Learning to Play Draughts using Temporal Difference Learning with Neural Networks and Databases

Jan Peter Patist and Marco Wiering
Cognitive Artificial Intelligence
Utrecht University
contact: peter.patist@phil.uu.nl

Abstract

This paper describes several aspects of using temporal difference learning (TD) and neural networks to learn game evaluation functions, and the benefits of using databases. Experiments in tic-tac-toe and international draughts have been done to measure the effectiveness of using databases. The experiment of Tic-Tac-Toe showed that training from database games resulted in better play than learning from playing against a random player. In the experiment of learning international draughts, the program reached after just a few hours of training a better level than a strong computer playing program and occasionally drew a very strong program. Thus, using temporal difference learning and neural networks on database games is a time-efficient way to reach a considerable level of play.

1 Introduction

Reinforcement learning (Sutton, 1988; Kaelbling et al., 1996) is a learning method which enables agents to learn to act in an environment by getting rewards. The agent chooses between actions based on the current sensory information and a function approximator to map sensory information to an output. This output can be an action or a value used in selecting an action. The agent uses a function approximator to map sensory information to an output which will result in the correct behavior. Rewards are used by the agent to adjust his function approximator to maximize his future cumulative reward. In game-playing the player or agent chooses a move which maximizes the chance to win minus the chance to loose.

Several ways exist to collect the games or examples needed to learn from. For example by self-play, playing against humans or by considering a database of already played games. However in games like draughts much look-ahead is needed to create a game which is good enough to learn from. Therefore, learning by playing an actual game is time consuming. Using databases of existing games is much less time consuming because look-ahead is not needed. Another benefit of using databases over self-play is that the examples are representative of good game-playing and do not suffer from start-up problems like in self-play when a randomly initiated evaluation function is used.

Games. For a long time now board games like draughts, checkers, backgammon, chess etc. have gained the interest of computer science. Nowadays there are programs which are playing at a high level. For example in chess the number one human player, Kasparov, has been beaten by Deep Blue[1997] and Deep Junior[2003]. Also in other board games the computer has gained ground, like in backgammon, othello, international draughts and checkers. In international draughts several grandmasters were defeated by the programs Buggy[2003] and Flits[2002].
The success of most of the strongest programs in these board games is due to the use of the possibility to look upon many thousands of positions per second by using computer power and efficient search algorithms. Because of the exponential growth of the amount of positions in respect to the search depth, it is impossible to calculate all possible continuations until a position which ends by the rules of the game. Several algorithms and heuristics have been developed to cut the continuations looked upon. Although the computer can search many positions it can not reach an end position from most of the positions. Therefore a good evaluation function is very important

to estimate 'the chance' to win given a certain board position. In most of the mentioned board games the evaluation function is tuned by the system designer and this may cost a lot of time. Automated learning can probably result in better evaluation functions in less time. For example the backgammon program TD-Gammon (Tesauro, 1995) reached the level of a world class player by learning using neural networks and TD-learning (Sutton, 1988).

Learning from database games. In reinforcement learning agents act in an environment and get feedback on their performance by reward signals. They use that feedback to adjust themselves to maximize their performance. However, when an agent learns from database games it does not apply its newly acquired insights directly to the board and is therefore not confronted with its behavior. Database games are only labelled with the end-results of a game (i.e. win, loss, or draw). We use a reinforcement learning method to estimate values of board positions that have occurred during a game. In many board-games like draughts, chess, etc. a database exists. Using temporal difference learning is a good way to make use of this rich collection of knowledge.

Overview. In the next section we describe and explain shortly the ingredients of a game-playing program and some learning game-programs. Reinforcement learning will be discussed in section 3. Experiments of learning Tic-Tac-Toe using TD(λ)-learning and databases will be presented in section 4, and in section 5 we present results on learning international draughts. Finally, section 6 concludes and describes possible future work.

2 Game playing

2.1 Parts of a game-playing program

Almost all programs which play board-games like draughts have the same architecture. The different parts of the programs are a move generator, a search engine and an evaluation function. Given a certain position, the move generator enables generating all possible successor positions. The search engine is an algorithm which calculates possible continuations. Examples of search algorithms are Alpha-Beta, Negascout, Principal Variation Search etc. See (Plaat, 1996) for more information on game search algorithms. Normally search algorithms are extended with heuristics to enhance speed. An evaluation function is used to value a position. In our experiments we used a neural network because of its ability to represent non-linear functions (Cybenko, 1989), which is important in draughts because a good evaluation function can probably not be represented by a linear function. It should be noted that the use of neural networks slows down the search speed. Because it is very difficult for programs to play well in the opening and end-game-phase they are usually equipped with a database for both phases. In most programs the parameters of the evaluation function are set by hand. However this costs a lot of time.

2.2 Learning Programs

Examples of automated learning evaluation functions are TD-Gammon in backgammon, KnightCap (Baxter et al., 1997; Baxter et al., 1998; Baxter et al., 2000) and NeuroChess (Thrun, 1995) in chess, and NeuroDraughts (Lynch and Griffith, 1997) in checkers. TD-Gammon became world class player by learning from self-play. KnightCap learned by TDleaf, a TD-method for game-trees and reached a reasonable level of play in fast-play chess. NeuroChess used an explanation-based neural network (EBNN) which predicted the position after some moves and was trained by grandmaster games. Gradient information of the EBNN was used to adjust the weights of another neural network V trained by self-play. The EBNN was trained before V. The level of play was weak. In Neurodraughts a cloning strategy was used to test, select and train networks with different input features. It is not known how strong this program plays against other checkers playing programs or humans.

3 Reinforcement Learning

3.1 Markov Decision Problems

Most work done in Reinforcement Learning (RL) is about solving Markov decision problems. A problem is a Markov decision problem whenever it satisfies the Markov property. The Markov property states that the transition probabilities to possible next states only depend on the current state and the selected action. A

Markov decision problem is characterized by the possible states and actions, a transition function, a reward function, and a discount factor. The transition function contains all transition probabilities of transitions that are made to a next state, under influence of some action. The reward function is determined by the system designer. The discounting factor affects the greediness towards immediate or future reward. Playing draughts can be seen as a Markov decision problem as long as the opponent uses a fixed strategy for selecting moves.

3.2 Temporal Difference Learning

To estimate the values of positions we used the temporal difference method (Sutton, 1988). The Temporal Difference method is a RL-method driven by the difference between two successive state values to adjust former state values which decreases the difference between all two successive state values. The change in the value of the state is equal to a learning parameter α multiplied by the sum of the temporal difference errors between two successive state values. These temporal differences are weighed exponentially according to the difference in time.

The state values are updated using the TD(λ) update rule, defined as:

$$\triangle V(s_t) = \alpha_{s_t} e_t^{\lambda} \tag{1}$$

$$e_t^{\lambda} = \sum_{i=0}^{T-t} \lambda^i e_{t+i} \tag{2}$$

$$e_t = (r_t + V(s_{t+1}) - V(s_t)) \tag{3}$$

where:

- $V(s_t)$:= Value of state s at time t
- T:=length of the trajectory
- r_t := reward at time t
- $e_t = TD(0)$ error between time-step t and t+1

Because we learn after each game all the needed TD(0)-errors are known, and in our work we therefore essentially use offline TD(λ) learning. In our research the TD-rule is implemented in an incremental way which is derivable from the TD-rule.

3.3 RL and Neural Networks

Because in draughts the amount of possible states or positions is too large to directly store and learn from, a function approximator must be used to approximate the value of the positions. In our research we used a feed-forward neural network as a function approximator. To learn from a game all positions are scored by propagating it through the neural network. New values of the positions are estimated after each game by using the TD-update rule. These new values are the target values of the positions seen in the game. Extended error back-propagation (Sperduti and Starita, 1993) is used to adjust the weights of the network in order to minimize the difference between the target value and the estimated value of the position, called error. In our experiments we used the activation function $\beta x/(1 + abs(\beta x))$ with neuron sensitivity β. The neuron sensitivity is adjusted according to the partial derivative of the error to the neuron sensitivity. Neuron sensitivity is used to speed up the learning. See (Bishop, 1995) for more information about feed-forward neural networks and (Wiering, 1999; Kaelbling et al., 1996) for more information on reinforcement learning.

4 Experiments

4.1 Tic-Tac-Toe

Tic-Tac-Toe is a simple children's game. It is played on a 3 by 3 squared board. The players put turnwise the mark X, O on a empty square on the board. If one player makes a move, which brings about three of his marks in diagonal, horizontal or vertical order, he wins. If all the squares are filled and the position is not a winning position, it is a draw. By learning the game, we mean to be able to play near optimal. To play optimal is to get the highest theoretical win-loss. For that we define 'equity' by the number of wins minus the number of losses divided by the games played. In our experiment we compared the performance of play of neural networks with different amounts of hidden units trained by playing against a random player versus training from database games. The performance is tested by playing against a predefined player, who we will call Expert that plays according to the following rules:

-If there are moves X which makes a win Then choose random between moves X
-Else If there are moves possible for opponent to win Then choose random a move which blocks one of these moves by playing on the same field
-Else play a random move

4.2 Representation

For each field on the board there is an input node reserved in the input vector of the neural network. An empty field is represented by the value zero, a field with X the value 1 and a O by -1. A win by X is evaluated by the reward function by 1, a win by O by -1, a draw by 0. And 0 reward for all other moves. The representation of 1, 0, -1 is chosen because of symmetry considerations.
The neural network player selects a move by calculating all his possible moves. These moves result in a new position. This position is the input of the neural network. The output of the neural network is the value of the position. Whenever no exploration is used the agent chooses the move which results in the position with the highest value (this is used in testing the networks).

4.3 Simulations

In all of the above experiments networks with 40, 60 and 80 hidden neurons are used. All experiments are averaged over 10 simulations. In the case of learning from playing against a random player a simulation amounts to 40,000 training games. In case of database learning the amount of 160,000 games is used, which are randomly selected from the database. The database is filled with 40.000 games played by Expert against a neural network during the training phase of the neural network. When a database is used in learning no games have to be played. So move generation and evaluation is not needed and this saves time. In the same time more games can be used in learning. This is the reason why 160,000 games are used in respect to the 40,000 games in the other experiments. After each 2,000 games, in the case of playing against random and 10.000 games in the case of learning from the database, the neural network is tested by playing 2,000 test games against Expert. The average of the equity of the 10 simulations per 2,000 games is taken.

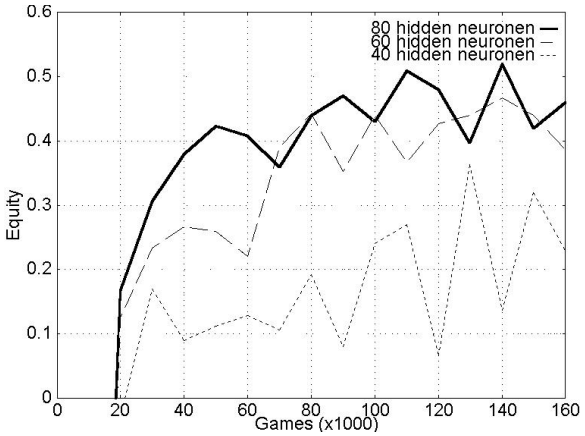

Figure 1: Learning from a database tested against Expert

4.4 Network and learning parameters

- Hidden units : 80,60,40
- Learning rate : 0.008
- Initial weights : random between -0.2 and 0.2
- Initial hidden neuron sensitivity β : 5
- Activation function hidden neurons : $\frac{\beta x}{1+abs(\beta x)}$
- Activation function output neuron : unity function
- Learning rate sensitivity hidden neurons : 0.00003
- Lambda: 0.8

4.5 Experimental Results

The maximal theoretical equity against Expert is about 0.61 (Wiering, 1995). All the networks which learned from playing against a random player reached all near optimal play against random of 0.928. After a network has learned to play near optimal against Random of course he will not play near optimal against Expert. The reason is very simple, namely because optimal play against random is different than playing optimal against Expert. But the optimal play against Expert and Random have some common strategies. The Network is able to learn some forks, which is the only way to win again Expert, from playing against Random.

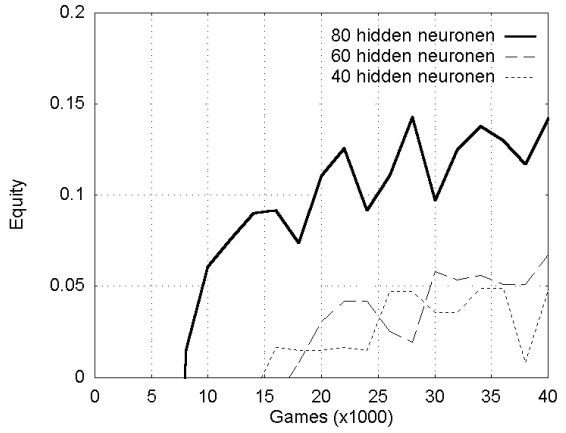

Figure 2: Learning from playing against a random player tested against Expert

Learning by database results in play which is not as good as a network trained by playing against Expert, which achieved a near optimal equity of 0.58(Wiering, 1995). This is because the database consists of bad and inconsistent games. In the case of the networks with size 80 learning from the database gives the best performance. In the test-phase the networks reached a high win-percentage however the loss-percentage hampered a higher equity.

The variance of the Equity after database learning is very high. Probably this is because the database contains many inconsistent and bad games. Learning from playing Expert yields the best play. This is because it is trained and tested against the same opponent. The biggest network performs the best. Database learning achieves higher equity than learning by playing against Random.

5 Learning to Play Draughts

It should be noted that with draughts we mean international draughts played on a 10x10 squared board. The game is played by two opponents on the 50 dark squares of the board (as shown in figure 3). In the starting position both players occupy the first four rows of either white or black checkers. The players move turnwise beginning with the white player. The goal of the game is to achieve a position in which the opponent can not make a move. If neither player can accomplish this it is a draw. Checkers can move one square at a time in a diagonal forward direction, to an unoccupied square or field. Checkers capture by jumping over an opposing man on a diagonal adjacent square to a unoccupied square directly beyond it. Checkers can jump in any diagonal direction and may continue as long as they encounter opposing checkers with unoccupied squares immediately beyond them. The capture with the maximal amount of captured checkers is obliged. When a checker reaches the far side of the board it becomes a king. Kings can move in a diagonal line to an unoccupied square only if all squares on the diagonal line are unoccupied. King capture at any distance along a diagonal direction with at least one unoccupied square immediately beyond it. Kings continue jumping from one of these squares till no jumping is possible. Kings and checkers are not allowed to jump more than one time over an opposing checker or King.

5.1 Experimental Setup

The games used in training are extracted from Turbo Dambase[1], the only draughts-games database and contains approximately 260.000 games. The database contains games between players of all different levels. For testing purposes the neural network was incorporated in the draughts program Buggy[2] which is probably the best draughts program of the world and uses state-of-the-art search algorithms. Because it is almost impossible to learn from the raw board position alone, we represented the board position by features.

5.2 Input features

We used 3 different kinds of features namely global-features, structural features and raw board representation. In the raw board representation every field of the game position is decoded to represent the presence of the kind of piece. Every square is represented by two bits. One bit to represent the occupance of a white single checker and one for a single black checker. We did not use raw-board presentation for kings. Structural features are boolean combinations of occupied or empty squares. Unlike structural features and the raw-board representation, global features model knowledge which is

[1]For information on Turbo Dambase, see http://www.turbodambase.com/

[2]For information on Buggy, see http://www.buggy-online.com/

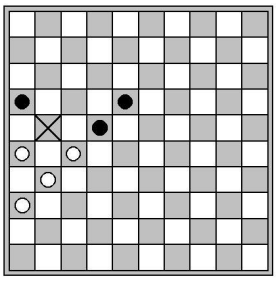

Figure 3: The structural feature 'Hekstelling'

not directly available from a board position. An example of a global-feature is material balance. Material balance is the difference in checkers plus 3 times the difference in kings. An example of a structural feature is 'Hekstelling' shown in figure 3.

The total amount of inputs, which are 23 global plus 99 structural features plus 100 raw board features, is 222. Five neural networks were trained. The difference between the first four neural networks lies only in the representation of a board position. We define the four neural network players in respect to the input, namely:

- NN1: Global features
- NN2: Global features plus all possible 'three-on-a-diagonal'
- NN3: All features except the raw board representation
- NN4: All features

The fifth neural network consists of three different neural networks. The neural networks were used in training or playing on the basis how many pieces were on the board. One for the position with more than 25 pieces, one for more than 15 till 25 pieces and one for the resting positions, for which all three were equipped with all features. All the neural networks were trained a single time over the first 200.000 games of Turbo Dambase. Positions in which a player was obliged to hit were eliminated from the game. Also all positions were eliminated which are a part of a tactical combination or a local discontinuity in material balance. This is for example when a player successively sacrifices two checkers in two moves and regains material balance on the third move by hitting two checkers. The reason why we did this is because learning on these positions distorted the learning process. The neural network has great difficulty in learning specific board settings in order to compensate the temporal difference in material.

5.3 Network and learning parameters

- Hidden units : 80
- Learning rate NN1, NN2 : 0.001
- Learning rate NN3, NN4, NN5 : 0.0005
- Initial weights : random between -0.2 and 0.2
- Initial hidden neuron sensitivity β: 3
- Activation function hidden neurons : $\frac{\beta x}{1+abs(\beta x)}$
- Activation function output neuron : unity function
- Learning rate sensitivity hidden neurons NN1, NN2 : 0.01
- Learning rate sensitivity hidden neurons NN3, NN4, NN5 : 0.005
- Lambda: 0.9

5.4 Testing

The four neural networks, created by learning from 200.000 games, were tested against two draughts programs available on the internet. DAM 2.2[3] is a very strong player and GWD[4] a strong player on the scale of very weak, weak, medium, strong and very strong (according to http://perso.wanadoo.fr/alemannia page76_e.html). Each network played one match of 10 games, 5 with white and 5 with black against both computer programs. In all the games both players got 4 minutes time. In the programs GWD and DAM 2.2. it was only possible to give the computer the amount of time per move. So GWD and DAM 2.2 got 4 seconds per move. Furthermore a round tournament was held with only the neural network

[3]For information and download, see http://www.xs4all.nl/ hjetten/dameng.html

[4]For information and download, see http://www.wxs.nl/ gijsbert.wiesenekker/gwd4distrib.zip

program:	*GWD*			
NN-...	WIN	LOSS	DRAW	RESULT
NN1	3	4	3	9-11
NN2	4	1	5	13-7
NN3	7	3	0	14-6
NN4	8	1	1	17-3

Table 1: The result of the matches against the program GWD. A win is awarded with 2 points, a draw with 1 point.

program:	*DAM 2.2*			
NN-...	WIN	LOSS	DRAW	RESULT
NN1	0	9	1	1-19
NN2	0	9	1	1-19
NN3	0	7	3	3-17
NN4	0	9	1	1-19

Table 2: The result of the matches against the program DAM 2.2. A win is awarded with 2 points, a draw with 1 point.

players. So each player played 10 games against another player using 4 minutes per game per player. No program made use of an opening book.

5.5 Results

All neural networks, except NN1 were able to beat GWD, the networks occasionally drew against DAM 2.2. as shown in Tables 1 and 2. In the round tournament NN3 and NN4 have beaten NN1 and NN2 as shown in Table 3.

5.6 Discussion

The Networks NN2, NN3 and NN4 were able to defeat GWD on a regular basis. Most of the time NN1 had a superb position against GWD. In many of the games the advantage in development for NN1 was huge and NN1 was controlling all the center fields. However sometimes NN1 made it possible for GWD to get a clear way to promotion. Also sometimes it built uncomfortable formations. The reason why it did this is because it does not have enough input features. All networks were not able to compete against Dam 2.2. Some of the networks were able to draw sometimes. NN4 is probably the best network. NN2 had also the problem of building bad structures sometimes. NN3 and NN4 are much better playing networks than NN1 and NN2. A problem of NN3 and NN4 is the evaluation of positions with structures where break off and rebuilding of the structure plays an important role. Probably this is because of the input features and the lack of an opening book. The networks NN3 and NN4 only have one feature for these kind of positions. This feature is only set on when the structure is on the board. That's probably why it is very difficult to learn using these features whether the positions, where the strategy is applicable to obtain the structure, are good or bad for the opponent. It is to be noted that this is of course when look-ahead does not reach to the structure. Another problem in the play is the opening. In some openings the networks want to take a front on 22 or 24 for white, 27 or 29 for black. The numbers correspond to fields on the board. If we look at the board with the black pieces going down, the number count is 1 to 50 from left above to right below in reading order. In the opening this is most of the time not a good idea. One simple strategy is to attack this piece a couple of times and exchange it. This results immediately in disadvantage in development and splitting the opponent's structure. The problem of the opening is that the positions are the most far positions from the end position. Because of this, the variance in the value of the features is very small. Therefore some small differences on the board can lead to a considerable advantage. The opening has been kind of a problem also for normal draughts playing computers. The networks were trained over a whole game. However because an endgame has nothing to do with the middle game inference can be a problem. We also tried an experiment with 3 neural networks for the opening, the middle game and endgame. However the network was not able to learn the material function in the opening. The reason is

x	NN1	NN2	NN3	NN4
NN1	x	3-3-4	2-6-2	4-6-0
NN2	3-3-4	x	1-6-3	3-5-2
NN3	6-2-2	6-1-3	x	4-5-1
NN4	6-4-0	5-3-2	5-4-1	x

Table 3: The results of the round tournament between the neural networks. A cross means no games are played. 3-3-4 means the row player won 3 games, lost 3 games and drew 4 games.

that there are too few examples of positions in the opening with piece (dis)advantage.

6 Conclusion

We have seen that the neural networks, trained on draughts database games, were able to beat on a regular basis a strong computer program and occasionally able to draw a very strong program. It only used 200.000 training games and lasted approximately 5 hours. We showed that adding a raw board presentation and global- and structural- information improved the level of play and probably adding more features can improve the level of play even further.

6.1 Future Work

We trained a neural network on a database of 200.000 games. This resulted in an evaluation function which is able to beat a good draughts program and occasionally draw a very good program. However now rises the question how we can improve this. First the amount of features can be extended. Research can be done in ordering the database in such a way that it will lead to better play. The games can be ordered in result, common positions, different level of players etc. Although the experiment of 3 neural networks for the opening, the middle-game and endgame failed to learn the material function we should not forget this idea. The reason to learn three networks was to overcome possible unpleasant inference. However some features we want to infer. For example the material feature. Some features we want to model locally in respect to some kind of position and some globally. Perhaps it is also an idea to use a minimum of look-ahead to also learn on concurrent positions. Other things to research is the combination of learning from database games and learning by self-play and playing against other computer programs or human players.

References

Baxter, J., Tridgell, A., and Weaver, L. (1997). Knightcap: A chess program that learns by combining TD(λ) with minimax search. Technical report, Department of Systems Engineering, Australian National University.

Baxter, J., Tridgell, A., and Weaver, L. (1998). Experiments in parameter learning using temporal differences. *International Computer Chess Association Journal*, 21(2):84–99.

Baxter, J., Tridgell, A., and Weaver, L. (2000). Learning to play chess using temporal differences. *International Computer Chess Association Journal*, 40(3):243–263.

Bishop, C. M. (1995). *Neural Networks for Pattern recognition*. Oxford University, New york.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Signals and Systems*, 2:303–314.

Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Leouski, A. (1995). Learning of position evaluation in the game of othello. Technical Report UM-CS-1995-023, Department of Computer Science University of Massachusetts, Amherst, MA.

Lynch, M. and Griffith, N. (1997). Neurodraughts: the role of representation, search, training regime and architecture in a TD draughts player. *Eighth Ireland Conference on Artificial Intelligence*, pages 64–72.

Plaat, A. (1996). *Research Re:Search and Research*. PhD thesis, Erasmus University, Amsterdam.

Sperduti, A. and Starita, A. (1993). Speed up learning and network optimization with extended back propagation. *Neural Networks*, 6:365–383.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

Tesauro, G. (1995). Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58–68.

Thrun, S. (1995). Learning to play the game of chess. *Advances in Neural Information Processing Systems (NIPS)*, 7:1069–1076.

Wiering, M. (1995). *TD-Learning of Game Evaluation Functions with Hierarchical Neural Architectures*. Department of Computer Systems, University of Amsterdam April, 1995

Wiering, M. (1999). *Explorations in Efficient Reinforcement Learning*. PhD thesis, University of Amsterdam, The Netherlands, 1999.