Temporal Difference learning in Backgammon

Monday, June 27

Neural Nets, Summer I 2016

Justin Beach

Patrick Loftus

Contents

Description of Experiment

We used the temporal difference learning algorithm to train a neural network to evaluate backgammon positions.

A backgammon board consists of 24 points and the bar. Following Tesauro's design choices, we designed a network with 198 inputs and 50 hidden nodes. We assigned 8 inputs in our neural network to each of the 24 points. The first 3 checkers on any point get their own input. The remaining checkers are summed halved and assigned to the 4th input. There are 4 inputs for each player. Thus, 8 white checkers on a point would generate 1 1 1 2.5 0 0 0 0 as its corresponding inputs. Likewise, 2 black black checkers would yield 0 0 0 0 1 1 0 0. Of the remaining 6 inputs, 2 represented the player on roll; 2, the number of checkers on the bar; and 2, the number of checkers that had been cleared.

After training our network on nearly 15000 games, we pitted the trained network against an untrained network to quantify the success of the training routine.

Data

Table 1 shows the number of wins for the trained and untrained players in 5 100 game series. Table 2 provides statistical insights into the series' results.

Table 1: Number of wins for trained and untrained Neural networks.

	wins	
series	untrained	trained
1	33	67
2	40	60
3	43	57
4	19	81
5	44	56

Table 2: Trained network win statistics.

	trained win %
Mean	64
Std Error	4.6
95 % C.I.	51 - 77

Results

The results are a bit disappointing. We can only be 95% confident that the trained network actually out performs the untrained network. Furthermore, we are 95% certain that the trained network will win fewer than 80% of its games against the untrained network.

A potential cause of the poor performance is the small training set. Although 15,000 matches sounds like a lot, previous implementations suggest that the network continues to improve for the first 100,000 games. Even at the 20,000 game mark performance is still improving quite rapidly. Unfortunately we didn't have time to play 100,000 training games with our network. Moreover, we didn't have an opportunity to optimize our learning parameters.

If given more time, we would have implemented the backgammon game engine in python. The external game engine that we used forced us to perform time consuming I/O operations after each turn. Because of this, we didn't have time to fully train our network or to optimize learning parameters.