Using Machine Learning to Teach a Computer to Play Backgammon

David Molin[†] , Viking Flyhammar[††] and Saman Bidgol[‡‡]

***Abstract*—This paper presents two methods to teach a computer to play backgammon. Both methods are based on neural networks which are trained by playing games against itself until reaching convergence. The first method uses an approach similar to TD-Gammon to train a neural network while the second method uses a bayesian approach to train a neural network.**

Based on the results, it is concluded that both methods succeed to improve its performance after training. The network generated by the bayesian method is shown to be superior to network created by the approach similar to TD-Gammon.

I. Introduction

The analysis of board games has become an increasingly interesting field for researchers and engineers. The theory in Machine Learning such as neural networks has made it possible to make computers play board games better than ever, beating the best professional players in the world.

The first strong computer player was BKG 9.8, written by Hans Berliner [1, p. 6-9]. It managed to beat the reigning world champion in backgammon back in 1979, becoming the first computer program beating a world champion in any board game. Berliner states it was largely a matter of luck because of more favorable rolls of dices. In the late 1980's backgammon programmers explored an approach using neural networks. The most notable papers in this area were published by G, Tesauro in 1992, 94 and 95 [2] [3] [4] describing how to make a computer play backgammon using a neural network. The new approach resulted in the computer program called TD-Gammon playing equal level or above of the best human players.

A. The Rules of Backgammon

Backgammon is a board game for two players. Backgammon is played on a board with 24 triangles, called points, where the players can move their checkers in a specific direction (see Figure 1). Each player has 15 checkers. Depending on the outcome of the two dices the player decides how to move his checkers on the board. The opening move cannot be done with two identical rolls. One of the most basic rules, and also one of the most important rules in backgammon is that it is possible to hit an opponent's checker if it is left alone at a specific point. If this is the case, that checker will be moved outside the board and placed at the bar, located in the middle of the game board. The next move for the opponent is to get his hit checker back into the game before he can continue to play. Also, a player can block a point by having at least two checkers at that particular point, which means that the other player cannot move his checkers to that point.

When a player has moved all of his 15 checkers to the last six points, he can start removing checkers from the board. When this is the case, a roll of one may be used to remove a checker from the 1-point, a two from the 2-point and so on. A die may be used to remove checkers from a lower-numbered point if there are no checkers on any higher points. I.e. if a player rolls a six and a five, but has no checkers on the 6-point, but two checkers remaining on the 5-point, then the rolls must be used to remove the two checkers from the 5-point. The first player to remove all of his checkers wins the game. Backgammon is not a deterministic game in the sense that dice rolls determines how the checkers may be moved.

B. Goals

The objective of this project is to explore how to use a bayesian approach in order to train a neural network to play backgammon.

C. Scope of the project

In this project the approach to teach a computer to play backgammon is to find a function which maps a certain state of the board to the probability that a player wins the round. Once the neural network is trained, the moves that generates the highest probabilities to win are picked.

The main objective of backgammon is to score as many points as possible by winning the game in different ways. However, these rules are out of the scope of this project. Therefore, only the basic rules and the win/loss ratio are considered in this project.

II. Theory

Define $s \in \mathbb{R}^n$ as a state vector that represents a specific game board, and also define $y \in \mathbb{R}$ to be the probability to win the game from a given state. Furthermore, let the function $h_\theta(s) : \mathbb{R}^n \mapsto \mathbb{R}$ be the hypothesis that estimates the probability to win from a given state. $h_\theta(s)$ is the output of a neural network where the last layer activiation function is a sigmoid function.

[†]davmo@stanford.edu

[††]vikingf@stanford.edu

[‡‡]bidgol@stanford.edu

A. Modelling of the Neural Network Parameters

In TD-Gammmon the weights are updated using

$$\theta^{t+1} = \theta^t + \alpha(Y_{t+1} - Y_t)\sum_{k=1}^{t}\lambda^{t-k}\nabla_\theta Y_k. \quad (1)$$

.

Since the case where $\lambda \to 0$ lends itself to the interpretation

$$\theta^{t+1} = \theta^t + \alpha\nabla_\theta J, \quad (2)$$

where J is the square error of the difference between $h_\theta(s)$ and Y_{t+1}. For this reason this case is explored in this paper. However, since the outcome of a game is bernoulli distributed rather than gaussian distributed the cost function used in this paper is

$$J = h(s)^y(1 - h(s))^{1-y}. \quad (3)$$

Since there are too many states in backgammon to store the value function, $V(s)$ explicitly for each state. $V(s)$ is approximated with a neural network $h_\theta(s)$. (The goal is that this network will output the probability that black wins given the current state and that it is blacks turn.) The weights of this network is denoted θ in this paper.

B. A Bayesian approach

Another method explored in this paper was using a bayesian model to update θ. The prior is assumed to be gaussian, the mean is initialized to small random values to avoid issues with symmetry in the parameters. The posterior given one observation is assumed to be gaussian with a mean which is the maximum a posteriori and the covariance is the hessian of the log of the posterior evaluated at the maximum a posteriori. This procedure is used in [5, p. 279]. This posterior can then be used to create the prior in the next estimate of θ. The mean prior in iteration i will be denoted $\hat{\theta}^{(i)}$ and the covariance will be denoted σ_{e_i}.

Using the prior from step i it is possible to create a training set by letting the computer play against itself and for every visited state assigning the value $y^{(i)} = \max_a \sum_{s'} P_{sa}(s')V(s')$. Where V is the estimated value given the prior. The theta which minimizes the cost function for this training set as the size of the training set goes to infinity will be denoted $\theta^{(i+1)}$. But the state transition given an action is deterministic, thus P_{sa} will be degenerate. The relation between $\theta^{(i+1)}$ and $\theta^{(i)}$ is assumed to be

$$\theta^{(i+1)} = \theta^{(i)} + \mathcal{N}(0, \sigma^2_{c_i}). \quad (4)$$

Which means that the difference between $\theta^{(i+1)}$ and $\theta^{(i)}$ is white noise. This assumption is made because when the prior estimate is close to the optimal set of parameters the new information from one more iteration should not change the value of $\theta^{(i)}$ much.

The noise in the estimates are assumed to be white gaussian noise with variance σ_N. This implies

$$P\left(y\middle|s,\theta,\sigma^2_N\right) \propto \exp(-\frac{1}{2\sigma^2_N}(h_\theta(s) - y)^2). \quad (5)$$

Equation (4) implies

$$P\left(\theta^{(i+1)}\middle|\hat{\theta}^{(i)}, \sigma^2_{e_i}, \sigma^2_{c_i}\right) \propto$$

$$\exp(-\frac{1}{2}(\theta^{(i+1)} - \hat{\theta}^{(i)})^T(\sigma^2_{e_i} + \sigma^2_{c_i})^{-1}(\theta^{(i+1)} - \hat{\theta}^{(i)})). \quad (6)$$

Furthermore, Equation (5) and Equation (6) used together imply

$$P\left(\theta^{(i+1)}\middle|\hat{\theta}^{(i)}, \sigma^2_{e_i}, \sigma^2_{c_i}, \sigma^2_N, s, y\right) \propto$$
$$P\left(\theta^{(i+1)}\middle|\hat{\theta}^{(i)}, \sigma^2_{e_i}, \sigma^2_{c_i}\right) P\left(y\middle|s,\theta,\sigma^2_N\right) \quad (7)$$

Thus the update rule is

$$\hat{\theta}^{(i+1)} := \arg\max_\theta P\left(\theta^{(i+1)}\middle|\hat{\theta}^{(i)}, \sigma^2_{e_i}, \sigma^2_{c_i}, \sigma^2_N, s, y\right), \quad (8)$$

$$\sigma^2_{e_{(i+1)}} := ((\sigma^2_{e_{(i)}} + \sigma^2_{c_{(i)}})^{-1} + 1/\sigma^2_N H)^{-1}, \quad (9)$$

where H is the hessian of $\log(P\left(y\middle|s,\theta,\sigma^2_N\right))$. Lastly σ_{c_i} is assumed to be proportional to $(\hat{\theta}^{(i)} - \hat{\theta}^{(i-1)})^2$.

III. Method

This project explores two different approaches to the problem of finding a neural network which plays backgammon. The first approach uses gradient descent in order to update the parameters, this approach is called the TD-Gammon approach in this paper, although this is a misnormer since the TD-Gammon derived by G.Tesauro is not identical to this approach, while the second approach uses the bayesian approch described in Section II. This approach is called the bayesian approach.

A. Defining the Feature Vector

In order to apply Machine Learning to this problem, it is necessary to calculate a feature vector. The feature vector used in this problem represents the current game board, thus the feature vector is the state vector described in Section II, where $n = 26$. The first 24 elements represent the 24 points on the game board while the last two represent if a checker is hit and placed outside the game. The value for each element in the state vector represents the number of checkers each player has at the actual points. Positive integers represent the current player's checkers while negative integers represents the opponent's checkers. For example, the initial game board shown in Figure 1 has the following state vector

$$s = [2,0,0,0,0,-5,0,-3,0,0,0,5,$$
$$-5,0,0,0,3,0,5,0,0,0,0,-2,0,0], \quad (10)$$

where the representation is counter clockwise and starts from the top-right corner. The positive integers represent the black player's checkers and the negative integers represent the white players checkers.

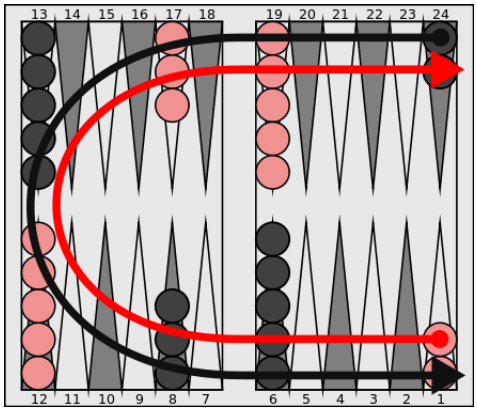

Fig. 1. The initial game board setup.

B. Calculating all Possible States

Given two dice rolls and the current state vector, an algorithm calculates all the possible state vectors by using recursive formula. Each possible state vector represents a possible move for a specific player from a given state and two dice rolls.

C. Calculating the Gradient of a Neural Network

The modeling of the probability to win in a given state is done by a neural network. The neural networks in this project have 40 hidden nodes. Since this network has to be updated, the gradient with respect to the weights in the neural network is calculated. This is done by forward- and backward propagation through the network.

D. Generating a Training set

In order to update the weights of the neural network training data is generated. This is done by letting the computer play against itself. At the end of every game a training set is generated by assigning a target variable, y to each game state visited during the game. The last game state is assigned the target variable 1, in other words, the probability to win from the last state is assumed to be 1. For the other states the target variable is the probability that black wins from the following state according to the neural network. Once this training set is generated the neural network is updated with respect to it.

E. Assumptions to Speed up Computations of the Bayesian Model

In order to reduce the computational complexity of the bayesian approach described in Section II, and thereby speeding up the simulations, certain approximations are made. One such assumption is that the hessian of the cost function with respect to the weights is $H = h_\theta(1-h_\theta)\nabla h_\theta(\nabla h_\theta)^T$ [5, p. 251]. Another assumption is that the covariance of the posterior is diagonal. The main reason for this assumption is to speed up computations since even for small networks the number of parameters grow quickly and computing inverses of large matrices is time consuming. Since the optimization problem in neural networks are nonconvex the step of finding the argmax with respect to the weights need to be simplified to find a local optima. This local optima is then approximated by the result of two iterations of Newton's method, calculated with the approximation of the hessian described earlier.

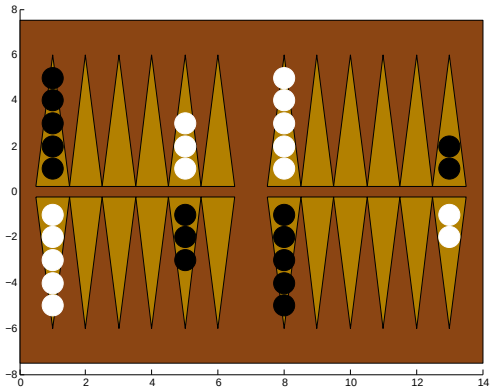

Fig. 2. The GUI used for testing.

Rolls	Opening move	Optimal
1-2	13/11, 6/5	Yes
1-3	8/5, 6/5	Yes
1-4	13/9, 6/5	Yes
1-5	13/8, 6/5	Yes
3-5	8/3, 6/3	Yes
4-6	8/2, 6/2	Yes
1-6	13/7, 6/5	No
2-3	13/11, 6/3	No
2-4	13/11, 13/9	No
2-5	13/11, 8/3	No
2-6	13/11, 13/6	No
3-4	13/9, 6/3	No
3-6	13/7, 6/3	No
4-5	13/9, 8/3	No

TABLE I
THE OPENING MOVES MADE BY THE FINAL BAYESIAN NEURAL NETWORK. SEVERAL MOVES ARE OPTIMAL COMPARED TO MOVES MADE BY THE PROFESSIONAL BACKGAMMON PLAYERS [6]. THE OPENING MOVES ARE DESCRIBED AS FROM/TO.

F. Backgammon GUI

To know whether the algortihm plays correctly and to visualize the game, a simple GUI is created in MATLAB. It plots a backgammon board and the associated checkers. The GUI is visualized in Figure 2.

IV. Results

The bayesian approach and the TD-Gammon approach have been trained for 2,000,000 games each. Figure 3 and Figure 4 show percentages of games won against neural networks which have been playing fewer games. Opening moves of the final bayesian neural network is presented in Table IV. Furthermore, Figure 5 shows the win ratio of the bayesian approach against the TD-Gammon approach.

V. Discussion

As shown in Figure 3, the bayesian model indicates convergence after approximately 1,200,000 games played, while the TD-Gammon model indicates convergence after approximately 800,000 games played as seen in Figure 4. Both models indicates an improvement of the game play as the number

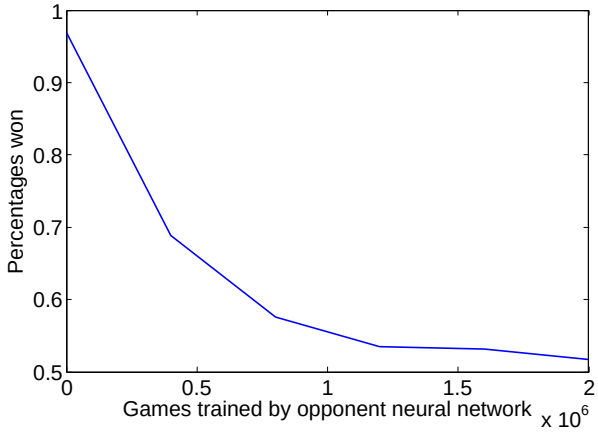

Fig. 3. Win ratio of the final bayesian neural network, trained 2,000,000 games. The number of games the opponent has played ranges from zero (random neural network) up to 2,000,000 games.

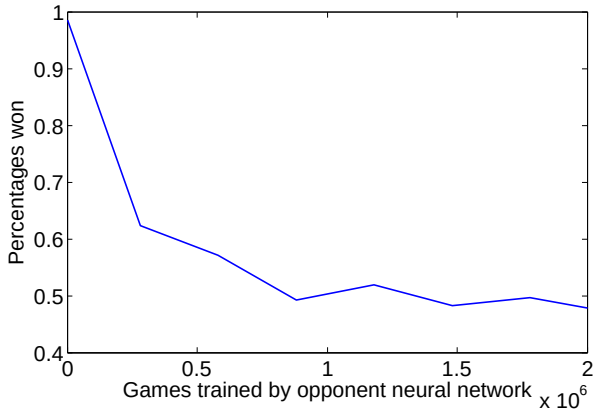

Fig. 4. Win ratio of the TD-Gammon neural network trained 2,000,000 games playing against different trained TD-Gammon neural networks. The opponent neural networks range from zero (random neural network) up to 2,000,000 games.

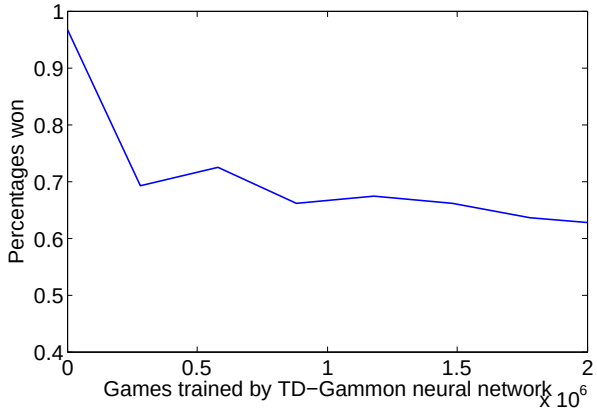

Fig. 5. Win ratio of the bayesian neural network trained 2,000,000 games playing against different trained TD-Gammon neural networks. The TD-Gammon neural networks range from zero (random neural network) up to 2,000,000 games.

of games played increase. There are two indicators of how well they play. One indicator is how well they play against each other, indicating which approach is the best of the two. Another one is studying the opening moves of the model.

By analyzing Figure 5 it is easily seen that the bayesian neural network is superior to neural networks created by the TD-Gammon model. Although this result seems promising the TD-Gammon model in this paper differs from the TD-Gammon approach developed by G.Tesauro

The optimal openings done by the bayesian model are compared to the best openings considered by the professional backgammon players. These moves are considered to be a part of the opening moves that maximizes the probability to win given a throw of two dices [6]. The Bayesian model succeeds to do 40 % of the optimal opening moves which indicates that the model plays good. However, the non optimal moves are in some sense good because the bayesian model manages to move one checker optimally. The reason why the final bayesian network does the non-optimal moves are because it seems to favor moving a checker to point three on the home board, in order to be able to secure this point on the next turn. Since the performance seems to converge, but the performance it converges to does not seem to be optimal, it should be possible to archieve a higher performance by increasing the number of hidden nodes in the neural network.

In addition the final bayesian neural network also demonstrates a tactical game play, such as blocking the opponent player from reentering the game if the opponent player has a hit checker. Furthermore, the neural network also demonstrates a defensive play style, e.g. Avoids to be hit by the opponent player.

A. *Future work*

At this stage of the project it is not verified whether the bayesian approach plays well against human players. Therefore, further studies could include developing a GUI to make this possible.

Since many assumptions are made about the noise in the bayesian model, additional studies on how the noise behaves in practice is of great importance for further improvement of the model. Another intresting thing to explore would be to see which parameters of the bayesian neural network has a low or high variance, respectively.

VI. Conclusion

Due to the test results and the improvement in the bayesian neural network as the training increases and the way of playing it is concluded that the algorithm plays well. This project also shows that a bayesian approach to reinforcement learning problems are viable.

Acknowledgement

Special thanks goes to Prof. Andrew Ng and the TAs for showing the thruth and beauty of Machine Learning.

REFERENCES

[1] e. a. Berliner, Hans, "Backgammon program beats world champ," *ACM SIGART Bulletin*, vol. Issue 69, January 1980.

[2] G. Tesauro, "Temporal difference learning of backgammon strategy," in *Proceedings of the Ninth International Workshop on Machine Learning*, ser. ML92. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 451–457. [Online]. Available: http://dl.acm.org/citation.cfm?id=141975.142085

[3] ——, "Td-gammon, a self-teaching backgammon program, achieves master-level play," in *Neural Computation*, ser. Vol. 6, No. 2. MIT Press., 1994, pp. 451–457. [Online]. Available: http://www.mitpressjournals.org/doi/pdf/10.1162/neco.1994.6.2.215

[4] ——, "Temporal difference learning and td-gammon," in *Communications of the ACM, March 1995 / Vol. 38, No. 3.*, 1995, pp. 58–68. [Online]. Available: http://www.informatik.uni-osnabrueck.de/barbara/lectures/selforganization/papers/tdgammon.ps.gz

[5] C. Bishop, *PATTERN RECOGNITION and MACHINE LEARNING*, 1st ed. Springer, 2006.

[6] T. Keith, "How to Play the Opening Rolls," http://www.bkgm.com/openings.html, viewed: 12/13/2013.