Deep Machine Learning in Gamesin Artificial Intelligence

S.Saravanan[1], Vishal Gupta[2], Aakash K Singh[3], Asraar Contractor[4]

CSE,CSE,CSE,CSE,VTU,VTU,VTU,VTU
RRIT,Bangalore,RRIT,Bangalore,RRIT,Bangalore,RRIT,Bangalore
[1]drsaranphd@gmail.com,[2]vishuplk66@gmail.com, [3]aakash.singh.as130@gmail.com,
[4]asraar91@gmail.com

Abstract— **This paper provides a survey of previously published work on machine learning in game playing. The material is organized around a variety of problems that typically arise in game playing and that can be solved with machine learning methods. This approach, we believe, allows both, researchers in game playing to find appropriate learning techniques for helping to solve their problems as well as machine learning researchers to identify rewarding topics for further research in game-playing domains. The paper covers learning techniques that range from neural networks to decision tree learning in games that range from poker to chess. However, space constraints prevent us from giving detailed introductions to the used learning techniques or games. Overall, we aimed at striking a fair balance between being exhaustive and being exhausting.**

Keywords— **machine learning, game paying, decision tree, neural networks, space constraints**

I. Introduction

In this paper, we will attempt to survey the large amount of literature that deals withmachine-learning approaches to game playing. Unfortunately, space and time do notpermit us to provide introductory knowledge in either machine learning or game playing. The main goal of this paper is to enablethe interested reader to quickly find previous results that are relevant for her researchproject, so that she may start her investigations from there.

There are several possibleways for organizing the material in this paper. We could,for example, have grouped it by the different games (chess, Go, backgammon, shogi,Othello, bridge, poker to name a few more popular ones) or by the learning techniquesused (as we have previously done for the domain of chess (Furnkranz 1996)). Instead,we decided to take a problem-oriented approach and grouped them by the challengesthat are posed in different aspects of the game. This, we believe, allows both, researchersin game playing to find appropriate learning techniques for helping to solvetheir problems as well as machine learning researchers to identify rewarding topics forfurther research in game-playing domains.

We will start with a discussion of book learning, i.e., for techniques that store precalculatedmoves in a so-called book for rapid access in tournament play.

Next, we will address the problem of using learning techniques for controlling thesearch procedures that are commonly used in game playing programs, we will review the most popular learning task, namely the automatic tuningof an evaluation function. We will consider supervised learning, comparison training,reinforcement and temporal-difference learning. In a separate subsection, we will discussseveral important issues that are common to these approaches. Thereafter, we will survey various approaches for automatically discovering patterns and plans,moving from simple advice-taking over cognitive modeling approaches to the inductionof patterns and playing strategies from game databases. Finally, wewill briefly discuss opponent modeling, i.e., the task of improving the program's playby learning to exploit the weaknesses of particular opponents.

II. Learning to choose opening variations

The idea of using opening books to improve machine-play has been present since theearly days of computer game-playing. Samuel (1959) already used an opening bookin his checkers playing program, as did Greenblatt, Eastlake III, and Crocker (1967) intheir chess program. Opening books, i.e., pre-computed to replies for a set of positions,can be easily programmed and are a simple way for making human knowledge, whichcan be found in game-playing books, accessible to the machine. However, the questionwhich of the many book moves a program should choose is far from trivial.Hyatt (1999) tackles the problem of learning which opening his chess programCRAFTY should play and which it should avoid. He proposes a reinforcement learningtechnique (cf. Section 4.3) to solve this problem, using the computer's position evaluation after leaving the book as an evaluation of the playability of the chosen line. Inorder to avoid the problem that some openings (gambits) are typically underestimatedby programs, CRAFTY uses the maximum or minimum (depending on the trend) of theevaluations of the ten positions encountered immediately after leaving book.

A. Learning from mistakes

A straight-forward approach for learning to avoid to repeat mistakes is to remembereach position in which the program made a mistake so that it is alert when this positionis encountered the next time. The game-playing system HOYLE (Epstein 2001)implements such an approach. After each decisive game, HOYLE looks for the lastposition in which the loser could have made an alternative move and tries to determinethe value of this position through exhaustive search. If the search succeeds, the stateis marked as ―significant‖ and the optimal move is recorded for future encounters withthis position. If the search does not succeed (and hence the optimal move could not bedetermined), the state is marked as ―dangerous‖. More details on this technique can befound in (Epstein 2001).

In conventional, search-based game-playing programs such techniques can easilybe implemented via their transposition tables (Greenblatt et al. 1967; Slate and Atkin1983). Originally, transposition tables were only used locally with the aim of avoidingrepetitive search efforts (e.g., by avoiding to repeatedly search for the evaluation of aposition that can be reached with different move orders). However, the potential ofusing global transposition tables, which are initialized with a set of permanently storedpositions, to improve play over a series of games was soon recognized.

Once more, it was Samuel (1959) who made the first contribution in this direction.His checkers player featured a rote learning procedure that simply stored every positionencountered together with its evaluation so that it could be reused in subsequentsearches. With such techniques, deeper searches are possible because on the one handthe program is able to save valuable time because positions encountered in memorydo not have to be re-searched. On the other hand, if the search encounters a stored. Actually, Buro suggests to discern between public draws and private draws. The latter—being a resultof the program’s own analysis or experience and thus, with some chance, not part of the opponent’s bookknowledge—could be tried in the hope that the opponents makes a mistake, while the former may lead toboring draws when both programs play their bookmoves (as is known from many chess grandmaster draws).

A very similar technique was used in the BEBE chessprogram, where the transposition table was initialized with positions from previousgames. It has been experimentally confirmed that this simple technique learning infact improves its score considerably when playing 100-200 games against the sameopponent (Scherzer et al. 1990).In game like chess, such rote learning techniques help in opening or endgame play.
In complicated middlegamepositions,where most pieces are still on the board, chancesare considerably lower that the same position will be encountered in another game.

Thus, in order to avoid an explosion of memory costs by saving unnecessary positions,Samuel (1959) also devised a scheme for forgetting positions that are not or only infrequentlyused. Other authors tried to cope with these problems by being selective inwhich positions are added to the table. For example, Hsu (1985) tried to identify thefaulty moves in lost games by looking for positions in which the value of the evaluationfunction suddenly drops. Positions near that point were re-investigated with a deepersearch. If the program detected that it had made a mistake, the position and the correctmove were added to the program’s global transposition table. If no single move couldbe blamed for the loss, a re-investigation of the game moves with a deeper search wasstarted with the first position that was searched after leaving the opening book. Frey(1986) describes two cases where an Othello program (Hsu 1985) successfully learnedto avoid a previous mistake. Similar techniques were refined later (Slate 1987) andwere incorporated into state-of-the-art game playing programs, such as the chessprogram CRAFTY (Hyatt 1999).

Baxter, Tridgell, and Weaver (1998b) also adopt this technology but discuss someinconsistencies and propose a few modifications to avoid them. In particular, theypropose to insert not only the losing position but also its two successors. Every time aposition is inserted, a consistency check is performed to determinewhether the positionleads to another book position with a contradictory evaluation, in which case bothpositions are re-evaluated. This technique has the advantage that only moves that havebeen evaluated by the computer are entered into the book, so that it never stumbles―blindly‖ into a bad book variation.

III. LEARNING FROM SIMULATION

The previous techniques were developed for deterministic, perfect information gameswhere evaluating a position is usually synonymous for searching all possible continuationsto a fixed depth. Some of them may be hard to adapt for games with imperfectinformation (e.g., card games like bridge) or a random component (e.g., dice gameslike backgammon) where deep searches are infeasible and techniques like storing precomputedevaluations in a transposition table do not necessarily lead to significantchanges in playing strengths. In these cases, however, conventional search can bereplaced by simulation search (Schaeffer 2000), a search technique which evaluatespositions by playing a multitude of games with this starting position against itself. Ineach of these games, the indeterministic parameters are assigned different, concretevalues (e.g., by different dice rolls or by dealing the opponents a different set of cardsor tiles). Statistics are kept over all these games which are then used for evaluating thequality of the moves in the current state.

Tesauro (1995) notes that such roll-outs can produce quite reliable comparisonsbetween moves, even if the used program is not of master strength. In the case ofbackgammon, such analyses have subsequently led to changes in opening theory (Robertie1992, 1993). Similar techniques can be (and indeed are) used for position evaluationin games like bridge (Ginsberg 1999), Scrabble (Sheppard 1999), or poker (Billingset al. 1999), and were even tried as an alternative for conventional search in the gameof Go (Brugmann 1993). It would also be interesting to explore the respective advantagesof such Monte-Carlo search techniques and reinforcement learning (see (Suttonand Barto 1998) for a discussion of this issue in other domains).

A.Evaluation Function Tuning

The most extensively studied learning problem in game playing is the automatic adjustmentof the weights of an evaluation function. Typically, the situation is as follows:
the game programmer has provided the programwith a library of routines that computeimportant properties of the current board position (e.g., the number of pieces of each4Inductive logic programming (ILP) refers to a family of learning algorithms that are able to inducePROLOG programs and can thus rely on a more expressive concept language than conventional learningalgorithms, which operate in propositional logic (Muggleton 1992; Lavraˇc and Dˇzeroski 1993; De Raedt1995; Muggleton and De Raedt 1994). Its strengths become particularly important in domains where astructural description of the training objects is of importance, like, e.g., in describing molecular structures(Bratko and King 1994; Bratko and Muggleton 1995). They also seem to be appropriate for many gameplaying domains, in which a description of the spatial relation between the pieces is often more importantthan their actual location.kind on the board, the size of the territory controlled, etc.). What is not known is howto combine these pieces of knowledge and how to quantify their relative importance.

The known approaches to solving this problem can be categorized along severaldimensions. In what follows, we will discriminate them by the type of training informationthey receive. In supervised learning the evaluation function is trained on informationabout its correct values, i.e., the learner receives examples of positions ormovesalong with their correct evaluation values. In comparison training, it is provided witha collection of move pairs and the information which of the two is preferable. Alternativelyit is given a collection of training positions and the moves that have been playedin these positions. In reinforcement learning, the learner does not receive any directinformation about the absolute or relative value of the training positions or moves. Instead,it receives feedback from the environment whether its moves were good or bad.
In the simplest case, this feedback simply consists of the information whether it haswon or lost the game. Temporal-difference learning is a special case of reinforcementlearning which can use evaluation function values of later positions to reinforce or correctdecisions earlier in the game. This type of algorithm, however, has become sofashionable for evaluation function tuning that it deserves its own subsection. Finally,in Section 4.5, we will discuss a few important issues for evaluation function training.

B. Supervised learning

A straight-forward approach for learning theweights of an evaluation function is to providethe program with example positions for which the exact value of the evaluationfunction is known. The program then tries to adjust the weights in a way that minimizesthe error of the evaluation function on these positions. The resulting function,learned by linear optimization or some non-linear optimization technique like backpropagationtraining for neural networks, can then be used to evaluate new, previouslyunseen positions.
Mitchell (1984) applied such a technique to learning an evaluation function for thegame of Othello (see also (Frey 1986)).

These values were then used for computing appropriateweights of the 28 features of a linear evaluation function by means of regression.In the game of Othello, Lee and Mahajan (1988) relied on BILL, a—for the time—very strong program5 that used hand-crafted features, to provide training examplesby playing a series of games against itself. Variety was ensured by playing the first20 plies randomly. Each position was labeled as won or lost, depending on the actualoutcome of the game, and represented with four different numerical feature scores (Leeand Mahajan 1990). The covariance matrix of these features was computed from thetraining data and this information was used for learning several Bayesian discriminantfunctions (one for each ply from 24 to 49), which estimated the probability of winningin a given position. The results showed a great performance improvement over the5In 1997, BILL was tested against Buro’s LOGISTELLO and appeared comparably weak: running onequal hardware and using 20 minutes per game BILL was roughly on par with 4-ply LOGISTELLO, whichonly used a couple of seconds per game (Buro 2000, personal communication).original program. A similar procedure was used by Buro (1995b). He further improvedclassification accuracy by building a complete position tree of all games. Interior nodeswere labelled with the results of a fast negamax search, which he also used for hisapproach to opening book learning (see Section 2.2 and (Buro 2001)).

Tesauro and Sejnowski (1989) trained the first neural-network evaluation functionof the program that has later developed into TD-GAMMON by providing it with severalthousand expert-rated training positions.

In fact, overfittingthe training data did hurt the performance on independent test data, a common phenomenonin machine learning. Likewise, in (Dahl 2001), a neural network is trainedto evaluate parts of Go positions, so-called receptive fields. Its training input consistsof a number of positive examples, receptive fields in which the expert played into itscenter, and for each of them a negative example, another receptive field from the sameposition, which was chosen randomly from the legal moves that the expert did not play.

IV. COMPARISON TRAINING

Tesauro (1989a) introduced a new framework for training evaluation functions, whichhe called comparison training, thelearner is not given exact evaluations for the possiblemoves (or resulting positions) butis only informed about their relative order. Typically, it receives examples in the formof move pairs along with a training signal as to which of the two moves is preferable.

However, the learner does not learn an explicit preference relation between moves as in(Utgoff and Heitman 1988), but tries to use this kind of training information for tuningthe parameters of an evaluation function (Utgoff and Clouse 1991). Thus, the learnerreceives less training information than in the supervised setting, but more informationthan in the reinforcement learning setting.

A. Reinforcement learning

Reinforcement learning (Sutton and Barto 1998) is best described by imagining anagent that is able to take several actions whose task is to learn which actions aremost preferable in which states. However, contrary to the supervised learning setting,the agent does not receive training information from a domain expert. Instead,it may explore the different actions and, while doing so, will receive feedback fromthe environment—the so-called reinforcement or reward—which it can use to rate thesuccess of its own actions. In a game-playing setting, the actions are typically the legalmoves in the current state of the game, and the feedback is whether the learner winsor loses the game or by which margin it does so. We will describe this setting in more7A genetic algorithm (Goldberg 1989) is a randomized search algorithm. It maintains a population ofindividuals that are typically encoded as strings of 0’s and 1’s. All individuals of a so-called generationare evaluated according to their fitness, and the fittest individuals have the highest chance of surviving intothe next generation and of spawning new individuals through the genetic operators cross-over and mutation.

For more details, see also (Kojima and Yoshikawa 2001), which discusses the use of genetic algorithms forlearning to solve tsume-go problems.detail using MENACE, the Matchbox Educable Noughts And Crosses Engine (Michie1961, 1963), which learned to play the game of tic-tac-toe by reinforcement.

MENACE has one weight associated with each of the 287 different positions withthe first player to move (rotated ormirrored variants of identical positionswere mappedto a unique position). In each state, all possible actions (all yet unoccupied squares)are assigned a weight. The next action is selected at random, with probabilities correspondingto the weights of the different choices. Depending on the outcome of thegame, the moves played by the machine are rewarded or penalized by increasing ordecreasing their weight. Drawing the game was considered a success and was alsoreinforced (albeit by a smaller amount).

However, the idea is that after many games, good positions willhave received more positive than negative reward and vice versa, so that the evaluationfunction eventually converges to a reasonable value.

B. Linear vs. non-linear evaluation functions

Most conventional game-playing programs depend on fast search algorithms and thusrequire an evaluation function that can be quickly evaluated. A linear combination ofa few features that characterize the current board situation is an obvious choice here.

Manual tuning of the weights of a linear evaluation function is comparably simple, butalready very cumbersome. Not only the individual evaluation terms may depend oneach other, so that small changes in oneweightmay affect the correctness of the settingsof other weights, but also all weights depend on the characteristics of the program inwhich they are used. For example, the importance of being able to recognize tacticalpatterns such as fork threats may decrease with the program’s search depth or dependon the efficiency of the program’s quiescence search.

However, advances in automated tuning techniques have even made the use of nonlinearfunction approximators feasible. Samuel (1967) already suggested the use ofsignature tables, a non-linear, layered structure of look-up tables. Clearly,non-lineartechniques have the advantage that they can approximate a much larger class of functions.

C. Evaluation function learning and search

In backgammon, deep searches are practically infeasible because of the large branchingfactor that is due to the chance element introduced by the use of dice. However,deep searches are also beyond the capabilities of human players whose strength liesin estimating the positional value of the current state of the board. Contrary to thesuccessful chess programs, who can easily out-search their human opponent but stilltrail her ability of estimating the positional merits of the current board configuration,TD-GAMMON was able to excel in backgammon for the same reasons that humansplay well: its grasp of the positional strengths and weaknesses was excellent.

However, in games like chess or checkers, deep searches are necessary for expertperformance. A problem that has to be solved for these games is how to integratelearning into the search techniques. In particular in chess, one has the problem that theposition at the root of the node often has completely different characteristics than theevaluation of the node. Consider the situation where one is in the middle of a queentrade. The current board situation will evaluate as ―being one queen behind‖, while alittle bit of search will show that the position is actually even because the queen caneasily be recaptured within the next few moves. Straight-forward application of anevaluation function tuning algorithm would then simply try to adjust the evaluation ofthe current position towards being even. Clearly, this is not the right thing to do becausesimple tactical patterns like piece trades are typically handled by the search and neednot be recognized by the evaluation function.

The solution for this problem is to base the evaluation on the dominant position ofthe search. The dominant position is the leaf position in the search tree whose evaluationhas been propagated back to the root of the search tree. Most conventional searchbasedprograms employ some form of quiescence search to ensure that this evaluationis fairly stable. Using the dominant position instead of the root positionmakes sure thatthe estimation of the weight adjustments is based on the position that was responsiblefor the evaluation of the current board position. Not surprisingly, this problem has alreadybeen recognized and solved by Samuel (1959) but seemed to have been forgottenlater on. For example, Gherrity (1993) published a thesis on a system architecture thatintegrates temporal-difference learning and search for a variety of games (tic-tac-toe,Connect-4, and chess), but this problem does not seem to be mentioned.

D. Feature construction

The crucial point for all approaches that tune evaluation functions is the presence ofcarefully selected features that capture important information about the current stateof the game which goes beyond the location of the pieces. In chess, concepts likeking safety, center control or mobility are commonly used for evaluating positions, andsimilar abstractions are used in other games as well (Lee and Mahajan 1988; Ender-ton 1991). Tesauro and Sejnowski (1989) report an increase in playing strength of 15to 20% when adding hand-crafted features that capture important concepts typicallyused by backgammon experts (e.g., pipcounts) to their neural network backgammonevaluation function. Although Tesauro later demonstrated that his TD()-trained networkcould surpass this playing level without these features, re-inserting them broughtyet another significant increase in playing strength (Tesauro 1992b). Samuel (1959) alreadyconcluded his famous study by making the point that the most promising road towardsfurther improvements of his approach might be ―. . . to get the program to generateits own parameters for the evaluation polynomial‖ instead of learning only weightsfor manually constructed features. However, in the follow-up paper, he had to concedethat the goal of ―. . . getting the program to generate its own parameters remains as farin the future as it seemed to be in 1959‖ (Samuel 1967).

The disadvantage of the features constructed in the hidden layers of neural networksis that they are not immediately interpretable. Several authors have worked on alternativeapproaches that attempt to create symbolic descriptions of new features. Fawcettand Utgoff(1992) discuss the ZENITH system, which automatically constructs featuresfor a linear evaluation function for Othello. Each feature is represented as a formula infirst-order predicate calculus.

E. Advice-taking

The learning technique that requires the least initiative by the learner is learning bytaking advice. In this framework, the user is able to communicate abstract conceptsand goals to the program. In the simplest case, the provided advice can be directlymapped on to the program's internal concept representation formalism. One such exampleisWaterman's poker player (Waterman 1970). Among other learning techniques,it provides the user the facility to directly add production rules to the game-playingprogram. Another classic example of such an approach is the work by Zobrist andCarlson (1973), in which a chess tutor could provide the program with a library ofuseful patterns using a chess programming language that looked a lot like assemblylanguage. Many formalisms have since been developed in the same spirit (Bratko andMichie 1980; George and Schaeffer 1990; Michie and Bratko 1991),most of them limitedto endgames, but some also addressing the full game (Levinson and Snyder 1993;

While in the above-mentioned approaches thetutoring process is often more orless equivalent to programming in a high-level game programming language, typicallyadvice-taking programs have to devote considerable effort into compiling the providedadvice into their own pattern language. Thus they enable the user to communicatewith the program in a very intuitiveway that does not require any knowledge about theimplementation of the program nor about programming in general.

The most prominent example for such an approach is the work by Mostow (1981).He has developed a system that is able to translate abstract pieces of advice in the cardgameHearts into operational knowledge that can be understood and directly accessed .The basic pattern for a knight fork is a knight threatening two pieces, thereby winning one of them. Inthe endgame, these might simply be two unprotected pawns (unless one of them protects the other). In themiddlegame, these are typically higher-valued pieces (protected or not). However, this definition might notwork if the forking knight is attacked but not protected or even pinned. But then again, perhaps the attackingpiece is pinned as well. Or the pinned knight can give a discovered check . . .by the machine. For example, the user can specify the hint ―avoid taking points‖ andthe program is able to translate this piece of advice into a simple, heuristic search procedurethat determines the card that is likely to take the least number of points (Mostow1983). However, his system is not actually able to play a game of Hearts. In particular,his architecture lacks a technique for evaluating and combining the different pieces ofadvice that might be applicable to a given game situation.

F. Cognitive models

Psychological studies have shown that the differences in playing strengths betweenchess experts and novices are not so much due to differences in the ability to calculatelong move sequences, but to which moves they start to calculate (de Groot 1965; Chaseand Simon 1973; Holding 1985; de Groot and Gobet 1996; Gobet and Simon 2001)

For this pre-selection of moves chess players make use of patterns and accompanyingpromising moves and plans. Simon and Gilmartin (1973) estimate the number of achess expert's patterns to be of the order of 10,000 to 100,000. Similar results havebeen found for other games (Reitman 1976; Engle and Bukstel 1978;Wolff et al. 1984).

This seems to indicate that CHUMP re-usesonly few patterns, while it continuously generates new patterns.TAL (Flinter and Keane 1995) is a similar system which also uses a library ofchunks, which has also been acquired from a selection of Tal's games, for restrictingthe number of moves considered. It differs in the details of the representation of thechunks and the implementation of their retrieval. Here, the authors observed a seeminglylogarithmic relationship between the frequency of a chunk and the number ofoccurrences of chunks with that frequency.

Epstein (1994b, Epstein (2001) A special Advisor—PATSY—is able to make use of anautomatically acquired chunk library, and comments in favor of patterns that are associatedwith wins and against patterns that are associated with losses.These chunks areacquired using a collection of so-called spatial templates, a meta-language that allowsto specify which subparts of the current board configuration are interesting to be consideredas pattern candidates. Patterns that occur frequently during play are retainedand associated with the outcome of the game (Epstein et al. 1996). HOYLE is alsoable to generalize these patterns into separate, pattern-oriented Advisors. Similar toPATSY, another Advisor—ZONE RANGER—may support moves to a position whosezones have positive associations, where a zone is defined as a set of locations that canbe reached in a fixed number of moves. The patterns and zones used by PATSY and

ZONE RANGER are attempts to capture and model visual perception. There is alsosome empirical evidence that HOYLE exhibits similar playing and learning behaviour than human game players (Rattermann and Epstein 1995).

V. Conclusions

In this paper, we have surveyed research in machine learning for computer game playing. It is unavoidable that such an overview is somewhat biased by the author’s knowledge and interests, and our sincere apologies go to all authors whose work had to be ignored due to our space constraints or ignorance. Nevertheless, we hope that we have provided the reader with a good starting point that is helpful for identifying the relevant works to start one’s own investigations. If there is a conclusion to be drawn from this survey, then it should be that research in game playing poses serious and difficult problems which need to be solved with existing or yet-to-be-developed machine learning techniques

Acknowledgment

The heading of the Acknowledgment section and the References section must not be numbered.

Causal Productions wishes to acknowledge Michael Shell and other contributors for developing and maintaining the IEEE LaTeX style files which have been used in the preparation of this template. To see the list of contributors, please refer to the top of file IEEETran.cls in the IEEE LaTeX distribution.

References

[1] ALLIS, V. (1988, October). A knowledge-based approach of Connect-Four — thegame is solved: White wins. Master’s thesis, Department of Mathematics andComputer Science, VrijeUniversiteit, Amsterdam, The Netherlands.

[2] ANGELINE, P. J. & J. B. POLLACK (1994).Competitive environments evolve bettersolutions for complex tasks. In Proceedings of the 5th International Conferenceon Genetic Algorithms (GA-93), pp. 264–270.

[3] BAIN, M. (1994). Learning Logical Exceptions in Chess.Ph. D. thesis, Departmentof Statistics and Modelling Science, University of Strathclyde, Scotland.

[4] BAIN, M. & A. SRINIVASAN (1995). Inductive logic programming with large-scaleunstructured data. In K. Furukawa, D. Michie, and S. H. Muggleton (Eds.), MachineIntelligence 14, pp. 233–267. Oxford University Press.

[5] BAXTER, J., A. TRIDGELL, & L. WEAVER (1998a). A chess program that learnsby combining TD(lambda) with game-tree search. In Proceedings of the 15thInternational Conference on Machine Learning (ICML-98), Madison, WI, pp.28–36. Morgan Kaufmann.

[6] BEAL, D. F. & M. C. SMITH (1997, September). Learning piece values usingtemporal difference learning. International Computer Chess Association Journal20(3), 147–151.

[7] BERLINER, H., G. GOETSCH, M. S. CAMPBELL, & C. EBELING (1990). Measuringthe performance potential of chess programs. Artificial Intelligence 43,7–21.

[8] BHANDARI, I., E. COLET, J. PARKER, Z. PINES, R. PRATAP, & K. RAMANUJAM(1997). Advanced Scout: Data mining and knowledge discovery in NBA data.Data Mining and Knowledge Discovery 1, 121–125.

[9] BILLINGS, D. (2000a, March). The first international RoShamBo programmingcompetition. International Computer Games Association Journal 23(1), 42–50.

[10] BISHOP, C. M. (1995). Neural Networks for Pattern Recognition. Oxford, UK:Clarendon Press.

[11] BOYAN, J. A. (1992). Modular neural networks for learning context-dependentgame strategies. Master’s thesis, University of Cambridge, Department of Engineeringand Computer Laboatory.

[12] BRAFMAN, R. I. & M. TENNENHOLTZ (1999). A near-optimal polynomial timealogorithmfor learning in stochastic games.In Proceedings of the 16th InternationalJoint Conference on Artificial Intelligence (IJCAI-99), pp. 734–739.

[13] BRATKO, I. & R. KING (1994). Applications of inductive logic programming.SIGART Bulletin 5(1), 43–49.

[14] THRUN, S. (1995). Learning to play the game of chess. In G. Tesauro, D. Touretzky,and T. Leen (Eds.), Advances in Neural Information Processing Systems 7, pp.1069–1076. Cambridge, MA: The MIT Press.

[15] TIGGELEN, A. V. (1991). Neural networks as a guide to optimization. Thechess middle game explored. International Computer Chess Association Journal14(3), 115–118.