Temporal Difference Learning in Score-Four

Matthew Hlavacek  
Student, Northwestern University  
2420 Campus Drive  
Evanston, IL 60201  
(847) 650-3876  
Matthewhlavacek2014@u.northwestern.edu

Benedict Lim  
Student, Northwestern University  
615 Garrett Place W34  
Evanston, IL 60201  
(224) 730-2026  
Benedict@u.northwestern.edu

John Ngo  
Student, Northwestern University  
912 Noyes Street Apt. D  
Evanston, IL 60201  
(712) 898-8148  
Johnngo2014@u.northwestern.edu

ABSTRACT
We have developed a machine-learning taught computer player for the game Score-Four, a three-dimensional variant of the aptly named Connect Four. This project was constructed at Northwestern University as a final project for the EECS 349: machine Learning course taught by Professor Bryan Pardo, where we applied temporal difference learning onto a neural network.

Categories and Subject Descriptors
I.2.6 [Artificial Intelligence]: Learning – knowledge acquisition, connectionism and neural nets.

General Terms
Algorithms, Design, Experimentation, Human Factors

Keywords
Temporal Difference, Reinforcement Learning, Abstract Strategy

1. INTRODUCTION
Score Four is an abstract board game played on a 4x4x4 grid where the objective, similar to that of Connect Four, is to attain a winning state. The game has a vast game state space, with a crude upper bound of 3^64 unique configurations. Neither a modeling of the underlying Markov decision process nor brute-force accumulation of the policies for all states is feasible in modeling a state space of this magnitude. Therefore, the most efficient approach to teach a computer to play Score-Four was determined to be a delayed reinforcement learning model with an approximation of the value function.

2. INTEREST AND UTILITY
While there have been several applications of temporal difference learning to board games such as chess, backgammon and tic-tac-toe, the application of the learning method on Score-Four has not been examined.[1] Furthermore, while games like Connect-Four and checkers are “solved” games with published best strategies, Score-Four has yet to have a proven winning strategy published. We are interested in the artificial neural net approach’s ability to respond to variations in game rules and feature representations.

Particularly interesting is the ability for artificial neural nets to learn a value function approximation without an expert defined feature set. This was shown in the first version of TD-Gammon, a temporal difference-based backgammon player.[2] By simply inputting the board state, the neural network can possibly be trained on a variety of rules for wins and moves.

3. IMPLEMENTATION
3.1 Task to learn
Our project specifically learns to implement an effective strategy of playing one side of a Score-Four game. For each of its side’s turn in the game, it takes as input the state of the board and outputs some particular move, attempting to eventually move into a winning state.

3.2 What will be optimized
Our project is primarily concerned with optimizing the estimation of the value function for the possible moves. The challenge lies in having the learner determine which of the legal next-moves gives it the greatest likelihood of reaching a winning state.[3]

3.3 Learning
The learner is based primarily on the TD(λ) and back-propagation algorithm as proposed by Sutton.[4] A neural network is first used to play both sides of the Score-Four game to completion. Upon completion, the TD(λ) algorithm is applied to modify the network’s weights at each time step. The neural network structure used is as follows:

```
  Output  y[k]  k= 0, …, m-1
               w[i][j][k]
                     v[i][j][k]
  Input  x[i]  i= 0, …, n
```

Figure 1: Conceptual Image of a Neural Network identifying the net’s inputs and outputs, hidden nodes, and accompanying weight vectors.

We selected 64 hidden nodes based on papers by Swingler[5] as well as Berry and Linoff[6], where they proposed a ‘rule of thumb’ that the number of nodes in the hidden layer should never be more than twice the number of inputs (in this case the number of inputs is 64 representing all the positions on the Score-Four board). Evaluations showed however that 128 hidden nodes were too

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission.

EECS 349’12, November 10, 2012, Evanston, IL, USA.
many for learning to have any significant impact, so the number of hidden nodes was reduced to 64.

Key parameters that influence the learning capabilities of the learner are the learning rates $\alpha$, $\beta$ as well as the decay parameter $\lambda$. $\alpha$ and $\beta$ determines how much updating of the weight occurs at each time step, where smaller values of $\alpha$ and $\beta$ implies slower learning rates. The decay parameter $\lambda$ determines how relevant past moves matter, with a value of 0 implying that no feedback occurs beyond the final step, and a value of 1 implying a feedback without decay arbitrarily far in time. We initially used learning rates and decay parameters calculated by Sutton's methodology, a methodology designed for a learner training over large number of training rounds, but the results were not satisfactory. Through a process of experimentation, we refined the optimal values for $\alpha$, $\beta$ and $\lambda$ for early (~5000 games) learning to improve the initial performance of the learner, then we experimented with $\alpha$ and $\beta$ decay from our early learning parameters to reach the optimal parameters proposed by Sutton. We also increased $\lambda$ from 0 to 0.6 so as to encourage the learner to initially learn the quickest method to win a game before learning more sophisticated strategies later in the learning process. Ultimately, the results from our training scenarios suggested that static variables outperform the decaying variables in developing the learner to a significant degree.

Lastly, we modified the code to partially reward the learner when it attains three pieces in a row, to allow the learner to recognize the importance of reaching the prerequisites to win states. This modification also allowed the learner to recognize the threat of the opponent attaining three pieces in a row.

3.4 Training

Though play against an expert would be ideal, this would be a slow, time-consuming process. Self-play has been shown in several temporal difference game-learning applications to be an effective means of training a model.\(^7\)

For the initial results, our learner was trained through self-play against a series of different opponents. During the training period, the learner also alternates between starting first and starting second. In the first session the opponent is a random player who makes a random legal move at each turn. At the beginning of the subsequent sessions, the opponent would take on the weights of the neural network learned so far and utilize them for the entirety of the session – thus the opponent will increase in sophistication throughout the training process.

Modifications to the training session included updating the opponent’s weights after each game, so that the learner is always playing against an identical match. This approach was ultimately scrapped for unforeseen reasons later discussed.

We augmented our learner with an $\varepsilon$-greedy policy while training to prevent the learner from being stuck in a local maxima and to ensure it explores the state space. The probability of making an off-policy move will also decay throughout training, recognizing that the learner is able to exploit more strategies later in the training rounds and would not need to keep exploring off-policy moves.

3.5 Evaluating Performance

The performance of the learner is evaluated by statistical data gained from extended play, weight progression from its initial to final states, and a snapshot demonstration that allows human players to interact with the finished product.

![Figure 2: Learner Win Percentage versus an opponent with static random weights](image1)

4. RESULTS

Results after processing our player through various training scenarios have been both illuminating and discouraging. Sustained game training have shown that the learner is capable of learning to compete versus a static opponent at a proficient level, gradually recognizing the opponent’s typical behavior and sequence of moves and eventually reaching a point where the learner can exploit this knowledge and win the majority of played games. Playing against non-static opponents prove to be extremely difficult for our learner in that it is not able to learn a singular effective strategy to play Score-Four.

![Figure 3: Learner Win Percentage versus an opponent with updating weights for each game iteration](image2)
rudimentary strategy. Furthermore, the current-standing reward vector only provides rewards and punishments for patterns that have three or four aligned pieces. This fails to truly emulate the marginal returns one receives from playing other possible moves which are not perceivably beneficial. Later work on this project therefore requires further fine-tuning to establish a more extensive and superior reward vector.

5. FUTURE WORK
If we were to recommend immediate next-steps, our primary suggestion would be to encode some expert knowledge in the input vector, such as checking for four in a row, or checking the number of “win-vectors”, or rows of four adjacent spaces, which are one piece away from a win. This would allow the neural net to better predict the outcome of a game and to better block opponent wins.

6. REFERENCES