Advances in Computer Science and Information Technology (ACSIT)
p-ISSN: 2393-9907; e-ISSN: 2393-9915; Volume 3, Issue 3; April-June, 2016, pp. 213-215
© Krishi Sanskriti Publications
http://www.krishisanskriti.org/Publication.html

AI for Checkers using Artificial Neural Networks

Darshan Patel[1], Shashank Rao[2] and Aayush Kubitikar[3]

[1,2,3]Department of Information Technology, Sardar Patel Institute of Technology

Abstract—*The idea behind this project is to develop an AI for the game of checkers that uses its experiences or certain events to implement moves in its lifetime. These events are then used to influence the future behavior of the program i.e. the program's decision about whether a move to be made is a good move or not is influenced by both the program's understanding of the rules of the game of checkers and by the history of games played. Instead of giving the program a static idea of how to play a game of checkers we will be making its understanding more fluid in which the code can get better or worse at playing the game.*

Keywords*: Board Games, Reinforcement Learning, TD (λ), Self-play*

1. INTRODUCTION

Artificial Intelligence approaches can be explained and can be experimented using board games because of their complexity of game tree space. The performance of various algorithms and techniques can be evaluated by these board games. Checkers incorporates a stochastic process therefore the game tree space is too large to lend itself to the traditional method of search. A computer program(Blondie24) that succeeded in doing so is a major milestone. The goal of this project was to recreate this major milestone using a different algorithm. In some cases, reinforcement learning algorithms perform better. We will be training the network with help of self-play and testing it against humans. The AI will then update the network weights using back propagation, thus improving its gameplay. We will be combining temporal difference method with back propagation for error correction. The truly remarkable aspect of this approach is that the computer program is self-taught. The weights are randomly initialized (initialized to 0) and then temporal difference algorithm is used to train the network.

2. RELATED WORK

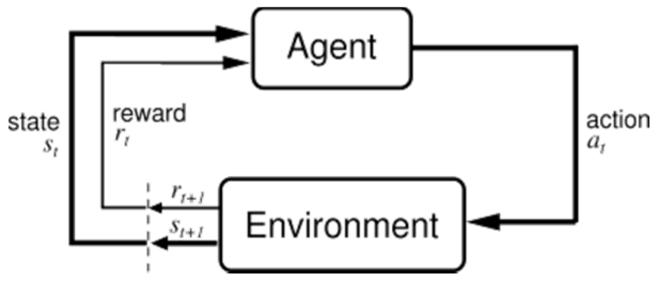
There has been significant work done in this field which includes creation of AI like "Blondie24" and "TD-Gammon".

Blondie24: It is an artificial intelligence checkers-playing computer program named after the screen name used by a team led by David B. Fogel. The purpose was to determine the effectiveness of an artificial intelligence checkers-playing computer program. Blondie24 played against some 165 human opponents and was shown to achieve a rating of 2048, or better than 99.61% of the playing population of that web site. The design of Blondie24 is based on a minimax algorithm of the checkers game tree in which the evaluation function is an artificial neural network. The neural net receives as input a vector representation of the checkerboard positions and returns a single value which is passed on to the minimax algorithm. The weights of the neural network were obtained by an evolutionary algorithm (an approach now called neuroevolution). In this case, a population of Blondie24-like programs played each other in checkers, and those were eliminated that performed relatively poorly. Performance was measured by a points system: Each program earned one point for a win, none for a draw, and two points were subtracted for a loss. After the poor programs were eliminated, the process was repeated with a new population derived from the winners. In this way, the result was an evolutionary process that selected programs that played better checkers games. The significance of the Blondie24 program is that its ability to play checkers did not rely on any human expertise of the game. Rather, it came solely from the total points earned by each player and the evolutionary process itself. The evolving players did not even know which individual games ended in a win, loss, or draw.

TD-Gammon is a neural network that trains itself to be an evaluation function for the game of backgammon by playing against itself and learning from the outcome. Although TD-Gammon has greatly surpassed all previous computer programs in its ability to play backgammon, that was not why it was developed. Rather, its purpose was to explore some exciting new ideas and approaches to traditional problems in the field of reinforcement learning. Each turn while playing a game, TD-Gammon examines all possible legal moves and all their possible responses (two-ply look-ahead), feeds each resulting board position into its evaluation function, and chooses the move that leads to the board position that got the highest score. In this respect, TD-Gammon is no different than almost any other computer board-game program. TD-Gammon's innovation was in how it learned its evaluation function.TD-Gammon's learning algorithm consists of updating the weights in its neural net after each turn to reduce the difference between its evaluation of previous turns' board positions and its evaluation of the present turn's board position—hence "temporal-difference learning". TD-Gammon was designed as a way to explore the capability of multilayer neural networks trained by TD(*lambda*) to learn complex nonlinear functions. It was also designed to provide a detailed comparison of the TD learning approach with the alternative approach of supervised training on a corpus of expert-labelled exemplars.

PROPOSED PLAN

In this project the trained program will act both as evaluator and a controller by using board positions to choose that move that will lead to the best value.



The architecture of a RL agent

On the left, the arrows labeled ’state’ and ’reward’ denote the two signals that the agent received from the environment. On the right, the arrow labeled ’action’ denotes the only signal the environment receives from the agent. For each step, the agent receives state and reward signals and then produces an action signal that changes the environment. The dotted line denotes the time horizon of a single step with the new state and reward signals after action at has been performed.

3. ALGORITHM AND METHODOLOGY

In designing the inputs to the network we have mapped the board position onto an input vector of 91 elements.

A checkers board consists of 64 blocks and 32 positions for placing checkers. The board is decomposed into smaller sub-boards of size 8 x 8, 7 x7, 6 x 6,…, 3x3. This comprises of total 91 inputs and 1 input is given for piece difference. These 92 input units are fully connected to a hidden layer of 40 units, and this hidden layer is in turn connected to the single output neuron. Each hidden layer neuron, and the output layer neuron, also has bias inputs whose values are held at unity.

The network was trained using a version of TD-backpropagation. Weight changes were calculated following every move except the first, and the changed weights were used in the next move's evaluation. The desired output during backpropagation was set to the evaluation of the neural network at the following move (after the opponent had made a move, that is.) The procedure at each step was:

Given vector of weights D eligibility trace vector e(s).

```
1.) Evaluate board positions using the neural
    network. Choose the move with the highest
    (lowest as black) evaluation. Move.
2.) If this is the end of the game:
Backpropagate , with reward of 1 or 0 depending
    on whether white won or lost.
3.) Else if this was not the first move, then:
a) Evaluate board.
b) Calculate error between current evaluation
    and previous evaluation.
c) Backpropagate, using the current evaluation
    as desired output and the board position
    previous to the current move as the input.
End.

Backpropagation procedure:
Given an input vector V and a desired output O.
1) Calculate error E between the network's
    output on V and the desired output O.
2) e(s) = (lambda)*e(s) + grad(V)
3) V = V + (alpha)*error(n)*e(s)
where error(n) is:
For the weight between hidden node i and the
    output node,
error(i)=E*activation(i)*weight(i)
For the weight between input node j
    and hidden node i,
error(j,i)=error(i)*activation(j)*weight(j,i)
```

4. RESULTS

The AI was trained by making it play 20,000 games against beginner level opponents. The evolved neural network had the ability to defeat players rated 1200 and lower, and had almost as many losses as wins against opponents rated between 1200 and 1300. With successive Iterations the neural network will give better results and can defeat further higher ranked players. The final rating of the AI is 1202, and there’s still possibility of an increase in the ratings as it faces further opponents.

The largest increase in rating occurs when a weak player defeats a strong player, while the largest decrease in rating occurs when a strong player loses to a weak player.

The best performance of the evolved network was likely recorded in a game against a player rated 1274.

5. CONCLUSION

As a result of successive gameplays, the ability of the AI can be seen. The AI was given no pre-programmed knowledge (except the possibility for using piece differential) and it learned to play at level that is challenging many humans after the training phase. Moreover, the coordinated action of even two pieces moving to pin down a single piece can necessitate a long sequence of moves where it is difficult to ascribe advantage to one position over another until the final result is in view. Finally, it is well known that many end game sequences in checkers can require very high ply, and all of these cases were simply unavailable to the neural network to assess. Still it played above expected level during the end games after it was completely trained.

REFERENCES

[1] “Some Studies in Machine Learning using the game of Checkers”, A. Samuel, IBM Journal, Vol 3, No. 3, July 1959.

[2] Samuel’s Checkers Illustration is reprinted from “Reinforcement Learning: An Introduction’”, R. Sutton and A. Barto, MIT Press, available at: http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node109.html

[3] “Temporal Difference Learning and TD-Gammon”, G. Tesauro, Communications of the ACM, Vol. 38, No. 3, March 1995.

[4] “Reinforcement Learning: An Introduction”, A. Barto. MIT Press, 1998. Available online at: http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html

[5] “Evolving Neural Networks to Play Checkers without Relying on Expert Knowledge”, K. Chellapilla and D. Fogel, IEEE Transactions on Neural Networks, Vol. 10, No. 6, 1999.

[6] “Blondie24: Playing at the Edge of AI”, Fogel DB, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2002

[7] "A Self-Learning Evolutionary Chess Program," Fogel DB, Hays TJ, Hahn SL, and Quon J, Proceedings of the IEEE, Vol.92 (12), pp. 1947-1954.

[8] “The Hex Player Project: An Experiment in Self Directed Machine Learning”, J. Zurita, MSCS Thesis, Villanova University, 2012.

[10] “Reinforcement learning in board games”, I. Ghory, CSTR-04-004, Department of Computer Science, University of Bristol, 2004. Available at: http://www.cs.bris.ac.uk/Publications/Papers/2000100.pdf