Recent Advances in Machine Learning and Game Playing

Johannes Fürnkranz
TU Darmstadt
Hochschulstrasse 10
D-64289 Darmstadt

Abstract

Game-playing applications offer various challenges for machine learning, including opening book learning, learning of evaluation functions, player modeling, and others. In this paper, we briefly highlight the most important achievements of this field and summarize some important recent advance.

1 Introduction

Game playing is a major application area for research in artificial intelligence in general [24], and for machine learning in particular [14, 6]. Traditionally, the field is concerned with learning in strategy games such as tic-tac-toe [19], checkers [22, 23], backgammon [34], chess [3, 5, 10, 21], Go [28], Othello [7], poker [4], or bridge [2]. However, recently computer and video games have received increased attention [18, 27, 20].

Since the early days of the field, game playing applications have been popular testbeds for machine learning. This has several reasons:

- *Games allow to focus on intelligent reasoning.* Other components of intelligent agents, such as perception or physical actions can be ignored.
- *Games are easily accessible.* A typical game-playing environment can be implemented within a few days, often hours. Exceptions are real-time computer games, for which only a few open-source test beds exist.
- *Games are very popular.* It is not very hard to describe the agent's task to the general public, and they can easily appreciate the achieved level of intelligence.

Game-playing applications offer various challenges for machine learning, including opening book learning, learning of evaluation functions, player modeling, and others. A wide variety of learning techniques have been used for tackling these problems. We cannot provide details on the learning algorithms here, but will instead focus on the problems.

This short communication is primarily meant to be an update to previous surveys [26, 12, 11]. We will briefly focus on the highlights of the history of this field and on the most relevant advances in in the last ten years.

2 Learning of Evaluation Functions

The most extensively studied learning problem in game playing is the automatic adjustment of the weights of an evaluation function. Typically, the situation is as follows: the game programmer has provided the program with a library of routines that compute important features of the current board position (e.g., the number of pieces of each kind on the board, the size of the territory controlled, etc.). What is not known is how to combine these pieces of knowledge and how to quantify their relative importance. Most frequently, these parameters are combined linearly, so that the learning task is to adjust the weights of a weighted sum. The main problem is that there are typically no direct target values that could be used as training signals. Exceptions are games or endgames that have been solved completely, which are treated further below. However, in general, algorithms use preference learning (where pairs of moves or positions are labeled according to which one is preferred by an expert player [32, 31, 13]) or reinforcement learning (where moves or positions are trained based on information about the eventual outcome of the game [30, 33]) for tuning the evaluation functions.

The key problem with reinforcement learning approaches is the credit assignment problem, i.e., even though a game has been won (lost), there might be bad (good) moves in the game. Reinforcement learning takes a radical stance at this problem, giving all positions the same reinforcement signal, hoping that erroneous signals will be evened out over time. An early classic in this area is MENACE, a tic-tac-toe player that simulates reinforcement learning with delayed rewards [19] using a stack of matchboxes, one for each position, each containing a number of beads in different colors, which represent the different legal moves in the position. Moves are selected by randomly drawing a bead out of the box that represents the current position. After a game is won, all played moves are reinforced by adding beads of the corresponding colors to these boxes, in the case of a lost game corresponding

beads are removed, thereby decreasing the probability that the same move will be played again.

The premier example of a system that has tuned its evaluation function to expert strength by playing millions of games against itself is the backgammon program TD-Gammon [33, 34]. Its key innovation was the use of a neural network instead of a position table, so that the reinforcement signal can be generalized to new unseen positions. Many authors have tried to copy TD-Gammon's learning methodology to other games [15]. None of these successors, however, achieved a performance that was as impressive as TD-Gammon's. The reason for this seems to be that backgammon has various characteristics that make it perfectly suited for learning from self-play. Foremost among these are the fact that the dice rolls guarantee sufficient variability, which allows to use training by self-play without the need for an explicit exploration/exploitation trade-off, and that it only requires a very limited amount of search, which allows to ignore the dependencies of search algorithm and search heuristic. These points have, for example, been addressed with limited success in the game of chess, where the program KnightCap [3], which integrates temporal difference learning [29] into a game tree search by using the final positions of the principal variation for updates, and by using play on a game server for exploration.

Many aspects of evaluation function learning are still discussed in the current literature, including whether there are alternatives to reinforcement learning (e.g., evolutionary algorithms), which training strategies should be used (e.g., self-play vs. play against a teacher), etc. One of the key problems, that has already been mentioned in Samuel's famous Checkers Player [22, 23], namely the automated construction of useful features, remains still largely unsolved. Some progress has, e.g., been made in the game of Othello, where a simple algorithm, very much like APriori [1] has been shown to produce valuable conjunctions of basic features [7].

3 Learning Search Control

A more challenging, but considerably less investigated task is to automatically tune the various parameters that control the search in game-playing programs. These parameters influence, for example, how aggressive the search algorithm is in pruning unpromising parts of the search tree and which lines are explored in more depth. The key problem here is that these parameters are intertwined with the search algorithm, and cannot be optimized independently, making the process very tedious and expensive.

There have been a few attempts to use explanation-based learning [8] to automatically learn predicates that indicate which branches of the search tree are the most promising to follow. These approaches are quite related to various uses of explanation-based learning in planning, but these could not be successfully be carried over to game-tree search.

Björnsson and Marsland [5] present a *gradient descent* approach that minimizes the total number of game positions that need to be searched in order to successfully solve a number of training problems. The idea is to adjust each parameter in proportion to its sensitivity to changes in the number of searched nodes, which is estimated with additional searches. The amount of positions that can be searched for each training position is bounded in order to avoid infinite solution times for individual problems, and simulated annealing is used to ensure convergence.

4 Opening Book Learning

Human game players not only rely on their ability to estimate the value of moves and positions but are often also able to play certain positions "by heart", i.e., without having to think about their next move. This is the result of home preparation, opening study, and *rote learning* of important lines and variations. As computers do not forget, the use of an opening book provides an easy way for increasing their playing strength. However, the construction of such opening books can be quite laborious, and the task of keeping it up-to-date is even more challenging.

Commercial game-playing programs, in particular chess programs, have thus resorted to tools that support the automatic construction of opening from large game databases. The key challenge here is that one cannot rely on statistical information alone: a move that has been successfully employed in hundreds of games may be refuted in a single game. Donninger and Lorenz [10] describe an approach that evaluates the "goodness" of a move based on a heuristic formula that has been found by experimentation. This value is then added to the result of a regular alpha-beta search. The technique has been so successful, that the chess program Hydra [9], probably the strongest chess program today, has abandoned conventional large man-made (and therefore error-prone) error books. Similar techniques have also been used in games like Othello [7].

5 Pattern Discovery

In addition to databases of common openings and huge game collections, which are mostly used for the tuning of evaluation functions or the automatic generation of opening books (see above), many games or subgames have already been solved, i.e., databases are available in which the game-theoretic value of positions of these subgames can be looked up [35]. For example, in chess, all endgames with up to six pieces, in checkers all 10-piece endgames have been solved [25]. In other games, such as Connect-4, are solved completely, i.e., all possible positions have been evaluated and the game-theoretic value of the starting position has been determined. Many of these databases are readily available, some of them (in the domains of chess, Connect-4 and tic-tac-toe) are part of the UCI repository for machine-learning databases [16].

The simplest learning task is to train a classifier that is able to decide whether a given game position is a game-theoretical win or loss (or draw). In many cases, this is insufficient. For example, in the chess endgame king-rook-king, any position in which the white rook cannot be immediately captured, and in which black is not stalemate is, in principle, won by white. However, in order to actually win the game it is not sufficient to simply make moves that avoid rook captures and stalemates. Thus, most databases contain the maximal number of moves that are needed for winning the position. Predicting this is a much harder, largely unsolved problem (some recent work can be found in [21]). In addition to the game-specific knowledge that could be gained by the extraction of patterns that are indicative of won positions, another major application could be a knowledge-based compression of these databases (the collection of all perfect-play chess endgame databases with up to 6 man is 1.2 Terabytes in a very compressed database format, the win/loss checkers databases with up to 10 men contain about 4×10^13 positions compressed into 215GB [25]).

6 Player Modeling

Player modeling is an important research area in game playing, which can serve several purposes. The goal of *opponent modeling* is to improve the capabilities of the machine player by allowing it to adapt to its opponent and exploit his weaknesses. Even if a game-theoretical optimal solution to a game is known, a system that has the capability to model its opponent's behavior may obtain a higher reward. Consider, for example, the game

of *rock-paper-scissors* aka *RoShamBo*, where either player can expect to win one third of the games (with one third of draws) if both players play their optimal strategies (i.e., randomly select one of their three moves). However, against a player that always plays *rock*, a player that is able to adapt his strategy to always playing *paper* can maximize his reward, while a player that sticks with the "optimal" random strategy will still win only one third of the games. One of the grand challenges in this line of work are games like poker, where opponent modeling is crucial to improve over game-theoretically optimal play [4]. A recent approach to opponent modeling in poker can be found in [17].

Player modeling is also of increasing importance in commercial computer games (see below). For one, behavioral cloning techniques could be used to increase the playing strength or credibility of artificial characters by copying the strategies of expert human players. Moreover, the playing strength of the characters can be adapted to the increasing skill level of the human player. Finally, agents that can be trained by non-programmers can also play an important role. For example, in Massive Multiplayer Online Role-playing Games (MMORGs) an avatar that is trained to simulate a user's game-playing behavior, could take his creator's place at times when the human player cannot attend to his game character.

7 Commercial Computer Games

In recent years, the computer games industry has discovered Artificial Intelligence as a necessary ingredient to make games more entertaining and challenging and, vice versa, AI has discovered computer games as an interesting and rewarding application area [18]. In comparison to conventional strategy games, computer game applications are more demanding, as the agents in these game typically have to interact with a large number of partner or enemy agents in a highly dynamic, real-time environment, with incomplete knowledge about its states. Tasks include off-line or on-line player modeling (see above), virtual agents with learning capabilities, optimization of plans and processes, etc.

Despite the large commercial potential, research in this area has just started, and the number of workshops and publications on this topic is rapidly increasing. One particularly interesting approach is *dynamic scripting* [27], an on-line reinforcement learning technique that is designed to be integrated into scripting languages of game playing agents. Contrary to conventional reinforcement learning agents, it updates the weights of all actions

for a given state simultaneously. This sacrifices guaranteed convergence, but this is desirable in a highly dynamic game environment. The approach was successfully applied to improving the strength of computer-controlled characters, and increasing the entertainment value of the game by automated scaling of the difficulty level of the game AI to the human player's skill level. Similar to the problem of constructing suitable features for the use in evaluation functions, the basic tactics of the computer player had to be handcoded. Ponsen et al. [20] extend dynamic scripting with an evolutionary algorithm for automatically constructing the tactical behaviors.

References

[1] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1995.

[2] Asaf Amit and Shaul Markovitch. Learning to bid in bridge. *Machine Learning*, 63(3):287–327, 2006. Special Issue on Machine Learning and Games

[3] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Learning to play chess using temporal differences. *Machine Learning*, 40(3):243–263, September 2000.

[4] Darse Billings, Lourdes Peña, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, January 2002. Special Issue on Games, Computers and Artificial Intelligence.

[5] Yngvi Björnsson and T. Anthony Marsland. Learning extension parameters in game-tree search. *Information Sciences*, 154(3-4):95–118, 2003.

[6] Michael Bowling, Johannes Fürnkranz, Thore Graepel, and Ron Musick. Special issue on machine learning and games. *Machine Learning*, 63(3), June 2006.

[7] Michael Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1-2):85–99, January 2002. Special Issue on Games, Computers and Artificial Intelligence.

[8] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1:145–176, 1986.

[9] Chrilly Donninger and Ulf Lorenz. The Hydra project. *XCell Online Journal*, Second Quarter, 2005.

[10] Chrilly Donninger and Ulf Lorenz. Innovative opening-book handling. In H. Jaap van den Herik, Shun-Chin Hsu, and H. H. L. M. Donkers, editors, *Advances in Computer Games 11*. Springer-Verlag, 2006.

[11] Johannes Fürnkranz. Machine learning in games: A survey. In Fürnkranz and Kubat [14], chapter 2, pages 11–59.

[12] Johannes Fürnkranz. Machine learning in computer chess: The next generation. *International Computer Chess Association Journal*, 19(3): 147–160, September 1996.

[13] Johannes Fürnkranz and Eyke Hüllermeier. Preference learning. *Künstliche Intelligenz*, 19(1):60–61, 2005.

[14] Johannes Fürnkranz and Miroslav Kubat, editors. *Machines that Learn to Play Games*. Nova Science Publishers, Huntington, NY, 2001.

[15] Imran Ghory. Reinforcement learning in board games. Technical Report CSTR-04-004, Department of Computer Science, University of Bristol, Bristol, UK, 2004.

[16] Seth Hettich, Catherine L. Blake, and Christopher J. Merz. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1998. Department of Information and Computer Science, University of California at Irvine, Irvine CA.

[17] Levente Kocsis and Csaba Szepesvári. Universal parameter optimisation in games based on SPSA. *Machine Learning*, 63(3):249–286, 2006. Special Issue on Machine Learning in Games.

[18] John E. Laird and Michael van Lent. Human-level AI's killler application: Interactive computer games. *AI Magazine*, 22(2):15–26, 2001.

[19] Donald Michie. Experiments on the mechanization of game-learning – Part I. Characterization of the model and its parameters. *The Computer Journal*, 6:232–236, 1963.

[20] Marc Ponsen, Héctor Muñoz-Avila, Pieter Spronck, and David W. Aha. Automatically generating game tactics via evolutionary learning. *AI Magazine*, 27(3):75–84, 2006.

[21] Aleksander Sadikov and Ivan Bratko. Learning long-term chess strategies from databases. *Machine Learning*, 63(3):329–340, 2006. Special Issue on Machine Learning in Games.

[22] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

[23] Arthur L. Samuel. Some studies in machine learning using the game of checkers. II — recent progress. *IBM Journal of Research and Development*, 11(6):601–617, 1967.

[24] Jonathan Schaeffer and H. Jaap van den Herik, editors. *Chips Challenging Champions: Games, Computers and Artificial Intelligence*. North-Holland Publishing, 2002. Reprint of a Special Issue of *Artificial Intelligence* 134(1-2).

[25] Jonathan Schaeffer, Yngvi Björnsson, Neil Burch, Robert Lake, Paul Lu, and Steve Sutphen. Building the checkers 10-piece endgame databases. In H. Jaap van den Herik, Hiroyuki Iida, and Ernst A. Heinz, editors, *Advances in Computer Games 10*, pages 193–210. Springer-Verlag, Graz, Austria, 2003.

[26] Steven S. Skiena. An overview of machine learning in computer chess. *International Computer Chess Association Journal*, 9(1):20–28, 1986.

[27] Pieter Spronck, Marc J. V. Ponsen, Ida G. Sprinkhuizen-Kuyper, and Eric O. Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006. Special Issue on Machine Learning in Games.

[28] David Stern, Ralf Herbrich, and Thore Graepel. Bayesian pattern ranking for move prediction in the game of Go. In *Proceedings of the 23rd International Conference on Machine Learning (ICML-06)*, pages 873–880, 2006.

[29] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[30] Richard S. Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[31] Gerald Tesauro. Comparison training of chess evaluation functions. In Fürnkranz and Kubat [14], chapter 6, pages 117–130.

[32] Gerald Tesauro. Connectionist learning of expert preferences by comparison training. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 1 (NIPS-88)*, pages 99–106. Morgan Kaufmann, 1989.

[33] Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–278, 1992.

[34] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995.

[35] H. Jaap van den Herik, Jos W. H. M. Uiterwijk, and Jack van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, 2002. Special Issue on Games, Computers, and Artificial Intelligence.