Finding a winning strategy for Monopoly

Christopher Berry

August 30, 2013

Contents

Abstract

This is the abstract

1 Introduction

Modern computers work in a very different way to the human brain. They can quickly and accurately answer large and complex mathematical problems which would take humans significantly longer. However, there are many other areas where human ability exceeds the capability of machines. Humans are required to make many thousands of decisions every day, from simple decisions such as to whether to have tea or coffee in the morning to much more complex decisions like whether to move house. There is a high parallelism with these problems as many pieces of different information need to be associated together to make a final decision [**?**]. Solutions to these kind of problems are extremely difficult using traditional programming methods for developing AIs.

Combining Reinforcement (RF) learning with neural networks could provide a new approach to programming computers. It could allow computers to achieve more; giving them the power to adapt to their environment and answer complex questions where many factors need to be considered simultaneously.

RF learning algorithms have the potential to change the way we think about computers and have many real applications. It has always been in the realm of

science fiction for computers to be making decisions and acting for themselves yet these methods are beginning to be applied to real world problems. Some of these applications already include diverse fields such as cell phone channel assignment, financial trading systems and even an application which can help doctors diagnose cancer cells [**?**] [**?**] [**?**]. The 2012 It is not unreasonable to expect these applications to grow with time as more computational power becomes readily available to design more complex networks. One of the most interesting and successful applications of RF learning was Tesauros. Tesauro used a method of RF learning known as Temporal Difference Learning (TD) to produce an AI which eventually was able to compete with the most skilled backgammon players. Despite this huge success many other applications have failed to beat conventional AIs ability to tackle a given problem.

I aim to apply the TD method of RF learning to the game of Monopoly. This will provide a greater insight into the conditions effecting the algorithms applications to other problems. Through tackling this project I intend to learn more about the TD(λ) and its applications as well as improving my own knowledge of the Java programming language. In this paper I will be describing the problem I aim to solve, reviewing some of the previous works in the field, outlining the steps I intend to take to solve the problem, looking at the possible approaches for tackling the problem, talking about how I will set up the AI, examining the different results I obtained before concluding my findings and highlighting opportunities for additional work.

2 Description of the problem

2.1 Problems to solve

I will need to design an AI capable of analysing the outcomes of all possible moves at every stage of the game and able to choose the one which will lead to the largest payoff. In a simple implementation of TD(λ), lookup tables may be used to store value of each state [**?**]. However, this is not appropriate for a game like Monopoly with such a huge space state, where a player having $1 more than a previous state would mean a whole different state. This issue is often resolved by using neural networks. Neural networks are made up of a series of nodes spread across a number of layers. The first layer is the input layer which takes inputs directly from the environment, these inputs are then passed one or more layers of hidden nodes. It is the presence of this hidden layer which allows the AI to learn complex tasks by combining the inputs in different combinations [**?**]. Although it is feasible to use more than one hidden layer the Kolmogrov representation theorem states that any function, however complex, can be represented with a 3 layer neural network [**?**]. The information then leaves the hidden layers to the output layer. The AI can utilise the values which come out at the output layer to assess a move and make decisions.

The other main problem in the RF Learning field is the Temporal Credit Assignment Problem [24]. This occurs when an must AI performs a series of

actions before receiving a payoff, such as in board games where a number of moves must be completed before a winner is declared. The problem appears because is it is difficult to assign a blame or reward to each of the actions that lead it to the games end point.

One method which resolves this problem is the Monte Carlo method. Learning occurs at the end of a game at which point the AIs predictions from throughout the game are assessed against the actual outcome and given the appropriate payoffs [24]. In games like Monopoly this has a considerable disadvantage. Enough memory will need to be made available to store the predicted outcome of each move as well as all of the moves that have been made. When games can go on for a large number of rounds this becomes a large problem. This is where the TD(λ) algorithm can be really beneficial.

The TD(λ)algorithm was developed by Sutton and Barto as an extension to the earlier work by Arthur Samuel [24]. Like Monte Carlo methods, TD(λ) requires only experience in a model of an environment to learn however it is able to learn while the program runs. The model updates its estimates based on previously learned estimates - without having to wait for the final outcome (known as bootstrapping) [24]. A brief example of this would be if a player chooses to make a move which leads to the state a and on its next turn the state is b then an adjustment is made to to the value of a moving it towards b. If the move leads to the final payoff of the game, the predicted outcome of the move the adjustment is made towards the final payoff value. This effectively means that the final payoff value gets propagated back through the sequence and all moves eventually receive credit for the outcome. The value of the λ in the equation is what defines the speed at which this process occurs. This means the value of λ acts as a rate of discounting, and a higher λ means future payoffs matter more to the current state. Sutton and Barto were able to prove that both methods converge to the correct result with an infinitely large number of games and even found through experimentation that TD(λ) appears to have a faster convergence rate than the Monte Carlo methods [24].

The final learning method I considered was Genetic algorithms. An AI using genetic algorithms plays a version of itself with a slightly mutated neural network. The most successful (fittest) is kept and plays against different versions of itself with slight mutations and the process is repeated. The key difference between Genetic Algorithms and RF Learning methods are that Genetic Algorithms aim to create many AIs and find the best whereas RF Learning methods aim to produce a single AI and continually try and improve it [17]. Although there has been some success such as [6] they havent had the same success as TD(λ). Due to their limited success at this time, the focus of this project will be to implement a TD(λ) algorithm.

2.2 The Monopoly board game

Monopoly is a board game for 2 to 6 players. Players take in it turns to move around the board. If a player lands on a property they are given the option to buy it, otherwise it is auctioned off between the other players. If another player

owns the property they have to pay rent to the owner. Properties are divided into 8 groups of colours, if a player holds all properties in a single colour they are said to own a monopoly. Owning a monopoly causes the rent of properties in the group to double. A player can also buy houses and hotels on the properties, further increases the rent other players must pay. A player also makes money for completing a lap of the board (passing the space called Go) and can make smaller amounts by landing on a Community Chest or Chance space which make the player draw from a deck of cards which can cause either beneficial or negative effects. The aim of the game is to bankrupt all the other players. One of the key elements of the game is deal making. Players make deals to try and obtain monopolies which requires good negotiation and people skills. The full rules I used were the official game rules found in the U.K boxed version of the game [**?**].

As my literature review will show, board games are a popular problem to apply RF learning to. They make good targets for RF learning as their rules are well established and their sequential turn based nature lends itself to RF Learning algorithms. Many board games, including Monopoly, have established strategies which can be used to help measure the success of an AI.

Monopoly contains some aspects which model real world problems such as bidding and trading. Tesauro believed that studying games that apply these factors in could lead to more immediate applications of RF learning in the real world [26].

Although there are widely accepted strategies for Monopoly [13] there has been no attempt to solve the game completely for all scenarios. Both Stewarts and Collins present a mathematical analysis of which spaces are landed on most and therefore which should be the best to buy. This analysis and the opinion of an expert judge both confirm the value of the orange coloured properties [21][7][13]. The Monopoly documentary Under the Boardwalk: The Monopoly Story highlighted several more key strategies including the value of buying up to 3 houses on a property as quickly as possible as rent doubles at this point and that although red and orange properties are considered the best, owning any other monopoly can win you the game due to the large amount of randomness introduced by the dice rolls. The winner of the 2009 world champion initially only had a monopoly on the low valued light blue properties [27].

Public and critical opinion on the current AIs used in Monopoly video games are very poor. Some believe that the developers simply make the AIs more difficult to beat by giving them a higher probability of making beneficial dice rolls. There is no official confirmation of this yet it is also a widely held opinion on forums, online shops and critical reviews [19] [2] [23]. There are also complaints about the developers creating artificial difficulty in other ways such as having AIs only trading with each other [1]. The other main complaint is that the AIs are just too simple; often making extremely poor trading decisions [1].

2.3 Objectives

3 Review of previous work

Tesauros paper showed how a neural network could be trained using the TD(λ) learning algorithm, a raw encoded version of the board state and playing many thousands of games against itself to become the most powerful backgammon AI ever developed. The eventual result exceeded the ability of other AIs which had learnt from data obtained from human expert games [25]. Tesauro went on to improve the performance of his program using hand crafted inputs to produce an AI which could compete with world champion backgammon players [26]. One of the most impressive features was that if a certain dice numbers were rolled in the opening play the AI would play differently from the accepted strategy at the time. This has subsequently influenced the strategies used by the worlds top backgammon players [26]. Beyond this, the TD(λ) algorithm has been applied successfully to some other games such as Chess [3], Robocup keepaway [22] and Tic Tac Toe [8]. Unfortunately, many other implementations arent quite as successful; failing to beat conventional AIs built for the board games. Wai-Kit and King found that when they tried to apply the algorithm to Go, using an implementation based on Tesauros work, their AI was ultimately unsuccessful [15]. Similar failures have been experienced for Tetris [9], and Othello [16]. Papers which have been unable to successfully apply TD(λ) to a game can still be extremely useful to examine as they teach us about potential reasons for the failure of TD(λ) and give us a clearer idea of the requirements of an environment for a successful implementation.

In Wai-Kit and Kings paper on Go they found they found the AI could handles some aspects of the game very well such as predicting the outcome of the smaller sub-games that occur in Go in the corners, but due to the huge number of state spaces of the full game the AI was unable to cope [15]. They only ran the AI for 100,000 training games though which may have not been enough time to learn the full game [15]. Given the huge state space and the complexity of the game more training may have been beneficial to the network.

There has been several attempts to identify the properties of a problem which means this approach can be successful. Fürnkranz produced a comprehensive survey of the literature produced for machine learning in games in 2001 [11]. The author identified the main reason for the success of Tesauros implementation were the limited search space and the dice rolls guaranteeing all of the space was explored [11].

Ghorys 2004 paper aimed to identify the important features of a board game making it an eligible target for TD-Learning [12]. These are:

1. Smoothness of boards: A neural networks has a greater potential to learn if the function for making decisions in a game is mathematically smooth, so for the function f(x1,x2,x3,...), a small change in x implies a small change in f(x1,x2,x3,...) [12]. This effectively means that for board games, a small change in the board leads to only a small change in the function used to

analyse the board. Unfortunately, this may not be true for Monopoly as owning one additional property of a colour (a small change in the board) may give a player a monopoly on a colour. This significantly alters the players chance of winning.

2. Divergence rate of boards at single-ply depth: The principle behind this is that all moves should make as little impact on the board as possible. Ghory compares some examples such as Connect 4 where the only change a move can make is to add another piece to the board, to chess with a higher divergence rate (one piece moving can lead to another being removed) or backgammon (one piece moving can lead to more than one being removed) [12]. In Monopoly this may be a problem. A player can take many actions on their turn before the dice is rolled. Making deals with other players especially can drastically alter the board state.

3. State-space complexity: This is the number of unique states in a game [12]. The more state spaces a game has the more complex the evaluator function has to be to correctly predict the outcome of a game. Ghory believes that the more states there are the longer the training function will take to become a good approximator of state values [12]. As money is a continuous variable and the different combinations of spaces players can own means Monopoly has quite a complex state space.

4. Forced exploration: Learning tasks often run into the problem of AIs not fully exploring some states as they believe them to be of low value [12]. Some games force exploration due to their non-deterministic nature, often using dice rolls such as backgammon. Monopoly forces a degree of exploration with the dice rolls but as a player has a choice of moves on a turn I will need to force some degree of exploration.

Other authors, such as Pollack and Blair believe the success of Tesauros TD-Gammon had little to do with how well TD(λ) was suited to backgammon but rather how well suited backgammon is for any kind of RF learning [18]. They were able to produce good results up to the standard of Tesauros original TD-Gammon using a much simpler hill climbing algorithm. They believe the key to TD-Gammons success actually comes from the properties of backgammon, specifically the way that the dice rolls can dramatically alter the outcome of the game means the AI has to explore the full space [18].

I aim to see how true the thoughts of these authors are and whether the problem presented by a game of Monopoly is an appropriate target for RF Learning using the TD(λ) to tackle.

4 Steps I will take

To achieve my goals I will undertake a number of distinct tasks. I will create a model of the game of Monopoly in Java as this is where I have the most experience therefore I can have the best chance of getting the system up and

running as quickly as possible. I will start with the 2 player game as learning can occur quicker and it will allow me to experiment with a number of variables before moving onto a games which are set up the same way as the official Monopoly tournaments, with 4 players [27]. The most important step will be designing an AI. I will be experimenting with a number of different versions which I will discuss in the next section. Each AI will be trained for a period and tested regularly. As I want other people to play against my AI I will also be implementing methods for human players to play against the AI.

4.1 Measuring success

I will need to have a number of different methods to monitor the performance of my AIs. I will design an AI which makes decisions randomly to use as a benchmark for the progress of my other AIs. Alongside the random AI I will be utilising an ELO system similar to that used in chess. This will allow me to monitor the progress of my AIs against the random AIs and against other versions of my AI. I will design a series of queries to measure how my AI will respond in certain situations. This will allow me to see if my AI has established a solid strategy. The final method for measuring the progress of my AI will be to have it play against human players. I will publish my best AI online and challenge people to play against it and give me their feedback.

5 Possible Approaches

There are a number of factors to consider when evaluating which learning model to utilise for a given problem and reading the literature gave me an insight into the best choice to apply to the game of Monopoly. I will need to decide how I will train my neural network, what value of λ to use in the TD(λ) algorithm, the number of hidden nodes to use, the payoffs a player receives at the end of the game, the round limit to impose on the game and the state of the game when the AI begins playing.

5.1 Training the network

The methods used most often for training neural networks are supervised learning or self play. Using supervised learning, the AI learns from data, usually produced by experts in the field, and attempts to figure out which of their moves has lead to their success. Supervised learning from expert data would be very difficult to implement for Monopoly as there arent any available databases containing information on past games. In the self-play method the AI learns by playing against itself and receiving feedback from the environment on its performance [26]. Tesauro highlights some other advantages:

> While it may seem that forgoing the tutelage of human masters places TD-Gammon at a disadvantage, it is also liberating in the

> sense that the program is not hindered by human biases or prejudices that may be erroneous or unreliable. *page 185 [26].*

Dayan, Schraudolph and Sejnowski found that initially starting an AI learning from databases before moving onto self play will speed up the learning process. It may even be possible to learn from databases made from two random players playing against each other [20], which would allow me to avoid the problem of a lack of Monopoly move databases being available. In Baxter, Tridgell and Weavers Knightcap chess AI [3], the authors believed that self-play would not be a good choice for teaching their AI as the deterministic nature of chess would mean most games were not substantially different to learn the complete game. This is only a large problem where the games the AI starts playing are too different from human games. In this case the AI learnt from playing human players in an online environment and the authors believe this was hugely beneficial [3]. It allowed the AI to learn from its mistakes much more than self play as human player would often try and exploit the AIs weaknesses forcing it to correct them. Also, because players are usually playing against an opponent of similar strength, the AIs weights were gradually guided to an optimum point [3]. In an ideal world I would love to apply a similar methodology to Monopoly but currently no free platform exists for playing Monopoly.

I will attempt various methods of training for my own AI. The main method will be self play as this simple method has proven power and is simple to implement. I will be trying database learning from databases generated by random AIs as suggested by Dayan, Schraudolph and Sejnowski [20]. The final method I will be trying is playing against a pre-programmed bot. This might help push the AIs abilities further than either database learning or self play may be capable of in this game.

5.2 The number of hidden nodes in the neural network

The exact number of hidden nodes very much depends on the model that needs to be interpreted. Linoff and Berry advise that there should not be more hidden nodes than inputs as this could overfit the network, causing it to memorise its training games and therefore unable to generalise from them [4]. I will be experimenting with different numbers of hidden nodes to find which work best with my model while keeping this information in mind.

5.3 The value of λ

The value of λ controls how credit is assigned temporally. A higher value of λ lead to longer lasting traces and so future payoffs effect earlier moves more, with the extreme cases of $\lambda = 1$ resulting in the error feeding back all the way to [24]. In their paper on Go, Wai-Kit and King were able to show that using a high value of λ for the AIs (therefore making past states more important to the current state) produced AIs which were able to beat other AIs more often [15]. However, this is in conflict to many other authors who use different values

for λ, including Tesauro I believe it will depend entirely on the model that the algorithm is applied to and therefore experimentation will need to be performed with different values of λ to observe the outcome.

5.4 Round limit on the game

Friedman, Henderson and Byuens paper showed that an AI using simple strategies could get stuck playing a game of Monopoly lasting an infinite amount of time. Players can earn more by passing go than they are spending paying rent to other players [10]. I expect that my AI will start with a very simple strategy and so I will be implementing a round limit of 65 rounds. This round limit aims to simulate the time limit used in tournaments of 90 minutes (check this) for a game [27].

5.5 Different payoffs

Ghory believed there were several ways to improve the learning rate of the algorithm, and confirmed these for both Tic-Tac Toe and Connect 4 [12]. The first was to assign a value to a win corresponding to how much the player won by [12]. In Monopoly this could follow the rules used in the tournaments, where a greater number of points is awarded to a player if they bankrupt other players rather than winning by having the most assets when the round limit is reached [27]. I will be attempting a number of versions of this with my own AI to try and encourage it to learn the best way to play the game. The first version will involve 3 different rewards depending on how the game was finished. If the winning player bankrupted all other players he will receive a large payoff. If the winning player won by having the most assets at the round limit then he will receive a smaller postive payoff. If the player lost he will receive a reward of 0. The second version will use a sliding scale. A players payoff for winning will be based on the number of rounds the player took to win the game. A smaller number of rounds will lead to a larger reward whereas hitting the round limit will lead to a much smaller reward. Losing will still give a player a reward of 0.

5.6 Starting games from different positions

Ghory believed that starting games from random positions will force a degree of exploration and means the AI sees states it may not otherwise see [12]. I have a number of ideas to implement something similar in Monopoly. I will try starting the game from random positions and starting from a position where all players have a Monopoly on a set of properties. The first would simply force more exploration while I belive the second could improve the rate at which the AI gains an understanding of the value of holding a Monopoly and buying houses.

6 Design and Testing

6.1 Design of the interface

To allow other people and myself to interact with the system I will need to design a basic interface. I do not intend to this to be overly complex as it is not the focus of my project but I should make it usable and familiar to anybody who has played Monopoly in the past. An example of the completed interface can be seen in figure 1.

Figure 1: Final interface for the game

6.2 Design of the AI

For the majority of actions the AI follows a simple procedure to decide which move to take. This includes:

1. Feed the current board state into the network to get the result of performing no actions.
2. Find all possible moves from a situation
3. Update the board as if the AI had performed one of these actions
4. Get the results from the neural network of this new board state
5. Put the board back to its original state
6. Repeat this for all moves available
7. Perform the action which produced the best result. If the best result is obtained by not acting then the AI chooses not to do anything.

   This is applied exactly as described to the following actions:

   - Deciding whether to buy a property or not
   - Buying a house
   - Selling a house
   - Mortgaging a property
   - Buying back a mortgaged property
   - Assessing whether to accept a deal offered by another player
   - Deciding whether to pay to leave jail or roll to leave jail

A number of the players other actions require a more complex implementation. As too much memory would be used if every single option for deals and auctions were to be considered I needed to come up with individual ways to handle these problems.

When a property or a house comes up for auction the AI will use the following method to decide on an action. Firstly, the AI measures its own value of any of the other bidders getting the property for the minimum price. An average of the results is taken and then compared against the AIs expected value of owning this asset for the minimum bid plus a cash increment of $50. If the second value is greater than the first, the AI will bid on the asset, otherwise it will leave the bidding process.

There is a huge number of potential deals a player could make. I tackled this in two different ways. In the first method the AI found the property which would give it the most value from owning. It then looked at all potential deals it could make the properties owner and chose the one which minimised the owners payoff. This is a very aggressive strategy which worked very nicely in head to head games where the player needed to hinder the others progress as much as possible but could cause problems in 4 player games where this behaviour could just result in both AIs losing to another player.

My second method tries to rectify this problem. This goes as follows: enumerate

8. A random set of 40 deals is generated
9. Any deals which would result in the other player being in a worse situation than they are currently are removed (calculated from the perspective of the deal maker)
10. The remaining deals are then valued by the current player
11. The deal which gives the best payoff to the current player is chosen If the value of making the deal is greater than the value of doing nothing then the deal is sent to the other player to assess.

Along with the advantage for 4 player games, this method leads to a significant speed increases over the previous method I discussed. This is the style I will be using in the results I will discuss later and is the one which I would recommend for future use.

6.2.1 Forcing exploration

Monopoly has both nondeterministic (provided by the random dice rolls) and deterministic elements (buying houses, making deals etc). This can lead to situations where an AI doesnt fully explore its options and gets to stuck in a local minimum where it performs what it believes are the best actions repeatedly but never explores the rest of the state space. I will be introducing a small random

element to my AI. 5% of the time the AI will play an option completely at random while training. This help to correct the issues caused by the deterministic elements of the game. This method is know as the ϵ-soft method [24].

6.2.2 Inputs to the neural network

The neural network takes an array of values representing the board state as an input. During my project I have used numerous different structures for encoding this information. The information common to all my input structures include:

- One input representing each players money (divided by an amount so it is not so large).
- A set of inputs for each site, representing:
  - the player who owns the site.
  - the number of houses on the site.
  - whether the site is mortgaged or not.
- A set of inputs for each station and utility, representing:
  - the player who owns the site.
  - whether the station or utility is mortgaged or not.
- A set of inputs representing whether each player is in jail or not.
- A set of inputs representing how long each player has been in jail for.
- A set of inputs representing the number of get out of jail cards each player has divided the maximum number of get out of jail cards.
- An input representing how many rounds of the game have been played divided by the maximum number of rounds.

Throughout the project I continuously developed and changed this area. One change which gave a huge boost to the performance of my AI was splitting inputs I had initially grouped together. Berry and Linoff recommend that categorical inputs should be split across separate inputs, rather than all being part of the same input [4]. This is because a neural network does not understand the inputs are unordered. In my network I had inputs including who owns a property represented by a single input node with the players number being the input. This would mean the network believes that player 5 or player 6 owning a property produces a much more similar board states than if player 1 owned the property. Because this isnt true I split up the inputs up so that each player has their own input that can be off or on if they own the property or not. The performance increase after making this change was huge. After playing only 10,000 games the AI using these new inputs was able to beat an AI using the original inputs, and trained for 10 times the amount of games, 64.80% of the time. The AIs I discuss in my results will all be using this second system.

Berry and Linoffs paper also brought my attention to the benefits of keeping all inputs below 1. When inputs are larger than 1 they tend to dominate calculations and the network has to spend time reducing the weights to lessen their effects [4]. Most of these inputs were simple to reduce below 1 by just dividing them by their maximum amount (i.e number of houses, number of get out of jail cards). Money caused more problems as this could take a much wider range of values. I initially started with money divided by the starting value, although this meant often that the money input was greater than 1. I ran some tests and found that money rarely rises over 5,000. I tried dividing the input by 7,500 and by 6,000. I also ran experiments using logarithms to scale the input. These were used so that the difference between lower amounts of money was more pronounced than the difference between higher amounts of money. Experimentation with these variables I found dividing money by 7,500 and 6,000 provided very similar results while using logarithms greatly reduced the performance of my AI. I chose to divide the money input by 7,500 for future AIs as this would ensure money is less than 1 even in the more exceptional cases.

Due to the non-linear nature of property colours I omitted these from the inputs. To properly implement the inputs I would need 8 extra inputs for each property to denote which colour it is in. Therefore including colours would make the network too big and I dont have the time to train networks using this many inputs. However, this shouldnt be a problem as the board state provides enough information to works these out given enough time.

UP TO THIS POINT.

I could resolve these color issues by using additional inputs. These would represent the various states a property could take such as whether nobody owns a property in the group, if some of the properties in the group are owned by one player and others are unowned, if the properties are owned by more than one player and others are available and finally if they are all owned by one player. Unfortunately, using all of these inputs at once creates an extremely slow AI. This method may have potential but due to time constraints I was unable to test it properly. I did however test some similar methods which were less intensive. I added a set of extra input nodes for each colour group. In the first version, if any player had a monopoly on a group of colours then the input for that group would be on. The second version used the same idea but the input was on if a player was one property off owning a monopoly. I believed this would allow the AI would be able to associate a combination of these inputs with whether it owns the properties in the corresponding colour, and therefore increase the speed at which the AI would learn the conditions to gain a monopoly and put itself in a winning position.

6.2.3 Outputs from the neural network

My neural network originally started with 6 output nodes, each one representing the probability of one of the players winning. Each player would aim to try and maximise its own payoff. In Ghorys paper it is suggested that the learning rate can be increased by having a players inputs swapped [12]. This inspired me

to redesign my system allowing me to reduce the number of outputs down to a single node. The network is used to evaluate the state of the board from the point of view of one player. Every time a position needed to be evaluated, the state of the board is switched so it looked like the network could see what was happening if the current player became player 0. This change caused the network to complete games significantly faster and greatly increased the rate of learning. This increase in learning was caused by the games being much more evenly matched (as players were all of the same skill level) and the network was effectively getting trained 6 times more than before due to not having to distinguish between the individual players. At 5,000 games the network with a single output node achieved a win rate of 100% against a random AI, while the AI with 6 output nodes only achieved a win rate of 55% even after 40,000 games.

6.3 ELO Rating System

As discussed earlier my AI will be using an ELO rating system similar to the one used in chess. I will be using the table provided by Pradu Kannan on his website to calculate the ELO difference between players [14].

To establish an AIs final ELO it will be play 5 sets of 100 games against 5 different AIs. The AIs change in ELO is calculated at the end of each of these sets of games and added to its total ELO rating. The 5 AIs will have been created using different variables and have proven good performance. I will discuss them in more detail in the results section.

6.4 Design of the AI queries

To test the ability of my AIs I wanted to add some tests of a number of scenarios that a player could find itself in during a game of Monopoly. The tests were designed to assess whether the AI could properly analyse the whole board state and choose the correct decisions based on this. All of these decisions will happen as if the the game was at round 30, close to the midpoint of the game.

The first test is designed to test whether the AI will trade some of the properties they own to obtain a Monopoly. They have enough money to start buying houses if they make the deal. If they refuse the deal they are left in a situation which they are unlikely to win as landing on the other players properties will probably bankrupt the player. If the AIs have a complete understanding of the game they should accept both deals.

- P1 owns all the pink properties, each with a hotel on them and $300.
- P2 owns all the orange properties, each with a hotel on them and $300.
- P3 owns all the Mayfair and Regent St. (1 green 1 blue), and $1500.
- P4 owns all the Bond St, Oxford St and Park Lane (2 green 1 blue),and $1500.

- P4 is asked if they will accept a deal of Regents St for Park Lane, giving both players a Monopoly.
- P3 is asked if they will accept a deal of Regents St for Park Lane, giving both players a Monopoly.

In the second test none of the players are near to making a Monopoly except one. That player can almost make a Monopoly on some low value properties (light blue). Owning a full Monopoly before any other players is a large advantage. The player with the Monopoly should trade his higher value properties to try and gain a Monopoly. The players without a Monopoly should refuse the offer made as it puts the other player in too strong a position.

- P1 owns Bond St, Euston and Angel and $1000.
- P2 owns Bow St and Pentonville Rd. and $1000
- P3 owns Piccadilly and $1000.
- P4 owns Whitehall and $1000.
- P2 is asked is they will trade Pentonville Rd for Bond St with P1.
- P1 is asked is they will trade Pentonville Rd for Bond St with P2.

The third test puts two players in control of Monopolies but without enough money to build houses. The remaining two players can trade to give both of them a Monopoly. The two without Monopolies have a large amount of cash and could start building immediately. By trading they would both end up in much stronger positions, even if trading does mean one player has to give up a higher valued property.

- P1 Owns all the pinks, all with hotels and $300.
- P2 owns all the oranges, all with hotels and $300.
- P3 owns two reds and a green and $1500.
- P4 owns a red and 2 greens and $1500.
- P3 asked if they will trade a red for a green, giving both players a Monopoly.
- P4 asked if they will trade a green for a red, giving both players a Monopoly.

In the fourth test the AI is asked if they could own any property, which would they prefer to own. The fifth test asks them which Monopoly they would like to own if they could choose and the sixth test asks them which Monopoly they would like to own with a set of hotels on it.

The final test makes the players incremental offers for a mid range property (Bow Street), testing the minimum amount of money they will accept for the property from another player. This test is otherwise set up in the same manner as test 2.

6.5 Testing

To test my model and my AI I performed a large number of unit tests. I tested the AI's progress frequently myself as well as enlisting others to help me test the system and the performance of the AI.

7 Results and analysis

7.1 Database learning vs Self Play

Initially, I ran two AIs. One had learnt completely from self play and the other had learnt from a database of games generated by two random AIs playing each other. The AIs were trained for 5,000 games. As these games involved 4 player, the AIs could learn much more than the equivalent number in 2 player games. 2 AIs learning from the databases was able to win a game vs 2 AIs learning from self play 43

Both of these were added to a new 4 player ELO gauntlet along with a random AI where an AI would play against 3 AIs of the other type and its victories recorded.

7.2 Changing the value of variables

Different values of λ and different number of hidden nodes can produce better results in different models. I experimented with a number of different variables here but unfortunately this was all done before I fixed a bug in the model which meant AIs couldnt buy houses on properties after the first game in a series of games. This meant that the early AIs couldnt learn the full game however as the rest of the game was implemented correctly these tests still provide me with a good guide to the variables which provide the best learning results. In an ideal world I would have liked to rerun them when the issue was fixed however due to the large amount of time required to run these tests I thought it was best if I focussed my attention elsewhere. The different values were tested learning from a database of random games produced by 4 random AIs playing each other and each network used 60 hidden nodes.

7.2.1 Changing the value of λ

To test the effect of different λs I trained networks which were identical other than their λ values. These were played against

I found that keeping $\lambda = 0.9$ provided the best solution. For the subsequent tests I continued to keep this variable fixed. It is possible that changing the inputs and changing the model could have an effect on which λ is better, however I do not have the time to test different λs for every separate test.

AI Description	Wins vs random AI	Wins vs TD V5	Wins vs TD V5.1	Average wins	Final ELO
0.00 λ	0%	0%	0%	0.00%	-3600
0.65 λ	100%	12%	26%	0.00%	672
0.70 λ	100%	22%	30%	46.00%	833
0.75 λ	100%	20%	18%	50.67%	696
0.80 λ	100%	26%	26%	46.00%	836
0.85 λ	100%	37%	34%	50.67%	993
0.90 λ	100%	43%	52%	65.00%	1165
0.95 λ	99%	8%	34%	47.00%	259

Table 1: Performance differences caused by different λ values

7.2.2 Different numbers of hidden nodes

Another variable for which the best value depends on the implementation is the number of hidden nodes to use. I performed a number of tests with different numbers of hidden nodes to find which showed the most learning after 5,000 games. Once again the AIs were trained all from the same database of games generated by 4 random players playing games against each other.

The results from the above experiments show the 160 node AI as the most successful (79.33

Although the 160 node AI was the most successful, the amount of time required to train it was much larger and so I believed it would be better to experiment with the 80 hidden node AI. The best result was seen at 5,000 games for the AIs so I allowed them to go through another period of self play

Testing the 80 hidden node AI for periods of self play after database games clearly affects the AIs performance very negatively. The AI unlearns everything very quickly and loses to the other AIs most of the time. I believe the failure is probably due to the very different type of games that would be played by a normal AI to that played by a random AI. This new situation causes a huge amount of confusion for an AI which has established weights and causes it to lose the strategy it has developed.

7.3 Standard self play

7.3.1 80 hidden nodes

The AI running with 80 hidden nodes started performing well but performance peaked at 15,000 runs before dropping significantly by 25,000 runs. The AI had good results against the ELO AIs but was still scored worse than previous AIs, despite previous AIs being unable to learn the full game. Playing against the AI I was still able to exploit it as it does not have an understanding of the power of owning a Monopoly. This effectively puts it in a situation where it is no better than the AI trained without being able to buy houses.

AI Description	Wins vs random AI	Wins vs TD V5	Wins vs TD V5.1	Average wins	Final ELO
20 hidden nodes	100%	4%	22%	42.00%	428
40 hidden nodes	99%	32%	38%	56.33%	582
60 hidden nodes	100%	43%	52%	65.00%	1165
80 hidden nodes	100%	58%	74%	77.33%	1438
100 hidden nodes	100%	28%	48%	58.67%	1022
120 hidden nodes	100%	38%	62%	66.67%	1200
140 hidden nodes	100%	60%	62%	74.00%	1355
160 hidden nodes	99%	56%	82%	79.33%	1505
180 hidden nodes	99%	52%	60%	70.33%	882

Table 2: Performance differences caused by different numbers of hidden nodes values

AI Description	Wins vs random AI	Wins vs TD V5	Wins vs TD V5.1	Wins vs TD V5.32	Wins vs TD V5.36	Average wins	Final ELO
2,500 runs	100%	41%	55%	51%	38%	57.00%	1094
5,000 runs	100%	30%	50%	52%	58%	58.00%	1123
10,000 runs	100%	40%	48%	43%	39%	54.00%	989
15,000 runs	100%	52%	55%	53%	44%	60.80%	1228
20,000 runs	100%	50%	54%	54%	40%	59.60%	1186
25,000 runs	100%	24%	43%	36%	39%	48.40%	773

Table 3: Test table

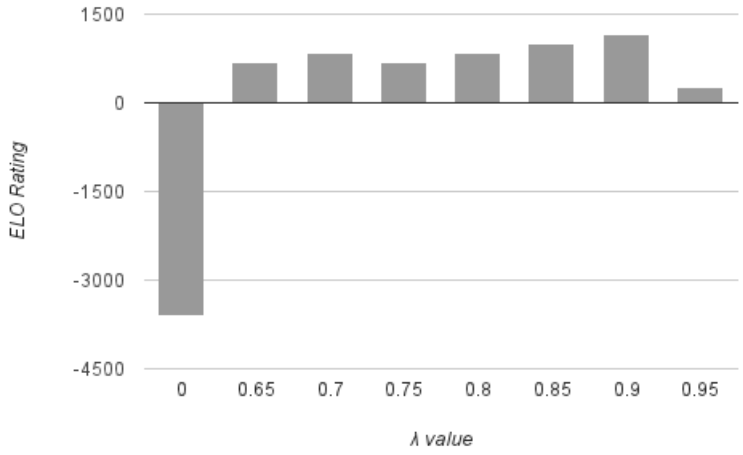

Figure 2: ELO Rating for different values of λ

7.3.2 160 hidden nodes

The AI using 160 hidden nodes for a longer period of self play never quite achieved the level of 80 hidden nodes but followed a very similar pattern (see 7). The AI showed improvement up to 15,000 runs before beginning to tail off. By 5,000 runs, both versions are choosing the same properties in the Monopoly queries. Both AIs believe that the Green properties to be the best to own a set of with or without hotels on them. At 5,000 runs they both believe the blue property Park Lane to be the best to own but by 20,000 runs both have changed their mind to the other blue property Mayfair. This choice corresponds with the AI learning which of the properties cost the most and gives the most rent however not necessarily the property that gets landed on a most. Due to the AI learning the 4 player games I could see it having a preference for properties which win it games in a smaller number of cases rather than win games consistently as in training games all AIs will be using the same strategies.

Through playing against these AIs I found neither were learning the appropriate strategies for becoming expert Monopoly players and both players prefer higher value properties over properties which would give them a Monopoly. I needed to move onto different methods which would resolve this issue. The 160 Hidden Node AI was able to perform better in the Monopoly query tests, answering 4 of the questions correctly but this didnt reflect its performance as the 80 hidden node which only answered 2 correctly was able to perform more.

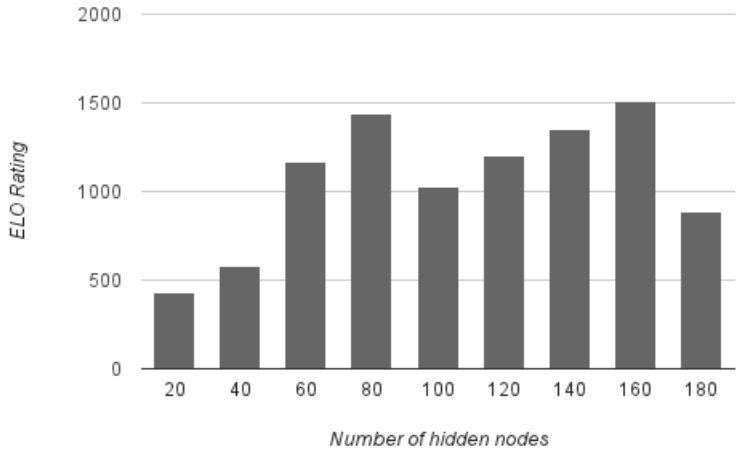

Figure 3: ELO Rating for different numbers of hidden nodes

7.4 Different win rewards

My initial thoughts was to try and punish losing more severely, giving the AI a negative reward for losing and different levels of positive win rates for winning. This produced very poor results though and the AI struggled even against the random AI so I wrote this off early.

I implemented two other win rewards in full. The first type used a flat win reward, but with the AI receiving a payoff of 1 for winning, 0.5 for winning by hitting having the most assets at the round limit and 0 for losing. The second type uses a sliding scale, so the AI receives a smaller payoff for winning the game in 60 rounds than if it wins the game in 30 rounds.

As you can see from 8, both of the AIs performed relatively poorly. The better result was produced by the first version yet this was still worse than other AIs trained for a similar period. One large problem with using this strategy is that the AI almost always plays to maximum round limit as it hasnt learnt the value of holding a Monopoly. This causes the AI to only ever receive half (or in the case of V2, even less) of the win reward it would normally which means their winning moves dont get as enforced as much as before and learning becomes slower. It may be possible to get around this problem by increasing the maximum round limit and allowing the AI to get better gradually but this is not a guaranteed solution and training an AI would take a much longer time. There may be other difficulties with learning as the temporal credit assignment problem becomes much larger the longer the game runs for.

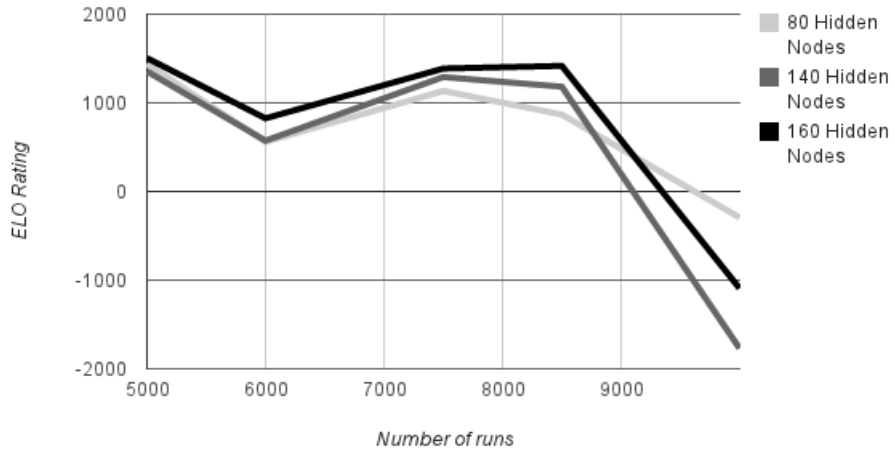

Figure 4: Change in ELO for 80 Hidden Node AI

7.5 Additional input nodes

As I described in the design section the two extra inputs set I considered is having an input for when a player is one property away from completing a monopoly on any group of properties and another for when a player has completed a monopoly on a set of properties. The idea of these inputs was to try and force an understanding of players owning a Monopoly or players being close to owning a Monopoly and see how this effects the AIs valuations.

Unfortunately, neither produced great results. As the graph 9 shows. The AI with one additional input for Monopoly showed signs of learning for a bit but its ability decreased rapidly after 5,000 games suggesting the input was confusing the network. The AI with an additional input denoting when a player is one away from a Monopoly performed better but it showed no signs of learning the additional factors I needed it to. Since these results were collected I ran the AI for a longer period and found that the abilities began to severely decrease at 25,000 runs (check). The logical extension of this would be to try and combine multiple inputs but this ends up taking a huge amount of time and is something I would recommend for future work.

7.6 Learning from a smart bot

In another attempt to force the AI to learn about the importance of holding a monopoly I wrote my own scripted AI. Although I didnt spend a large amount of time writing this I found the performance to be quite good when I played

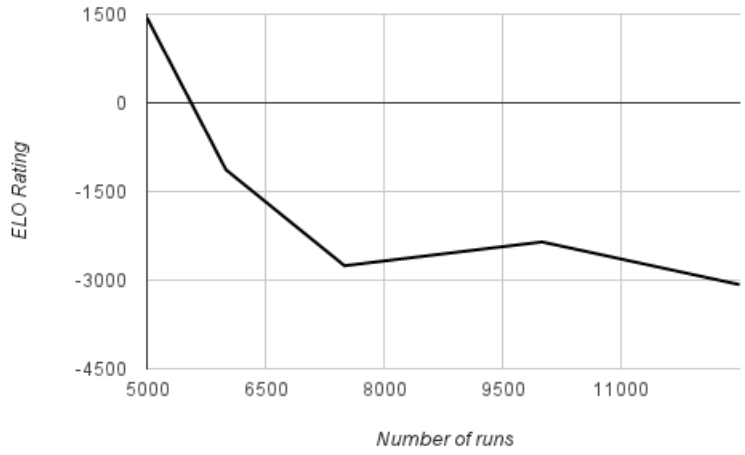

Figure 5: Change in ELO for 80 hidden node AI learning through self play after database games

against it. The bot used a strategy where it would try and get a monopoly on a group of properties as quickly as possible, and then build houses on these properties. If its money gets too low it will mortgage many of its properties that it doesnt have a monopoly on to allow it to keep buying more houses and hope another player lands on them.

The performance of this smart bot against the other AIs was very successful. It scored 1718 in the ELO tests which puts it as the most powerful AI so far and was able to beat my previous best AI the majority of the time.

One method I tried to allow my AI to learn from this bot was to let it play many games against it. This produces extremely poor results with an AI losing against the random AI a majority of the time. I believe this is due to the fact that the AI loses every game against the bot. This gives it an extremely low valuation of every state and every move and it becomes very difficult for it to learn anything of value.

The next method I tried was having the AI learning from various types of database produced by the bot. I was hoping that this would remove the problem of the AI losing all the time and the AI would be able to gain an understanding of the required strategy to win at Monopoly. The first of these databases was a set of 5,000 games generated by 4 bots playing each other. The AI performed very poorly, with a win rate of 6

I considered it a possibility that this was caused by the AI not understanding the basic game before trying to learnt the more complex strategy played by the

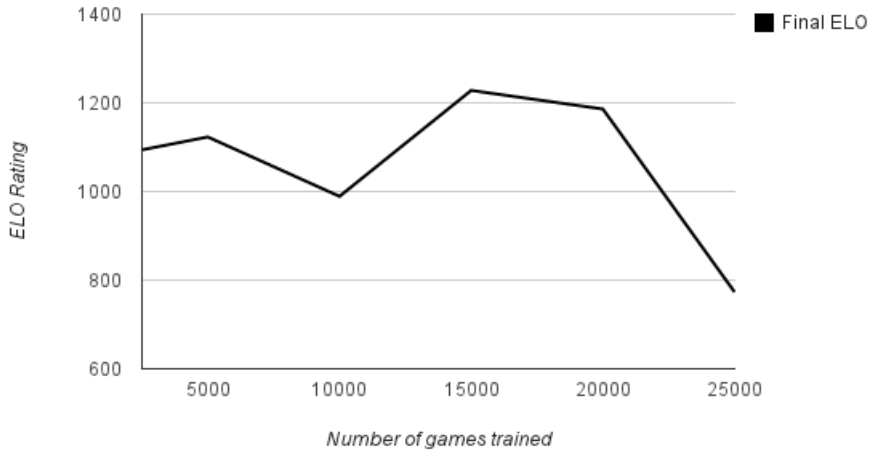

Figure 6: Change in ELO for 80 Hidden Node AI

bot. For this reason my second attempt had the AI learn from a database of produced by random AIs for 5,000 games before learning from a database produced by bots for 5,000 games. The results were still extremely poor with the AI losing all but 4

My other theory for the failures of these results was that the AI was unable to learn a strategy due to the fact the bots were all playing identical strategies. This could mean that even though one bot wins with the strategy every game there are always 3 bots playing the same strategy and losing causing a large decrease in the AIs valuation of this. To test this theory I had 2 random AIs play against 2 bots and generated a database of 5,000 games from this. Unfortunately, this wasnt enough once again and the AI was able to win only 7

7.7 What querying the AIs has taught me

Using the AI queries I have learnt much about how the AI is making decisions. In the majority of cases the AIs do not get a full understanding of the game of Monopoly, as they are unable to see the value of holding a monopoly. The AIs present a good knowledge of the value of individual properties but no matter what inputs I use they are unable to comprehend the value of owning a Monopoly. This is highlighted by the fact that after sufficient learning has taken place they are not willing to make certain trades. The AIs performing at a good level all make the sames mistakes. The AI generally only succeeds at test 1 v2 and test 3 v2 where the AI is gaining a higher value property for a

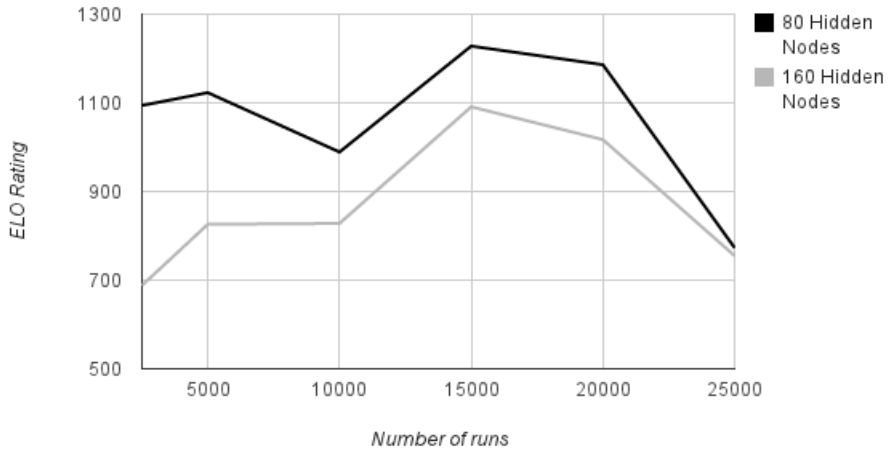

Figure 7: Comparison of 80 Hidden node AI and 160 Hidden Node AI

lower one.

The other tests asking which properties the AI wants also gives us insight into the way the AIs work. Figure refcolorchoices and **??** show the choices of the 33 AIs which obtained an ELO rating greater than 1,000.

In figure refcolorchoices the AI was asked if they could own any set of properties 30 rounds into the game which would it be. This shows the AIs huge preference towards having the green properties. This is a reasonable choice as although individually they are worse less than the blue properties, there are 3 of them as oppose to 2 giving them a higher worth. The other properties some of the AIs like are red, yellow and orange. These properties are not as valuable but due to their positioning they are often the properties that are landed on most and all seem like sensible choices.

Figure refpreferedmonopoly shows that these AIs have a huge preferences towards Mayfair and Regent Street. Mayfair is an obvious choice as the property with the highest rent and face value on the board. The green property, Regent Street, is more interesting. Green is the highest face value monopoly, yet Regent Street isnt the property that gives the most flat rent. I see little reason to choose this over Bond Street.

7.8 Adding hand-crafted features to the AI

One method for fixing the weaknesses of my AI is to add hand crafted features to elements where I felt the AI was lacking. The Handcrafted features make

AI Description	Wins vs random AI	Wins vs TD V5	Wins vs TD V5.1	Wins vs TD V5.32	Wins vs TD V5.36	Average wins	Final ELO
2,500 runs	100%	33%	37%	34%	26%	46.00%	688
5,000 runs	100%	34%	47%	34%	33%	49.60%	826
10,000 runs	100%	22%	53%	44%	32%	50.20%	828
15,000 runs	100%	33%	53%	51%	48%	57.00%	1091
20,000 runs	100%	37%	45%	48%	44%	54.80%	1017
25,000 runs	100%	33%	41%	41%	23%	48.00%	755

Table 4: Test table

checks on a number of things.

- When making a deal, before it is offered to another player, the deal is checked to make sure the AI isn't giving the other player a Monopoly except if the player is getting a Monopoly for themselves.
- When assessing a deal, it is checked to make sure the player isn't giving the other player a Monopoly unless the player is also gaining a Monopoly.
- The player handles making a deal differently if it is one property off gaining a Monopoly. The player tries to make a deal that means he gets a Monopoly and looks at a subset of deals, using the one which the AI believes the other AI will accept and maximize its own payoff.

The results from testing this AI put it below the ability of the bot I designed (winning only 38

7.9 Combining AI with bot

The next step in trying to make a better AI would be to combine my bot, with its proven performance, and my best AI to see if the two working together could create something more powerful. I set up an AI which could use the bot to make decisions in a number of circumstances while using the TD AI to make decisions in other circumstances. To test which of the two should be making decisions I initially had every decision handled by the bot, I then allowed the TD AI to make control of one decision and play 100 games against the original bot. The results can be seen in 5. The results often showed little difference between the bot and the TD AI but I still went with whichever produced the

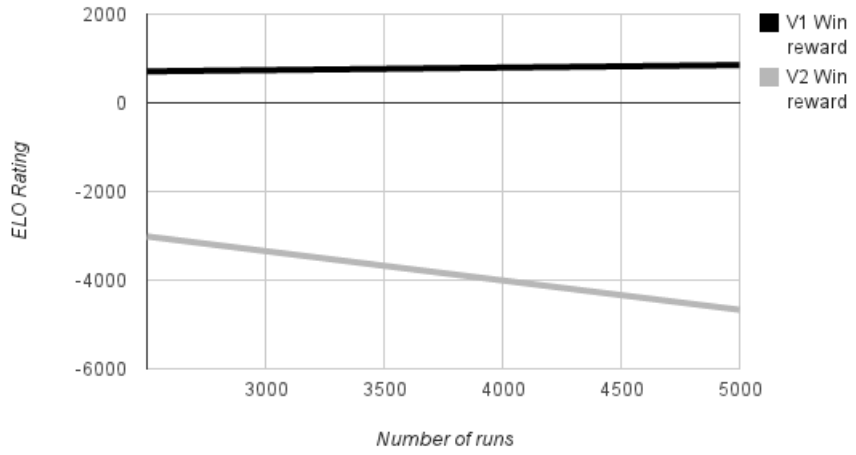

Figure 8: Change in ELO for the different win rewards

best results. This meant that in the duel AI the TD AI was responsible for deciding whether to buy properties and mortgaging properties while the bot was responsible for assessing offers, making bids on properties and handling receiving properties from bankrupt players. Because of the large amount of options presented for a normal turn and a turn in jail I decided to split these up and perform further experiments to test whether the bot or the TD AI handles the individual decisions better. The results of the 5 were used to set which component of the combined AI made the appropriate decisions and then whether the bot or TD AI controlled the mortgage decisions, house buying decisions, buying back mortgaged property decisions and deal making decisions were switched between the bot and TD AI one at a time.

The table 6 shows that the bot is better at handling whether to mortgage properties, buying houses and especially buying back mortgaged properties. The TD AI is better at deal making decisions. The final version of the combined decision making AI uses the AI which had performed best in a given situation to make these decisions.

The results of the final AI were good. As 12 shows, the AI combining the TD AI and bot produced the highest ELO rating I was able to achieve in the ELO tests, along with being able to beat the bot on its own 57.20

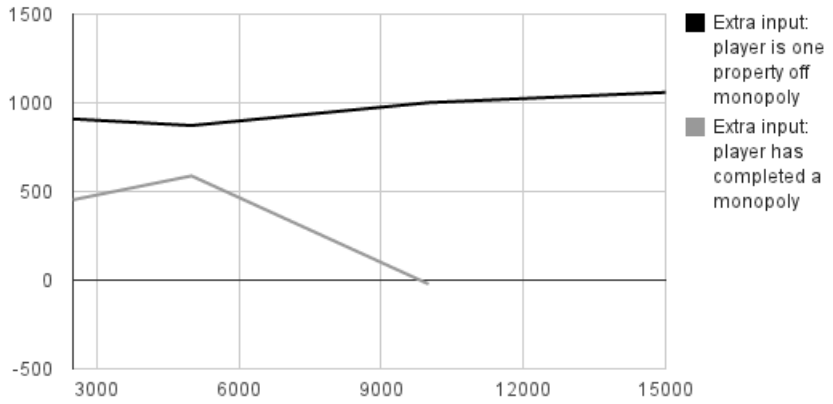

Figure 9: Change in ELO for the AIs with additional input nodes

Figure 10: AIs with ELO greater than 1,000s monopoly with hotels choices

7.10 Reviews from users

I had a number of users play against my AI to see how well it performed. The people who played against it had all played the game before but not of them claimed to be exceptional players. In total 29 games were played by other people. Of these games the players were able to beat the AI 4 times, 13.8%. This is quite a good result but I noticed that 3 of the victories were achieved by a single player. This player was able to identify the weaknesses of my AI and exploit it to win games. The AI can play to a decent level but has some flaws in its knowledge of the game. Once a player works these out he or she can easily exploit them and get a much higher win rate against the AI. However, if a player is unable to find this weakness they will have a hard time beating the AI as shown by the player who relentlessly kept playing against the AI for 14 games and was unable to beat it.

Figure 11: AIs with ELO greater than 1,000s property preferences

Decision TD AI handles	Wins vs bot
Decides whether to buy property	53%
Handles options from jail	51%
Handles standard move options	15%
Handles assessing offers made to player	44%
Makes bids on properties	46%
Handles decisions when receiving properties from bankrupt players	49%
Handles mortgaging properties	51%
Handles selling houses	53%

Table 5: Combining aspects of my bot and the best TD AI player

Decision TD AI handles	Wins vs bot
Handles mortgage decisions	48%
Handles house buying decisions	47%
Handles buying back mortgaged property decisions	33%
Handles deal making decisions	59%

Table 6: Combining aspects of my bot and the best TD AI player, standard and in jail turns.

7.11 Pre and Post bug performance

For a while during my development I had a bug which caused the AI to be unable to buy houses on monopolies after the first game it played. As the AIs were usually left to learn for 5,000 games before stopping this ability was never learnt in the earlier AIs. What is most interesting is that even when the bug was fixed the AIs still didnt learn the importance of holding a monopoly and their learning actually suffered as shown by 13. What is even more interesting was the AIs performed worse in all cases. I believe this is because the game has become more complex and attempting to learn this just causes the AI more confusion. Generally, the AI also took longer to peak in performance. The 80 hidden node AI reached its rating after only 1,000 runs pre bug fix while it took 15,000 runs post bug fix. This result helps support my theory that the AI is learning the more simple game at a faster pace.

8 Conclusion and evaluation

8.1 Conclusion

During this project I have shown that a Neural Network trained using TD($/lambda$) methodology produces a capable AI. The AI is able to perform to a good level, consistently able to beat human players until they discover how to exploit its

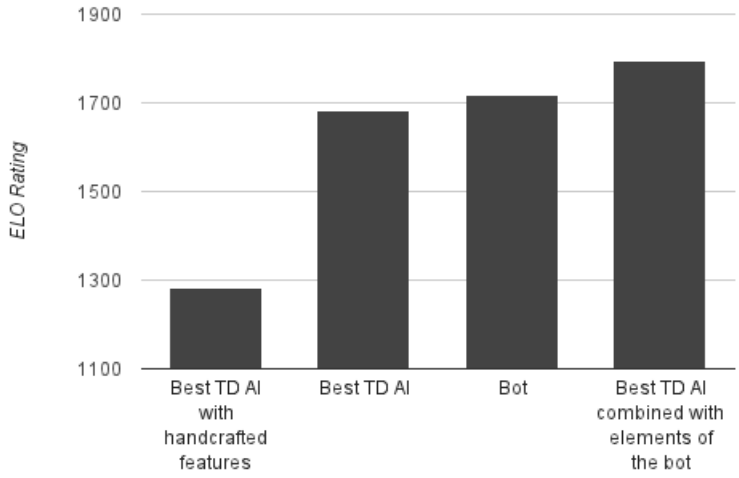

Figure 12: ELO ratings of AIs with additional features

weaknesses. These weaknesses do show some large gaps in the AIs performance. The main deficiency is the AIs inability to recognise the value of holding a Monopoly. Despite realising this early in the process all of the different methods I tried were unable to rectify this. The AI followed a similar pattern in most cases, continuing to learn about the game up to a point where the network would become overtrained and the performance would deteriorate. This suggests the AI is not able to work out the more complex elements of the game before getting too confused. Further evidence for this is supplied by the lacklustre results obtained from the AI learning from a database produced by the bots I designed. This produced extremely poor results suggesting the AI wasnt able to work out what the bot was doing.

8.2 Evaluation

I believe the game of Monopoly is quite a difficult case to train an AI to understand. One of the largest problems is the number of different ways to win. A player can win using a range of different strategies and some luck of the dice roll. This can result in a lot of noise as often games can be won by just holding onto properties while others may be won by holding a monopoly. It may difficult for the network to work out this strategy. During self play the winner is almost always the AI which had the most assets at the round limit which helps to reinforce the AIs strategy that there is no need to try and bankrupt other players by gaining monopolies. Even with this knowledge I wasnt able to manipulate

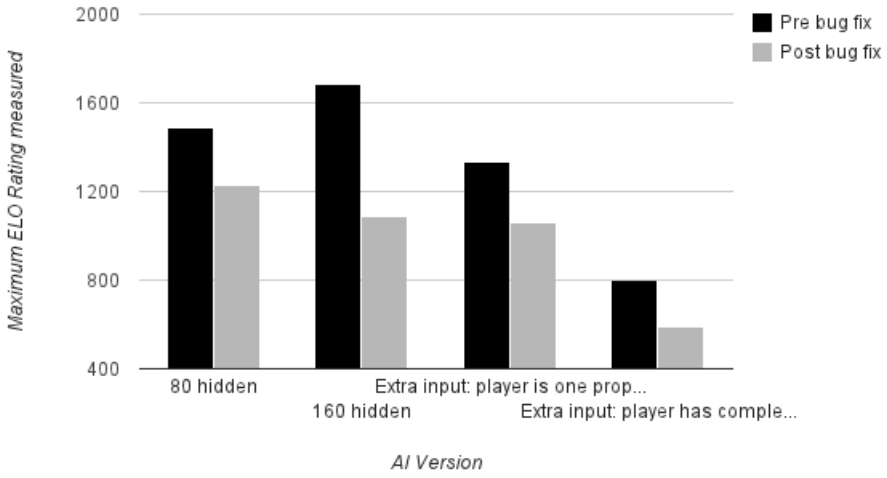

Figure 13: AIs best ELO score pre and post bug fix

the AI through any other strategy to prevent this from happening. This sort of behaviour highlights one of the weaknesses of self play, getting stuck in a strategy and being unable to deviate as even if the AI learnt the importance of holding a Monopoly, as all of the AIs would learn it this would no longer increase an AIs chance of winning any more than the previously held strategy.

8.3 Objectives

Through this project I have been able to show that although it may be possible to produce an AI which plays Monopoly to a high standard using RF learning it is not a simple case and requires a significant amount of work. Unfortunately, Monopoly just doesnt seem as well suited for the application of RF learning as other board games have been.

From undertaking this project I have learnt much about neural networks and the TD(λ) algorithm. I have a much better understanding of how to manipulate inputs and the environment to allow for better learning, and the potential applications of these methods. Dealing with bugs has also proved my programming skills further and I feel I am better prepared

9 Future work

Due to the time limit of my project and the problems I ran into I was unable to fully test everything I would have liked to. Potential changes could include

introducing a more in depth search to certain elements of the game. The AI could perform better if it searched through deals at a greater depth? There are numerous other different inputs and variable combinations which could have produced better results. Many of these are unlikely to produce a significantly better AI. The key to doing this would be to introduce inputs which would help fix the AIs deficiencies, namely forcing it to learn the value of holding a Monopoly and to give it more of an understanding of money in a number of situations. The key input types to be tested would be the one with 4 extra inputs for each property group which I was unable to test due to the huge amount of time learning took. It would also be beneficial to run any of the other input types for a longer period as this could produce better results.

Another aspect of the final program worth mentioning was the effect of a four player game. Most literature on RF Learning suggests keeping win rewards between 0 and 1. I believe that this is true for 2 player games but when a game involves 4 players a different approach should be taken. I found my AIs valuations were all below 0.5 in the Monopoly query tests and were all fairly close. Because an AIs decisions is based on relative valuations changing this wont suddenly make the AI much more successful but would hopefully allow it to better distinguish between states as they are further apart. This effect is caused because whenever 1 player wins and receives a win reward of 1, the other players receive a win reward of 0, resulting in an average payoff of 0.25 to the network. This causes the valuation of the average board state to move towards this value. Potential fixes could be assigning a player a value based on which position they came in, giving the winner a higher payoff, or giving player payoffs based on how many assets they had at the end of the game.

If the main objective of further research is to design a better Monopoly AI there is some evidence that combining some features of the TD AI and Monopoly AI produces a better AI. As this wasnt the point of my project I didnt want to invest much time pursuing this so I am sure much more could be achieved pursuing this. Given my own traditional AI wasnt highly optimised either it isnt proven that the method of combining the two would be able to exceed the abilities of a fully optimised traditional AI.

Genetic algorithms have recently been used to produce promising checkers players. Chellapilla and Fogel managed to produce an AI which was able to beat expert level checkers players and even draw with a master [6].This could applications to Monopoly as it prevents the stagnation I noticed in self play RF learning where all of the AIs are playing the same strategy by having a number always playing a different strategy to the others. A similar idea could still be implemented with TD($lambda)$methodspossiblystartingtheAIplayingagainstmoresimplebotsbeforeeventuallyhav

A different structure for Neural Networks was suggested by Boyan in a paper covering modular neural networks [5]. An application using Modular Neural Networks uses a different neural networks for different parts of the game which can be very useful in games where the different aspects are very different.The author found that modular neural networks was more successful than a single (monolithic) neural network in backgammon and was able to provide context sensitivity to the evaluation function but they often end up lacking the deep

understanding of structural features which a single (monolithic) neural network is capable of [5]. This could have applications in Monopoly in a number of different ways. For example, different networks could handle different stages of the game where deal making becomes more important or different neural networks could handle the individual decisions open to the player. I would hope this would allow the AI to make smarter deals and have a better understanding of money in the appropriate situations.

References

[1] Amazon.co.uk. Monopoly (xbox 360), 2013. [Online; accessed 28-June-2013].

[2] Touch Arcade. Monopoly here & now - does the ai cheat?, 2009. [Online; accessed 28-June-2013].

[3] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Knightcap: a chess program that learns by combining td (lambda) with game-tree search. *arXiv preprint cs/9901002*, 1999.

[4] Michael J Berry and Gordon S Linoff. *Data mining techniques: for marketing, sales, and customer relationship management*. Wiley. com, 2004.

[5] Justin A Boyan. *Modular neural networks for learning context-dependent game strategies*. PhD thesis, Citeseer, 1992.

[6] Kumar Chellapilla and David B Fogel. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, 87(9):1471–1496, 1999.

[7] T Collins. Probabilities in the game of monopoly, January 2005. [accessed 25-February-2013].

[8] P Ding and T Mao. *Reinforcement Learning in Tic-Tac-Toe Game and Its Similar Variations*. PhD thesis, Thayer School of Engineering at Dartmouth College, 2009.

[9] Kurt Driessens and Sašo Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.

[10] Eric J Friedman, Shane G Henderson, Thomas Byuen, and Germán Gutiérrez Gallardo. Estimating the probability that the game of monopoly never ends. In *Winter Simulation Conference*, pages 380–391. Winter Simulation Conference, 2009.

[11] Johannes Fürnkranz. Machine learning in games: A survey. *Machines that learn to Play Games*, pages 11–59, 2001.

[12] Imran Ghory. Reinforcement learning in board games. *Department of Computer Science, University of Bristol, Tech. Rep*, 2004.

[13] R Humphries. World monopoly expert phil orbanes tells rusty about the new tokens and winning game strategy., January 2013. [Online; accessed 28-June-2013].

[14] Pradu Kannan. Elo table, November 2008. [Online; accessed 01-June-2013].

[15] Irwin King. *Application of Temporal Difference Learning and Supervised Learning in the Game of Go*. PhD thesis, Citeseer, 1996.

[16] Anton V Leouski and Paul E Utgoff. What a neural network can learn about othello. *University of Massachusetts, Amherst, MA, Tech. Rep*, pages 96–10, 1996.

[17] Clinton Brett Mclean. *Design, evaluation and comparison of evolution and reinforcement learning models*. PhD thesis, Rhodes University, 2002.

[18] Jordan B Pollack and Alan D Blair. Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32(3):225–240, 1998.

[19] G Roush. Monopoly here & now iphone review, December 2008. [Online; accessed 29-June-2013].

[20] Nicol N Schraudolph, Peter Dayan, and Terrence J Sejnowski. Learning to evaluate go positions via temporal difference methods. In *Computational Intelligence in Games*, pages 77–98. Springer, 2001.

[21] I Stewart. Mathematical recreations - monopoly revisited. pages 116–119. Nature Publishing Group, 1996.

[22] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[23] Google Play Store. Monopoly reviews, 2013. [Online; accessed 29-June-2013].

[24] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[25] Gerald Tesauro. *Practical issues in temporal difference learning*. Springer, 1992.

[26] Gerald Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1):181–199, 2002.

[27] K Tostado. Under the boardwalk: The monopoly story. [Film], 2011.