Formatting Template for the 3rd Year Project Paper

Student Name: A.N. Other

Supervisor Name: Y.A.A.N. Other

Submitted as part of the degree of [e.g. MEng Computer Science] to the

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

Abstract — These instructions give you guidelines for preparing the final paper. DO NOT change any settings, such as margins and font sizes. Just use this as a template and modify the contents into your final paper. Do not cite references in the abstract.

The abstract must be a Structured Abstract with the headings **Context/Background**, **Aims**, **Method**, **Results**, and **Conclusions**. This section should not be longer than half of a page, and having no more than one or two sentences under each heading is advised.

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

This section briefly introduces the general project background, the research question you are addressing, and the project objectives. It should be between 2 to 3 pages in length. 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 20 pages in length. The system will not accept any report longer than 20 pages. It should be noted that not all the details of the work carried out in the project can be represented in 20 pages. It is therefore vital that the Project Log book be kept up to date as this will be used as supplementary material when the project paper is marked. There should be between 10 and 20 referenced papers—references to Web based pages should be less than 10%.

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 (**?**, **?**) 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 (**?**). eXtreme Gammon (**?**), Snowie (**?**), and GNUBG (**?**) are some of the strongest backgammon programs that use Neural Networks; eXtreme Gammon is currently the supreme software (**?**). Different variants of backgammon(**?**, **?**), 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 (**?**) and Wiering (**?**), 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 (**?**). GNUBG made use of modular neural network (**?**), 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 (**?**, **?**, **?**).

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 (**?**). It has been proven in the work of Wiering (**?**) 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 (**?**).

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 (**?**). 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 (**?**) and eXtreme Gammon (**?**) 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 (**?**), Co-evolution methods - Hill Climbing (**?**) and incremental fuzzy clustering method (**?**). 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 (**?**). 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 (**?**). 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 (**?**); 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.

A Main Text

The font used for the main text should be Times New Roman (Times) and the font size should be 12. The first line of all paragraphs should be indented by 0.25in, except for the first paragraph of each section, subsection, subsubsection etc. (the paragraph immediately after the header) where no indentation is needed.

B Figures and Tables

In general, figures and tables should not appear before they are cited. Place figure captions below the figures; place table titles above the tables. If your figure has two parts, for example, include the labels "(a)" and "(b)" as part of the artwork. Please verify that figures and tables you mention in the text actually exist. make sure that all tables and figures are numbered as shown in Table **??** and Figure 1.

Table 1: UNITS FOR MAGNETIC PROPERTIES

Symbol	Quantity	Conversion from Gaussian

C References

The list of cited references should appear at the end of the report, ordered alphabetically by the surnames of the first authors. References cited in the main text should use Harvard (author, date) format. When citing a section in a book, please give the relevant page numbers, as in (**?**, 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" (**?**). Papers that have been submitted or accepted for publication should be cited as "submitted for publication" as in (**?**) . You can also cite using just the year when the author's name appears in the text, as in "but according to Futher (**?**), we ...". Where an authors has more than one publication in a year, add 'a', 'b' etc. after the year.

III SOLUTION

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

This section should be between 4 to 7 pages in length.

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 (**?**). TD Gammon used the gradient-decent form of the TD(λ) algorithm with the gradients computed by the error backpropagation algorithm (**?**). 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 $(0, 1)$. The ideal value of λ is between 0.7 and 0.6 based on Tesauro's (**?**) results. α on the other hand, should ideally decay. $e_t(s)$ is the eligibility trace for state s, it marks s as eligible for undergoing learning changes when a TD error, δ_t occurs (**?**).

A.2 1-ply search 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 (**?**). Initially, the AI agent will use a 1-ply search algorithm to pick the best legal action for the current turn. Each action will be evaluated in the neural network, forward feeding, and the action with the maximum outcome will be returned (**?**).

B System Components

Python 3.6 will used as the language for this project. The neural networks will be implemented using TensorFlow package. TensorFlow will be used as it allows to generate training progress summary, to save and to restore a trained network. Figure **??** shows the expected structure and component dependencies of this project.

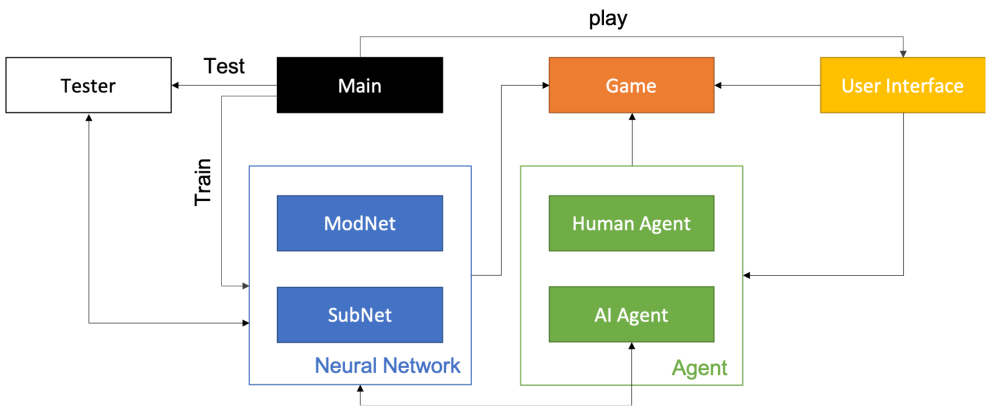

Figure 1: System components and dependencies

B.1 Main

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

B.2 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 a GitHub repository, backgammon (**?**), of this module will be refactored and modified for the use of this project.

B.3 User Interface

This module will be handle generating the command line interface (textual interface) and the graphical user interface. The use of either interface will depend on the availability of pygame, python package; pygame allows generating a graphical user interface in python. Similarly, to the Game module, an open source implementation taken from a GitHub repository, backgammon (**?**), of this module will be refactored and modified for the use of this project.

B.4 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 neural 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.

B.5 Modnet

This module will define the operations for extracting features from the game board, testing and training neural 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 neural 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 neural network won't need the gating program.

B.6 Subnet

This module will include the Neural 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 feeding and backpropagation algorithms. All networks created for this project will use an instance of this module; networks used in modular neural network and monolithic neural network. The architecture of these networks will be explained in the next section. An open source implementation taken from a GitHub repository, td-gammon (**?**), will used as the basis for this module.

B.7 Tester

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

C Neural Network Architecture

C.1 Monolithic Neural Network

For this network, it will be based on Tesauro's (**?**) TD Gammon implementation; a fully-connected feed-forward neural 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 their own configuration represented separately making the total 288. In addition, 8 input units will be included as expert features, table-**??**. Including expert features proved to provide better outcomes from the network (**?**). Lastly, 2 input units to represent the current player's token. As part of the network architecture, there will be

Table 2: 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 borne off
off_pieces_2	percentage of pieces that opponent has borne 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

one hidden layer H with 50 units and one output layer O with 1 unit representing the winning probability. At later stages the output layer will be extended to include 4 units, 2 units for the player winning the game and winning by a gammon and 2 units for the player losing the game and losing by a gammon. 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 initialized 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 initialized to an array of constant values of 0.1. Figure **??** includes the layout of the neural network.

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 bases from GitHub, backgammon (**?**) and td-gammon (**?**), will be used and referred throughout the life cycle of this project. The main challenge with using open source code will be debugging the code and validating it.

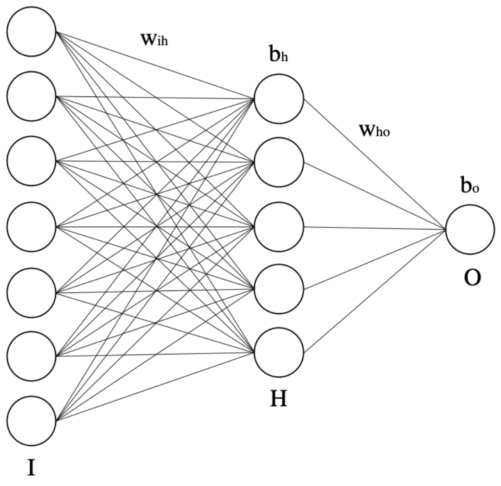

Figure 2: Neural Network architecture

C.2 Modular Neural Network

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

1. One network for back game positions; the player is behind in the race (pip count) 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.

2. One network for priming games; if the player has a prime of 4-5 fields (a long wall of checkers)

3. One network for a hybrid priming and back game; combines the conditions for both games

4. One default network for all other positions

Initially each network will have the same layout/architecture as the monolithic neural network, however the networks don't necessarily need to have the same layout.

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

- **Designer Domain Decomposition Network (DDD) -**This architecture will be used in the first stages of the project. The DDD network consists of n monolithic neural networks and a hard-coded gating program, figure **??**. The gating program, based on the work of Boyan (**?**), partitions the domain space into n classes. For this project the n classes are represented by the different backgammon strategies. In both forward feeding and backward propagation, the gating program will be called to select an appropriate network for the current board extracted features/inputs. Exactly one network will be activated at any time.

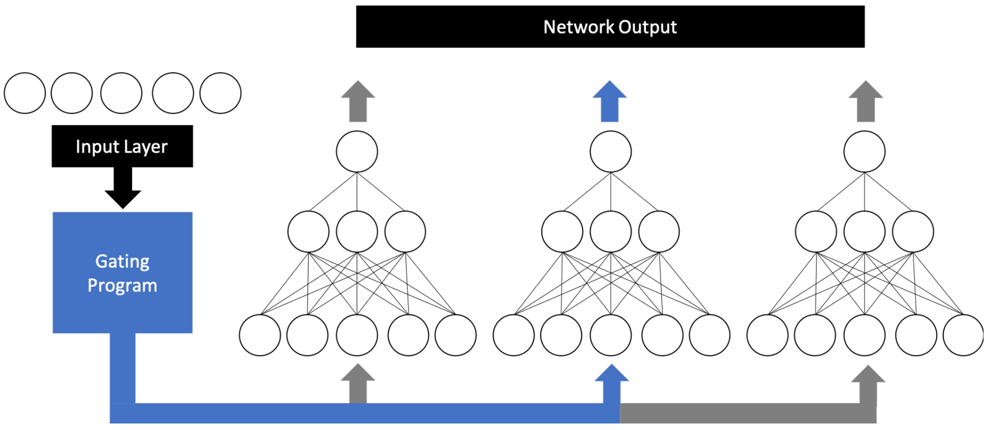

Figure 3: The DDD Network

- **Meta-Pi Networks -** The gating program in the DDD network will suffer from a blemish effect; the stiffness introduced by hard coding the triggers for the networks results in a non-smooth evaluation as noted by Boyan (**?**). The Meta-Pi network is a trainable gating network, figure **??**. This network will be used to determine the most suited network to be

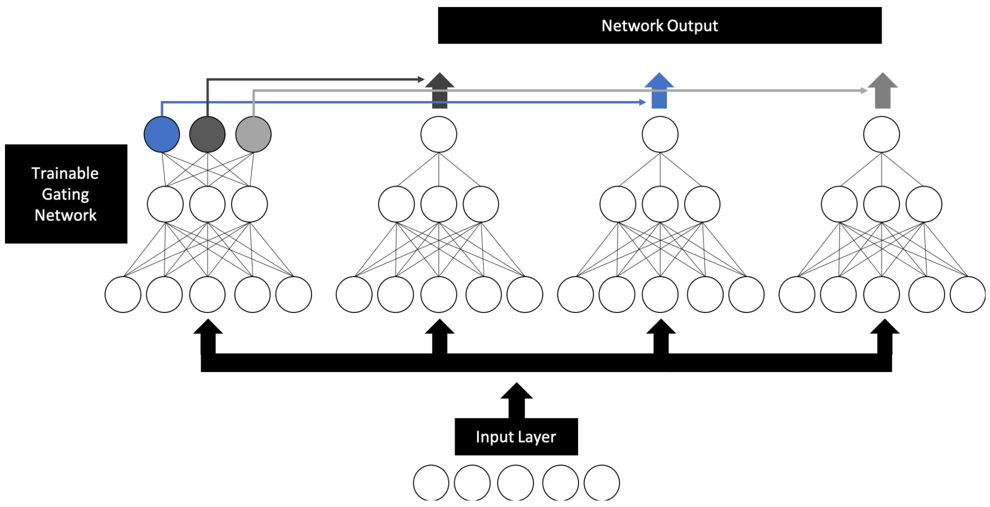

Figure 4: Meta-Pi gating network

triggered based on a given input. The benefit of this approach is that it will allow the agent to develop a smoother evaluation function. This network will be introduced in later stages of the project once all networks have been trained.

D Training

A total of 3 networks will be trained; 1 monolithic network and 2 modular networks. The training strategy will be based on the work of Tesauro (**?**), Boyan (**?**) and Wiering (**?**). The monolithic network and the modular networks with the DDD architecture will be trained by self-play using TD(λ) learning with a decaying learning rate α starting from 0.1 until 0.01 and a decaying value for λ starting from 0.9 until 0.7; exponential decay will be used for both learning rate α and λ. Each network will be trained on 500,000 games. After each 5000 games, the network will be tested for 1000 games against the previously stored version of the network itself. This will allow to monitor the progress of the network's training.

Before running the full training with 500,000 games, different configurations for each network will be tested e.g. the addition of expert features as part of the input layer. The number of training games will be reduced to 100,000 and the resultant network will then be tested against the other configurations of that network for 2000 games. The best combination of configurations will be used to fully train each network.

Once DDD networks are finished training, meta-pi gating network will be trained following the same strategy.

E Evaluation

Both modular networks will be tested for 5000 games against the monolithic network. The result obtained will give an indication of the general performance of the networks and effectiveness of the strategies. A random sample of those games will be recorded and analysed to evaluate the strategies followed by both modular networks. In addition, both modular networks will be tested for 5000 against each other to measure the performance of the hybrid strategy.

To further evaluate the networks, few test games against an expert-level backgammon human player will be conducted. The expert will be asked to provide feedback about the AI agent's actions and strategy of each modular network. The expert won't be told any details regarding the network that the AI agent will be using at first. After the feedback is obtained from the expert player, they will be made aware of the AI agent that used the network with the new strategy and will be asked to do few more test games and the validate if the network performs a hybrid strategy successfully and it is indeed a new strategy.

IV RESULTS

this section presents the results of the solutions. It should include information on experimental settings. The results should demonstrate the claimed benefits/disadvantages of the proposed solutions.

This section should be between 2 to 3 pages in length.

V EVALUATION

This section should between 1 to 2 pages in length.

VI CONCLUSIONS

This section summarises the main points of this paper. Do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions. This section should be no more than 1 page in length.

The page lengths given for each section are indicative and will vary from project to project but should not exceed the upper limit. A summary is shown in Table **??**.

Table 3: SUMMARY OF PAGE LENGTHS FOR SECTIONS

Section		Number of Pages
I.	Introduction	2–3
II.	Related Work	2–3
III.	Solution	4–7
IV.	Results	2–3
V.	Evaluation	1-2
VI.	Conclusions	1