programming.

We describe the Minimax TD-learning method with Campaign in mind, but it should be obvious how it works for more general Markov games, featuring the general properties not present in Campaign. Minimax TD-learning trains a state evaluator to estimate the game-theoretic minimax value of states. This state evaluator, be it a lookup table, a neural net or whatever, is used for playing games, and standard TD(λ)-learning is used to improve estimates of state values based on the sequence of states visited.

The way that the state evaluator controls the game, however, is different from the MDP case. At each state visited a game matrix is assembled. For each combination of Blue and Red strategy the resulting state is calculated (which is why the algorithm requires knowledge of the rules). The evaluator's value estimate of that state is used as the corresponding game matrix entry. If the resulting state is a terminal one, the actual payoff is used instead. Then the matrix game is solved, and random actions are drawn for Blue and Red according to the resulting probability distributions. This procedure is repeated until the game terminates. A long sequence of games will normally be needed to get high quality estimations from the TD-learning procedure.

It is a well-known fact that TD-learning in MDPs may get stuck with a sub-optimal solution, unless some measures are taken that forces the process to explore the state space. This may of course happen with Markov games as well, being a superset of MDPs.

6 Implementation and Experimental Results

In this section we describe implementation issues concerning our state evaluator, different techniques used for solving matrix games, calculation of performance and experience with the learning algorithm itself.

6.1 Neural Net State-Evaluator

We have implemented our state evaluator as a neural net. The net is a "vanilla-flavored" design with one layer of hidden units, sigmoid activation functions, and

back-propagation of errors. The net has four input nodes associated to the state variables (b, r, p, n), each scaled by a factor 0.2 to get an approximate magnitude range of [0,1]. The net has one output node, which gives the estimated state value. The number of hidden nodes was set to eight.

6.2 Solving Matrix Games by Linear Programming

It is a well-established fact that matrix games can be solved by LP techniques, see e g Strang [8]. However, the practical problems encountered when implementing and using the simplex algorithm surprised us, and we would like to share this with the public. The problems would surely be less if we had used a commercial LP package, but that would require close integration of it into our program, which was not desirable. Instead we copied the simplex procedure published in [12].

Recall that our game matrix $\mathbf{M} \in \mathbf{R}^{m \times n}$ has as entry m_{ij} Blue's (expected) payoff when Blue uses his strategy with index *i* and Red uses his strategy *j*. We see the game from Blue's side, so we surely need variables $x_1, \ldots, x_m$ that represent his probability distribution. These will give the randomizing probabilities associated with his pure strategies. They must be non-negative (which fits the standard representation of LP problems), and sum to 1, to represent a probability distribution: $\sum_{i=1}^{m} x_i = 1$.

We also need a variable x_{m+1} for the game's value, which is not necessarily non-negative. A standard trick for producing only non-negative variables is to split the unbounded variable into its positive and negative parts: $x_{m+1} = v - u$. We do not have any convincing explanation of it, but from our experience this was not compatible with the simplex algorithm used, as it claimed the solution was unbounded. A different problem arose in some cases when the optimal value was exactly 0, as it was unable to find any feasible solution. This is probably due to rounding errors. To eliminate these problems we transformed the matrix games by adding a constant, slightly higher than the minimum matrix entry negated, to all matrix entries, thereby keeping the solution structure and ensuring strictly positive value. Afterwards the same constant must of course be deducted from the calculated value.

Minimax solutions are characterized by the fact that Blue's expected payoff is no less than the value, whichever pure strategy Red uses. For each $j \in \{1, \ldots, n\}$ this gives the inequality $\sum_{i=1}^{m} x_i \cdot m_{ij} - x_{m+1} \geq 0$. The objective function is simply the value x_{m+1}.

With this problem formulation the simplex procedure appeared to be stable, but only with double precision floating point numbers.

6.3 Solving Matrix Games by Fictitious Play

During our agonizing problems with the simplex algorithm we quickly implemented an iterative algorithm for matrix games called *fictitious play*. This algorithm is also far from new, see Luce and Raiffa [7]. Fictitious play can be viewed as a two-sided competitive machine learning algorithm. It works like this: Blue and Red sequentially find the most effective pure strategy, under the assumption that the opponent will play

according to the probability distribution manifested in the histogram of his actions in all previous iterations. The algorithm is very simple to implement, and it is completely stable. Its convergence is inverse linear, which does not compete with the simplex algorithm that reaches the exact solution in a finite number of steps. However, we do not need exact solutions in the minimax TD-training, because the game matrices are also not exact. From our experience fictitious play is faster than simplex for the required precision in training. But when it comes to calculating the Campaign solution, which is needed for evaluating the *Peq* of agents, the accuracy provided by the simplex algorithm is preferable. After this work had been done, we registered that Szepesvari and Littman [13] also suggests the use of fictitious play in minimax Q-learning to make the implementation "LP-free".

6.4 Calculating *Geq* and *Peq* Performance

To measure the progress of our Campaign-playing agent we need to evaluate its *Geq* and *Peq* performance.

The *Geq* calculations are very similar to the algorithm we use for calculating the solution. Because the behavior of the agent that is evaluated (Blue) is given, the problem of identifying its most effective opponent degenerates to an MDP problem, which can be solved by dynamic programming. First Red's most effective actions are calculated for states with one stage left $(b, r, p, 1)$. Then the resulting state values are used for identifying optimal actions at states with two stages left, and so on.

The *Peq* could be calculated in much the same way, except that no optimization is needed as both Blue and Red's strategies are fixed. However, it is more efficient to propagate probability distributions forwards in the state space, and calculating the expected outcome with respect to the probability distribution at the terminal states. This saves time because calculations are done only for states that are visited when this Blue agent plays against the given minimax Red player.

6.5 Experimental Results

When used without exploration the minimax TD-algorithm did not behave well. The first thing the net learned was the importance of profit points. This led to game matrices that result in a minimax strategy of taking profit only, for all states. The resulting games degenerated to mere profit-taking by both sides, and all games were draws. Therefore it never discovered the fact that units are more important than profit points, particularly in early game states. In retrospect it is not surprising that the algorithm behaved this way, because the action of using all units for profit is optimal in the last stage of the game, which is likely to be the first thing it learns about.

One way of forcing the algorithm to explore the state space more is by introducing random actions different from those recommended by the algorithm. Instead we have been randomizing the starting state of the game. Half of the games were played from the normal starting state (5,5,0,5), and the rest were drawn randomly. To speed up training, the random starting states were constructed to be "interesting". This was

done by ensuring that a player cannot start with a lead in both number of units and profit points.

TD-learning is subject to random noise. We were able to reduce this problem by utilizing a symmetry present in the game. If a given state (b, r, p, n) receives feedback v, it implicitly means that the state $(r, b, -p, n)$ seen by the opponent deserves feedback $1-v$. Adding this to the training procedure helps the net towards consistent evaluations, and automatically ensures that symmetric states (like the starting state) get neutral feedback (that is 0.5) on average. This reduced the random fluctuations in the net's evaluations considerably.

With these modifications the algorithm behaved well. Figure 2 shows the learning curves of the agent's *Geq* and *Peq*, with $\lambda = 0$ and learning rate decreasing from 1 to 0.1. The number of iterations in the fictitious play calculations was increased linearly from 100 to 500. The unit on the x-axis is 1000 games, and the curves are an average of five training batches.

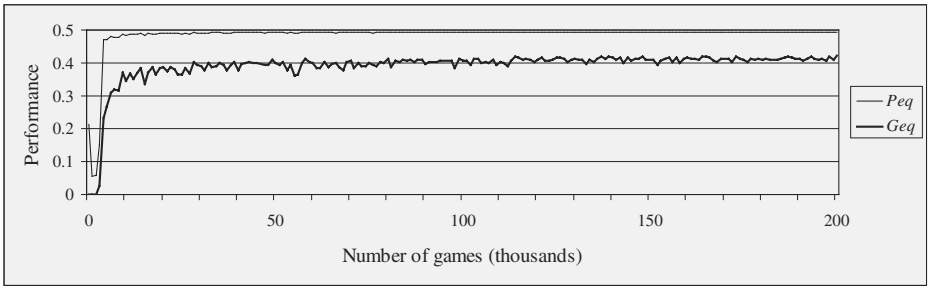

Fig. 2. *Geq* and *Peq* learning curves

We see that the *Peq* of the agent quickly approaches the optimal 0.5. The *Geq* values do not quite reach this high, but the performance is acceptable in light of the relatively small neural net used. The *Geq* is close to zero in the first few thousand games, and the *Peq* also has dip in the same period. This is because the agent first learns the value of profit points, and it takes some time before the exploration helps it to discover the value of units.

7 Conclusion

Our main conclusion is that minimax TD-learning works quite well for our Markov game named Campaign. Unlike the experience of Littman [6], the algorithm fails completely without forced exploration, but our exploration technique of randomizing the starting point of the game appears successful.

The results show that it is far easier to achieve high performance according to the *Peq* measure (expected outcome against a minimax-playing opponent) than according to *Geq* (expected outcome against the agent's most effective opponent).

Our experience indicates that the simple fictitious play algorithm can compete with LP algorithms for producing solutions for matrix games in cases where high precision is not needed. As a bonus fictitious play is also far simpler to implement.

References

1. Sutton, R.S.: Learning to predict by the methods of temporal differences, *Machine Learning* **3** (1988) 9–44.
2. Tesauro, G.J.: Practical issues in temporal difference learning, *Machine Learning* **8** (1992) 257–277.
3. Koller, D., Pfeffer, A.: Representations and solutions for game-theoretic problems. *Artificial Intelligence* **94**(1) (1997) 167–215.
4. Halck, O.M., Dahl, F.A.: On classification of games and evaluation of players – with some sweeping generalizations about the literature. In: Fürnkranz, J., Kubat, M. (eds.): *Proceedings of the ICML-99 Workshop on Machine Learning in Game Playing*, Jozef Stefan Institute, Ljubljana (1999).
5. Billings, D., Papp, D., Schaeffer, J., Szafron, D.: Poker as a testbed for machine intelligence research. In: Mercer, R., Neufeld, E. (eds.): *Advances in Artificial Intelligence*, Springer-Verlag, Berlin–Heidelberg–New York (1998) 228–238.
6. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the 11th International Conference on Machine Learning*, Morgan Kaufmann, New Brunswick (1994) 157–163.
7. Luce, R.D., Raiffa, H.: *Games and Decisions*. Wiley, New York (1957).
8. Strang, G.: *Linear Algebra and Its Applications*. *Second Edition.* Harcourt Brace Jovanovich, Orlando (1980).
9. Dahl, F.A., Halck, O.M.: Three games designed for the study of human and automated decision making. Definition and properties of the games Campaign, Operation Lucid and Operation Opaque. FFI/RAPPORT-98/02799, Norwegian Defence Research Establishment (FFI), Kjeller, Norway (1998).
10. Berkovitz, L.D.: The Tactical Air Game: A multimove game with mixed strategy solution. In: Grote, J.D. (ed.): *The Theory and Application of Differential Games*, Reidel Publishing Company, Dordrecht, The Netherlands (1975) 169–177.
11. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. *Artificial Intelligence* **72** (1995) 81–138.
12. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK (1988).
13. Szepesvari, C., Littman, M.L.: A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation* **11** (1999) 2017–2060.