REINFORCEMENT LEARNING ALGORITHM USING NEURAL NETWORKS FOR PLAYING OTHELLO

Richard M. Bateman and Simona Doboli
Department of Computer Science
Hofstra University
Hempstead, NY, USA 11549-1000
E-mails: RichardMBateman@aol.com, Simona.Doboli@hofstra.edu

KEYWORDS
Reinforcement Learning, TD learning, Neural Networks, Othello.

ABSTRACT
The paper presents a reinforcement learning algorithm using neural networks for learning an agent to play the Othello game. The system has no initial knowledge of the game, except of its rules. Reinforcement learning techniques allow an agent to learn successful game strategies by repeatedly playing the game. The only information received by the agent during learning is a reinforcement signal at the end of each game. Neural networks are known for their good generalization for untrained inputs. Since the state space for a game like Othello is very large, we use neural networks to represent the value function. The approach is similar to that of Tesauro for the game of Backgammon. The learned network can play Othello at a level close to that of expert programs as shown by its performance in world-championship games.

INRODUCTION
Reinforcement learning is an on-line method of learning from experience (Sutton and Barto, 1998). The aim is to learn an optimal value function that assigns to each state of the environment a value representing the estimated reward from that state until the end of an episode. All learning is done while the agent interacts with the environment: the agent repeatedly observes states, takes actions, receives rewards and updates its value function. The only information the agent receives about its actions is a reward signal, which is known only at the end of the game. Reinforcement learning has been used successfully in spatial navigation problems (Arleo and Gerstner, 2000), game playing (Backgammon) (Tesauro, 1995), trading agents and markets (Moody and Saffell, 2001; Tesauro and Kephart, 1999) and job-shop scheduling (Zhang and Dietterich, 1995).

Reinforcement learning works well for problems with a small number of states, where each state can be visited a large number of times - a requirement for convergence to the optimal value function (Sutton and Barto, 1998). Situations with a large number of states require a different approach, otherwise an unfeasibly long training time is needed. The value function for the visited states is updated during learning, while a function approximation method is used to evaluate the value function of the rest of the states (Sutton and Barto, 1998). We chose neural networks to represent the value function (Sutton and Barto, 1998; Arleo and Gerstner, 2000; Foster et al. 2000). Neural networks can learn any nonlinear mapping based on their well known property of universal approximators. Moreover, neural networks have very good generalization capabilities for inputs unseen during learning.

In this paper, we use a reinforcement learning algorithm implemented with neural networks for training an agent to play the Othello game. The value function is represented by a multi-layer feed-forward neural network trained with the back-propagation algorithm (Rumelhart and McClelland, 1986; Rumelhart et al. 1986). Similar to how Gerald Tesauro created an agent to play the game of Backgammon (Sutton and Barto, 1998), an agent is created for the purpose of learning how to play the game of Othello, which is a game with a massive state space and a small set of rules. It is a game played between two players with a simple objective: to end the game with more pieces of your color than your opponent’s on the board. Other approaches to playing Othello include using min-max trees (that uses a heuristic based on the number of pieces of your color) and by directly assigning values to specific tiles on the board (and the agent would then try to always place a piece on the most highly-valued tile) (Sutton and Barto, 1998). The problem with these heuristic methods is that it is unknown how useful these strategies might be, the agent’s play may be predictable, and in the case of game trees it might be impossible to build a tree beyond a small number of moves. By using reinforcement learning to allow the agent to develop its own strategy, human learning can be simulated.

The results of the best trained agent were compared to the performance of world champions. Several world championship games were examined to see how often the agent would choose the same move as the expert and how often the agent would believe that the expert’s move was a good one. This comparison is used to demonstrate that learning did indeed occur.

PROBLEM DESCRIPTION
Othello is a game played between two players, Black and White, on an eight by eight grid. Each player has pieces of his color on the board; the goal for the player is to have more pieces of his color than his opponent’s on the board at the end of the game. To do this, a player must outflank his opponent’s pieces, which then causes them to be flipped over. The following example diagram shows how White is able to outflank her opponent in three different ways, causing exactly five of Black’s pieces to become white.
Every turn, a player must make a move that outflanks at least one of his opponent’s pieces; if such a move cannot be made, the player loses his turn. If the other player also cannot move, the game ends (this most commonly occurs when the board is full). When the game is over, the number of pieces of each color is counted; whichever player has more pieces is the winner.

A game state for Othello is defined by the arrangement of black and white discs on the board, and the name of the current player. The next state would occur after the current player has placed a disc on the board, outflanked at least one piece, and flipped all outflanked pieces over to her color. (The upper bound of Othello’s state space is $3^{64}$; there are 64 tiles, each of which can bear a white or black tile, or be empty).

**METHODS**

The purpose of reinforcement learning is to find an optimal value function, where the value of it in each state ($V(s)$) represents the estimated discounted reward from that state until the end of the game (Sutton and Barto, 1998; Tesauro, 1995). The value function guides the action taking process: at every move in the game, the action that is taken is the one that leads to the best possible next state. After each action, the agent receives a reward signal based on which the value function is updated. In TD($\lambda$) - a variant of reinforcement learning called temporal difference - the observed reward at time $t$ is used to update the value of all states, not only of the current state (TD(0)). Previously visited states and states visited closer in time to $t$ - chosen by an eligibility trace - are affected more than the rest. In this way, learning updates not only the present state - either good or bad - but also the states on the path to it (Sutton and Barto, 1998). Learning is hard because the reward signal is received only at the end of the game indicating a win or a loss. Whether an intermediary move is good or bad is known only at the end of the game. By repeatedly playing the game, learning proceeds from the end of the game to the beginning, as reward slowly propagates back from later moves to earlier moves.

Since Othello has a very large state space, the value function is difficult to represent with a table, and more importantly, it is impossible to ensure that each state will be visited a very large number of times - a requirement for convergence to optimal value function. That is the reason why, for problems like the Othello game, a function approximation method is used to interpolate the value function for unvisited states. We use feed-forward neural networks to represent the value function.

The neural network has three layers as shown in Figure 2. There are $N$ nodes in the input layer, $H$ neurons in the hidden layer and a single neuron in the output layer. The input and hidden layers are both augmented with a bias neuron. (Nodes in the input layer are not considered neurons because no processing occurs within them).

**Figure 2:** The structure of the neural network.

The activity of the $N$ sensory nodes in the input layer represents the current state of the Othello board. There are 194 input nodes: 192 neurons represent White’s and Black’s pieces arranged on the 64 tiles, as well as which of the 64 tiles are empty or not; 2 neurons indicate whose turn it is. The hidden neurons receive projections from all input nodes, plus the bias neurons. The activation function of a hidden neuron is the sigmoidal function:

$$x_k = f(h_k) = \frac{1}{1 + e^{-h_k}}$$
where $h_k = S(w_{ki} \ast x_i)$ is the weighted sum into a hidden neuron, $w_{ki}$, the weights between input nodes and hidden neurons, and $x_i$, the values of the input nodes, including the bias neuron. The activation function of the output neuron is also the sigmoidal function. The output of the network represents the value function of the input state at time $t$: $V(s_t)$.

All weights are updated using the back-propagation algorithm (Rumelhart et al. 1986). The error ($\delta_t$) that is back-propagated from the output to the input is the difference between the estimated reward from state $s_t$ ($r_t + \gamma V(s_{t+1})$) and the current value of that state ($V(s_t)$):

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

where $r_t$ is the observed reward in state $s_t$, and $\gamma$ represents the rate at which reward from future states is discounted (Sutton and Barto, 1998). The value of $r_t$ is 0 all through the game, except for the last state in each game, where a reward signal of +1 is produced if Black has won, and a reward signal of 0 is produced if White has won. The update equation for all weights in the network is:

$$W_t = W_{t-1} + \alpha \delta_t e_t$$

with $W_t$ the vector of all weights in the network, $\alpha$ the learning rate, $\delta_t$ the eligibility trace, a vector of the same size as $W_t$ (Sutton and Barto, 1998):

$$e_t = \lambda e_{t-1} + \nabla W V(s_t)$$

with $\lambda$, representing the eligibility of a weight to being updated. At each time step, the eligibility trace goes down proportional with $\lambda$. The increment of the eligibility trace is proportional with the gradient of the network function with respect to the weights: $\nabla W V(s_t)$.

Equations (1-4) represent the TD($\lambda$) algorithm implemented on neural networks. The complete learning algorithm is described below:

**Initialize the weights of the network to small, random values.**

$p$: the policy for choosing moves.

$a$: the learning rate

$e$: the probability of choosing a random move (used in $p$)

**Repeat for each episode:**

$s_t$: The initial state of the episode

**Repeat for each step of the episode:**

$a$: action given by $p$ for $s_t$

Take action $a$, observe reward $r$, and next state, $s_{t+1}$.

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

$$e_t = \gamma \lambda e_{t-1} + \nabla W V(s_t)$$

$$W_t = W_{t-1} + \alpha \delta_t e_t$$

$s_t$: The initial state of the episode

**Until $s_t$ is terminal**

**Update $a$, $e$**

The above algorithm is implemented as follows: Training involves two players (Black and White) sharing a single neural network which is updated after every move is made. The policy, $p$, for choosing moves is as follows: choose a random move with $e$ probability; otherwise, choose from among the set of moves that differ in value from the best move by a small amount (usually 0.02). After each game, $a$ and $e$ are updated by the following equations:

$$a = (a_0) \ast (0.99 \frac{Ng}{200}) + 0.0025$$

$$e = (e_0) \ast (e \frac{Ng}{10000}) + .002$$

where $Ng$ is the number of games played.

**RESULTS**

Fifteen neural networks were trained differing in regard to four key parameters: the number of hidden neurons, the way the learning rate ($a$) decayed, the way epsilon ($e$) decayed, and the number of training epochs. The networks were then pitted against each other in a round-robin style tournament. In order to increase the variety of games played between any pair of networks, $e$ was reduced to 0 in all cases, but the tolerance for determining the set of best moves was increased to 0.05. After the tournament, the network that achieved the best performance was then chosen for further analysis.
The one selected had ninety-six hidden neurons, a learning rate of 0.5 that decayed the most slowly, an epsilon value of 0.9 that also decayed very slowly, and had played 100,000 training games with itself.

Figure 3 gives a sample board evaluation. The network, or agent, looks at each of the next states that can result from each of its possible legal moves. It then evaluates the worth of each of those next states, and then chooses the move that leads to the most valuable state. In this scenario, it is White’s turn and White can make 11 moves. Values close to 0 indicate states good for White, and values close to 1 indicate states good for Black. The network considers 3 moves particularly bad (marked by red X’s), 7 moves good (blue checkmarks), and 1 move excellent (the yellow circle). The network has learned that the corner can be a valuable place to put a piece. Moreover, it notes that tiles near the corner are very poor places to put pieces. (Placing a piece near the corner would allow the enemy to grab the corner the next turn).

The seemingly best network was further studied to see how its performance compared to the performance of world championship games (Mandt, 2003). The games were presented to the network state by state; what was analyzed was the relationship between the values of what the network thought was the best move and the move the world champion selected. The point of the analysis is to illustrate that genuine learning must have occurred if the network tends to agree with experts. Figure 4 shows the results of this analysis.

Around 30% of the time the network chose the exact same move the expert did, and around 64% of the time the network believed that the expert’s move was close in value to the move it chose. Considering that there are about an average of 8 moves a turn, the 30% exact match is much better than the 12.5% chance of choosing the move randomly. It is also important to note how these results would be different for an untrained network. An untrained network would not choose the exact same move as the expert with any great probability; however, since to an untrained network all states are equally valuable, the untrained network would believe that the expert’s move was close to the best move in value 100% of the time. The reason that the results are important for a trained network is that, for a given state, the trained network assigns widely different values to the possible next states (as can be seen from the sample evaluation). The high percentage of closeness of value of the expert’s move to the network’s move is only relevant to a trained agent.

After this comparison of the network’s choices to an expert’s, the network played several games against human players experienced with Othello (and thus probably playing at an intermediate level). The players consisted of a senior university student majoring in mathematics, a professor of computer science, and three on-line players who were playing in Yahoo!’s intermediate area for Othello (Reversi) players. The program successfully beat all but the professor, who conceded that the program offered a challenging level of play.

CONCLUSIONS

We have shown that a reinforcement learning algorithm using neural networks was able to learn to play Othello without an external teacher. Considering that the network started with only the rules of the game, it is significant that after training, the network tended to agree with world champions. There is a caveat, however: training an agent to play Othello using a neural network does not result in optimal game play. One problem with this method of learning is that the network only learns to play against one style of strategy. No matter whom the network’s opponent, it will always choose the same moves. In other words, the network never considers the opponent’s playing strategy. Moreover, against expert computer programs that might employ preprogrammed heuristics, the network is at a disadvantage. One possible improvement that could be made to the neural network program would be to augment its decision-making with a game tree. For example, perhaps during the last five moves of the game, the network could be abandoned and a game tree could be used to analyze those critical last few moves. Although the network only reaches a good level of play, it was able to reach this level on its own without any help from any strategist guiding it. The combination of neural networks and reinforcement learning is thus useful in applications in which no strategies are currently known; these strategies can then be learned by the network.
<table>
<thead>
<tr>
<th></th>
<th>Exact Match:</th>
<th>Close (Within 0.02):</th>
<th>Somewhat Close (Within 0.10):</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Opening: 21.8%</td>
<td>Opening: 52.5%</td>
<td>Opening: 93.5%</td>
</tr>
<tr>
<td></td>
<td>Midgame: 25.1%</td>
<td>Midgame: 41.3%</td>
<td>Midgame: 78.2%</td>
</tr>
<tr>
<td></td>
<td>Endgame: 41.4%</td>
<td>Endgame: 50.7%</td>
<td>Endgame: 72.6%</td>
</tr>
<tr>
<td></td>
<td>Average: 29.4%</td>
<td>Average: 48.2%</td>
<td>Average: 81.4%</td>
</tr>
</tbody>
</table>

Fairly Close (Within 0.05):
- Opening: 71.8%
- Midgame: 59.3%
- Endgame: 61.0%
- Average: 64.0%

Somewhat Close (Within 0.10):
- Opening: 93.5%
- Midgame: 78.2%
- Endgame: 72.6%
- Average: 81.4%

Note: The Opening, Midgame, and Endgame consist of 20 moves. Not all games last 60 moves.

REFERENCES


Zhang W., T.G. Dietterich. 1995. “A reinforcement learning approach to job-shop scheduling.” *Proceedings of IJCAI95*