Neural Network Temporal Difference Gammon

Ahmed MAZARI [*1] **Divya Grover** [*1]

Abstract

In this project, we study Backgammon game from a practical standpoint. We first, go through the theoretical tools in order to get a better understanding. Then, we examine an available code source implementing it. After that, we extend the code by adding our agent based neural network temporal difference, simulation driver to run simulation and test the strength of a given player and other functionalities required to run it. Finally, we evaluate our results, discuss and put them into perspective.

1. Our approach

We implemented TD-Gammon learning agent on top of a *code-base* [1] provided freely . Specifically We implemented the following classes:

1. BackPropPlayer: The player which uses a Neural Network to compute the next-best move, based on the current value-function, as given the current Neural Network. It applies TD-learning to update the parameters based on the outcome of the current game.
2. SimulationDriver and TestDriver: To run simulations of self-play for training and to test the strength of any given players.
3. Utility: A helper-class which implements other necessary functions, most importantly, BoardtoVector, to convert the board into an input representation for the Neural Network.

Overall **500 lines of code** were written by us, including significant changes in the original codebase.

[*]Equal contribution [1]University of Paris Sud 11, France. Correspondence to: Ahmed MAZARI <ahmed.mazari@outlook.com>, Divya Grover <revorg7@gmail.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, 2017. JMLR: W&CP.

[1] http://modelai.gettysburg.edu/2013/tdgammon/index.html

The NN architecture is described as follows :
during play-time, each player sees the board as follows:

- Opponent's pieces from my homeboard to spike-24
- MyBar, OopBar, myOff, oppOff
- My pieces from my homeboard to spike-24

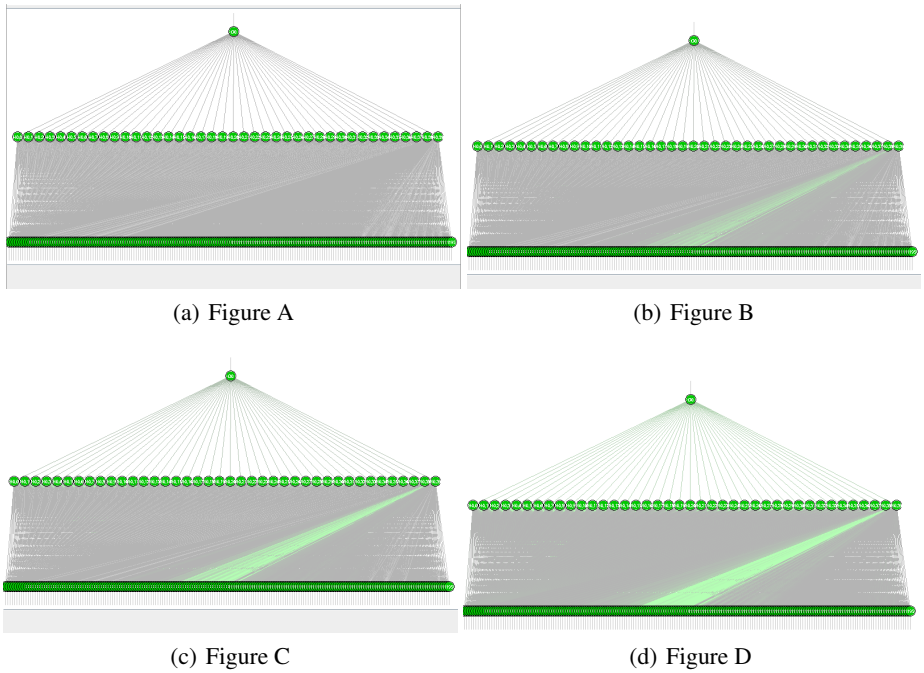
2. Results and discussion

The Neural Network training resulted in improvement of gameplay, which was then vetted against a HumanPlayer and a RandomPlayer. Neural Networks for different number of self-plays are visualized in (Fig.2). They should be read as follows: Green denotes positive weights and Red denotes negative weights. The intensity of color denotes the strength of weights.
As evident from the trained networks, it is the inputs in the middle that matter the most for winning probability prediction, and the input-signal here corresponds from left to right as follows:

1. The number of opponent's pieces on his homeboard.
2. The no. of bar pieces and no. of pieces already off board for each player.
3. The number of my pieces on my homeboard.

The above observation is reinforced in Fig.2 where the intensity of weights for those inputs increase as the training progresses. One important hypothesis that can be made is as follows: since during the initial stages of the game, there are not much pieces of any player on his own homeboard, and neither are any bar pieces or off pieces yet. The effective input-signal to NN corresponds to an input where the middle part is zero, hence the predicted value is almost random.

From (Fig.2), we notice that the more we play , the more perform better. However, for $k = 10000$ number of training games the result returned is not better than the one with $k = 1000$. There is a difference in the order of 5%. This result (of $k = 10000$) is a little bit strange because we expect that the more we play, the better percentage we get. In order to satisfy our curiosity and skepticism, we run the



(a) Figure A (b) Figure B

(c) Figure C (d) Figure D

Figure 1. Weights of Neural Network vs number of iterations of self play for 1000,10k,100k,1000k respectively for A,B,C and D

game several times with $k = 10000$ but we get always a percentage which is less than the one with $k = 1000$.
Despite the results of $TD - Gammon2$, we notice that our algorithm performs well, when we increase the number of training games. As a result, playing $1000,000$ times, the Neural Network Temporal Difference (λ) algorithm outperforms its opponent (84.1%).
During the training we encountered practical difficulties which are associated with the computational resources required to train our neural network. As you can see in (Fig.2) to reach 84.1% we need to play $1000,000$ times and it takes 886 minutes. We could get better results, if we had a powerful machine to be able to vary the hyperparameters : number of hidden units and training games. This is why we couldn't compare our results with those obtained *here,Table 11.1* [2]. Trained with 40 and 80 hidden units (our algorithm with only 40 hidden units), and up to $1,500,000$ training games (against $1000,000$ for our algorithm) against several opponent. It's tempting to confront our algorithm to the opponents presented in *Table 11.1* [3] to effectively asses its performance.
Our algorithm is executed on a personal computer (acer Aspire V) which is characterized by

- 6 GB DDR3 L Memory
- 128 GB SSD
- Intel HD graphics 520, up to 3148 MB Dynamic Video Memory
- Intel $Core^{TM}$ i5-6200U 2.3GHz with Turbo Boost up to 2.8 GHz

Program	Hidden	Training	Time	Opponents	Results
	Units	Games	In minutes		
TD-Gammon1	40	1, 000	Less than 1 minute	Random Player	77.3 %
TD-Gammon2	40	10, 000	6 minutes	Random Player	73.9%
TD-Gammon3	40	100,000	69 minutes	Random Player	79.8%
TD-Gammon4	40	300,000	280 minutes	Random Player	82,3 %
TD-Gammon5	40	1000,000	886 minutes	Random Player	84.1 %

Figure 2. Summary of our TD-Gammon Results

As a perspective,it would be interesting to try the following tracks:

1. Change the format of input representation. Then, try to train without eligibility traces.

[2] http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node108.html
[3] http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node108.html

2. Try to pretrain the network using played-games or other opponents before selfplay.

3. During action-selection time, use exploration-exploitation strategy and compare (perhaps) the TD-training against a rollout (Monte Carlo) learning strategy.

4. Use a different model for function approximation instead of Neural Network. Also, compare the results with 1-ply and 2-ply strategies.

5. Play against intelligent agent rather than Random player. For instance : Neural network agent of different architectures and a real human agent player (good).

3. Conclusion

This practical project, allowed us to deepen our understanding of one reinforcement learning algorithm which is neural network temporal difference and put it into practice. It helped us to get an acute insight of the close relationship between theory and practice and to what extend it's important to well understand an algorithm from its foundation in order to put it efficiently in practice.