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 School of Engineering and Computing Sciences, Durham University

Abstract —

A Context/Background

TD-Gammon of Tesauro (**?**, **?**) 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 (**?**). However, the monolithic nueral network soon reached its limitation an outcome studied by Boyan () and a modular nueral network becomes more suitable to over come this limitation. The newest software for Backgammon build on top of the modular architecture such as eXtreme Gammon (**?**) and GNUBG (**?**).

B Aims

The aim of this project is to study the influence of including a hybrid of known backgammon strategies such as Priming and Back games as part of the nueral network architecture and to find a combination of strategies to maximize the performance.

C Method

Initially the nueral network from Tesauro's TD Gammon will be implemented and trained. This network will be referred to as monolithic nueral network. Then, multiple Modular Nueral Networks that include hybrid of backgammon strategies will be implemented and trained.

D Proposed Solution

Keywords — Backgammon; Reinforcement Learning; Modular Nueral Network;

I INTRODUCTION

This section briefly introduces the project, the research question you are addressing. Do not change the font sizes or line spacing in order to put in more text.

Note that the whole report, including the references, should not be longer than 12 pages in length (there is no penalty for short papers if the required content is included). There should be at least 5 referenced papers.

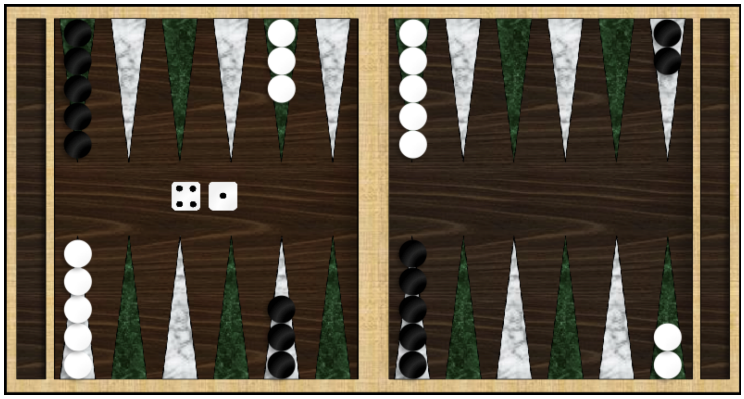

Figure 1: Backgammon board setup

A Backgammon Game

Game rules followed and game set up.

B TD Gammon

first implementation, limitations

C Searching Algorithm

depth of lookup to choose the best action for the current turn (1-ply, 2-ply ... etc)

D Learning Method

E Nueral Network architecture

F Research Questions

II DESIGN

This section presents the proposed solutions of the problems in detail. The design details should all be placed in this section. You may create a number of subsections, each focusing on one issue.

A Requirements

B Algorithms

B.1 Reinforcement Learning

- Define Reinforcement Learning components in term of Backgammon
  - Temporal difference learning
  - value function with nueral network (backprobogation)
  - after state value function

B.2 n-ply search algorithm

Following Tesauro's research and recent backgammon softwares, the search algorithm used to determine the current turn's move will effect the general training outcome; it is evident from (backgammon league table) that better results are expected from 2-ply and 3-ply search. 1-ply and 2-ply search algorithms will be tested in this project. The 2-ply algorithm algorithm will be implemented as follows:

First, to select a move, the ai agent will look ahead not only the positions that would immediately result, but also the opponent's possible dice rolls and moves. Assuming the opponent always make the best move, the expected value of each move was computed and the best was selected.

It is important to note that this computation could take some time and in timed games this could be unfavourable. The only difference between 1-ply and 2-ply is that the 1-ply won't check all the possible rolls of the opponents and stops after evaulating the immediate best action. 1-ply will be used initially for this project.

C System Components

Python 3.6 will used as the language for this project. The nueral networks will be implemented using tensorflow package. Tensorflow was picked as it is easy to use, to generate training progress summary, to save and to restore the trained network. Figure 2 shows the expected structure and component dependencie of this project.

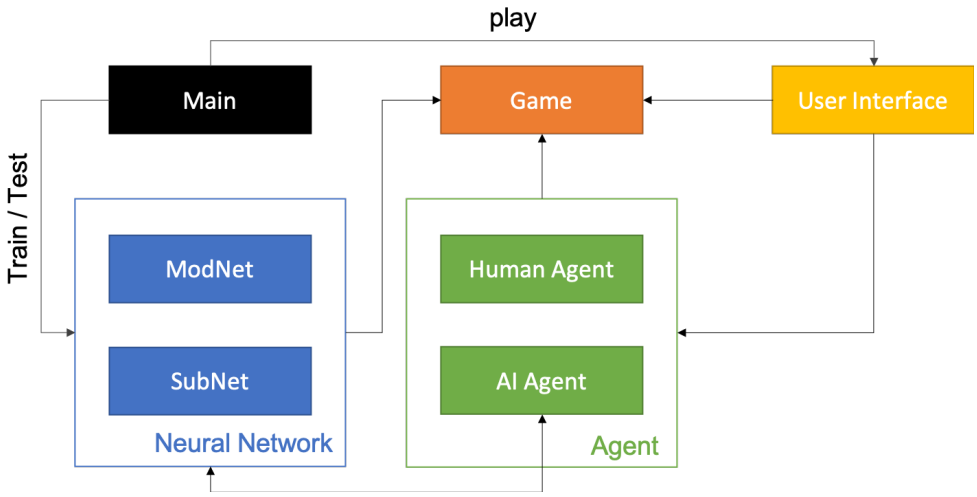

Figure 2: System components and dependencies

C.1 Game

This module will hold the game setup and define the rules and constraints of the game e.g. take an action, find legal moves and game board setup. An open source implementation taken from Awni github repository of this module will be refactored and modified for the use of this project.

C.2 Agents

There are 2 types of agents that will be implemented for this project:

- **A human agent**, an interactive agent which will take user inputs either from the command line or by capturing the user clicks though a GUI to make a move/action.
- **AI agent** will use a modular nueral network to determine the move/action for the current turn. A list of legal moves is obtained from the game module and an action will be picked based on the search algorithm.

C.3 Modnet

This module will define the operations for extracting features from the game board, testing and training nueral network/s. This module will heavily depend on Subnet module. For modular networks, a game-specific gating program will be implemented in this module to determine which sub-network will be suitable to a given input, set of extracted features. For the different modular nueral networks to be trained for this project, different instances of this module will be created as each modular network will require different gating program. The monolithic nueral network won't need the gating program.

C.4 Subnet

This module will include the Nueral Network implementation using tensorflow. It will provide routines for storing and accessing model, checkpoints and summaries generated by tensorflow. In addition, it will include the forward propagation and backpropagation algorithms. All networks created for this project will use an instance of this module; networks used in modular nueral network and monolithic nueral network. The architecture of these networks will be explained in the next section.

D Nueral Network Architecture

D.1 Monolithic Nueral Network

For this network, it will be based on Tesauro's TD Gammon implementation (Tesauro 1992, 2002); a fully-connected feed-forward nueral networks with a single hidden layer. Initially, the architecture will consist of one input layer I with 298 units which will consist of 288 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 thier own configuration represented seperately making the total 288. In addition, 8 input units will be included as expert features, table-3. Those expert features proved to provide better outcome from the network (). Lastly, 2 input units to represent the current player's token.

As part of the network architecture, there will be one hidden layer H with 50 units and one output layer O with 1 unit representing the winning probability. The network will have 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 will be intialized to be random values, hence the initial strategy is a random strategy. Each hidden unit and output unit will have a bias b_h and b_o with sigmoid activation. Each bias will be intialized to an array of constant values of 0.1.

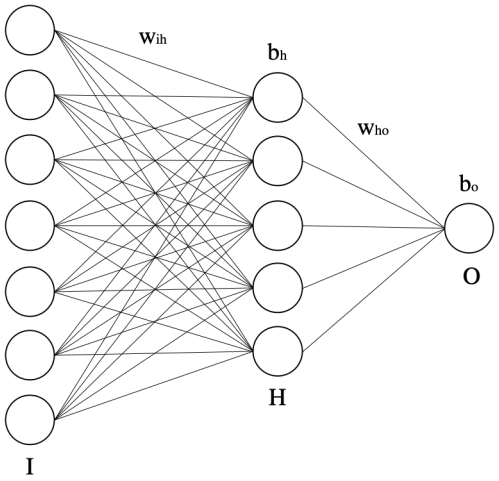

Figure 3: Nueral Network architecture

A lot of implementations of this network are available in the open source community and will be used as a reference for this project; two code basis from github _ and _ will be used and referred through out the life cycle of this project. The main challenge with using open source code will be debugging the code and validating it.

D.2 Modular Nueral Network

To incorporate backgammon startegies, modular nueral architecture will be implemented. The strategies will be represeneted by different monolithic nueral networks that will be activated when certain board configurations are reached. This approach has been implemented by Boyan() and what most recent softwares such as GNU-Backgammpn follow. The modular networks that will be implemented for this project will consist of a combination of the following networks:

1. One network for racing game; the checkers cannot be hit anymore by another checker or the checkers layout is close to this outcome (it becomes a race for who can bear off faster)

2. One network for back game positions; the player is behind in the race (pipcount) but has two or more anchors (two checkers at one field) in the opponent's home board. This network will also be used when there are many checkers on the bar

3. One network for priming game; if the player has a prime of 4-5 fields (a long wall of checkers). This is considered a defensive game

4. One default network for all other positions

Initially each network will have the same layout/architecture as the monolithic nueral network, however the networks don't necessarly need to have the same layout. At later stages, different layouts will be tested to configure each nueral network e.g. racing game network does not need inputs for all fields since most checkers will be beared off or close to being beared off. This could help reduce computing time and thus increase the efficiency of the training.

There are two types of Modular Nueral Network architectures that will be implemented in this project:

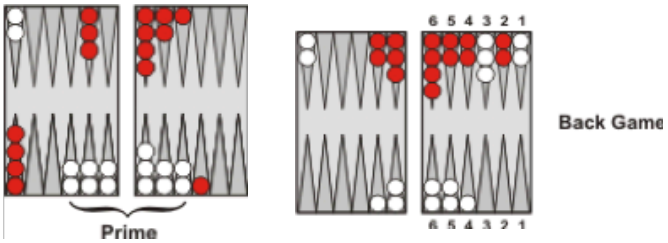

Figure 4: Illustration of Prime and Back games

- **Designer Domain Decomposition Network (DDD) -**This architecture will be used in the first stages of the project. The DDD network consists of n monolithic nueral networks and a hard-coded gating program, figrue 5. The gating program, based on the work of Boyan (year), partition the domain space into n classes. For this project the n classes are represented by the different backgammon strategies; there are other possible decompositions for the backgammon strategies but Racing, Back and Prime games will be the focus for this project. In both forward feeding and backward propagation, the gating program will

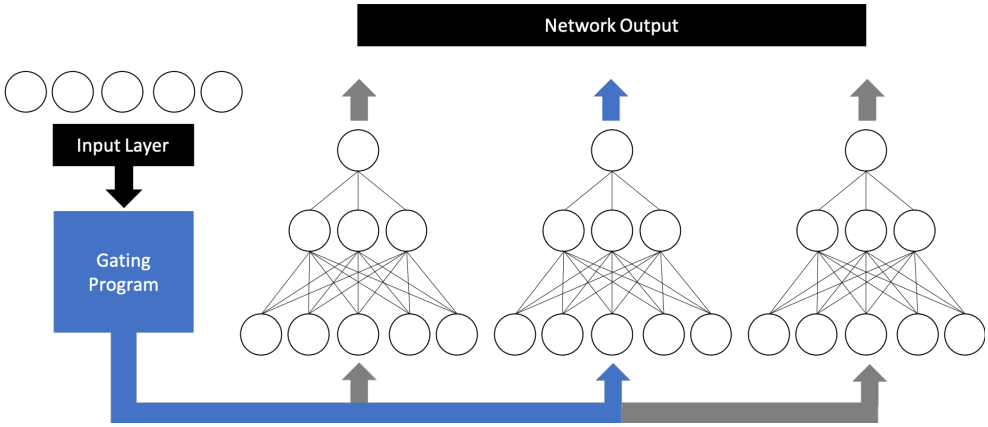

Figure 5: The DDD Network

be called to select an appropiate network for the current board extracted features (inputs). Exactly one network will be activate at any time.

- **Meta-Pi Networks -** The gating program in the DDD network will suffer from a blemish effect as noted by Boyan (year). The Meta-Pi network is a trainable gating network, figure 6. This network will be used to determine the most suited network to be triggered based on a give input. The benefit of this approach is that it will reduce the stiffness introduced by hard coding the triggers for the networks and will allow the agent to develop a smoother evaluation function. This network will require the other networks to be fully trained and only the meta-pi network would be updated in the training process. Thus, this network will be introduced in later stages of the project once all networks have been trained.

E Training

The training strategy will be based on the work of Tesauro, Boyan and Weiring. The monolithic network and the modular networks with the DDD architecture will be trained by self-play

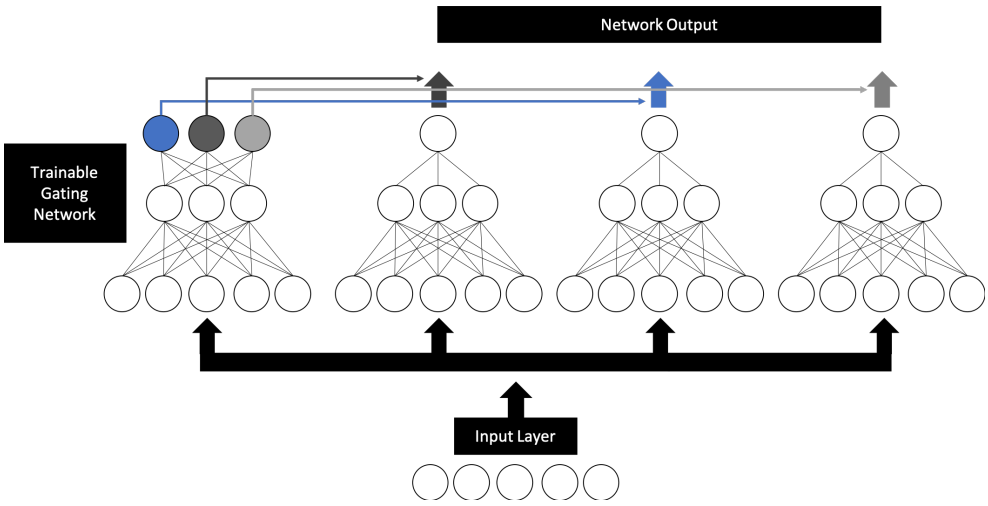

Figure 6: Meta-Pi gating network

using TD(λ) learning with a decaying learning rate α strating from 0.1 until 0.01, a discount factor γ of 1 and a decaying value for λ starting from 0.9 until 0.7; exponetial decay will be used for both learning rate α and λ. Each network will be trained on 1 million games. After each 5000 games, the network will be tested for 2500 games against the previously stored version of the network itself. This will allow to monitor the progress of the network's training.

F Evaluation

All modular networks will be tested for 5000 games against the monolithic network. The result obtained will give an indication of the improvement introduced by the modular architecture and strategy combinations. To evaulate the performance of the strategy combinations even further, 5000 test games will be run for each modular network against the other. To check that the networks do learn the strategy, a sample of the tested games will be thoroughfully exmamined and rolled out. The agent is expected to implement its own version of the strategy.

G Extensions

G.1 Doubling Cube

H References

The list of cited references should appear at the end of the report, ordered alphabetically by the surnames of the first authors. The default style for references cited in the main text is the Harvard (author, date) format. When citing a section in a book, please give the relevant page numbers, as in (Budgen 2003, p293). When citing, where there are either one or two authors, use the names, but if there are more than two, give the first one and use "et al." as in , except where this would be ambiguous, in which case use all author names.

You need to give all authors' names in each reference. Do not use "et al." unless there are more than five authors. Papers that have not been published should be cited as "unpublished" (Euther 2006). Papers that have been submitted or accepted for publication should be cited as "submitted for publication" as in (Futher 2006) . You can also cite using just the year when the

author's name appears in the text, as in "but according to Futher (2006), we ...". Where an authors has more than one publication in a year, add 'a', 'b' etc. after the year.

References

Budgen, D. (2003), *Software Design*, 2nd edn, Addison Wesley.

Euther, K. (2006), Title of paper. unpublished.

Futher, R. (2006), Title of paper 2. submitted for publication.

Table 1: List of Functional Requirements

ID	Requirement	Priority
FR1	A module for Backgammon game must be implemented. This will include the actual board setup with the rules and constraints of the game e.g. legal moves and the dice role	High
FR2	An AI agent should be created such that it uses a nueral network to evaulate legal moves/actions to play backgammon and a greedy search algorithm to pick the best legal move/action	High
FR3	A module for the nueral network should be created. It should support the funtionalities required for the nueral network e.g. updating weights through back-propagation, saving and restoring the network meta-data	High
FR3	A module for the training and testing the nueral network should be created	High
FR3	Monolithic Nueral Network should be implemented and trained based on Tesauro's TD Gammon	High
FR4	Modular Nueral Network that includes Racing Game strategy should be implemented and trained	High
FR5	Modular Nueral Network that includes Racing and Holding Game strategies should be implemented and trained	High
FR6	Modular Nueral Network that includes Racing and Priming Game strategies should be implemented and trained	High
FR7	Modular Nueral Network that includes Racing, Holding and Priming Game strategies should be implemented and trained	High
FR8	depth (n-ply) search algorithm should be implemented and incoporated into AI agent in FR2	Medium
FR9	Textual User interface for a Human agent to play against the AI agent from FR2 should be implemented	Medium
FR10	Graphical User interface for a Human agent to play against the AI agent from FR2 should be implemented	Low
FR10	Create a hueritic function for deciding when to include the doubling cube	Low

Table 2: List of Non-functional Requirements

ID	Requirement	Priority
NFR1	Trained networks should return the outcome from forward propagation within 40ms	Medium
NFR2	The AI agent should pick a legal move/action within 1s. In other word the search algorithm for the best move/action should return a value within 1s	Medium

Table 3: 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
pip_count_1	pip count for current player
pip_count_2	pip count for opponent
off_pieces_1	percentage of pieces that current player has beared off
off_pieces_2	percentage of pieces that opponent has beared off
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