ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

A brief tutorial on reinforcement learning: The game of Chung Toi

Christopher J. Gatti[1], Jonathan D. Linton[2], and Mark J. Embrechts[1] *

1- Rensselaer Polytechnic Institute
Department of Industrial and Systems Engineering
Troy, NY 12180 - USA

2- University of Ottawa - Telfer School of Management
Ottawa, ON K1N 6N5 - Canada

Abstract. This work presents a simple implementation of reinforcement learning, using the temporal difference algorithm and a neural network, applied to the board game of Chung Toi, which is a challenging variation of Tic-Tac-Toe. The implementation of this learning algorithm is fully described and includes all parameter settings and various techniques to improve the ability of the network to learn the board game. With relatively little training, the network was able to win nearly 90% of games played against a 'smart' random opponent. The aim of this work is to develop a general software framework for reinforcement learning with an aim to allow for the implementation of game playing strategies for managers that can be applied to option and portfolio management.

1 Introduction

Reinforcement learning is a learning technique in which an agent learns a task through repeated interaction with an environment. This method has been used to create computer programs that are capable of playing various board games at very high levels of expertise. The most notable example is that by Tesauro, who used reinforcement learning to create a backgammon player that could challenge world class human opponents [5, 6]. One game that reinforcement learning has not yet been applied to, is that of Chung Toi, a clever and challenging extension to Tic-Tac-Toe. The purpose of this work is to provide a basic general software framework for reinforcement learning, and to describe the implementation of this learning method to the game of Chung Toi. The availability of a general-purpose reinforcement learning software platform was motivated in order to have a game theoretic framework that can accommodate the implementation of game playing strategies for the selection of options and portfolios.

2 The game of Chung Toi

Chung Toi is similar to the game of Tic-Tac-Toe in that it is a two player game, it is played on the same 3×3 board, and the objective of the game is to get three of ones' pieces in a row. The game differs in that each player has

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

only three of either white or red octagonal pieces that are labeled with arrows [Figure 1]. Players initially take turns placing their pieces on open positions of the board, orienting the pieces either cardinally or diagonally. In the second phase of the game, players take turns moving and/or rotating their pieces to open positions on the board. However, pieces can only be moved in directions which correspond to the orientation of the respective piece. For example, a piece that is oriented cardinally can only be moved to open positions that are either vertical or horizontal relative to its current location.

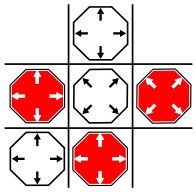

Fig. 1: The game of Chung Toi. Players first place pieces on the board, aligning them cardinally or diagonally, and then move and/or rotate pieces attempting to obtain three of ones' pieces in a row.

3 Reinforcement learning

Reinforcement learning is based on an agent repeatedly interacting with an environment and learning from rewards or penalties (i.e., feedback) resulting from good and poor decisions [Figure 2]. Consequently, the implementation of reinforcement learning requires models of both the environment and the agent.

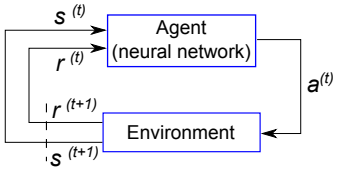

Fig. 2: The reinforcement learning paradigm. An agent selects an action based on the current state, and the environment provides rewards based on the actions pursued, which are used to update state values (adapted from [4]).

3.1 The environment

The model of the environment consists of a board representation, the rules of the game, and an evaluation function. As mentioned above, the Chung Toi board consists of 3×3 board, thus the pieces on the board can be coded using two 9×1 vectors: one indicating their presence, and one indicating their orientation. The positions of the pieces were encoded with a 1 or -1, indicating a white

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

piece or red piece, respectively, or a 0 indicating an open board location. The orientation of the pieces were encoded with a 1 or -1, indicating that a piece (in the corresponding position vector) was oriented cardinally or diagonally, respectively, or a 0 indicating an open board location. Additionally, a 2×1 vector was used to indicate which player was to play next; $[1\ 0]^T$ indicated player 1 was to play next, and $[0\ 1]^T$ indicated player 2 was to play next. The state of the game was therefore fully represented by concatenating these vectors, resulting in a 20×1 state vector. In this work, player 1 was assigned to the white pieces and always had the first turn during both training and evaluation games.

The rules of the game were implemented simply as those stated above. This required determining all allowable moves for each player, as well as developing an evaluation function which could determine if either player had won the game. One of the most challenging parts of applying reinforcement learning to a board game is that of explicitly coding the rules of the game, identifying all of the allowable moves, and employing playing strategies of the network.

3.2 The agent

The agent interacts with the environment by selecting moves and learning from feedback from winning and losing. The agent is represented using a multi–layer neural network which acts as a function approximator, with inputs corresponding to the 20 components state vector, a single hidden layer, and a single output node. The output can be considered as the value of the state, with respect to player 1, that was input to the network; greater output values correspond to states from which player 1 is more likely to win the game. Moves are selected for player 1 by first identifying all possible moves, and then evaluating the value of each of these moves (i.e., evaluating all possible next states through the network). The agent follows an ϵ-greedy policy, selecting the move with the greatest value with probability ϵ. This policy is used to allow the agent to both exploit its expertise, and explore moves that may be beneficial, but that aren't regarded as such by the current network.

The agent gains expertise by repeatedly playing games and using feedback to update the weights of the neural network and improve its state value estimations. In the case of board games, feedback is often only provided at the end of the game, and this feedback is the only information from which the agent can learn each state value. More specifically, feedback is used with the temporal difference algorithm to adjust the weights in the neural network. The temporal difference algorithm was formalized by Sutton and Barto [3, 4] and can be thought of as an extension of the backpropagation algorithm [2, 7]. The temporal difference algorithm uses discounted information from future states, using a temporal discount factor λ, to update the network weights, and thus the state values, of previous states. The weight update equation for a weight w_{jh} (weight from a neuron in layer h to a neuron on layer j) takes the general form:

$$\begin{pmatrix} weight \\ correction \\ \Delta w_{jh} \end{pmatrix}^{(t)} = \begin{pmatrix} learning \\ parameter \\ \alpha \end{pmatrix} \times (error) \times \sum_t \begin{pmatrix} temporal \\ discount \\ \lambda \end{pmatrix}^{(t)} \times \begin{pmatrix} local \\ gradient \\ \delta_j \end{pmatrix}^{(t)} \times \begin{pmatrix} input\ of \\ neuron\ j \\ y_h \end{pmatrix}^{(t)}$$

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

As $\lambda \to 0$, weight updates become based only on information from the subsequent state; as $\lambda \to 1$, weight updates amount to averaging information from all subsequent states. The above equation can be more explicitly written for updating the weights ($w_{jh}^{(t)}$) to the output layer j, from the hidden layer h:

$$\Delta w_{jh}^{(t)} = \alpha \left(P^{(t+1)} - P^{(t)} \right) \sum_{k=1}^{t} \lambda^{t-k} f'(v_j^{(k)}) y_h^{(k)}$$

where α is the learning rate, $P^{(t)}$ and $P^{(t+1)}$ are the state values (i.e., network output) at the current and subsequent time steps, respectively, λ is the temporal discount factor, $f'(v_j)$ is the transfer function derivative at node j evaluated at the induced local field $v_j = \sum_h w_{jh} y_h$, and $y_h = f(v_h)$ is the output of the hidden node h where $f(\cdot)$ is a transfer function. Note that the superscripts in parentheses are not exponents and are used to indicate time steps within the game for the corresponding variables; also note that the superscript on the temporal discount factor λ is an exponent. The above equation can be extended to update weights ($w_{hi}^{(t)}$) to the hidden layer h, from the input layer i:

$$\Delta w_{hi}^{(t)} = \alpha \left(P^{(t+1)} - P^{(t)} \right) \sum_{k=1}^{t} \lambda^{t-k} f'(v_j^{(k)}) w_{jh} f'(v_h^{(k)}) x_i^{(k)}$$

The game progresses as follows at the time step t: a player selects the next move for time step $t+1$ and the board is updated with the new piece location/orientation; the board is evaluated for a win, loss, or no current winner; and the network weights are updated using state values P from the current (t) and next time steps ($t+1$). At the end of the game, there is no next state value $P^{(t+1)}$, and this is instead replaced with the reward for the current game. Note that this method is an iterative weight updating scheme, in which weights are updated during the game.

3.3 Training and performance evaluation

Training is performed by having the network play against itself: player 1 selects moves with the greatest state value (with probability ϵ), and player 2 selects moves with the smallest state value (with probability ϵ). Greedy wins and blocks were also allowed for both players; these moves allow either player to take moves that lead to a win (1^{st} preference) or to blocking an opponent's win (2^{nd} preference), regardless of the state value estimate. The performance of the network was evaluated after every 100 training games by playing 300 evaluation games, which was sufficient to obtain a stable performance estimate. The performance of the network was quantified by the proportions of wins, losses, and draws (i.e., games with more than 100 moves; in reality, there are no draws in Chung Toi). During evaluation games, player 1 always played the first move, always selected moves corresponding to the greatest state value ($\epsilon = 1$), and was not allowed to

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

take greedy wins or blocks. The opponent selected moves at random, but was considered 'smart' such that it was allowed to take greedy wins or blocks.

3.4 Network and parameter settings

The above-described implementation requires numerous parameter settings, which were set as follows: ϵ=0.85; λ=0.7; α=0.0012; reward values consisted of 1 if player 1 wins, -1 if player 2 wins, and 0 for a draw. The parameters ϵ and α were annealed over the course of training (increased and decreased, respectively). The neural network was a 3 layer (1 hidden layer) fully connected network, with 20 input nodes, 30 hidden nodes, and 1 output node; the input and hidden layers each had a bias node with a constant input value of 1. All hidden nodes used a hyperbolic tangent transfer function of the form $f(x) = 1.7159 \tanh(\frac{2}{3}x)$ [1], and the output node used a linear transfer function. Weights were initialized by randomly sampling from a distribution with a mean of zero and a standard deviation σ_w=$m^{-1/2}$ where m is the number of weights leading into node w [1]. These parameters were set based on recommendations from literature sources regarding the backpropagation and temporal difference algorithms [1, 6, 8].

4 Results

The neural network was trained for 2000 training games and its performance during training is shown in Figure 3. Upon initialization, the network wins about 50% of the games. In the early stages of training, performance is often somewhat erratic however, after training, the network was able to win nearly 90% of the games. This performance gain was achieved with relatively few training games, and additional training should allow performance to improve further.

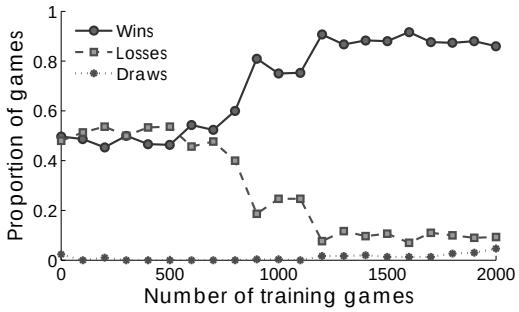

Fig. 3: Performance of network during 2000 training games.

5 Discussion

This work applied reinforcement learning, using the temporal difference algorithm, to train a neural network to play Chung Toi. The learning strategy was

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

based on a basic implementation of the reinforcement learning method, though it also required appropriate parameter settings and a number of techniques to improve the ability of the network to learn. The learning algorithm has been evaluated with many different parameter settings and implementation variations, and it was found that the values and methods used in the implementation described herein work well; however, and variations to these parameters may also allow the network to learn.

The use of an opponent which mainly selects random moves to evaluate the performance of the network may be viewed as a deficiency. Simply because the network can win the majority of games against such an opponent does not necessarily mean it would be able to challenge a human opponent. Increasing the playing expertise of the neural network further would require many more training games (upwards of hundreds of thousands as in [6]), and this would also significantly increase the computation time. The purpose of the current work however, was to show how a simple implementation of reinforcement learning, with relatively little training, could be used to train a network to learn how to perform well in a challenging environment. In this implementation, the network was trained using games played only against itself. Other training strategies are possible however, and these may be used increase the expertise and the speed of learning of the network [6, 8, 9].

In summary, this work presented a basic framework for implementing reinforcement learning, and this framework was used to train a neural network to play a challenging board game. This work successfully extended the reinforcement learning method to a new domain, and supports exploring its use in more challenging, real-world game-like environments, such as in options trading.

References

[1] Y. LeCun, L. Bottou, G. Orr, and K. Müller, Efficient BackProp. In G. Orr, and K. Müller, editors. *Neural Networks: Tricks of the Trade*, Springer, 1998.

[2] D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing* Vol. 1, MIT Press, Cambridge, MA, 1986.

[3] R. S. Sutton. Learning to predict by the method of temporal difference, *Machine Learning*, 3: 9-44, 1988.

[4] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* (1^{st} ed.) MIT Press, Cambridge, MA, 1988.

[5] G. Tesauro. Neurogammon: A neural network backgammon program. In proceedings of the *International Joint Conference on Neural Networks.* Vol. 3, pages 33-40, 1990.

[6] G. Tesauro. Practical issues in temporal difference learning, *Machine Learning*, 8: 257-277, 1992.

[7] P. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. Dissertation, Harvard University, Cambridge, MA, 1974.

[8] M. A. Wiering. TD learning of game evaluation functions with hierarchical neural achitectures. Master's Thesis, University of Amsterdam, 1995.

[9] M. A. Wiering. Self-play and using an expert to learn to play backgammon with temporal different learning, *Journal of Intelligent Learning Systems & Applications*, 2: 57-68, 2010.