reinforcement Learning, Looking for New Backgammon Strategies

Student Name: Fatema Alkhanaizi

Supervisor Name: Rob Powell

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

Abstract —

A Context/Background

Backgammon and many other board games have been widely regarded as an ideal testing ground for exploring a variety of concepts and approaches in artificial intelligence and machine learning. Using reinforcement Learning techniques to play backgammon has given insight to potential strategies that were overlooked in the past.

B Aims

The aim of this project is to find a new strategy for backgammon; a hybrid of known strategies will be used as the basis for the new strategy.

C Method

A modular neural network architecture will be used to incorporate the different backgammon strategies. The priming and back games will be used for this project. Two modular networks will be implemented and trained, one that will include the 2 strategies separately and another one that will include a hybrid of the 2 strategies. A single neural network based on TD Gammon will also be implemented and trained. The modular networks will be evaluated against the single network and against each other. Test games against an expert user will be included to validate the new strategy.

D Results

A python package that will include modules to train and test the networks using self-play. It will also include a module for setting both a textual and a graphical user interfaces to play against the trained networks.

E Conclusions

Keywords — Backgammon; reinforcement Learning; Modular Neural Network; Priming Game; Back Game; Strategy

I INTRODUCTION

A Backgammon

Backgammon is a game played with dice and checkers on a board consisting of 24 fields, in which each player tries to move their checkers home and bear them off while preventing the opponent from doing the same (Keith 1995). Figure 1 illustrates the setup that was be used for this project.

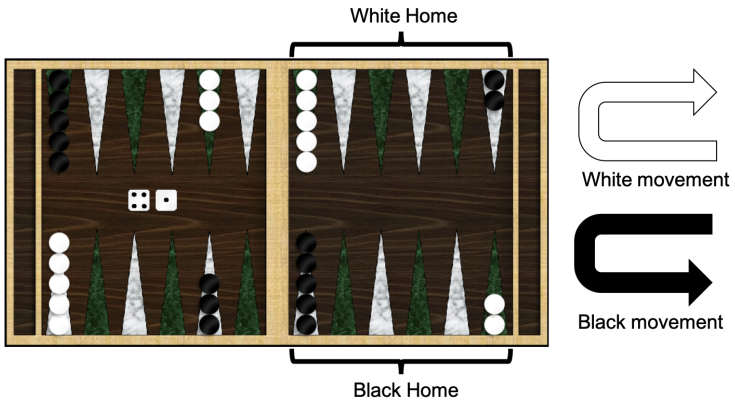


Figure 1: Backgammon board setup

There are many know backgammon strategies. The following strategies were the studied in this project:**The back game**, the player tries to hold two or more anchors, points occupied by two or more checkers in the opponent's home board, as long as possible and force the opponent to bear in or bear off awkwardly (Keith 1995). **The priming game**, a particular type of holding game that involves building a prime a long wall of the player's pieces, ideally 6 points in a row in order to block the movement of the opponents pieces that are behind the wall (Keith 1995).

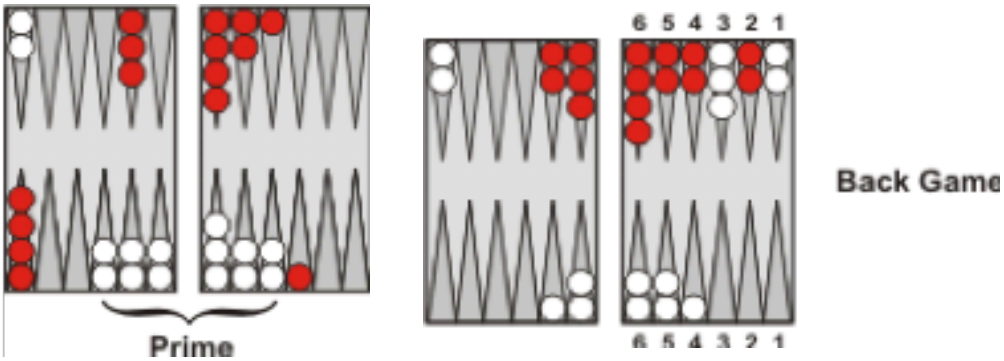


Figure 2: Illustration of Priming and Back games

B Reinforcement Learning

Backgammon and many other board games have been widely regarded as an ideal testing ground for exploring variety of concepts and approaches in artificial intelligence and machine

learning. Reinforcement learning is an often used approach. A reinforcement learning algorithm allows an agent to learn from its experience generated by its interactions with an environment (Sutton & Barto 1998).

Using reinforcement Learning techniques to play backgammon had given insight to potential strategies that were overlooked in the past (Sutton & Barto 1998). TD-Gammon of Tesauro (1992) used TD(λ) reinforcement learning methods with a single neural network that trained itself and learned from the outcomes to be an evaluation function for the game of backgammon. (add why this is significant) In backgammon, the dice rolls guaranteed sufficient variability in the games so that all regions of the feature space would be explored. This characteristic made it perfectly suited for learning from self-play as it overcame a known reinforcement Learning problem with the trade-off between exploration and exploitation (Frnkranz 2000). TD(λ) is part of Temporal difference algorithms which is a class of reinforcement learning that uses the difference between two successive positions for backpropagating the evaluations of the successive positions to the current position. TD(λ)-algorithms are parametrised by the value λ which makes the agent looks further in the future for updating its value function, strongly influencing the agent learning (Sutton & Barto 1998).

C Project Overview

Extending the work of Tesauro, Boyan (1992*a*) showed that the use of different networks for different subtasks of the game could out perform learning with a single neural network. Many AI software for Backgammon use similar modular neural network architecture i.e. GNU-Backgammon, a top performing open-source software, uses 3 different neural networks for their evaluation function (Silver 2006). This project will focus on using modular neural network architecture to incorporate a hybrid of backgammon strategies to find a new game strategy. In addition, the performance of the new strategy will be evaluated by comparing it to another network that includes the strategies separately. Combinations of 2 strategies will be studied: **the back game** and **the priming game**.

D Research Questions

The following questions were investigated in this project:

1. Would using hybrid strategies result in the agent learning a new strategy? or would one strategy be more dominant than the other?
2. Would including more features for the networks to evaulate result in better strategies?
3. How effective would the hybrid strategies be over the strategies being included separately?
4. How would be adding doubling cube to the network evaluation influence the learning outcome?

E Project Objectives

Basic objectives: Train a monolithic nueral network based on TD Gammon architecture. Train a modular nueral network that includes primining and back game strategies. Train a modular nueral network that includes hybrid of primining and back game strategies. Intermediate

objectives: Experiment with the learning parameters and the inclusion of more features extracted from the current state of the game to the networks. Use pubeval as a bench player to evaluate the performance of the networks during training. Use gnubg features to analyse a complete game, to evaluate the moves made by each network; this gives an estimated numerical evaluation to the performance of strategy followed, is it making the best possible moves for the given dice/situation. Advanced objectives: Including doubling cube decesion process to the networks. Changing the action selection depth (n-ply algorithm) Increasing the size of the hidden network. All basic and intermediate objectives were reached. The advanced objectives were left for future work.

F Deliverables

- **Trained neural networks**: A single neural network based on Tesauro's TD Gammon, a modular neural network that uses a new strategy, a hybrid priming and back strategy, and a modular neural network that uses two separate known strategies, priming and back strategies.
- **AI agent**: This agent was able to use the trained network to evaluate and make moves.
- **User interface and human agent**: This was a simple command line interface which took user inputs to make moves and to play against the trained networks. Another complex implementation was a graphical user interface which captured the user clicks for making moves.
- **Testing suit**: This was used to run all tests and evaluations to be done on the networks.
- **Project Report**: All tests and evaluations were recorded and analysed in this report.

II RELATED WORK

This section presents a survey of existing work on the problems that this project addresses. it should be between 2 to 4 pages in length. The rest of this section shows the formats of subsections as well as some general formatting information for tables, figures, references and equations.

PROBLEM BACKGROUND

The domain of complex board games such as Go, chess, checkers, Othello, and backgammon has been widely regarded as an ideal testing ground for exploring a variety of concepts and approaches in artificial intelligence and machine learning. TD-Gammon of Tesauro (Tesauro 1992, Tesauro 2002) had demonstrated the impressive ability of machine learning techniques to learn to play games. TD-Gammon used reinforcement learning techniques with a Neural Network (NN) that trains itself to be an evaluation function for the game of backgammon, by playing against itself and learning from the outcome (Tesauro 2002). eXtreme Gammon (GameSite 2017), Snowie (Egger 2017), and GNUBG (Silver 2006) are some of the strongest backgammon programs that use Neural Networks; eXtreme Gammon is currently the supreme software (Depreli 2012*a*). Different variants of backgammon(Papahristou & Refanidis 2011, Papahristou & Refanidis 2012), training techniques, learning methods and neural network architectures have been the focus of

some researches that followed the work of Tesauro. This project will focus on studying the effect of including hybrid of Backgammon strategies to the learning network and comparing it to including the strategies separately; the introduction of those strategies will be part of the NN architecture. Many studies do not include the doubling cube in their analyses. As an extension to this project, further analyses with doubling cube will be added.

NEURAL NETWORKS ARCHITECTURE

The architecture of the NN used for value function evaluation in the reinforcement learning problem plays a big role in the learning speed and performance. A large neural network with more hidden nodes provided a better equity results compared to a smaller network as shown in the work of researchers like Tesauro (Tesauro 2002) and Wiering (Wiering 2010), however the bigger the network the more computations are required thus the slower the learning. GNUBG is made of 3 neural nets: the contact net which is the main net for middlegame positions, the crashed net, and the race net (Silver 2006). GNUBG made use of modular neural network (Boyan 1992*b*), many work that followed TD-gammon made use of this neural network architecture. Backgammon game strategies were incorporated in the NN which had enhanced the results of the NN (Wiering 2010, Boyan 1992*b*, Silver 2006).

TRAINING TECHNIQUES

Training neural networks by self-learning is the most popular training approach that is used in many Backgammon softwares. Tesauro used self-learning to train the NN for TD-gammon; TD-gammon was able to reach master-level (Tesauro 1992). It has been proven in the work of Wiering (Wiering 2010) that by utilising an expert program, the speed of learning increases in comparison to self-learning. However, the end outcome for each self-learning and learning from an expert is almost the same. Learning from watching a match against two experts is by far the worst technique (Wiering 2010).

Doubling Cube

TD-gammon used a heuristic function for doubling cube (cubeful) computations. The formula that is based on a generalisation of prior theoretical work on doubling strategies by Zadeh?Kobliska (Tesauro 2002). On the other hand, GNUBG estimated the cubeful equity from the cubeless equity by using a generalised transformations as outlined by Rick Janowski. Both approaches to computing the cubeful equity made us of a formula instead of a neural network as commented by Tesauro and GNUBG creaters the NN approach is more complex so they used the former approach. Snowie (Egger 2017) and eXtreme Gammon (GameSite 2017) both can evaluate cubeful equities however as both softwares are commercial their approaches are not known.

LEARNING METHODS

Other approaches to learning includes genetic programming (Azaria & Sipper 2005), Coevolution methods - Hill Climbing (Pollack & Blair 1998) and incremental fuzzy clustering method (Tabrizi et al. 2015). Pollack and Blair claimed that their Hill Climbing algorithm had 40% winning factor against Pubeval, a moderately good public-domain player trained by Tesauro

using human expert preferences, however, this claim was countered by Tesauro proving that his TD algorithm was better and it deals with nonlinear structures (Tesauro 1998). This indicated that TD(λ) is the superior learning method. The GP approach have achieved better results against Pubeval of 58% on the other hand and had shown good results automatically obtained strategies, but it required more computational efforts and was more complex (Azaria & Sipper 2005). Incremental fuzzy clustering method, an adaptive artificial intelligence method, used in the work of Tabrizi, Khoie and Rahimi was designed to learn to play backgammon from its opponent (Tabrizi et al. 2015); learning his movements and tactics which is something none of the previous methods provided, however there is no evidence of this approach being better than the previous learning approaches in terms of playing the game of backgammon. For the focus of this project, TD(λ) algorithm will be used along with modular neural network architecture.

III SOLUTION

A Algorithms

A.1 Temporal Difference, TD(λ)

In backgammon, the dice rolls guarantee sufficient variability in the games so that all regions of the feature space will be explored. This characteristic made it perfectly suited for learning from self-play as it overcomes the known reinforcement Learning problem with the trade-off between exploration and exploitation (Frnkranz 2000).

TD Gammon used the gradient-decent form of the TD(λ) algorithm with the gradients computed by the error backpropagation algorithm (Sutton & Barto 1998). The update rule is as follows

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\delta_t\vec{e}_t$$

where δ_t is the TD error,

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

and $\vec{e}_t$ is a column vector of eligibility traces, one for each component of $\vec{\theta}_t$, updated by

$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t)$$

The expression $\nabla_{\vec{\theta}_t} V_t(s_t)$ refers to the gradient. For backgammon, $\gamma = 1$ and the reward is always zero except when winning, reducing the update rule to

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha[V_t(s_{t+1}) - V_t(s_t)][\lambda\vec{e}_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t)]$$

α, the learning rate, and λ are constraint by the range $(0, 1)$. $e_t(s)$ is the eligibility trace for state s, it marks s as eligible for undergoing learning changes when a TD error, δ_t occurs (Sutton & Barto 1998). This exact algorithm was used in this project.

A.2 Action selection algorithm

A ply is one turn taken by one user; n-ply refers to how far the player will look ahead when evaluating a move/action (Keith 1995). The implemented AI agent used a 1-ply search algorithm, (Tesauro 1992), to pick the best legal action for the current turn; no look ahead. Each action was evaluated in the neural network, forward feeding, and the action with the maximum outcome was returned.

B Neural Network Architecture

B.1 Monolithic Neural Network

This network was based on Tesauro's (1994) TD Gammon implementation; a fully-connected feed-forward neural networks with a single hidden layer. The basic architecture consisted of one input layer I with 288 units for raw inputs representing the checkers configuration on the board,each field in the board is represented by 6 units as there are 24 fields so 144 total units and each player has their own configuration represented separately making the total 288. For basic network architecture the first 4 features from table 1 were used, the other 4 were included at a later stage. Lastly, 2 input units were included to represent if it was the player's turn or the opponent's turn.

Table 1: Possible expert features for input layer

Feature name	Description
bar_pieces_1	number of checkers held on the bar for current player
bar_pieces_2	number of checkers held on the bar for opponent
off_pieces_1	percentage of pieces that current player has borne off
off_pieces_2	percentage of pieces that opponent has borne off
pip_count_1	pip count for current player
pip_count_2	pip count for opponent
hit_pieces_1	percentage of pieces that are at risk of being hit (single checker in a position which the opponent can hit) for current player
hit_pieces_2	percentage of pieces that are at risk of being hit (single checker in a position which the player can hit) for opponent

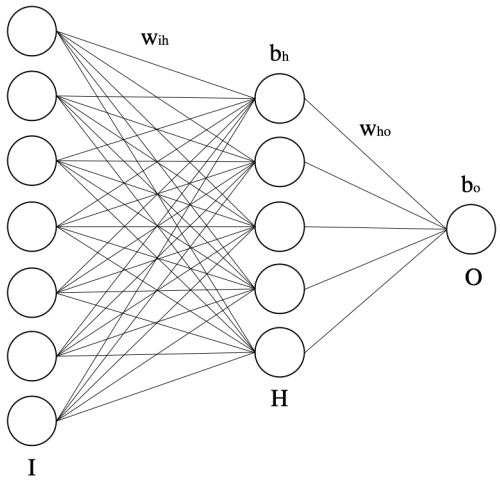


Figure 3: Neural Network architecture

As part of the network architecture, there was one hidden layer H with 50 units and one output layer O with 1 unit representing the winning probability. The number of units for each layer was based on previous implementations, taking into account the complexity introduced by

the increase of the units for each layer. The network had weights w_{ih} for all input units I_i to hidden unit H_h and weights w_{ho} for all hidden units H_h to output unit O_o. The weights were initialized to random values; hence the initial strategy was a random strategy. Each hidden unit and output unit had a bias b_h and b_o. Sigmoid function was used for the activation function between the layers. Each bias was initialized to an array of constant values of 0.1. Figure 3 includes the layout of the neural network.

A lot of implementations of this network were available in the open source community and were referred throughout the implementation of this project; two code bases from GitHub, backgammon (Hannun 2013) and td-gammon (Fleming 2016), were used as the starting point for this project. The main challenge with using open source code was debugging the code and validating it.

B.2 Modular Neural Network

Designer Domain Decomposition Network (DDD) was the Modular Neural Network architectures that was followed in this project. The DDD network consists of n monolithic neural networks and a hard-coded gating program, figure 4. The gating program, based on the work of

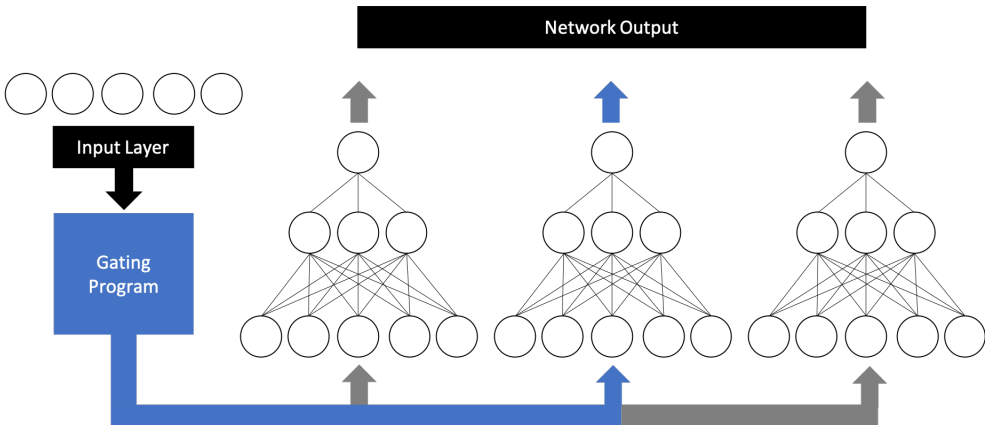


Figure 4: The DDD Network

Boyan (1992*a*), partitioned the domain space into n classes. For this project the n classes were represented by the different backgammon strategies. The each strategy was activated when certain board configurations were reached. This approach was implemented by Boyan (1992*a*) and what most recent software such as GNU-Backgammon (Silver 2006) follow. Both modular networks, seperate strategy and hybrid, implemented in this project consisted of the following basic networks:

- One network for racing game. This network addressed a known weakness of the monolithic network implementation and was only activated when both players' checkers were past each other.
- One default network for all other positions

The seperate strategy modular network included the following additional networks:

- One network for back game positions: the player is behind in the race, the pip count difference is more than 90 points, but the player has two or more anchors, checkers at one field, in the opponent's home board. This network took into account the number of checkers on the bar.
- One network for priming games: if the player has a prime of 4-5 fields, a long wall of checkers, with at least 2 checkers on each field.

On the other hand, the hybrid modular network included this network:

- One network for a hybrid priming and back game. This network combined the conditions of both back and priming games.

In both forward feeding and backward propagation, the gating program was called to select an appropriate network based on the current board configuration and extracted features i.e. pip count. Exactly one network was activated at any time. Each network had the same structure as the monolithic network including the paramaters.

C Implementation

Python 3 was used as the language for this project. The neural networks was implemented using TensorFlow package. TensorFlow was used as it allowed the generateration of training progress summary, easily save and restore the trained networks through checkpoins. Figure 5 shows the structure and the component dependencies.

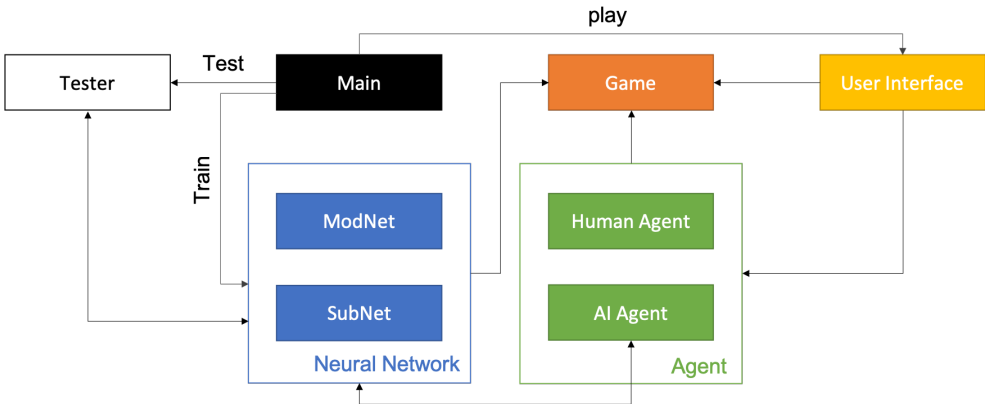


Figure 5: System components and dependencies

C.1 Main

This module was used to invoke other components in the system and handle the general actions required from the system: play, train and test.

C.2 Game

This module held the game setup, the rules and constraints of the game e.g. taking an action, finding legal moves and game board setup. An open source implementation taken from a GitHub

repository, backgammon (Hannun 2013), of this module was refactored and modified for the use in this project.

C.3 User Interface

This module handled generating the command line interface (textual interface) and the graphical user interface. The use of either interfaces depended on the availability of pygame, a graphical user interface package for python. Similarly, to the Game module, an open source implementation taken from a GitHub repository, backgammon (Hannun 2013), of this module was refactored and modified.

C.4 Agents

There are 4 types of agents that were implemented:

- **Random Agent**, an agent that would randomnly select a legal move.
- **Human agent**, an interactive agent which took user inputs either from the command line or by capturing the user clicks though a GUI to make a move/action.
- **AI agent** that used a neural network to determine the move/action for the current turn. A list of legal moves was obtained from the game module and the best action was selected based on the search algorithm.
- **Pubeval Agent**, an intermediate level benchmark agent provided by Tesauro (1993). Pubeval had two set of weights that it used to select a move: weights for racing game and weights for the remaining positions.

C.5 Modnet

This module defined the operations for extracting features from the game board and training neural networks. This module heavily depended on Subnet module. An instance of this module was used for the monolithic network. For modular networks, a game-specific gating program was implemented in this module to determine which sub-network was best suited for a given board configuration represented as a set of extracted features. For the different modular neural networks to be trained for this project, different instances of this module were created as each modular network required different gating program.

C.6 Subnet

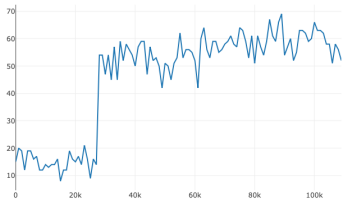
This module included the Neural Network implementation using TensorFlow. It provided routines for storing and accessing the network model, checkpoints and summaries generated by TensorFlow. In addition, it included the forward feeding and backpropagation algorithms. All networks created for this project used an instance of this module; networks used in modular neural network and monolithic neural network. An open source implementation taken from a GitHub repository, td-gammon (Fleming 2016), was used as the basis for this module.

C.7 Tester

This module included all evaluations and test routines for the neural networks.

D Strategy Validation

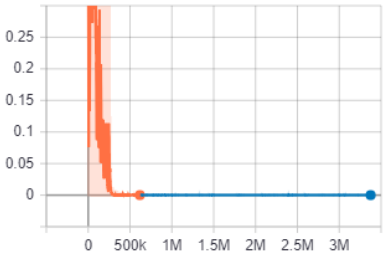
The main aim of this project was to find a new backgammon strategy. To validate the strategy followed by the networks, a measure for the general performance of the strategy against a benchmark agent was taken, and few complete matches for the best networks were analysised. Before making any measurements or analysis, it was important to determine at which stage should the training stop i.e. convergence of the network. There were 3 indicators that determined the convergance of a network: consistent test games results, total number of turns taken and convergance of the loss plot, figure 6. The loss plot was for the mean squared of TD error, δ_t. A low total for number of turns per games indicated that the training was going in the right direction (Tesauro 2002); very long games, over 500 turns, were a red flag.



(a) Consistent test games results

```
[Train 56367/1000000] (Winner: x) in 37 turns
[Train 56368/1000000] (Winner: x) in 45 turns
[Train 56369/1000000] (Winner: o) in 40 turns
[Train 56370/1000000] (Winner: x) in 54 turns
[Train 56371/1000000] (Winner: x) in 35 turns
[Train 56372/1000000] (Winner: o) in 42 turns
[Train 56373/1000000] (Winner: x) in 38 turns
[Train 56374/1000000] (Winner: o) in 42 turns
[Train 56375/1000000] (Winner: o) in 48 turns
```

(b) Low number of turns for each game



(c) Converged loss plot

Figure 6: Convergance indicators

To measure the performance of the networks, the random agent was used in early stages of this project but it was replaced by pubeval agent as it was too weak. To analyse the strategies of the networks, GNU-Backgammon software was used. GNU-Backgammon software provided a complete analysis highlighting bad moves in red and marking them with question marks 7. Further analysis was provided by clicking on the move with a breakdown of the points gained or lost by making that move and its ranking. There were many agents with varied strengths provided by the software. The world class agent was picked to provide a good challenge for the trained networks; it played a really strong game close to the best human players in the world by using 2-ply lookahead, no noise, and a normal move filter (Silver 2006).

E Training and Testing

Each network had 3 types of checkpoints that were used throught the evaluation process of the project: latest, previous, test. The latest checkpoint stored the most recent trained network. The previous checkpoint stored the latest 1,000s game trained network. The test checkpoint stored all the trained networks after every 1,000s game.

E.1 Monolithic Nueral Network

The number of training games was set to 500,000. Based on previous work of Tesauro (2002), and Wiering (2010), the best weights could be reached at a game far before the 500,000 game.

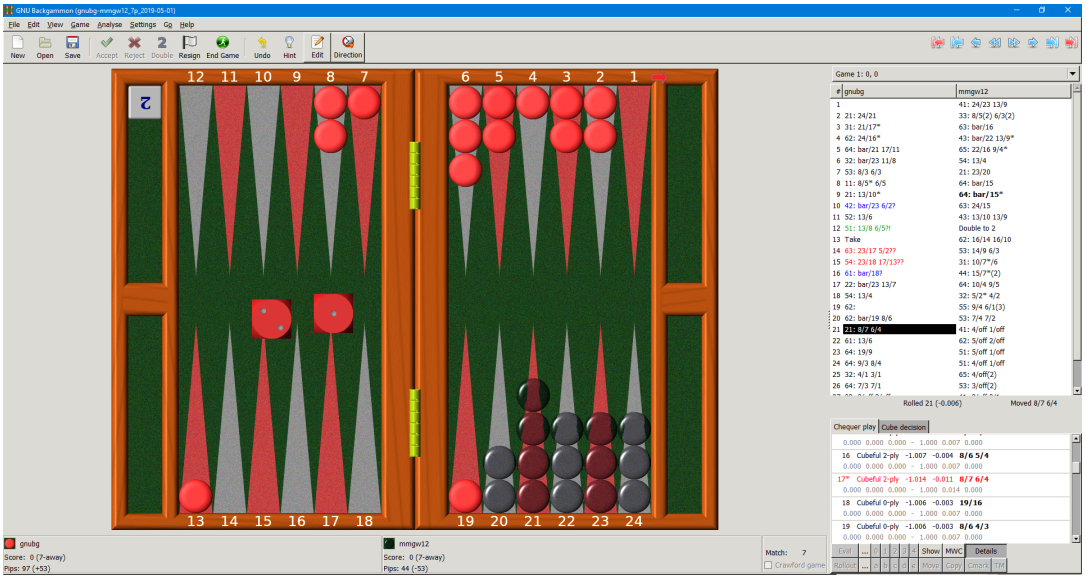


Figure 7: GNU Backgammon software

After every 1,000 games, the network was tested by playing 100 games against Pubeval agent. This test was set as an indicator of the training progress.

E.2 Modular Nueral Network

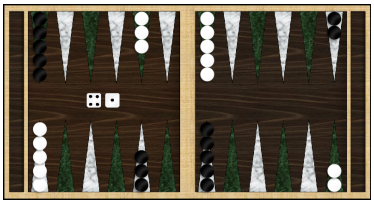
Similarly to the monolithic network training, the number of training games was set to 500,000 and after every 1,000 games, the current network would be tested by playing 100 games against Pubeval agent. However, for modular networks, after every 2,000 games the layout of the game was changed, figure 8. Due to the gating program and the conditions implied for the activation of each sub-network, 4 extra layouts were included: racing layout, prime against prime, back game from the player perspective and back game from the opponent perspective. With these layouts, each sub-network was trained for relatively the same number of games.

IV RESULTS

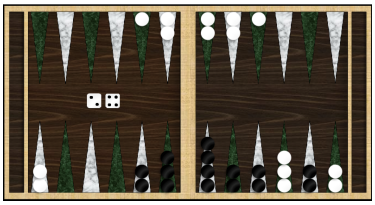
All Networks were trained on machines with the following specifications: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz with 32.0GB RAM and Hyber-V support. Multiple learning parameters and input features were tested which required the networks to be retrained with each change.

A General performance

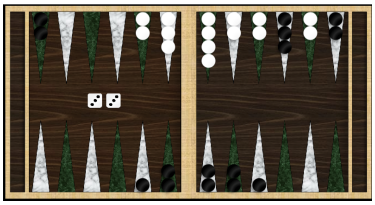
In the first stage of experiments, parameters for λ and α were varied. The values used were based on previous related works. The basic input features were used in all experiments. Only monolithic network was used at this stage, mainly because of the dependecy of the modular network to the monolithic network paramaters and structure. Initially, the random agent was used as the benchmark player. However after experimenting with different paramaters, the random agent proved to be too weak, so it was disregarded as a testing agent, table 2. Pubeval agent was used for the remainder of the tests.



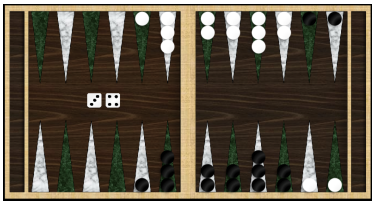
(a) Basic layout



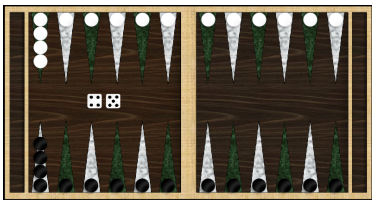
(b) Player back game layout



(c) Opponent back game layout



(d) Prime against prime layout



(e) Racing layout

Figure 8: Modular Network training layouts

Table 2: Best trained parameters for Monolithic network performance after 1000 test games against pubeval (P) and random (R) agents

$\lambda = 0.7, \alpha = 0.01$			$\lambda = 0, \alpha = 0.1$		
Max after	Win rate (P)	Win rate (R)	Max after	Win rate (R)	Win rate (P)
100,000	66.0%	3.0%	159,000	98.0%	60.00%

When $\lambda = 0.7$ and $\alpha = 0.01$, performance did not improve even after 100,000s games and only resulted in a faster convergance for the network; the training wasn't improving the performance. When λ was closer to 0 and $\alpha = 0.1$, performance significantly improved after the first 1000 games and continued to improve after 100,000 games, figure 9a. Changing α did not influence the performance as much as λ. so this parameter was kept for the remainder of the experiments and was used for the modular networks training.

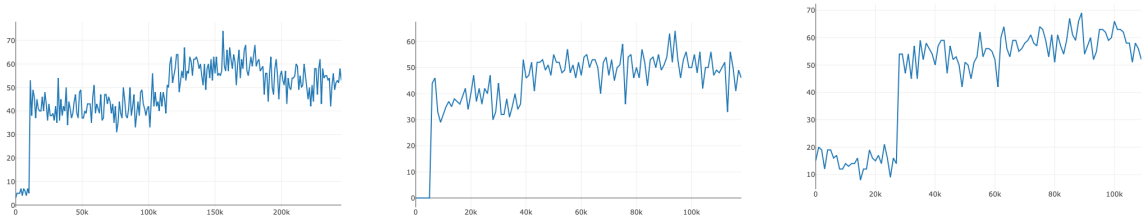
For the second stage of experiments, the input features were varied. All networks were trained at the same time. Table 3 shows the performance of the networks with basic input features, with pip count included and with both pip count and hit probability included against Pubeval agent after 1000 test games. λ and α were not changed. The networks were left to run for as long as possible. The results in the table were for the latest training game reached by each network.

Table 3: Performance after 1000 test games against Pubeval agent

	Basic		Pip count		Pip count + hit probability	
Architecture	Max after	Win rate	Max after	Win rate	Max after	Win rate
Monolithic	159,000	61.20%	94,000	54.00%	100,000	60.00%
Seperate Modular	111,000	36.20%	34,000	16.20%	12,000	24.20%
Hybrid Modular	46,000	19.30%	143,000	35.30%	29,000	20.00%

Based on the results of table 3, for the monolithic network, the inclusion of pip count and hit probability features did not improve the performance of the network. The basic features resulted in the best performance against pubeval agent and converged at a much earlier stage, figure 9.

Continuing training however, resulted in making the performance slightly worst as shown in figure 9a after the 190,000 game mark.



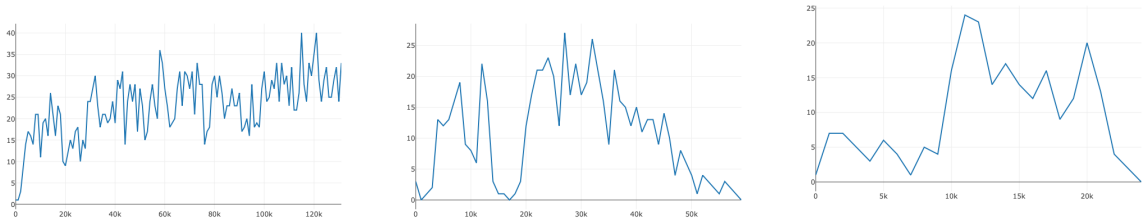
(a) Basic features

(b) Pip count included

(c) Pip count and hit probability included

Figure 9: every 1000s trained game for monolithich network against pubeval agent for 100 test games

For seperate modular network, the inclusion of the extra features resulted in the divergance of the network. A single game would take over 1,000 turns to finish which costs a lot of time, figure 10; the main reason behind the low number of training games.



(a) Basic features

(b) Pip count included

(c) Pip count and hit probability included

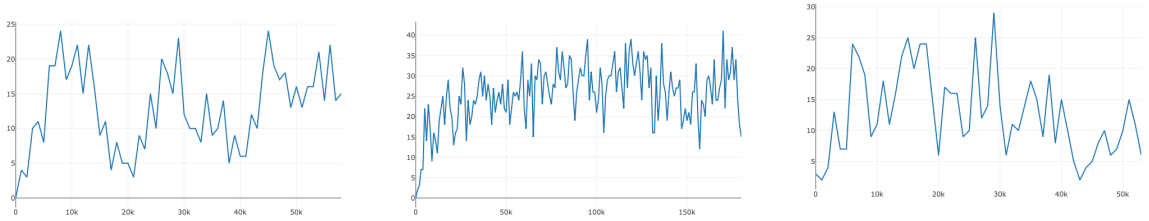
Figure 10: every 1000s trained game for seperate modular network against pubeval agent for 100 test games

Only the hybrid modular network benefited from the inclusion of the pip count feature, but it diverged in the other cases, figure 11.

Overall, these result proved that the monolithic network outperformed both modular networks, and the best seperate modular network performed slightly better than the best hybrid modular network. This result contradicted the expected result that the modular networks would perform better than the monolithic network which was based on related works and notes included in GNU-Backgammon software.

B Match analysis

The best performing networks were tested against GNU-Backgammon's world class level agent for 10 games. The best performing network were: the monolithic network with basic input features, the seperate modular network with basic input features and the hybrid modular network with pip count included.

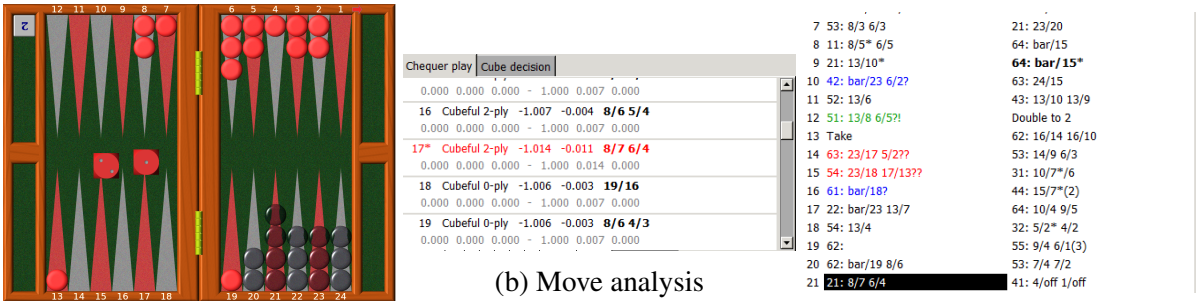


(a) Basic features (b) Pip count included (c) Pip count and hit probability included

Figure 11: every 1000s trained game for hybrid modular network against pubeval agent for 100 test games

B.1 Monolithic Network

Monolithic network made aweful decisions in some cases but generally picked one of 3 top moves during the game. The monolithic network tended to use primes as part of its strategy; this was observed by watching the network play against Pubeval and against GNU-Backgammon agent. As the game reached the racing stage, the network made average decisions by picking moves that were in the middle range, 17th move picked out of 25 possible moves, based on GNU-Backgammon analysis, figure 12. This empahsised the results of previous works regarding the weakness of a single network implementation (Frnkranz 2000). The monolithic network won 1 out 10 games against GNU-Backgammon agent.

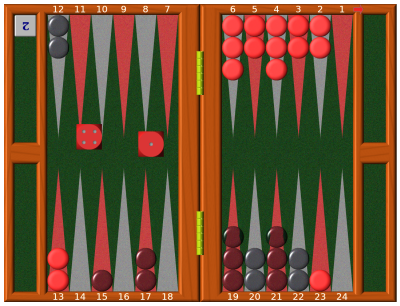


(a) Game Board (b) Move analysis (c) Full game analysis

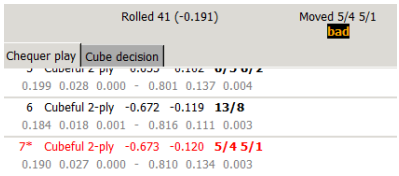
Figure 12: Monolithic network (Red) against GNU-Backgammon world class level agent (Black) - racing stage

B.2 Seperate Modular Network

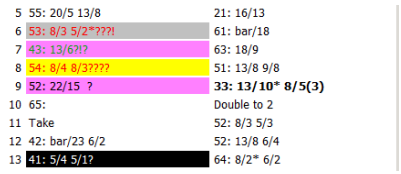
Seperate modular network played good first roll positions, made many top decisions at the racing stage. It occasionally picked one of 3 top moves during the game but made more bad decisions than the monolithic, figure 14. The hybrid network tended to pick moves that reduced the pip count which highlighted the influence of including the pip count to the features. In terms of strategy, the hybrid network seemed to build primes but also kept 2 checkers at the 13th field for the majority of the game. It didn't follow any resemblance to a back game strategy. The hybrid modular network won 1 out 10 games against GNU-Backgammon agent.



(a) Game Board



(b) Move analysis

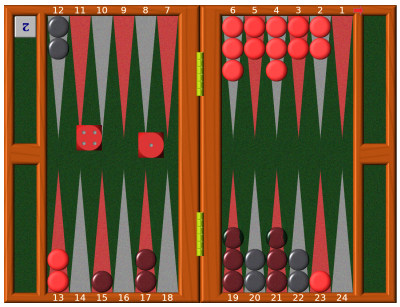


(c) Full game analysis

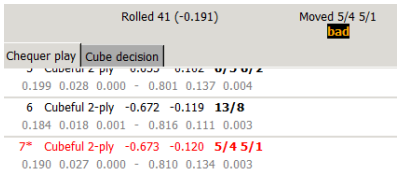
Figure 13: Hybrid modular network (Red) against GNU-Backgammon world class level agent (Black)

B.3 Hybrid Modular Network

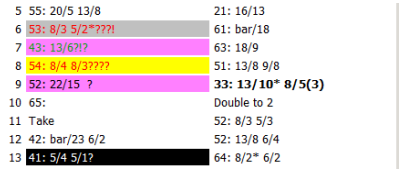
Hybrid modular network played good first roll positions, made many top decisions during the racing stage of the game. It occasionally picked one of 3 top moves during the game but made more bad decisions than the monolithic, figure 14. The hybrid network tended to pick moves that reduced the pip count which proved the influence of including the pip count to the features. In terms of strategy, the hybrid network tended to build primes but also kept 2 checkers at the 13th field for the majority of the game. It didn't follow any resemblance to a back game strategy. The hybrid modular network won 1 out 10 games against GNU-Backgammon agent.



(a) Game Board



(b) Move analysis



(c) Full game analysis

Figure 14: Hybrid modular network (Red) against GNU-Backgammon world class level agent (Black)

V EVALUATION

A Strength and limitations

The results for the monolithic network indicates that it is better than the original td gammon. however the modular networks did not give the expected results and generally performed worst than the monolithic network.

Time has been the major limitation for this project. For each network. For fast games with monolithic network around 65,000 games can be trained per day. This puts major limitation when testing different parameters and any errors in the code waste days of training. Hardware

limitations are one major aspect and highly influences training time. Training required a fast CPU to reach the goal of 500,000 games.

Bug are difficult to spot. As training takes time and good result happen after a couple of days, any bug can't be spotted until after days of training had gone by.

Modular Networks. Some games could take thousands of turns, this usually indicates that the network has diverged this incredebly increase the time a single game takes slowing down the training significantly. Although adjusting paramaters helps in some cases, convergences is not always garanteed and different runs for the same configurations could go either ways. This also explains the low number of games that were trained for those networks. Inclusion of expert however seemed to have fixed the divergance problem giving better results.

Action selection algorithm could be to blame for this behaviour as the first best action was always selected.

The results indicate that the modular networks did not out perform the monolithic network. however the hybrid module network did better than the seperate modula network.

It is difficult to decided when is it appropiate to stop learning.

Difficulty in interacting between the agent and gnu agent. required more manual effort which was incredibly ineffective and slow. Manually entering dice roll and entering move provided by both players.

B Improvements

If given a chance to repeat this project the advance objectives would have been included. Training strategy for the network re-evaluated. As most bugs in the code has been fixed and a better evaluation method had been included with using pubeval. Doubling cube will be included as part of evaluating the actions taken by the AI agent. This will be implemented by including a heuristic function as part of the action evaluation process. The implementation will be initially based on Tesauro's (2002) work, then an implementation using Crawford rule will be tested. The best trained network will be retrained with the doubling cube taken into consideration. The newly trained network will be tested for 5000 games against the older version of the network. Following Tesauro's (2002) work and recent backgammon software, the search algorithm used to determine the current turn's move will affect the general training outcome; it is evident from results collected by Depreli (2012*b*) that better results are expected from 2-ply and 3-ply search.

Use different package that is more optimized to use the CPU improving the training time.

VI CONCLUSIONS

References

Azaria, Y. & Sipper, M. (2005), 'Gp-gammon: Genetically programming backgammon players', *Genetic Programming and Evolvable Machines* **6**(3), 283–300.
URL: *https://doi.org/10.1007/s10710-005-2990-0*

Boyan, J. A. (1992*a*), Modular neural networks for learning context-dependent game strategies, Technical report, Masters thesis, Computer Speech and Language Processing.

Boyan, J. A. (1992*b*), 'Modular neural networks for learning context-dependent game strategies'.

Depreli, M. (2012*a*), 'Bot comparison final table'. http://www.extremegammon.com/studies.aspx\#D2012, accessed: 20/09/2018.

Depreli, M. (2012*b*), 'Bot comparison final table'.
URL: *http://www.extremegammon.com/studies.aspx#D2012*

Egger, O. (2017), 'Backgammon snowie'. http://www.bgsnowie.com/about/what-is-backgammon-snowie.dhtml,accessed: 28/09/2018.

Fleming, J. (2016), 'td-gammon'.
URL: *https://github.com/fomorians/td-gammon*

Frnkranz, J. (2000), Machine learning in games: A survey, *in* 'MACHINES THAT LEARN TO PLAY GAMES, CHAPTER 2', Nova Science Publishers, pp. 11–59.

GameSite (2017), 'extreme gammon'. http://www.extremegammon.com/default.aspx,accessed: 26/09/2018.

Hannun, A. (2013), 'backgammon'.
URL: *https://github.com/awni/backgammon*

Keith, T. (1995), 'Backgammon galore glossary'.
URL: *http://www.bkgm.com/glossary.html*

Papahristou, N. & Refanidis, I. (2011), Training neural networks to play backgammon variants using reinforcement learning, *in* C. D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, J. J. M. Guervós, F. Neri, M. Preuss, H. Richter, J. Togelius & G. N. Yannakakis, eds, 'Applications of Evolutionary Computation - EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Torino, Italy, April 27-29, 2011, Proceedings, Part I', Vol. 6624 of *Lecture Notes in Computer Science*, Springer, pp. 113–122.
URL: *https://doi.org/10.1007/978-3-642-20525-5_12*

Papahristou, N. & Refanidis, I. (2012), On the design and training of bots to play backgammon variants, *in* L. S. Iliadis, I. Maglogiannis & H. Papadopoulos, eds, 'Artificial Intelligence Applications and Innovations - 8th IFIP WG 12.5 International Conference, AIAI 2012, Halkidiki, Greece, September 27-30, 2012, Proceedings, Part I', Vol. 381 of *IFIP Advances in Information and Communication Technology*, Springer, pp. 78–87.
URL: *https://doi.org/10.1007/978-3-642-33409-2_9*

Pollack, J. B. & Blair, A. D. (1998), 'Co-evolution in the successful learning of backgammon strategy', *Machine Learning* **32**(3), 225–240.
URL: *https://doi.org/10.1023/A:1007417214905*

Silver, A. (2006), 'All about gnu'.
URL: *http://www.bkgm.com/gnu/AllAboutGNU.html*

Sutton, R. S. & Barto, A. G. (1998), *Introduction to Reinforcement Learning*, 1st edn, MIT Press, Cambridge, MA, USA.

Tabrizi, T. S., Khoie, M. R. & Rahimi, S. (2015), Incremental fuzzy clustering for an adaptive backgammon game, *in* '2015 Annual Conference of the North American Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing (WConSC), Redmond, WA, USA, August 17-19, 2015', IEEE, pp. 1–5.
URL: *https://doi.org/10.1109/NAFIPS-WConSC.2015.7284185*

Tesauro, G. (1992), Temporal difference learning of backgammon strategy, *in* D. H. Sleeman & P. Edwards, eds, 'Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1-3, 1992', Morgan Kaufmann, pp. 451–457.

Tesauro, G. (1993), 'Ftpable benchmark evaluation function'. `http://www.bkgm.com/rgb/rgb.cgi?view+610`, accessed: 15/04/2019.

Tesauro, G. (1994), 'Td-gammon, a self-teaching backgammon program, achieves master-level play', *Neural Computation* **6**(2), 215–219.
URL: *https://doi.org/10.1162/neco.1994.6.2.215*

Tesauro, G. (1998), 'Comments on "co-evolution in the successful learning of backgammon strategy"', *Machine Learning* **32**(3), 241–243.
URL: *https://doi.org/10.1023/A:1007469231743*

Tesauro, G. (2002), 'Programming backgammon using self-teaching neural nets', *Artif. Intell.* **134**(1-2), 181–199.
URL: *https://doi.org/10.1016/S0004-3702(01)00110-2*

Wiering, M. A. (2010), 'Self-play and using an expert to learn to play backgammon with temporal difference learning', *JILSA* **2**(2), 57–68.
URL: *https://doi.org/10.4236/jilsa.2010.22009*