

MSC THESIS · 2015

# Profitable Algorithmic Trading using Deep and Wide Reinforcement Learning

Retyped edition · enhanced scan figures

# Abstract

Since the first formal exchanges were established by the Dutch in 1611 [35] traders and speculators have tried to develop systems to profit from them. Profits can be made if a trader can correctly decide a good time to buy a commodity so he can subsequently sell that commodity at a higher price. In this thesis we successfully automated this decision making process using deep and wide reinforcement learning. We developed and trained reinforcement learning agents that made over £2,000 when deployed on the Betfair betting exchange to trade real money.

This thesis makes three main contributions; (i) We provide detailed new analysis on combining deep and wide artificial neural networks with reinforcements learning, (ii) We show that the same learning framework is successful in two very different environments and (iii) For the first time, to our knowledge, we validate academic research into trading markets with positive real world results.

A framework for using feedforward artificial neural networks with reinforcement learning was successfully built and tested in a grid world environment. We found that using both deeper and wider neural networks increases the decision making performance of the agents. The same framework was used to develop agents to trade profitably on historical greyhound and horse races. The agents were provided with information about the current prices and volumes available on Betfair. Using this information the agents were able to learn by trial and error profitable trading strategies autonomously. It was discovered that increasing the width of the neural networks increased profits whereas increasing the network depth did not add to profitability on the races tested.

Having successfully developed profitable agents validated on out of sample historical race data we deployed the best agents to trade on Betfair with real money. The agents made over £2,000 but only made about 15% of the profits suggested by the results validated using historical race data. This shows how important real world testing is to validate academic research in this field. It brings into doubt the potential profitability of the previous academic research on trading markets that has only been validated using historical data.

# Acknowledgements

I would like to thank my supervisor, Dr. Andrew Philippides, for all the hours he has spent advising me on this thesis. He has provided me with many insights into how to improve the research.

I would also like to thank my wife, and best friend, Rachelle. She has supported me throughout the project reading and correcting drafts. She has also kept me alive by bringing countless meals to my desk while I have been waiting for computer code to execute.

---

*Retyped from a scanned copy of the original 2015 MSc thesis; figures were cropped and image-enhanced (denoised, contrast-corrected, upscaled) from the scan, all other content transcribed verbatim. One physical page-ordering artifact in the source scan (pages 6/7) was corrected during transcription; a small number of the original's own typos are preserved as printed and are not transcription errors.*

# Table of Contents

---

## **1. Introduction**

- 1.1. Aims and Objectives
- 1.2. Overview

## **2. Background**

- 2.1. Introduction to Trading on Betfair
- 2.2. Relevant Reinforcement Learning literature
- 2.3. Relevant Machine and Reinforcement Learning for Trading Literature

## **3. Introduction to Reinforcement Learning**

- 3.1. RL Overview
- 3.2. Small Tabular Grid World Example
- 3.3. Larger Tabular Grid World Example

## **4. Results 1 : RL Using Deep and Wide ANNs for Grid World**

- 4.1. Introducing a Feedforward ANN Learning by Backpropagation
- 4.2. Learning Pre-calculated Q Values with an ANN
- 4.3. Learning Q values with ANNs Directly Through Interaction with the Environment
- 4.4. Conclusions

## **5. Results 2 : RL using Deep and Wide ANNs for Autonomous Trading**

- 5.1. Structure of the Automated Trading System
- 5.2. Developing Agents
- 5.3. Wider and Deeper ANNs
- 5.4. Conclusions

## **6. Results 3 : Real Money Results**

- 6.1. Agent Requirements
- 6.2. Agent Selection
- 6.3. Real Money Results
- 6.4. Summary and Conclusions

## **7. Results 4 : Extensions**

- 7.1. Adding Latency to the Model
- 7.2. Adapting for Different Types of Markets

## **8. Conclusions and Future Work**

## **Bibliography**

# 1. Introduction

---

Since the first exchanges were established by the Dutch in 1611 [35] traders and speculators have tried to develop systems to profit from them. Traders gather information about the market they wish to trade. They use this information to decide when to buy and when to sell a commodity. If they can buy a commodity at one price and sell it for a higher price at a later time they make a profit.

In the early years the information used by traders was based on rumours and inside information [35]. In the 18<sup>th</sup> century sophisticated charting systems were developed in Japan to help forecast the rice markets. In the early 20<sup>th</sup> century Richard Wyckoff wrote about his experiences trading using the information printed on the ticker tape [36]. The tape provided details of every share traded on the New York Stock Exchange in near real time. Using this information he decided when to buy and sell shares. In the 21<sup>st</sup> century high frequency traders use order book information to decide when to buy and sell stocks in a fraction of a second to beat retail investors to profitable trades [37]. The technology available has changed but the basic principle of traders using information to make a decision as to when to trade has been unchanged for centuries.

We wish to remove the human element of this decision making process. This project uses reinforcement learning to automate the process of converting the information available traders into profitable actions as to when to trade. Reinforcement learning (RL) is a machine learning method where agents are trained to learn the best action to take to maximise long term rewards. We use the Q-learning [3] RL method in this project which allows agents to learn which actions to take through experience. There is no need to build a model of the system the agent operates in. One particular method is to combine artificial neural networks (ANNs) with Q-learning. This has the potential to work well with trading markets where the reward of profit for opening a trade is only realised when that trade is closed at an unknown time the future.

We used the Betfair exchange to train and test the trading agents we developed. There are many advantages of using a betting exchange rather than a financial markets for initial testing in this field. Betting exchanges allow very small capital outlays, costs are fixed on profit per market rather than a fixed cost per transaction and there are hundreds of markets a week for testing. This allowed us to train agents using historical data and test them using real money in real market conditions.

## 1.1. Aims and Objectives

The overall objective of this project is to develop a profitable trading system for Betfair that requires no human involvement in the decision making process as to when to trade. Once the structure the agent operates in is set up the agent should be able to adapt to trade profitability in different market conditions and in different types of the markets.

The first goal was develop and test the Q-learning RL algorithm using ANNs. In order to do this we constructed a grid world environment to perform our analysis. Although there have been many examples of using ANNs with RL in the literature (e.g. 5, 6, 7, 10, 13, 22) we could not find any detailed evaluation of how and why they work. We look at the mechanics of how ANNs perform using Q-learning. We test how using wider and deeper ANNs effect the performance of different Q-learning agents.



The second goal is to use the knowledge gained from testing in the grid world environment to develop a profitable trading system using historical Betfair data. We want to use expert knowledge of the betting market domain to construct agents that can autonomously learn the best time to trade profitably using ANNs trained with Q-learning. Again, we test the effects of using deeper and wider ANNs. We wish to prove that this is possible using validation races on historical markets.

The final goal is to test the trained agents in real market conditions with real money. We could not find any papers in the literature where seemingly successful out of sample results were actually validated in the real world. Much of the research into artificial intelligence and reinforcement learning has been on playing games and toy problems. We want to demonstrate that the field can be used commercially for monetary gain.

## 1.2. Overview

The remainder of this project is split into five Chapters; Background, Introduction to RL, Results 1 RL in a grid world environment, Results 2 RL for autonomous trading, Results 3 real money profits, Results 4 Extensions and finally the Conclusions.

In Chapter 2, we start by giving an overview as to how the Betfair betting exchange works. We then look at recent work using machine learning in general for the trading problem and the limited papers that use reinforcement learning to help with trading decisions. In Chapter 3, we demonstrate how reinforcement learning works. We introduce the Q-learning algorithm which is used in this project and demonstrate it using a simple table to store Q-values. We use a one dimensional grid world example to explain the basic theory. This is followed by a more complex two dimensional example which is used with ANNs in Chapter 4.

In Chapter 4, the results of using ANNs trained using Q-learning for the two dimensional grid world example are shown. We briefly introduce ANNs and how they can be trained using backpropagation. A method for combining Q-learning with ANNs is explained. We examine how using deeper and wider ANNs effect the ability of Q-learning agents to learn. Results are presented for agents that learn from stored Q-values. These agents are then compared to agents that learn directly by interacting with their environment.

Chapter 5 presents the results for training agents that learn to trade autonomously. It contains the details of the methods used to develop the profitable trading agents. Details of how the model of Betfair is constructed and the algorithms developed are given. We show agents trained using both tabular and ANN Q-learning. Different widths and depths of neural networks are tested and show different levels of profitability.

Chapter 6 contains the results of real money testing on the Betfair exchange using automated trading. The results contain two different sets of results for agents trained using a simple and a more complex ANN structure. The methods to select best agents developed in Chapter 5 are shown. The practicalities of trading using autonomous agents are also discussed in the context of the real world results.

In Chapter 7 we extend the work. Following lower real profits in the real world tested than those seen in historical validation races we look at the effect of latency on the agents. We present results showing agents that learn with latency built into their environment. We also give results of using the same agent structures to learn to trade in different types of Betfair markets. This demonstrates the adaptability of the agents to different markets.

The final chapter contains our conclusions and limitations of the current work. We then make suggestions for future work in this field.

## 2. Background

---

This Chapter starts with a section on background information into how betting exchanges work. This is followed by a review of some of the relevant reinforcement learning and trading research in the literature. Readers unfamiliar with reinforcement learning should refer to Chapter 3 to clarify the terms used in the literature review if required.

### 2.1. Introduction to Trading on Betfair

Betfair is the world's leading betting exchange founded in 1999. It was designed to allow gamblers to bet directly against each other without the need for a bookmaker. If one gambler wished to bet on a selection to win his bet could be matched with another gambler wishing to bet on the same selection to lose. The gambler betting on the selection to win places "back" bets and is called a backer whereas the gambler betting on the selection to lose places "lay" bets and is called a layer. The price of each matched bet is called its "odds". The odds on Betfair are expressed in decimals.

An example of a matched bet is as follows:

A backer wishes to bet £100 on Manchester United to win at odds of 3.0. If Manchester United win the match the backer is returned £300. This represents the £100 bet on the team to win and a £200 profit. For this bet to work a layer opposing Manchester United has to be found. This gambler risks £200 if Manchester United win but gains £100 (the backer's stake) if Manchester United lose. Some early betting exchanges attempted to match backers and layers directly. The problem with this approach is that if a backer and a layer cannot be found at the same time the bet cannot be matched. Betfair decided to use a limit order book to match bets instead.

#### 2.1.1. Limit Order Book

Limit order books are used to bring buyers and sellers together directly in many stock markets around the world, for example, the NASDAQ and NYSE. The order book aggregates the price and amount of stock all the buyers and sellers are willing to transact at that point in time. An individual wishing to buy or sell a particular stock can examine the contents of the order book and decide to either to do a transaction at the best price currently on offer (a market order) or add their own offer to the order book for someone else to take (a limit order). A limit order runs the risk of not being matched if the market moves away from the price whereas a market order will always be matched.

The limit order book on Betfair is similar except that buyers are called backers, sellers are called layers and stocks are called selections. Figure 1 is an example of the price and volume information available to gamblers using the standard interface for a horse race.

15:10 Wincanton

Wed 25 Feb | 2m41 Hcap Chs

☐ Live Stream

☐ Radio

[Form & Results](#)

Win

Place

3 Places

AvB

More ▾

☒ ☒ Going In-Play ☐

Matched: **GBP 892,123**

☒ Betfair SP [?] ☒ Timeform

5 selections

101.7%

Back all

BSP

Lay all

99.9%

1 **Atlantic Roller**  
Sam Twiston-Davies

2.04  
£5330

2.06  
£6522

2.08  
£14635

SP

2.06

SP

2.1  
£3209

2.12  
£2313

2.14  
£1575

4 **Flaming Chарmer**  
Brendan Powell

5  
£963

5.1  
£1983

5.2  
£1520

SP

5.20

SP

5.3  
£1788

5.4  
£716

5.5  
£872

2 **Sun Wild Life**  
Daryl Jacob

3.9  
£1210

3.95  
£1331

4  
£3447

SP

4.20

SP

4.1  
£73

4.2  
£2007

4.3  
£6591

5 **Possibly Flora**  
Noel Fehily

14.5  
£174

15  
£571

15.5  
£556

SP

15.5

SP

16  
£505

16.5  
£712

17  
£199

3 **Moujik Borqet**  
Liam Treadwell

30  
£147

32  
£264

34  
£301

SP

31.1

SP

36  
£250

38  
£226

40  
£169

Figure 1 – Example of horse race win market 3 minutes before the race starts. The blue boxes contain the best odds on offer to back each horse with the amount of money available below. The pink boxes contain the best odds available to lay each horse.

Figure 2 shows the top three back and lay offers currently available for the race favourite called Atlantic Roller. A gambler wishing to back this horse with a stake of £10 can either take the best price of 2.08 or leave an offer in the order book for a layer to take. This offer could be at 2.1, 2.12, 2.14 or a higher price. The order then joins the back of the queue for the requested price. For example, if the offer was 2.1 the offer would be behind the current £3209 currently in the book then £3209 would have to either be matched by a layer or cancelled by the backer before the new bet was matched. If the backer wished to place a large £20,000 bet immediately he would have to accept a worse price for some of his bet because there is only £14,635 available at 2.08. The remaining £5,375 would be matched at 2.06 from the £6522 available.

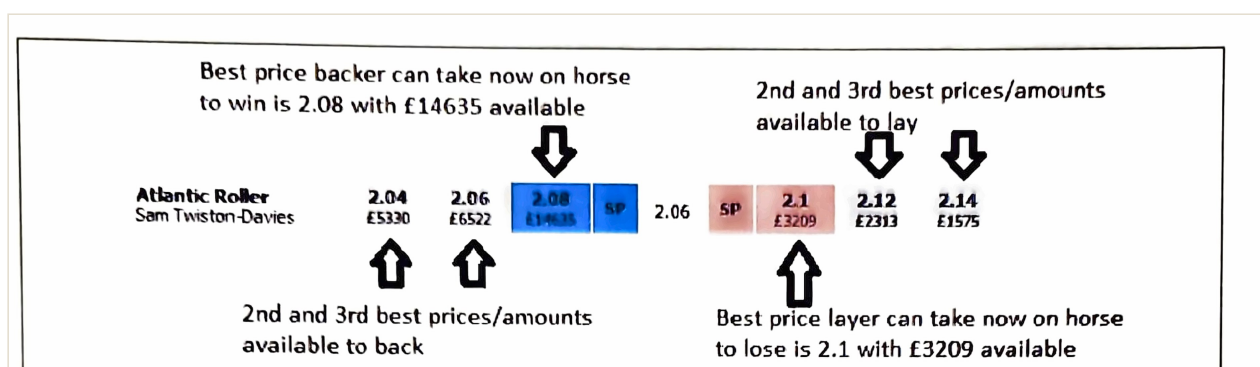


Figure 2 – Close up of the favourite for the race called 'Atlantic Roller'. Details of the top 3 back and lay price and volumes available at the current time.

### 2.1.2. Trading

The initial idea of betting exchanges was to bring together gamblers with opposing views as to outcome of an event and match their bets. The exchange itself collects a small commission from the winning gambler but,

unlike a bookmaker, has no risk of asymmetric information with the gambler. This meant that gamblers could get better odds than with the bookmakers. This in turn encouraged more and more volumes to the exchange.

The high volumes in the exchange attracted a new trading style of gambler betting on potential price movements rather than the actual outcome of the event. If a selection can be backed at one time step and layed at another time step for similar stakes than a guaranteed profit can be locked in regardless of the result of the event being bet on.

In Figure 3 an example of a profitable trade is demonstrated. The trader thinks that the price for a selection is currently too big so he backs the horse at 7.0 for a stake of £100 (Fig 3.1). The selection is then backed heavily by other gamblers and the price moves into 6.0. At this point the trader places a £100 lay bet. This cancels out the back bet's stake (Fig 3 2). The liability for the lay bet is only £500  $((6.0-1)*£100)$  whereas the potential winnings from the back bet is £600. This leaves a potential profit of £100 if the selection wins.

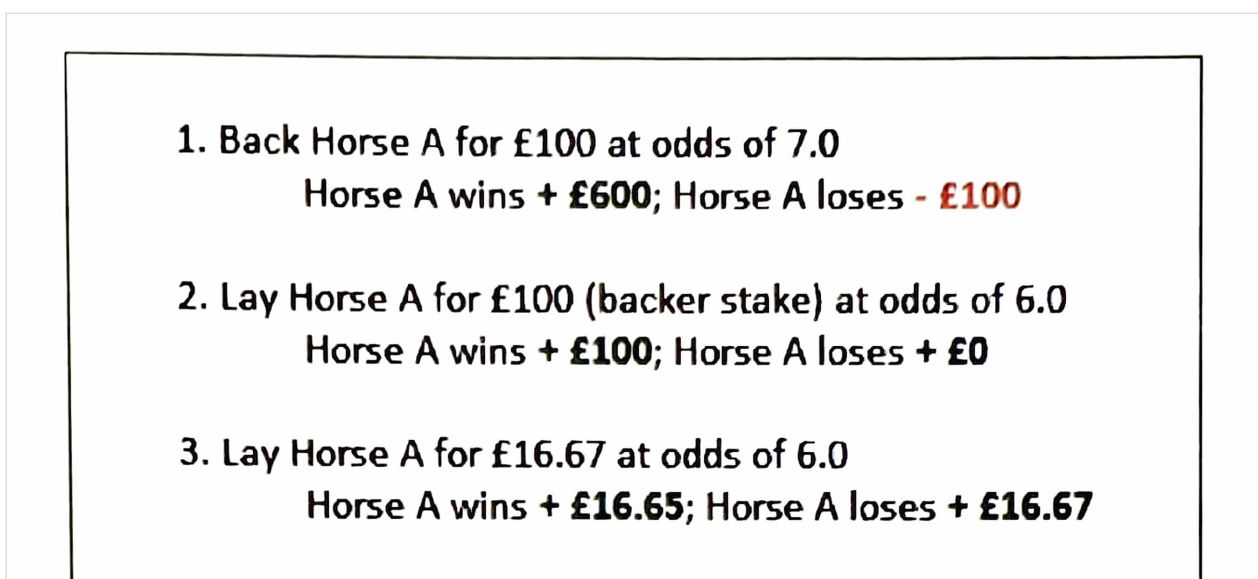


Figure 3 – The 3 stages of a profitable trade that results in a fixed profit regardless of the outcome of the event.

Having locked in a risk free bet on Horse A to win the trader now has the choice of stopping there or equalling the profit out regardless of who wins. This is done by laying Horse A again at 6.0 for a stake of £16.67 (Figure 3 part 3). This is an example of a winning trade but if the odds of the horse increased after an initial back bet a losing trade would be locked in.

### ***2.1.3. Potential Information for Traders***

A trader makes his money by correctly betting on the size and direction of price movements. One method of doing this is to use fundamental analysis to forecast the chance of each horse winning using form, jockey and trainer information. When the odds in the market differ from the forecast bets could be made hoping that the market will adjust for this information before the race starts. A second option is to use the price/volume information contained in the order book and its history to predict price movements.

The Betfair order book is updated at least every 200 milliseconds to take account new bets added to the market. The amount of money available to back and lay as well as the total amount traded is updated for each

selection at each price. Figure 4 shows a snapshot of the odds table for one selection for the prices close to the top prices available. There are 350 potential prices for each selection.

| Price | Traded and Available |          |
|-------|----------------------|----------|
|       | To back              | To lay   |
| 1.91  | £798                 | £2,134   |
| 1.92  | £1,953               | £10,262  |
| 1.93  | £2,559               | £32,771  |
| 1.94  | £2,404               | £27,480  |
| 1.95  | £1,945               | £28,716  |
| 1.96  | £1,457               | £20,739  |
| 1.97  | £856                 | £7,249   |
| 1.98  | £1,545               | £5,253   |
| 1.99  | £1,226               | £8,318   |
| 2.00  | £5,158               | £26,302  |
| 2.02  | £6,532               | £41,699  |
| 2.04  | £7,252               | £67,568  |
| 2.06  | £2,405               | £120,434 |
| 2.08  |                      | £2,538   |
| 2.1   |                      | £4,031   |
| 2.12  |                      | £5,416   |
| 2.14  |                      | £3,851   |
| 2.16  |                      | £836     |
| 2.18  |                      | £405     |
| 2.2   |                      | £1,996   |
| 2.22  |                      | £751     |
| 2.24  |                      | £633     |
| 2.26  |                      | £679     |

*Figure 4 – Odds table for a single selection.*

Figure 4 – Odds table for a single selection.

Potentially a trader could store this information each time step and track historic price changes, volumes matched and volumes available. This information could be added to the current state of the order book in order to make trading decisions. On the regular interface Betfair provide simple historical price volume charts for each selection in each market such as the one in Figure 5.



Figure 5 – Price volume chart since the start of the market. The red line is the matched price and the grey bars represent the volume traded. The x-axis shows each bet made through time.

Products such as 'Bet Angel' [27] and 'Geeks Toy' [28] provide traders with much more sophisticated visualisation and trading tools than those available on the Betfair website. Betfair also provide an application programming interface (API) [29] that allows users to directly read market information and place bets. Traders could potentially represent this information in an infinite amount of individual ways to help them trade profitably.

## 2.2. Relevant Reinforcement Learning literature

In this section we focus on three examples from the literature where ANNs have been combined with RL. We use these examples as they provide a brief overview of how ANNs have been used with RL over the last two decades. The first is Tesauro's [5-7] backgammon agents that were developed in the 1990s. These agents use simple feed forward ANNs. The second is was to control a slot car [30]. This utilised a deep auto-encoder. The third is Deepmind's use of convolutional ANNs to learn to play Atari games [13]. The three papers show an increasing trend to reduce the level of human involvement in the creation and training of RL agents.

The most well-known successful application of reinforcement learning are the series of backgammon agents developed by Tesauro [5-7]. A representation of the current board position was fed into a feed forward ANN with a single hidden layer. This state representation was then trained using backpropagation and reinforcement learning. The game of backgammon has hundreds of thousands of different potential board



positions. It also has up to 20 potential moves that the player can make from any position. This made the use of an ANN necessary because there were too many potential states of the game to store in a table.

Tesauro built several increasingly successful models. The initial model achieved a basic standard of play with a simple representation of board that incorporated no backgammon knowledge. It contained 198 inputs with just 40 hidden units in the hidden layer of the ANN. When expert knowledge was used to design the inputs, and the number of hidden units was increased to 160, master-level play was achieved [5]. Professional players have changed the way they play in the opening moves of a game after watching Tesauro's agent play.

In [30] reinforcement learning was successfully used to control a slot car to human level from raw camera images overlooking the track. The raw pixels from the camera were inputted into a deep auto-encoder [20]. This worked as unsupervised learning method to reduce the dimensionality of the inputs. A potential problem with using a deep auto-encoder is that the reduced dimensions of the inputs produced may leave out vital information required by the agent to learn to take the best actions [14].

A recent success is Deepmind's use of deep reinforcement learning to play Atari games [13]. The agent had access to the Atari emulator's screen images as pixels as the agent plays a game. These are fed in a convolutional neural network [20] to give a representation of the current game state. The network was then trained to learn the best action to make for any given image (or series of images) learning the delayed rewards of the game score.

Using the same networks and parameter settings the agent was able to beat all previous algorithms for a set of 49 games. No human intervention was required for the selection of features from the images. A great amount of work must have been done to determine the best parameter settings, setting up a network structure and deciding how to present the images to the network. It may be found that by hand-crafting the inputs, for example changing the image resolution or using changes in pixels between images, the learning speeds and game scores improve.

The three examples above all occur in relatively simple closed environments. When the state representation inputs consist of only a set of images and the task is simple end to end reinforcement learning, when seeing a picture leads to sensible actions being taken, is possible [13]. In more complex tasks, such as helicopter control [32] and trading [10], significant engineering of input features and knowledge of the task domain is still required to make the best use of reinforcement learning. Tesauro's work shows that adding expert knowledge can greatly increase agent performance.

### **2.3. Relevant Machine and Reinforcement Learning for Trading Literature**

There has been a huge amount of both academic and commercial research into forecasting the financial markets due to its potential for financial gain. In 1970 Fama's efficient market hypothesis (EMH) [33] stated that markets were efficient because their current price includes all currently available information. Academic studies into financial markets (e.g. [10] for the Brazilian stock market) and betting markets (e.g. [34] for UK betting markets) have demonstrated potentially profitable strategies that disprove the EMH. Although most of the papers show theoretical profit on out of sample data, to our knowledge, none of them demonstrate real money profits through real trading.



A summary of machine learning methods used to forecast the stock market was presented in [18]. Tools used include linear regression, artificial neural networks (ANNs), genetic algorithms, support vector machines (SVMs) and case-based reasoning. We look at four papers where machine learning has been used to successfully find profits in out of sample trading data. They provide a brief snap shot of the current work in the field.

In [10] an ANN was trained to forecast the high and low price for a stock the following day. The authors handpicked the inputs to include historical price information about the stock. Trading rules, decided by the authors, were set to buy the stock if it fell below the forecast minimum price and sell it if it went above the forecast maximum price. Using this strategy theoretically would have doubled investors' capital. In [19] an ANN model using a genetic algorithm to optimise network structure produced a 23.3% annualised return on the foreign exchange (forex) markets. In [15] SVMs were used to predict short term price movements from the NASDAQ limit order book. Trading the up and down prediction resulted in profit on historical data.

In [16] restricted Boltzmann machines (RBMs) [20] were used to extract features from individual stocks' history and applied to basic momentum strategy using an ANN. The inclusion of the RBMs for feature extraction increased theoretical annualised returns from 10.53% to 45.93%. This result is similar to the success of the use of a deep auto-encoder in the slot car example [30]. It shows there is potential for deep learning methods to be used to help with the representation of financial market data.

The machine learning methods reviewed so far are based on supervised learning. A target price at an arbitrary time in future is forecasted from historical data. A hard coded trading system, where a trade is opened if a forecast is above or below price  $X$ , is then decided by the author to set trading rules using that forecast. Reinforcement learning provides a mechanism to directly learn the best action (to trade or not to trade) based on the current situation. Agents can develop strategies by trial and error using reinforcement learning algorithms rather than having to be hard coded.

Dempster et al wrote three papers [23-25] on using RL to develop strategies for trading in the forex markets. In [24, 25] forex order flow data, order book data and technical indicators are used as potential inputs to represent the current state for RL. An evolutionary algorithm was used to select the best inputs through various trials of running different inputs into the RL algorithm. The results showed that adding order book and order flow data to technical indicators increased profitability of the agent. The model was extended in [25] to include more data and resulted in profits on out of sample test data.

In [12] RL was used to develop agents to optimise trade execution. The goal of the agent was to buy or sell a fixed volume of a particular stock in a fixed time frame at the best possible price. The agent had access to the NASDAQ limit order book. The agent used this public information and its own private information of how much stock it needed to trade and the time it had left to from a tabular presentation of its current state. The Q-learning algorithm was then used to train the agents so they could learn when, how much and at what price to trade. The trained agents resulted in decreased average price per share achieved compared to other strategies on historical data.

In [22] multiple Q-learning agents were trained to learn a trading strategy for the Korean stock market. The agent was provided with a series of technical indicators about the historical price movements for a number of different stocks. These inputs were then feed into a feed forward neural network that was trained by

backpropagation to learn approximate future returns for different actions. The 'Q-Trading' agents trained by the system beat the market on historical data.

For the development of our own RL agents using ANNs we learn several important things from the literature. From Tesauro's backgammon work we learn that expert knowledge can be hugely beneficial for performance. Deep learning methods (used in [13, 16 and 30] have been useful but there is no evidence for RL that they beat systems designed by expert knowledge. In this project we seek not only to demonstrate that it is possible to use ANNs with RL to profitably trade but also to make as much profit as possible. For this reason as much expert knowledge as possible will be incorporated in selecting the information the RL agents receive.

### 3. Introduction to Reinforcement Learning

---

In this Chapter we introduce the structure and algorithms required for reinforcement learning (RL). We introduce the components and algorithms needed and then demonstrate them working in a simple grid world environment. The grid world environment is then made more complex and the use of neural networks for function approximation is examined.

The grid world environments make it much easier to introduce the modelling theory and practical methods that will be used for the trading systems in Chapter 5.

#### 3.1. RL Overview

In RL an agent learns by interacting with its environment. Unlike supervised learning there are no fixed targets for the algorithm to learn. Instead the agent tries to learn the best policy in order to maximise its long term future reward over the course of a complete task. This task might be playing a game where the future reward of winning or losing may only be known a long time in the future.

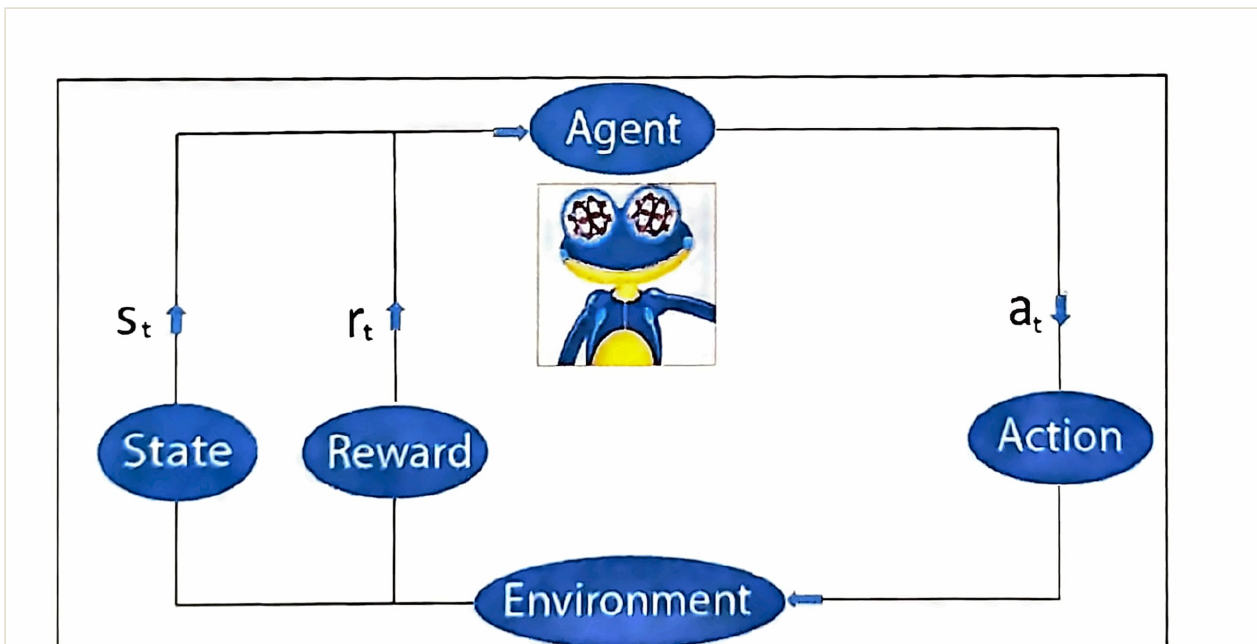


Figure 6 - Reinforcement Learning Setup. It depicts the relationships between the agent and its environment. At discrete time  $t$  the agent is presented with a representation of its environment  $s(t)$ . At time  $t$  the agent then chooses one action from a set of actions available to it. When the agent takes an action the environment then responds in the next time step by providing a reward ( $r(t+1)$ ) and a new state representation ( $s(t+1)$ ). The agent then assesses the new state and decides on a new action ( $a(t+1)$ ).

The key components of reinforcement learning algorithms are the state representation of the environment, the actions taken by the agent, the rewards received from the environment, and the update algorithm that links these together. We will describe each of these in detail before describing the Q Learning update algorithm used in this project.

### ***3.1.1. The State Representation of the Environment***

At discrete time  $t$  the agent is presented with a representation of its environment. This is unlikely to be a full description of the environment and is thus called partially observed. For a RL agent to perform well it is important that a quality state representation is provided as this is the only information that an agent can use to make an action decision.

The Markov assumption is required which assumes that the current state representation contains all relevant information about previous states and the state it is currently in. This is so that the agent can make decisions based on information only contained in its current state representation. History can be built into the current state representation to try to maintain the Markov assumption if required. In simple environments, such as finding the best path through a maze, the Markov assumption holds with no history because current position is all that matters. If the system was tracking a car the previous position would be required in order to know the future direction. In this case the position of the previous state could be added to the current state representation.

The number of possible states at each time step could be limited to a discrete number or it could be made up of a continuous set of variables with an infinite number of states. In order to cope with a large number of inputs a function approximation method is needed because the number of states grows exponentially with the number of inputs (see [1] for a detailed summary of the methods used). In this project we use neural networks as our function approximation method.

### ***3.1.2. The Actions Taken by the Agent***

At time  $t$  the agent chooses one action from a set of actions available to it. These could be anything such as where to move next, whether to buy a stock or shoot a gun. The goal of the agent is to learn which action to take at time  $t$  to maximise long term reward. For different algorithms this might mean initially exploring an environment randomly or attempting to build a model of the probability of the state changing given a particular action.

Generally a discrete number of actions are available. These can then be used by the agent to monitor the value of each action separately. Using continuous actions is much harder for a model to learn [see 2]. In the case of trading a continuous action would be an exact amount to bet whereas a discrete action would be just to open a bet or not open a bet.

### ***3.1.3. The Rewards Received from the Environment***

The goal of a RL algorithm is find a way of learning the best action to take at time  $t$  to maximise the total reward over the whole task. If the task is over a long time period, a discounting element can be added so that rewards in the future are worth slightly less than rewards right now.

It can be beneficial to introduce intermediate rewards into a system to help an agent learn. For instance, a chess playing agent might be rewarded for every piece taken rather than just when the game is won or lost in the distant future. This may encourage the model to learn faster but could also lead to inefficient non-optimal behaviour. As with state representation, choosing a suitable reward structure is vital to getting a RL system to work.

### 3.1.4. Q Learning Update Algorithm

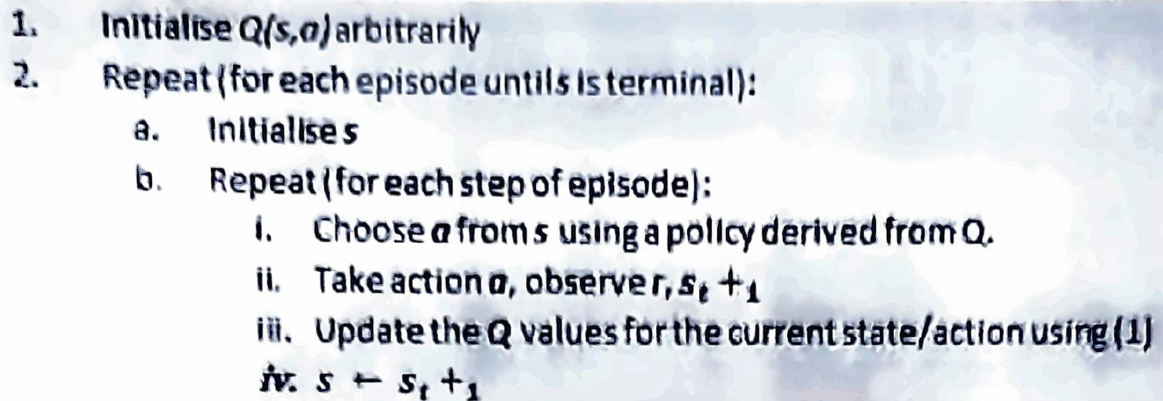
The Q Learning algorithm is a popular reinforcement learning algorithm introduced in [3]. It is a model free method of learning the best actions to take in any given state. A **Q** value is introduced as a measure of the future expected reward of taking a certain action when in a certain state at time  $t$ .

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

#### Equation 1 - Q Learning Update Function

The update algorithm above is made up of several parts. The Q value for the state and action at time  $t$  is  $Q(s_t, a_t)$ . The update starts with the original value and then adds a new value set at the learning rate  $\alpha$ . If  $\alpha$  is set at a low level (e.g. 0.01) then the updates are very slow. This can help with learning in very noisy environments. If  $\alpha$  is set high then updates are large and very little (or none if the rate is set at 1) of the previous Q value is retained.

The contents of the square brackets determine the update value.  $r_{t+1}$  is the immediate reward from the environment the next time step from taking action in the state of the current time step.  $\max_a Q(s_{t+1}, a)$  is the Q value of taking the action with the greatest value in the next time step after taking the action this time step. This is the part of update that allows future rewards to propagate back through time.  $\gamma$  is the discount of this future reward. If this is set to 1.0 future rewards are worth the same as immediate rewards. Finally the current  $Q(s_t, a_t)$  is subtracted from the new total discounted reward value [see 3.2 for a worked example of the update].

- 
1. Initialise  $Q(s,a)$  arbitrarily
  2. Repeat (for each episode until  $s$  is terminal):
    - a. Initialise  $s$
    - b. Repeat (for each step of episode):
      - i. Choose  $a$  from  $s$  using a policy derived from  $Q$ .
      - ii. Take action  $a$ , observe  $r, s_{t+1}$
      - iii. Update the  $Q$  values for the current state/action using (1)
      - iv.  $s \leftarrow s_{t+1}$

Algorithm 1 - Q Learning Algorithm adapted Sutton and Barto [4]

1. Initialise  $Q(s,a)$  arbitrarily
2. Repeat (for each episode until  $s$  is terminal):
  - a. Initialise  $s$
  - b. Repeat (for each step of episode):
    - i. Choose  $a$  from  $s$  using a policy derived from  $Q$ .
    - ii. Take action  $a$ , observe  $r, s_{t+1}$
    - iii. Update the  $Q$  values for the current state/action using (1)
    - iv.  $s \leftarrow s_{t+1}$

The learning algorithm is described in Algorithm 1. In line (1) the initial Q values for every state/action pair are set. These could be set to zero or a suitable value for the problem domain the agent is operating in. Line (2) then runs through a number of learning experiences for the agent. The initial state is set (2a).

From the initial state the agent then chooses an action to take (2bi). This action could be a random action or it could follow the action which currently has the best Q value for that state. In practice a combination of the two would be used. Random actions allow the agent to explore new environments and gain new experiences. This is particularly beneficial at the start of learning when the agent has little knowledge. Following the action with the best current Q value for the current state allows the agent to revisit rewarding states. This has the effect of propagating positive rewards back through time and reinforcing the best actions. This explore or exploit choice can make a big difference to how RL agents perform in real world environments.

This action is then taken (2bii) and the Q value for the current state/action pair is updated (2biii). The algorithm (2b) then loops and the agent takes another action. This continues until the agent is stopped. The stopping point can be a fixed number of steps or when the agent has reached a set reward level.

### 3.2. Small Tabular Grid World Example

We set up a simple one dimension grid world example, described in Figure 7, to explain how Q-learning works in practice.

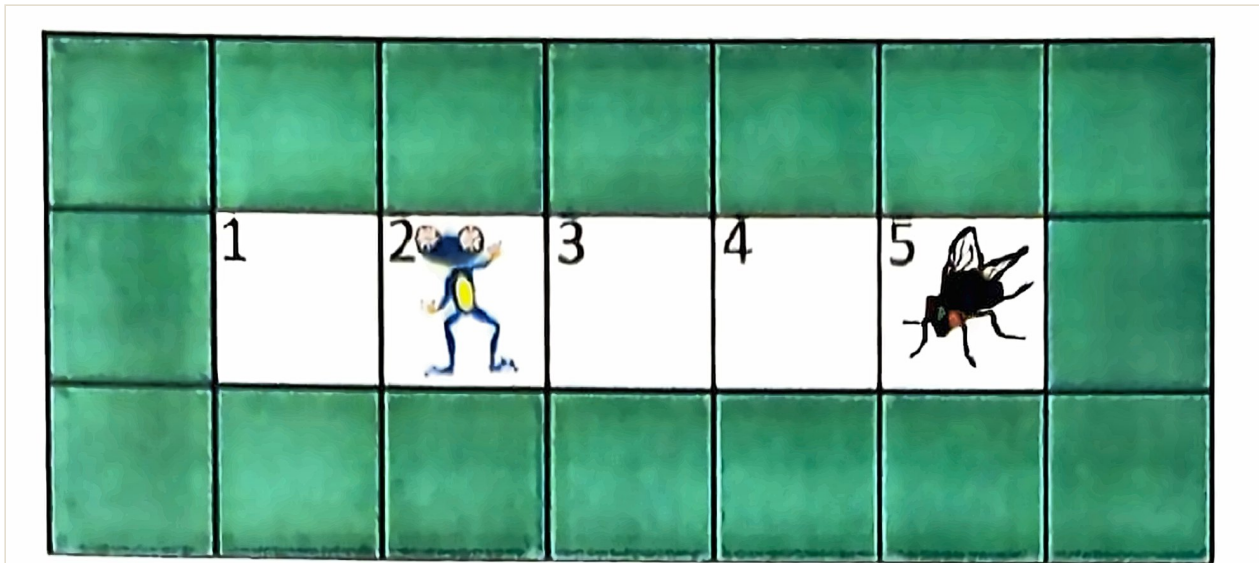


Figure 7 – Simple 1D grid world environment. There are five different states labelled 1-5. The agent starts in State 2. The agent can do two possible actions: move left or move right. If the agent moves to State 5 it receives a reward of 100 for catching the fly and returns to State 2. At this point a new fly is installed in State 5. Moving to any other state results in a zero reward.

The agent in Figure 7 can either explore the environment by taking random actions or exploit its current knowledge by take the action with the highest Q value in each state. In this simple example the learning rate is set to 1. This means that equation (1) above is reduced to:

$$Q(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \quad (2)$$

### Equation 2 - Q Learning updates with Learning Rate = 1

In this special case it means that the Q learning update for each state/action pair does not retain knowledge of its previous value. It is calculated as the reward for taking this action plus the discounted future Q value available for taking the action with the highest Q value in the next time step. With the discount rate set at 0.99 the Q learning algorithm was run for 200 random moves. The Q values for all state/action pairs were initially set at zero.

Figure 8 – Q Learning updates for learning rate = 1 with discount rate = 0.99.

| ACTION OPTIONS | Initialise Qs |       | Update 1 |       | Update 2 |       | Convergence |       |
|----------------|---------------|-------|----------|-------|----------|-------|-------------|-------|
|                | Left          | Right | Left     | Right | Left     | Right | Left        | Right |
| 1              | X             | 0     | X        | 0     | X        | 0     | X           | 97    |
| 2              | 0             | 0     | 0        | 0     | 0        | 0     | 96          | 98    |
| 3              | 0             | 0     | 0        | 0     | 0        | 99    | 97          | 99    |
| 4              | 0             | 0     | 0        | 100   | 0        | 100   | 98          | 100   |

Figure 8 shows how the Q values develop through time with these settings. With all Q values initially being set at zero the first update occurs when the agent is in State 4 and then moves to State 5. At this point the reward for this action of moving right is set to 100. The agent is then moved back to State 2. After some random movements the agent finds itself at State 3 and then moves right into State 4. There is no immediate reward for moving to State 4 but the maximum Q value for any action in State 4 is 100. This 100 is then multiplied by the discount factor of 0.99 leaving a Q Value for taking

action right in State 3 of 99. This is how the future rewards propagate through the states as the agent explores.

In this case after around 200 random movements the Q values stop changing as they have converged. For each state the Q value to move right is greater than the Q value to move left. This means that if the agent takes the action with the highest Q value from any state it starts in it will end up going directly to the fly. In this case the discount rate was set to 0.99. Had the discount rate been set to 1 every state/action Q value would have converged to 100. Without a discount rate there is no encouragement for the agent to catch the fly in the shortest number of moves possible. Setting a small negative reward for every move would also lead to an optimal route being formed.

### 3.3. Larger Tabular Grid World Example



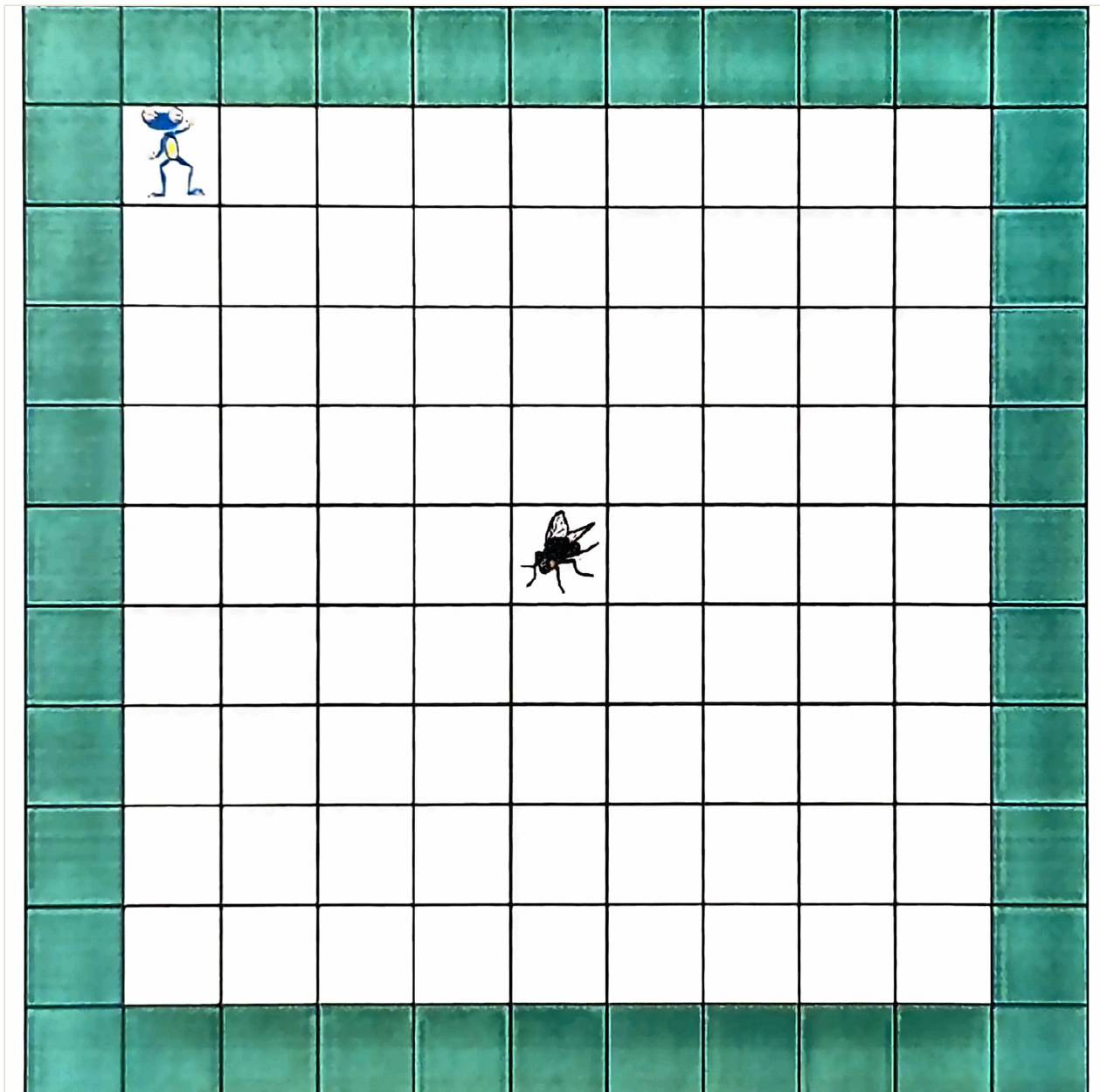


Figure 9 - 2D Grid world environment. The agent is placed at the top left of the grid. If it goes to the square in the middle and hits the fly it receives a reward and returns to the top left.

The agent in Figure 9 has the four available actions with the ability to move up and down as well as left and right. There are 80 possible non-fly states in the 9 by 9 grid. With four actions for each state that means there are 320 potential Q values for the agent to learn. The reward for catching the fly is set to 0.5 with a discount rate of 0.99 and a learning rate of 1.0.

If the agent is set to randomly move around the grid it takes about 200 moves before following the best Q values will lead the agent from the top left to the fly directly. This does not mean that the Q values for all 324 state/action combinations have converged. Some of the states may not even have been visited.

If the agent was placed at a different starting point following the best Q-Values after only 200 moves of exploring it is unlikely to lead the agent to the fly. If the agent's goal is to find the fly from any of the 80 non-



fly squares on the grid a different approach if needed. Now if the fly is not found after 30 random moves, the agent is reset to a random position on in the grid. This ensures that learning can start from anywhere.

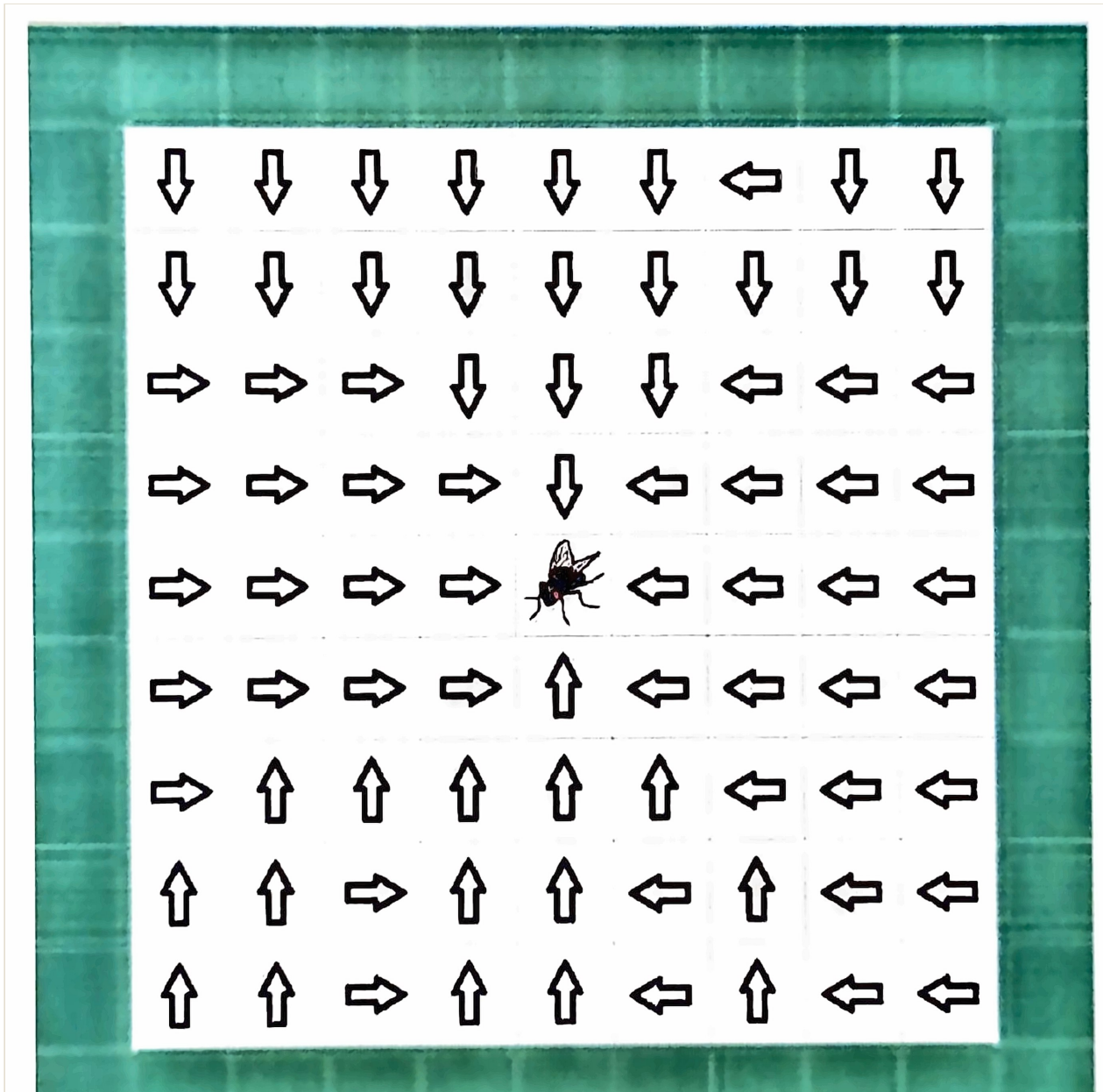


Figure 10 - The arrows show the direction of the action with the highest Q value in each square after 2500 random moves resetting after 30 moves without finding the fly.

Figure 10 shows that with a learning rate still set to 1.0 with the agent taking random actions and being randomly moved to a new state every 30 moves it does not find the fly leads to convergence. Starting at any of the 80 squares following the best action with the best Q value leads the agent to the fly. It took the agent 2500 moves to learn this solution.

Figure 11 depicts the Q-values that an agent has learnt for each of the actions for each position on the grid. The redder values represent a higher Q-value. The values in the chart have all been adjusted to be between zero and one. This will come in useful in 4.2 when a neural network attempts to learn these values.

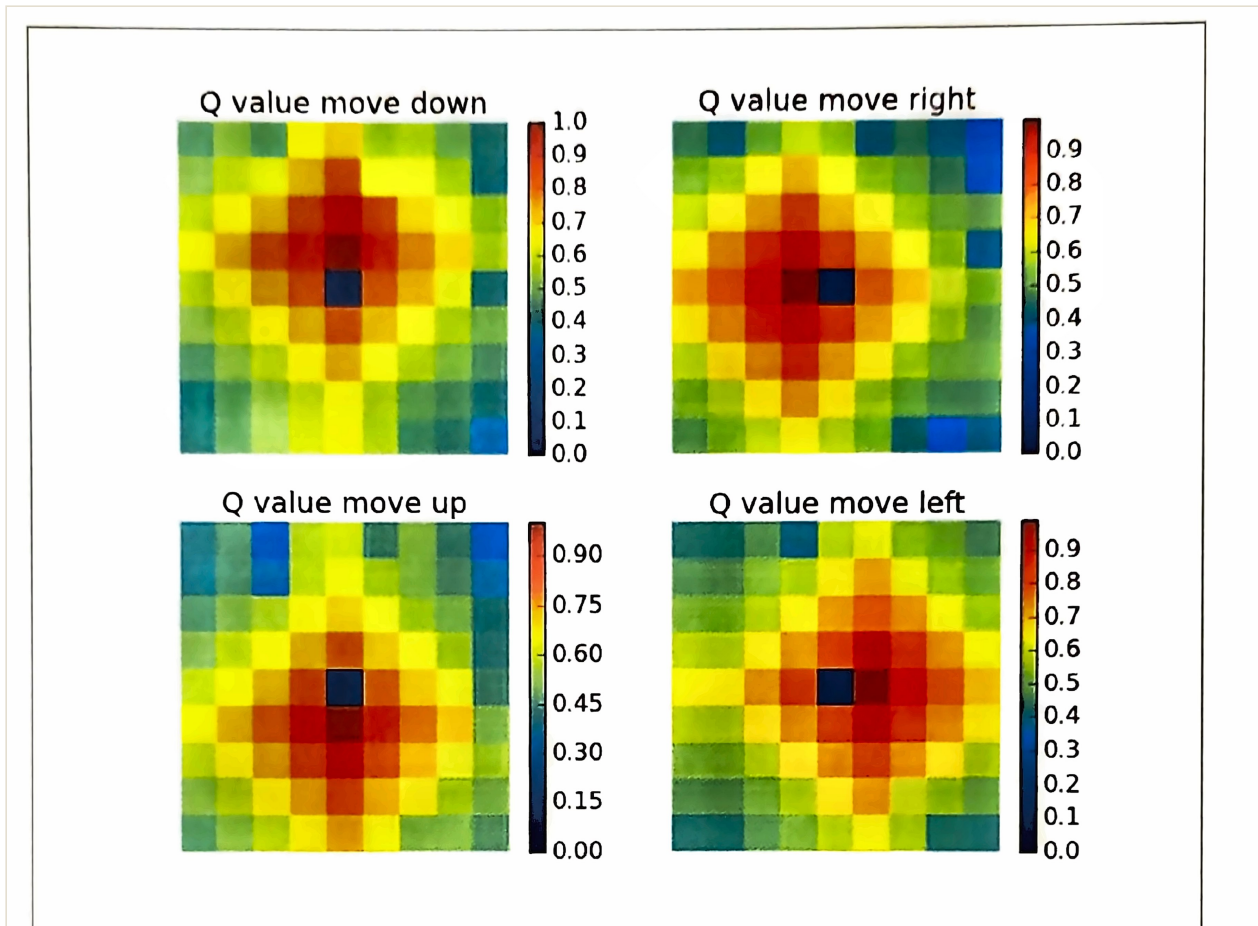


Figure 11 - The Q values at convergence using tabular Q-learning for each of the four potential actions. The dark blue square in the centre represents the fly. The agent selects the action with the highest Q value when deciding where to move next. In Section 3.5 four neural networks are used to try and approximate the functions contained in the four figures.

Setting the learning rate to 1.0 means that previous Q values for the current state/action pair are lost every update. This is not the fastest way to learn the optimal solution in most cases. Figure 12 shows the speed of convergence when different learning rates are used. In this case a learning rate of 0.15 or 0.4 reaches the 80 squares leading directly to the fly convergence fastest. The high (0.7 and 1.0) and the very low (0.01) learning rates converge more slowly.

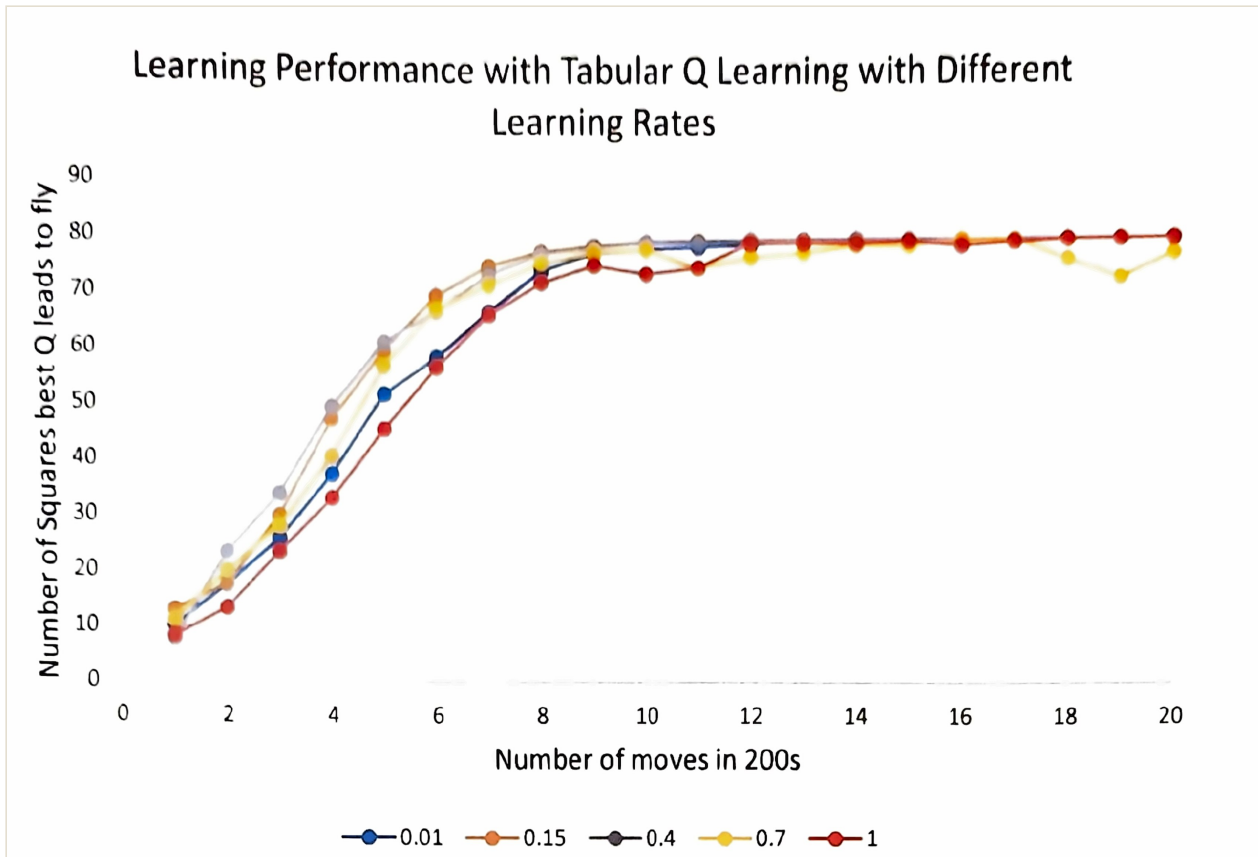


Figure 12 - Changing the learning rate. Each learning rate was trailed for 4000 moves repeated 10 times. The plots show the average performance of the ten samples for each learning rate. Standard deviations for the 10 samples has been not included for simpler viewing (We exclude it for similar graphs throughout the project for the same reason).

In all the tests above the agent explored by random action selection. If the problem becomes more complex it will become necessary to slowly decrease the randomness of the search and follow the actions with the best Q values more through the learning lifecycle.

## 4. Results 1 : RL Using Deep and Wide ANNs for Grid World

---

In this Chapter we demonstrate and analyse the use of wide and deep ANNs for RL using the grid world environment that was introduced in Chapter 3. Although there are a growing number of examples using ANNs with RL in the literature [e.g. 5, 6, 7, 10, 13] we could not find any rigorous analysis of the effects of using different ANN structures. The emphasis appears to be on publishing the final structures that are optimised for the particular task being examined.

We start by briefly introducing feed forward ANNs. This is followed by an analysis of using ANNs of different widths and depths to learn the Q-values learned by tabular Q-learning (Figure 11). We evaluate how successful different network structures are at learning four slightly different landscapes. We find that deeper ANNs perform better than wider ANNs for this task. This is followed by a description of how we incorporated ANNs to use Q-learning to learn by directly interacting with their environment. Again we find that adding layers to the network structure is superior to simply adding neurons to a single hidden layer.

### 4.1. Introducing a Feedforward ANN Learning by Backpropagation

In Chapter 3 we have stored the Q value of every state/action pair individually in a lookup table. The problem with this approach is that when the number of possible states and actions increase the number of individual pairs increase exponentially. Visiting all the states to learn individually the Q values becomes impossible. For example, in backgammon there are potentially of millions of states and actions. To overcome this Tesauro [5-7] used a single layer neural network to approximate the Q values for each state action pair. This meant that Q values could be estimated for unseen or rarely visited states from information learnt from visiting similar states.

Figure 13 depicts the components of a feed forward ANN [See 8 for introduction]. For the grid world problem the input layer contains the row and column numbers and a bias number (set to 1.0). Each of the inputs are then connected to a hidden layer. Each of the black connections contains a weight. So the top hidden unit has an input value of the different three inputs multiplied by the three different weights. This sum value is then fed into a transfer function to produce an output. The outputs from the hidden layer are then fed into the output layer and the process is repeated to produce output  $y$ . Figure 13 shows a two layer neural network with one hidden layer.

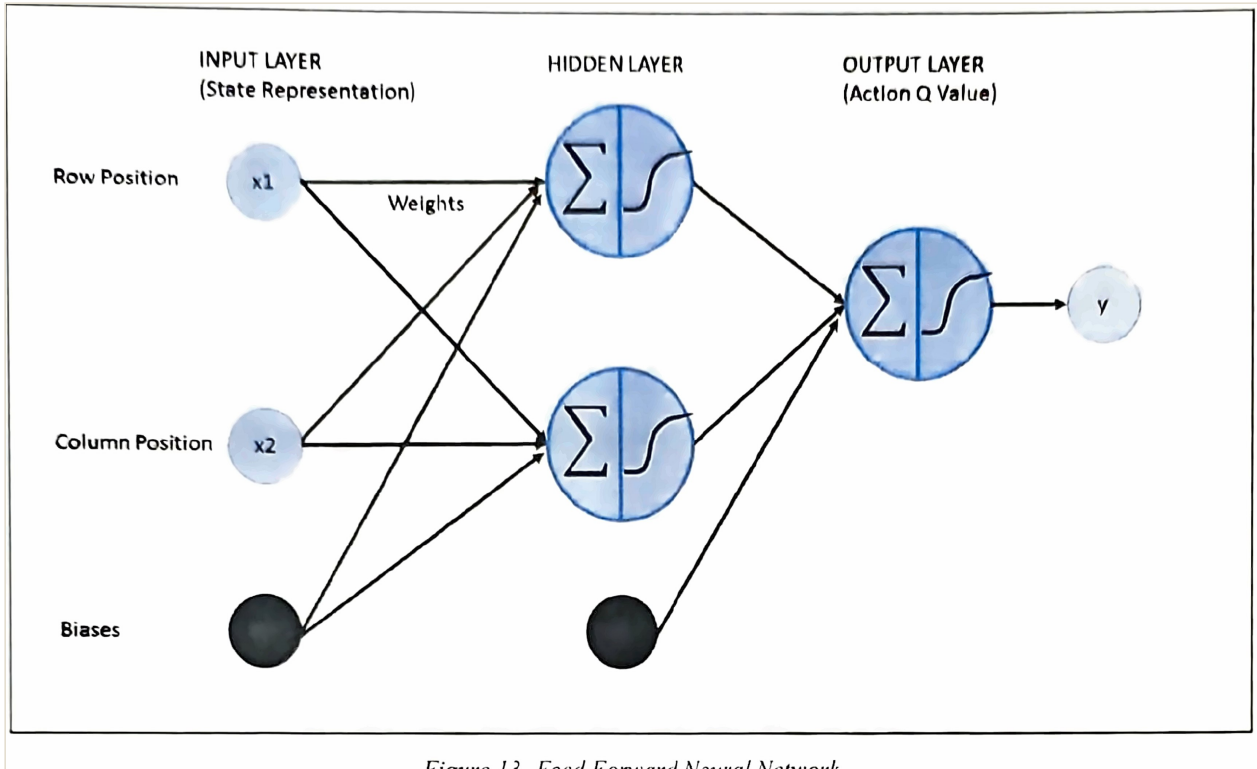


Figure 13 - Feed Forward Neural Network

Backpropagation is the method often used to train the weights for the network. It will be used extensively throughout this thesis. This method can be used with stochastic gradient descent (SGD) or batch gradient descent to update the weights. In SGD the weights are updated every training example to minimise the error between the actual and predicted outputs. In batch gradient descent all the learning samples are provided at once and mean error function is used. A third alternative is to use mini-batch updates where a proportion of the total batch is used each update. Setting sensible initial weights to assist the backpropagation algorithm can be vital to get it to train. Each method requires a target output to be compared to the predicted output and the creation of a loss function [See 8 for full explanation of SGB and backpropagation].

In theory feed forward ANNs are able represent any function provided they have enough hidden units [9]. This is because the transfer (activation) function allows non-linear relationships to be represented. Figure 14 shows the hyperbolic tangent function that is used for the neural networks in this thesis. Any value can be inputted into the neuron but the output will always be between -1 and 1. Other functions can be used such as the sigmoid which has outputs between 0 and 1. Despite any function being able to be represented by a feedforward neural network finding the correct number of hidden units, hidden layers and then training the weights is problematic. The rest of this Chapter will examine how and why ANNs with different widths and depths perform differently in the context of RL.

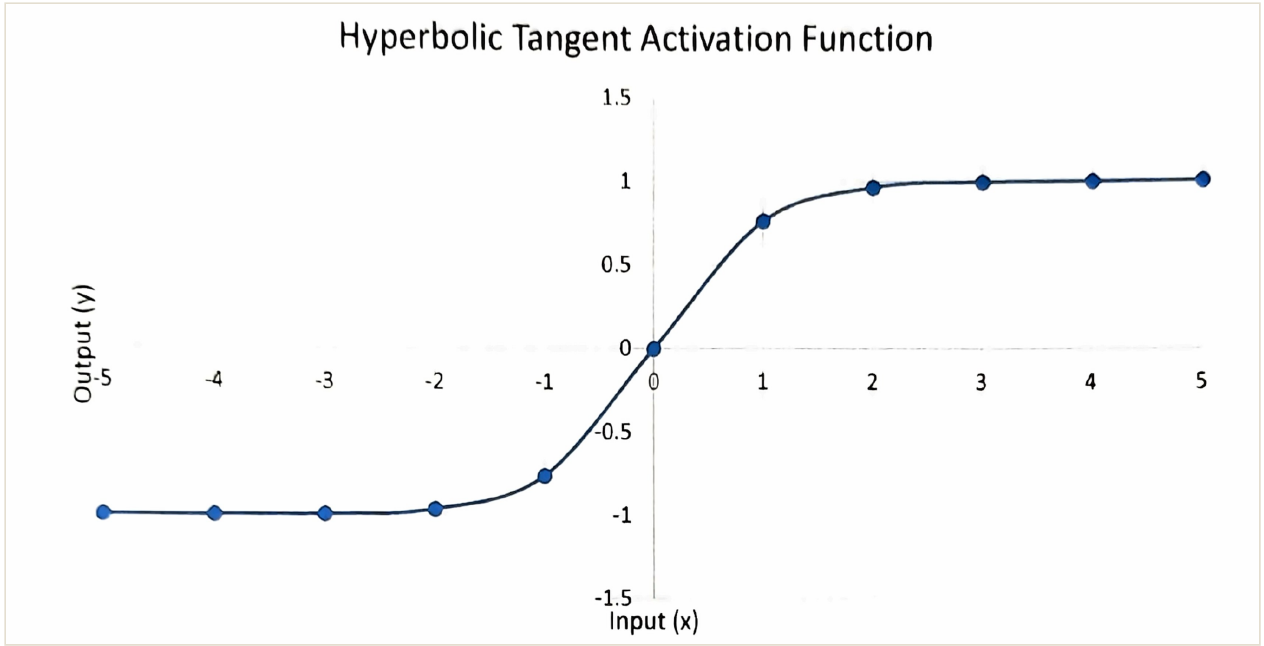


Figure 14 - Hyperbolic Tangent Transfer Activation Function

## 4.2. Learning Pre-calculated Q Values with an ANN

The results from Section 3.3 provided a Q value for all 4 potential actions for each of the 80 non terminal states. Instead of storing this information in a lookup table this section looks at storing it in the weights and architectures of ANNs. The reason for doing this is to test whether ANNs can be trained to learn different state/action Q-values that lead to the agent optimising its future rewards.

Four neural networks will be used, one for each of the actions (up, down, left and right). We could have used a single neural network with four outputs for the different actions. In [13] single networks were used for all actions to speed up learning. The decision to use different networks was made because we want different actions to learn independently from the inputs. For more complex tasks, such as trading, we may wish to provide different state representations for different actions. The grid world task is a good environment to test using multiple networks because learning is relatively quick. Each action ANN will have grid inputs representing the current state and output estimated Q-values.

The Q-Values for each action for each grid square depicted in Figure 11 form a non-linear landscape. The ANNs need to be able to learn these complex landscapes. Once learnt the neural networks can be tested to see if following the highest Q value action in each grid square results in the agent finding the fly. If all 80 squares lead to the fly then the ANNS can be said to have successfully learnt the best actions.

Stochastic gradient descent with the backpropagation algorithm is used to train the weights. The weights were set initially in a random normal distribution with a mean of zero and a standard deviation of 1. After some initial testing we divided these weights by two which resulted in increased performance. Providing small positive and negative weights helped avoid local minima. The biases were all initially set to one. The error function used was the squared difference between predicted and actual converged Q-Values from Figure 11. A learning rate of 0.005 with momentum set at 0.1 proved to work best for the dataset. The dataset for each of



the four action networks consisted of 80 data points (one for each non terminal state). A data point was picked at random for each update. For each data point the inputs to the ANNs were the grid and row position.

#### 4.2.1. Wide Networks

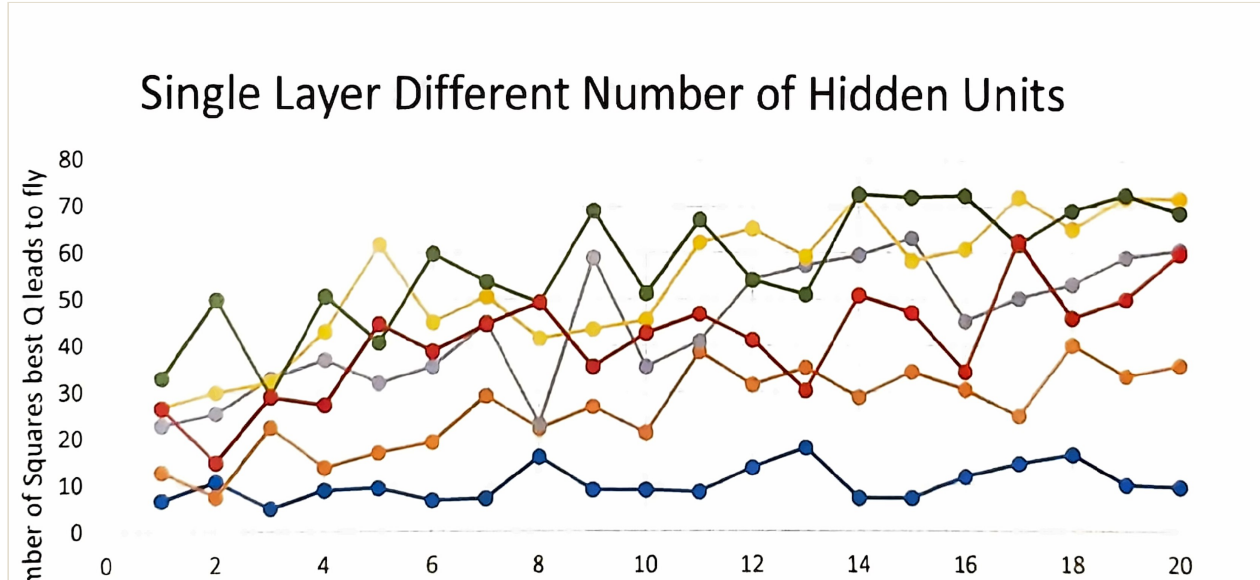


Figure 15 - Neural network performance learning directly from the tabular Q values. Split by number of hidden units in a single layer. Each dataset is the average of 10 trials.

For the first tests we used different numbers of hidden units in a single hidden layer. Figure 15 shows the number of squares that following the best Q action value leads to the fly through training. A total of 100,000 updates were performed for each network. Figure 15 shows that with a small number of hidden units the complex Q landscapes could not be represented. 2 and 6 hidden units results in poor performance by the agent. 20 and 40 hidden units performed the best reaching on average 70 of the 80 grid points leading to the fly. Performance dips when the number of hidden units is increased to 80.

We examined the Q-values for each action for each of the 80 squares after the 100,000 updates to gain some more insights into why different numbers of hidden neurons perform differently. The colour charts in Figure 16 depict the Q-values. The top line of the set of charts shows the targets the networks are trying to find. When just two hidden neurons are used the agents are only capable of working out which general direction to move. Two hidden neurons are not enough to separate the inputs to find the peak in the centre.

As the number of hidden units increase from 2 to 20 we can see complexity of the different Q-values found for each action increasing. The amount of red (higher Q-values) gets tighter around the fly in the centre of the pictures. The pictures with 20 neurons look very similar to the targets. When the number of hidden units is increased to 200 the red patches grow. This means the agent will perform worse as it cannot differentiate between the actions when it gets close to the fly.

Figure 17 examines the move down action in more detail. It shows how the Q-values and weight values of the more down ANN change through time. The Q-values are the outputs from the neural networks. The weight value metric provides a summary of all the weights in the network. Analysis of these metrics together given an indication of the stability of networks. ANNs with 2, 20, 80 and 200 hidden units in a single layer are



depicted. The 20 and 80 hidden unit networks show similar results for all of the charts. Their mean Q-values and weight values are steady throughout learning.

When just two hidden neurons were used the Q-values have a much lower standard deviation than the other networks. The mean and standard deviation of the weight values are also much lower. This shows that the ANN struggles to split the inputs into different output values. When 200 units are used the mean Q-value moves up and down rapidly between -0.6 and 0.6 during learning. The weight metrics are similar to the 80 hidden units. This shows that a very small change in weight value has a big change in the output of the network. This could be because a change in one of the 240 weights to the hidden layer has a direct effect on the output of the network. There are no extra layers in the network to smooth out the changes. The more hidden units there are it is also more likely that some may have become saturated at either +1 or -1. If a large number of units become saturated at these highest values any weight change leading to the output will have a greater effect.

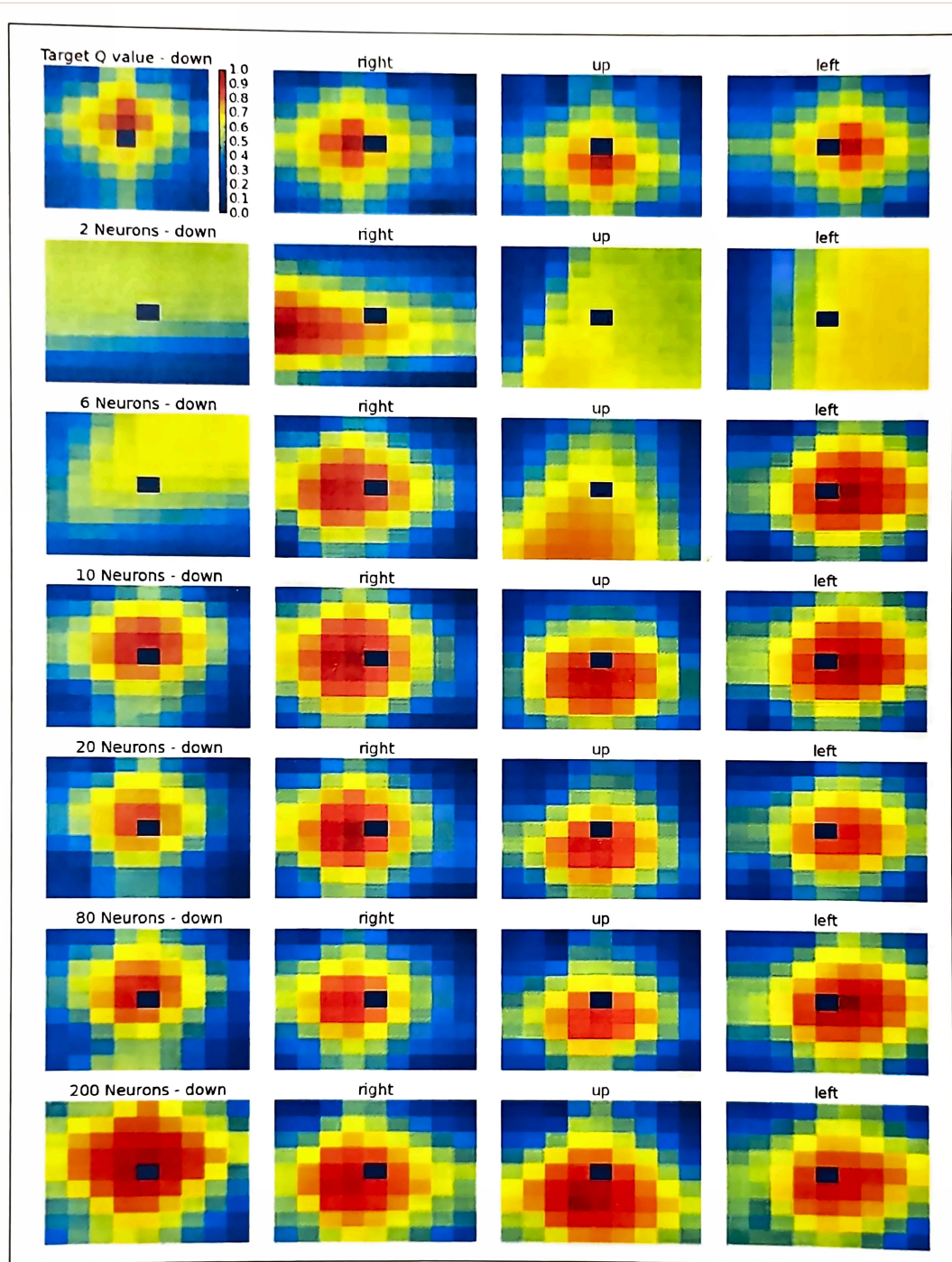


Figure 16 - Snap Shot Wide Neural Networks. The Q-Values after 20 times 5000 runs learning the tabular Q-learning results directly. The top row show the target values (See Fig 11). The four columns each represent an action. Line 2 starts with an example run with just 2 hidden neurons in the single hidden layer. In the final line the agent has 200 hidden neurons in the

Figure 16 - Snap Shot Wide Neural Networks. The Q-Values after 20 times 5000 runs learning the tabular Q-learning results directly. The top row show the target values (See Fig 11). The four columns each represent an action. Line 2 starts with an example run with just 2 hidden neurons in the single hidden layer. In the final line the agent has 200 hidden neurons in the single hidden layer. The blue square in the centre represents the fly. For each line of four diagrams the outputs (estimated Q-values) have been normalised between 0 and 1. We also do this for the 3 similar diagrams that follow in this chapter.

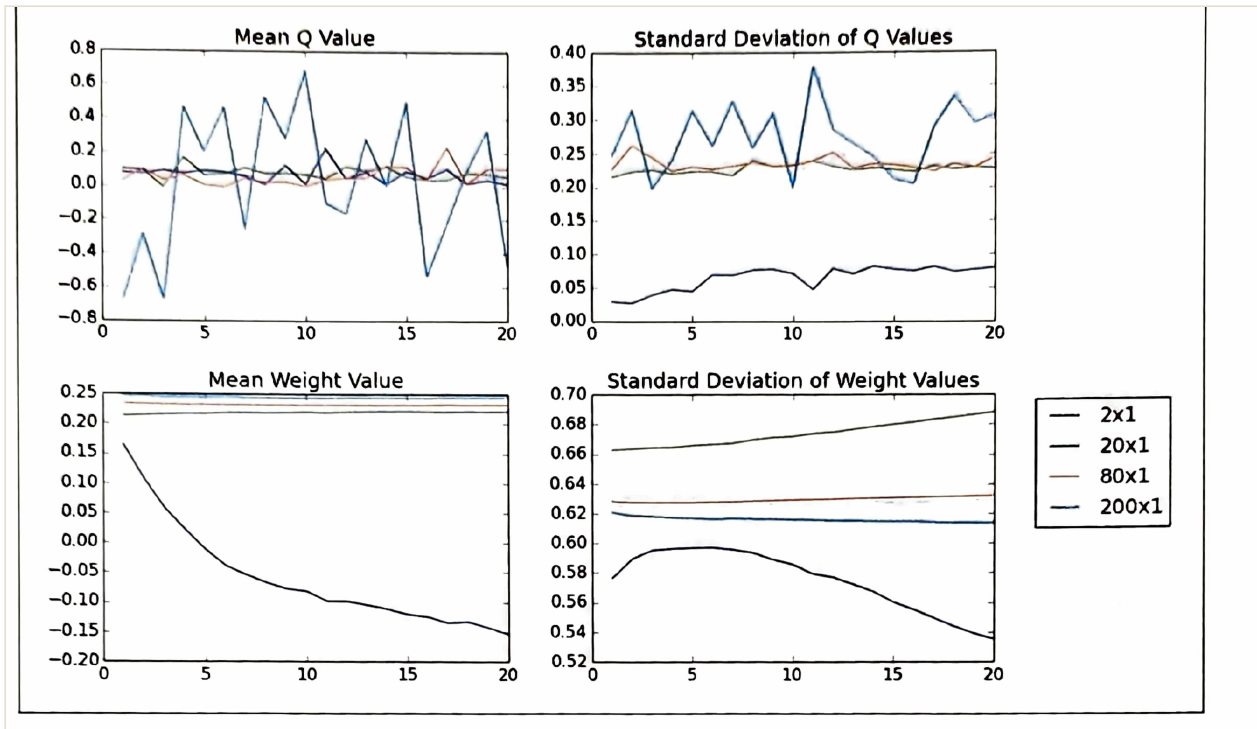


Figure 17 - Changing Q values and weight values during training for ANNs of different widths for the DOWN action. For each chart the four lines are for the different number of hidden neurons in the single hidden layer (2, 20, 80 and 200). The x-axis for each chart presents a snap shot for every 5000 updates. The mean Q-value is the mean Q-value for action move down for each of the 80 squares. The standard deviation of Q-values is the standard deviation of the Q-values for action move down for each of the 80 squares. The mean weight value is the mean of the weights for all of the weights in the move down ANN for all layers. The standard deviation of the weight values is the standard deviation of the weights for all of the weights in the move down ANN for all layers.

#### 4.2.2. Deep Networks

We now examine if increasing the number of hidden layers can improve performance. Figure 18 shows the results when extra hidden layers are included. The hidden layers each have 20 hidden units. Adding the second layer means that the agent is trained to find the fly from any of the 80 squares most of the time. Adding a third hidden results in even smoother learning with all ten trials converging after 50,000 updates.

The colour charts in Figure 20 show the outputs from the ANNs for each of the four actions for one, two, three and four hidden layers of 20 neurons each. The charts show that as every extra layer is added the outputs become more and more similar to the target values. With four hidden layers it is virtually a perfect representation.

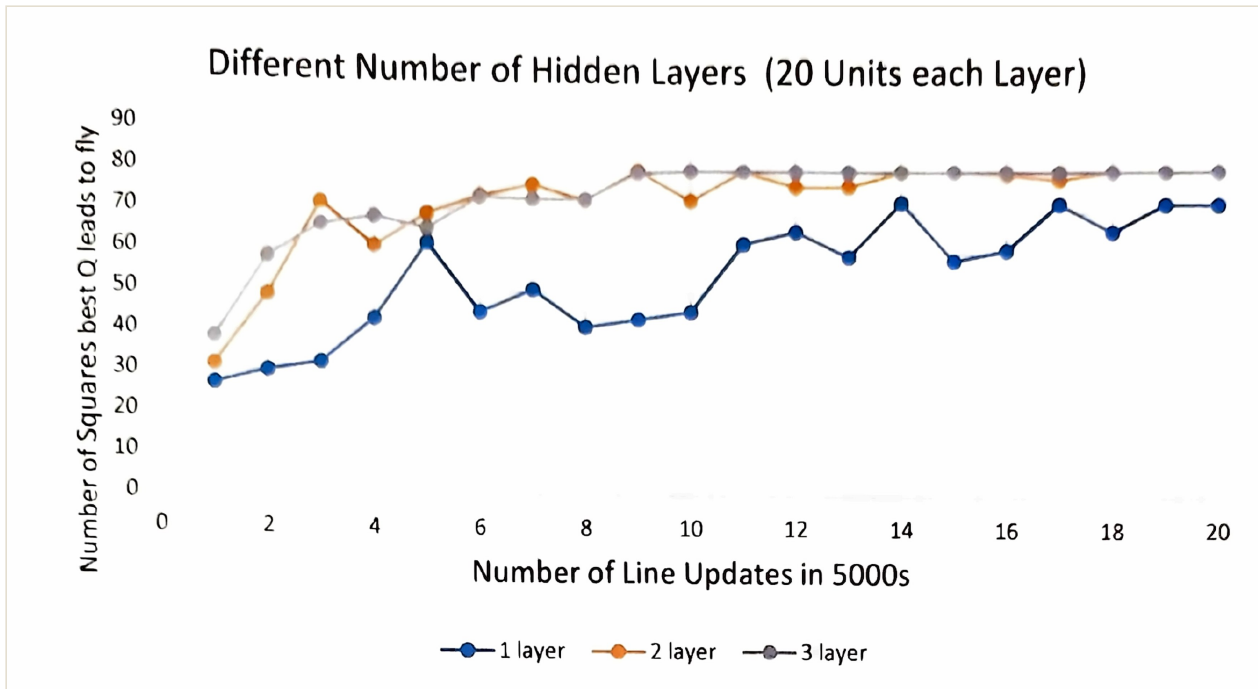


Figure 18 - Neural network performance learning directly from the tabular Q values. Split by number of hidden layers with 20 units in each layer. Each dataset is the average of 10 trials.

Figure 19 shows the Q-value and weight values through the learning process for each down action. The Q-values (the ANN outputs) are consistent as the number of layers increase. At the same time the average weight values and their standard deviation decrease as more layers are added. This means that the deep layers are made up of lots of smaller weights all slowly changing. One change in weight will not have a massive change on the output of the entire network because of these smaller weight changes. Also the network have extra layers to adapt for any neuron that has become saturated before the output layer.

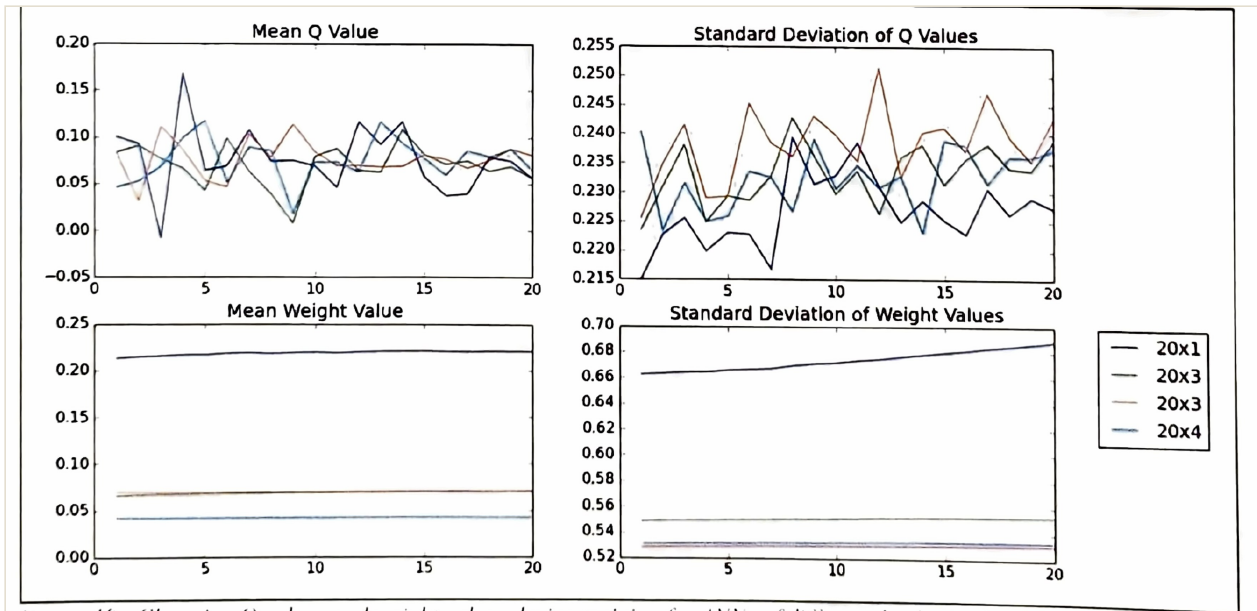


Figure 19 - Changing Q values and weight values during training for ANNs of different depths for the DOWN action. For each chart the four lines are for the different number of hidden layers with 20 neurons in each layer. See Figure 17 for full explanation of the different charts

Figure 19 - Changing Q values and weight values during training for ANNs of different depths for the DOWN action. For each chart the four lines are for the different number of hidden layers with 20 neurons in each layer. See Figure 17 for full explanation of the different charts.



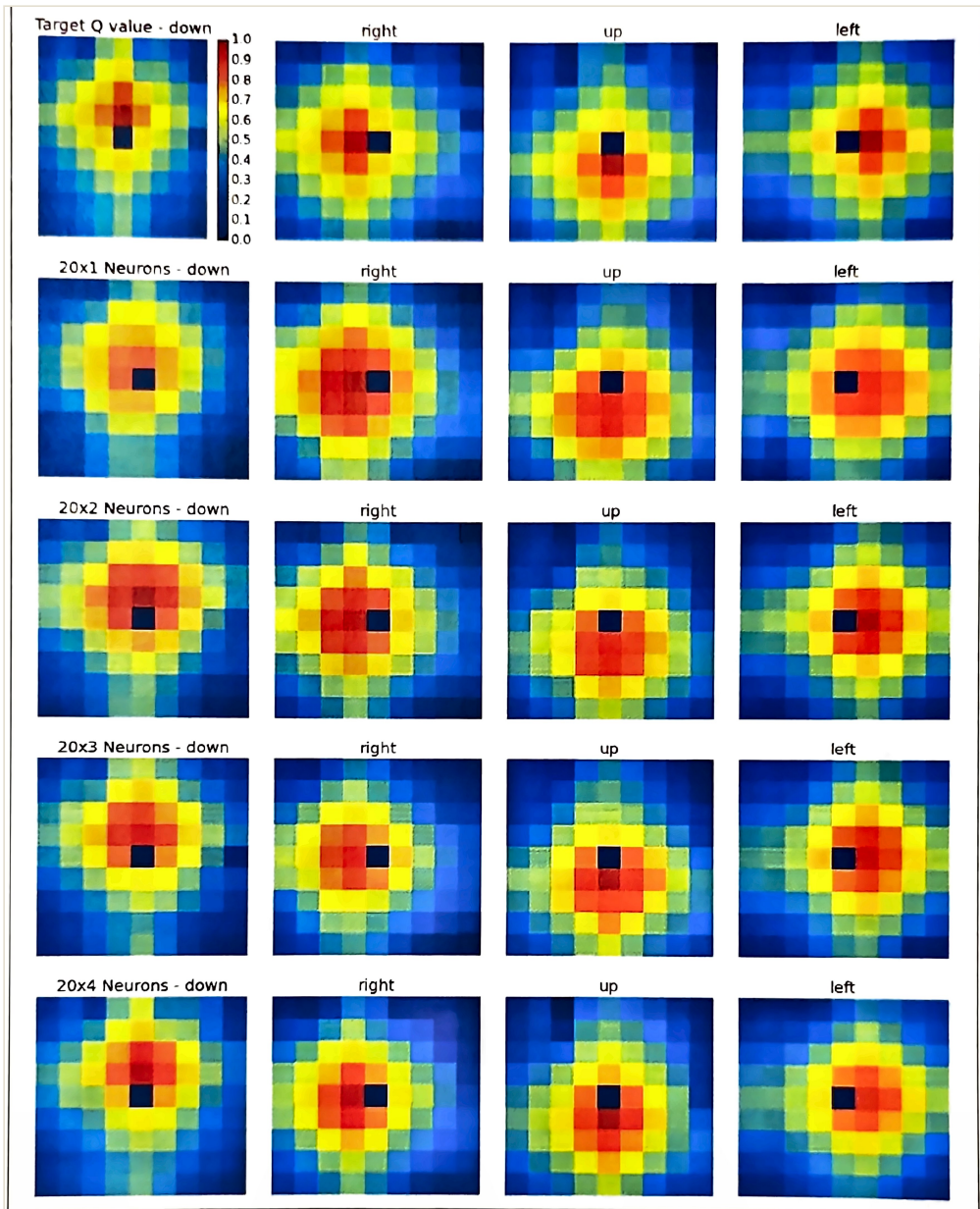


Figure 20 - Snap Shot Deep Neural Networks. The Q-Values after 20 times 5000 runs learning the tabular Q-learning results directly. The top row show the target values (See Fig 11). The four columns each represent an action. Line 2 starts with an example run with 20 hidden neurons in a single hidden layer. Each row after has an extra hidden layer of 20 neurons. The blue square in the centre represents the fly.

Figure 20 - Snap Shot Deep Neural Networks. The Q-Values after 20 times 5000 runs learning the tabular Q-learning results directly. The top row show the target values (See Fig 11). The four columns each represent an action. Line 2 starts with an example run with 20 hidden neurons in a single hidden layer. Each row after has an extra hidden layer of 20 neurons. The blue square in the centre represents the fly.

### 4.3. Learning Q values with ANNs Directly Through Interaction with the Environment

Transferring optimal Q-Values already determined by tabular Q-learning to a neural network has limited benefit. In the above case storing the weight values of all the connections for the four action ANNs would exceed the space required merely to store the values in a lookup table. The real benefit of using an ANN, or a set of ANNs, to approximate the Q-values is when the state/action space is very large or continuous state variables are used. In these cases all the possible actions for each state cannot be visited in reasonable time so a method to generalise Q values from any state/action input is required.

In this section we will start by describing how we have incorporated ANNs into the equations and algorithms for tabular Q-Learning that were introduced in Chapter 3. We will do this by explaining how ANNs can be used instead of a table to store the Q-values and how they can be updated as the agent interacts with the environment. This will be followed by a description of how we set the parameters to help the agents learn for this particular task. We will then examine how wider and deeper networks perform with the agent learning by directly interacting with their environment.

#### ***4.3.1. Updating the ANNs as the Agent Moves***

The Q learning update algorithm (Equation 1) for every move by the agent can be combined with the SGD updates used in 4.2 for training a feed forward ANN. As in the previous section we use four neural networks, one for each action, to estimate the Q-values for each state. These four action ANNs can be updated independently only when the agent takes that action.

Figure 21 depicts the process for the example of when the agent decides to move up. Instead of the target value being the Q value created by the tabular method the new target value is calculated from the four ANNs (Figure 21 part 1). The new target is calculated using the same components as it was for tabular Q learning. The only difference is that the Q values used are the outputs of ANNs rather than a Q-table. The Q value of the current state action (red on Figure 21) is the output of the ANN for the current action for the current state. The Q-value for the best action in the next state is the highest output of the four action ANNs in the next state (blue on Figure 21).



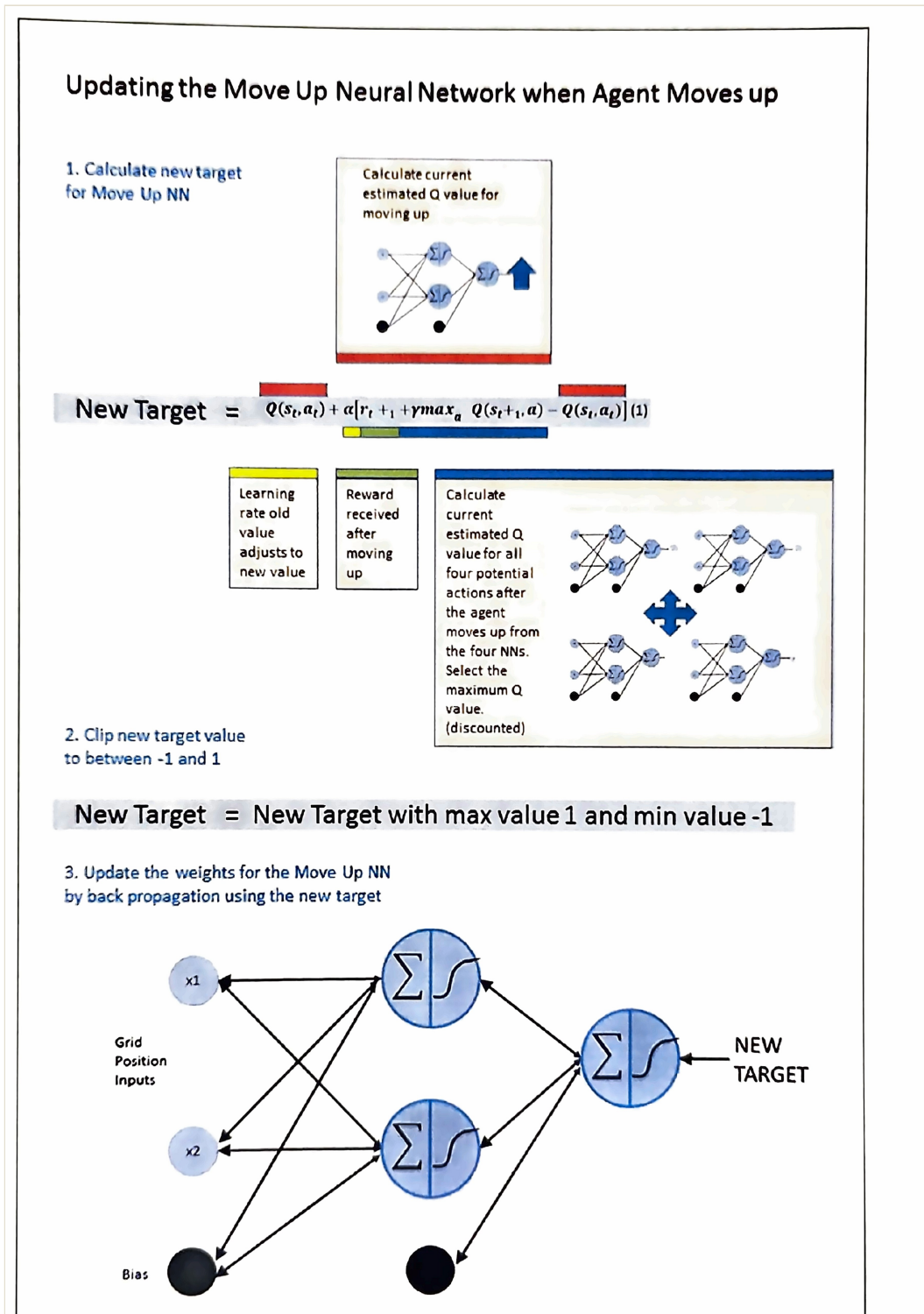


Figure 21 - depicts how the Q-learning update algorithm is used with the Q-Values stored in a set of action neural networks. (1) The new target is calculated when the agent moves up (same applies for any direction agent moves). (2) This new target is then clipped between -1 and 1. (3) Every time the agent moves up, the move up action neural network is updated.

This value is clipped between -1 and 1 (Figure 21 part 2) so that the target does not saturate the hyperbolic tangent activation function used (Figure 14). Without the clipping of the target values the outputs from the ANNs can explode to very big numbers. The new target value is then used as the target for the ANN update (Figure 21 part 3). The ANN for the action taken is updated whenever the agent takes that action. The ANNs are therefore learning from a moving target.

#### **4.3.2. Setting the Parameters for the Q-learning Algorithm**

The same grid world environment and reward system as 3.3 for tabular Q-learning is used for this section. The tabular Q-learning agent solved the task very easily by randomly searching the environment and updating its Q-table. The Q-value converged after 2500 moves. Using ANN agents to solve the task will prove more difficult as four ANNs have to be trained from moving targets to all learn the non-linear landscape.

Using ANNs with Q-learning requires setting two learning rates. One for the Q-learning updates and one for the ANN updates. After some experimentation a learning rate for the ANNs of 0.5 with momentum of 0.1 was chosen. This learning rate was combined with a low learning rate for updating the target Q values. The balance between the two learning rates is vital to getting the ANNs to learn Q values that lead to optimal actions being taken. This learning rate for the Q updates was initially set at 0.02 and was reduced every 10,000 moves the agent took. If  $k$  is every 10,000 moves the learning rate was set to  $0.2/(k+1)$ . This meant that after 200,000 moves  $k=20$  and the learning rate was reduced to 0.001.

In Chapter 3 we described the outline of the Q-learning algorithm (Algorithm 1). Setting how much the agent randomly explores compared to how much it exploits its current knowledge is important for how well the agent will learn. If it exploits its current knowledge too early it may never find the optimal solution. If it explores too much the ANNs will take too long to learn. The agent was initially set to follow the action with the best Q value just 10% of the time. The rest of the time the agent is set to move randomly. As in 3.3, the agent restarts from a random square in the grid if it fails to find the fly after 30 moves. As  $k$  increases the ratio of random exploring to exploiting the current best Q value is decreased linearly. After 200,000 moves the agent is set to follow the best Q values all the time. Slowly increasing the exploitation of the best Q values means the current best path is followed more often reinforcing the ANNs for the positive actions in each state.

#### **4.3.3. Wide Networks**

We will now analyse how well different structures of ANNs perform learning directly from their environment. The process is similar to results in 4.2. We start by examining how a different number of neurons in a single hidden layer perform.

Figure 22 shows the average performance from ten trials for different number of hidden units in the single layer. The results are far worse than learning directly from the lookup tables in Figure 15 with the best performance being 40 squares rather than 70 squares. The results show a similar pattern to Figure 15 with 20 hidden units performing the best. Smaller (2-10) and larger (40) numbers of hidden units perform worse.

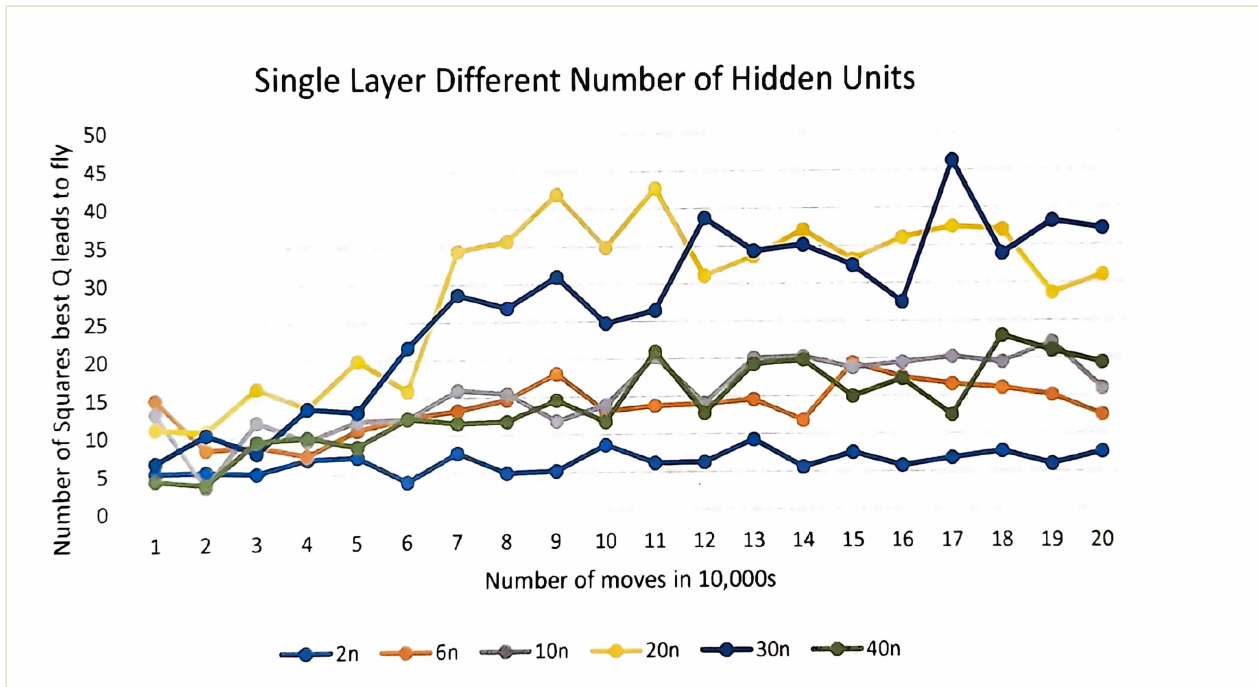


Figure 22 - Neural network performance learning directly from interacting with the environment. Split by number of hidden units in a single layer. Each dataset is the average of 10 trials.

Figure 23 shows the colour chart for outputs of the different action ANNs. These outputs represent the estimated Q-values for each state/action. The top line of charts show the Q-values stored by tabular Q-learning at convergence. We see a similar pattern to Figure 16. Using just 2 hidden units the agents are unable to learn anything. As the number of hidden units increases from 6 to 20 a pattern starts to emerge showing correct Q-values for some of the squares. With 80 hidden neurons more of the squares are dark red and by 200 hidden neurons virtually all the squares are red with the value of 1.

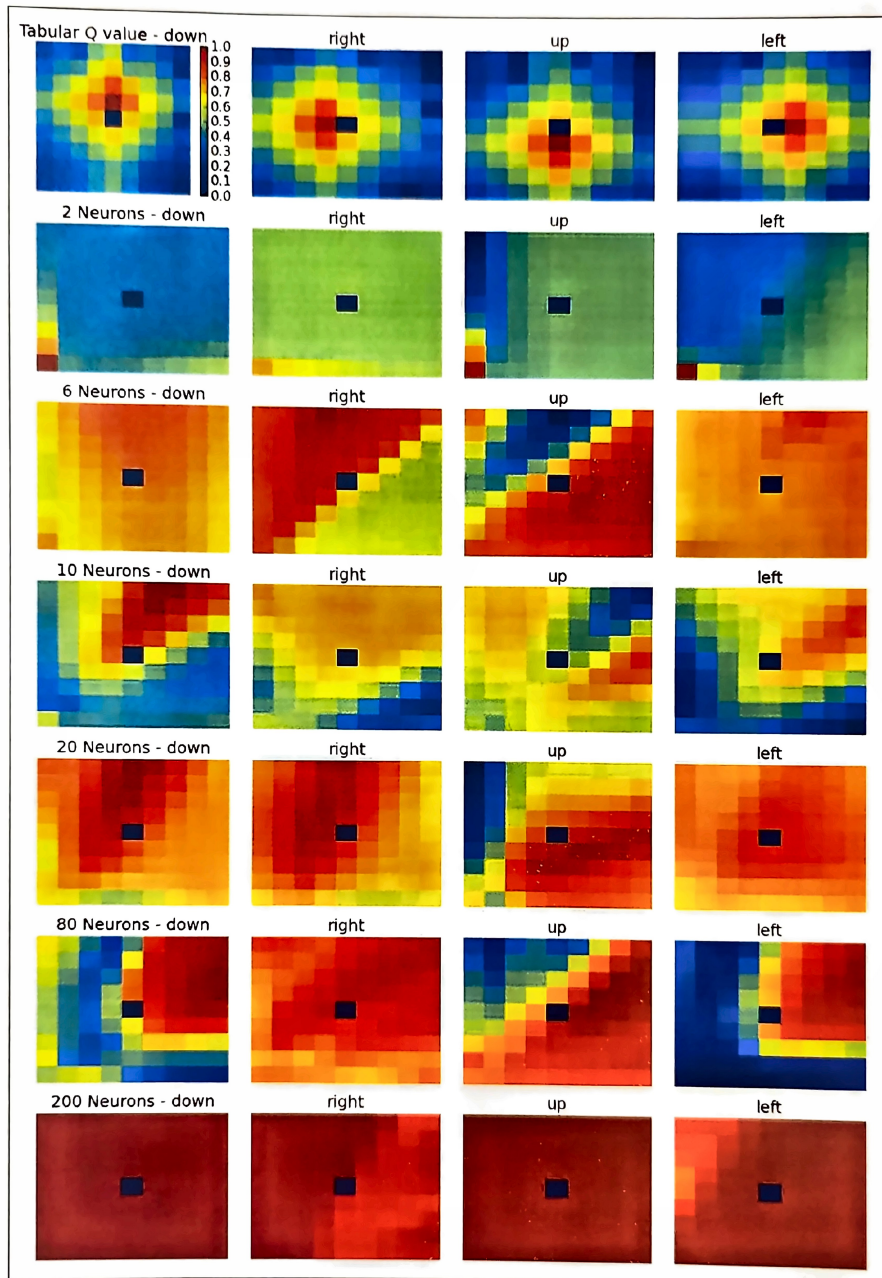


Figure 23 Snap Shot Wide Neural Networks. The Q-Values after 20 times 10000 runs learning by interaction directly with the environment. The top row shows the Q-value results from tabular Q-learning that found the optimal result (See Fig 11). The four columns each represent an action. Line 2 starts with an example run with just 2 hidden neurons the single hidden layer. In the final line the agent has 200 hidden neurons in the single hidden layer. The blue square in the centre represents the fly.

Figure 23 - Snap Shot Wide Neural Networks. The Q-Values after 20 times 10000 runs learning by interaction directly with the environment. The top row shows the Q-value results from tabular Q-learning that found the optimal result (See Fig 11). The four columns each represent an action. Line 2 starts with an example run with just 2 hidden neurons the single hidden layer. In the final line the agent has 200 hidden neurons in the single hidden layer. The blue square in the centre represents the fly.

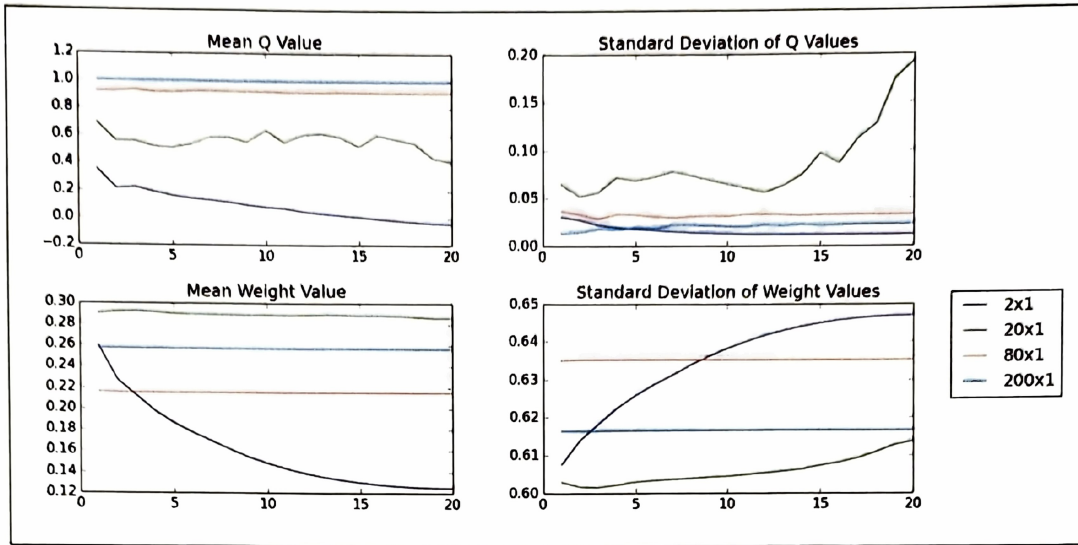


Figure 24 - Changing  $Q$  values and weight values during training for ANNs of different widths for the DOWN action. For each chart the four lines are for the different number of hidden neurons in the single hidden layer (2, 20, 80 and 200). The x-axis for each chart presents a snap shot for every 10000 moves. See Figure 17 for full explanation of the different charts.

Figure 24 - Changing  $Q$  values and weight values during training for ANNs of different widths for the DOWN action. For each chart the four lines are for the different number of hidden neurons in the single hidden layer (2, 20, 80 and 200). The x-axis for each chart presents a snap shot for every 10000 moves. See Figure 17 for full explanation of the different charts.

Figure 24 shows what happens to the  $Q$ -values and weight values during the learning process for the move down action ANNs. The mean  $Q$ -value for both 80 and 200 hidden units are almost 1 from the start. This indicates that the output neuron is saturated. The agent fails to learn anything useful because of this. For the two hidden neuron agent we see that the mean and standard deviations of the  $Q$ -values gradually decreases through time. This shows how the agent fails to learn anything while interacting with the environment. 2 hidden units is not enough to learn how to represent the grid inputs. For 20 hidden units the mean  $Q$ -value slightly decreases and the standard deviation increases. This shows that for 20 hidden units learning is occurring. The agent is able to learn and get better through time.

#### 4.3.4. Deep Networks

In 4.2.2 it was found that increasing the number of hidden layers increased the agents' performance. Figure 25 shows that this also occurs when the ANNs are trained directly. Here 20 units are placed in each hidden layer. Performance and speed of learning increases as the number of hidden layers is increased from 1 to 4. After 4 hidden layers the performance appears to stop increasing. The average performance of the 10 trials is 70 of the 80 squares leading the fly when the action with the best  $Q$  value is taken.



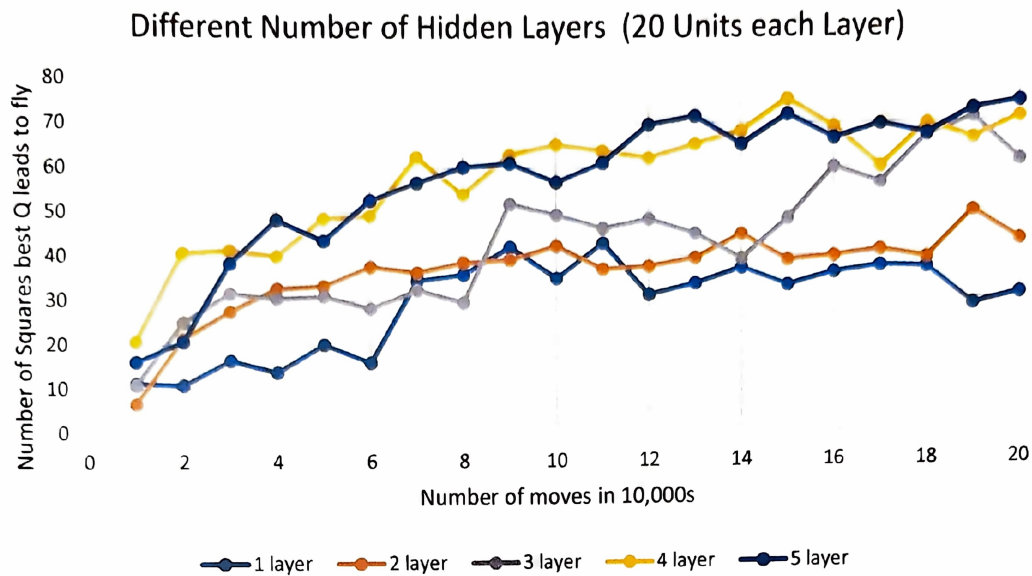


Figure 25- Neural network performance learning directly from interacting with the environment. The number of hidden layers was set to 1-5 with 20 hidden units in each layer. Each dataset is the average of 10 trials.

Figure 25 - Neural network performance learning directly from interacting with the environment. The number of hidden layers was set to 1-5 with 20 hidden units in each layer. Each dataset is the average of 10 trials.

Figure 26 shows the Q values and weight values through time for networks of different depths and compares them to the shallow wide network (200 units in one hidden layer).

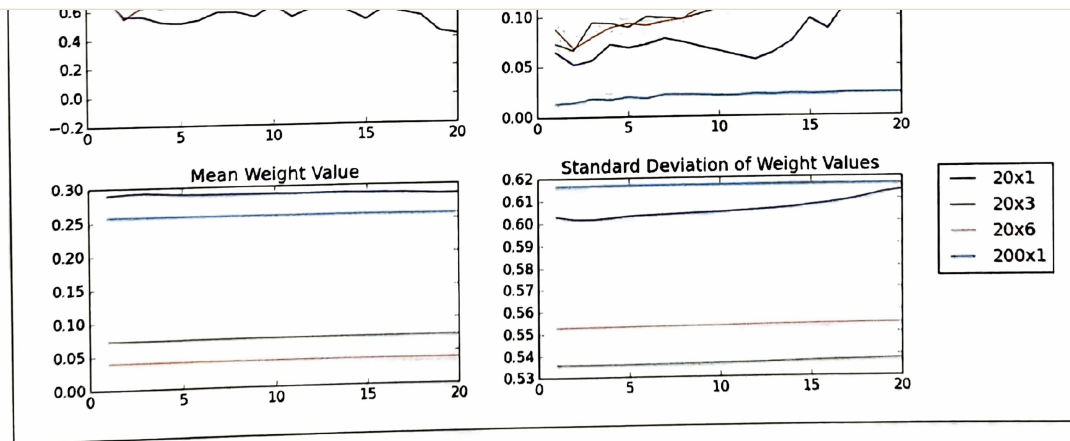


Figure 26 - Changing Q values and weight values during training for ANNs of different depths for the DOWN action. For each chart three of the four lines are for the different number of layers of 20 neurons each. We also include the wide, shallow 200 unit network for comparison. The x-axis for each chart presents a snap shot for every 10000 moves. See Figure 17 for full explanation of the different charts.

As the number of hidden layers increase the mean and standard deviation of the weights decrease.

Figure 26 - Changing Q values and weight values during training for ANNs of different depths for the DOWN action. For each chart three of the four lines are for the different number of layers of 20 neurons each. We also include the wide, shallow 200 unit network for comparison. The x-axis for each chart presents a snap shot for every 10000 moves. See Figure 17 for full explanation of the different charts.



As the number of hidden layers increase the mean and standard deviation of the weights decrease. This is the same result as we found for ANNs learning from the converged tabular Q-values. These very low weights enable the ANNs to slowly learn as they explore their environment and exploit their knowledge.

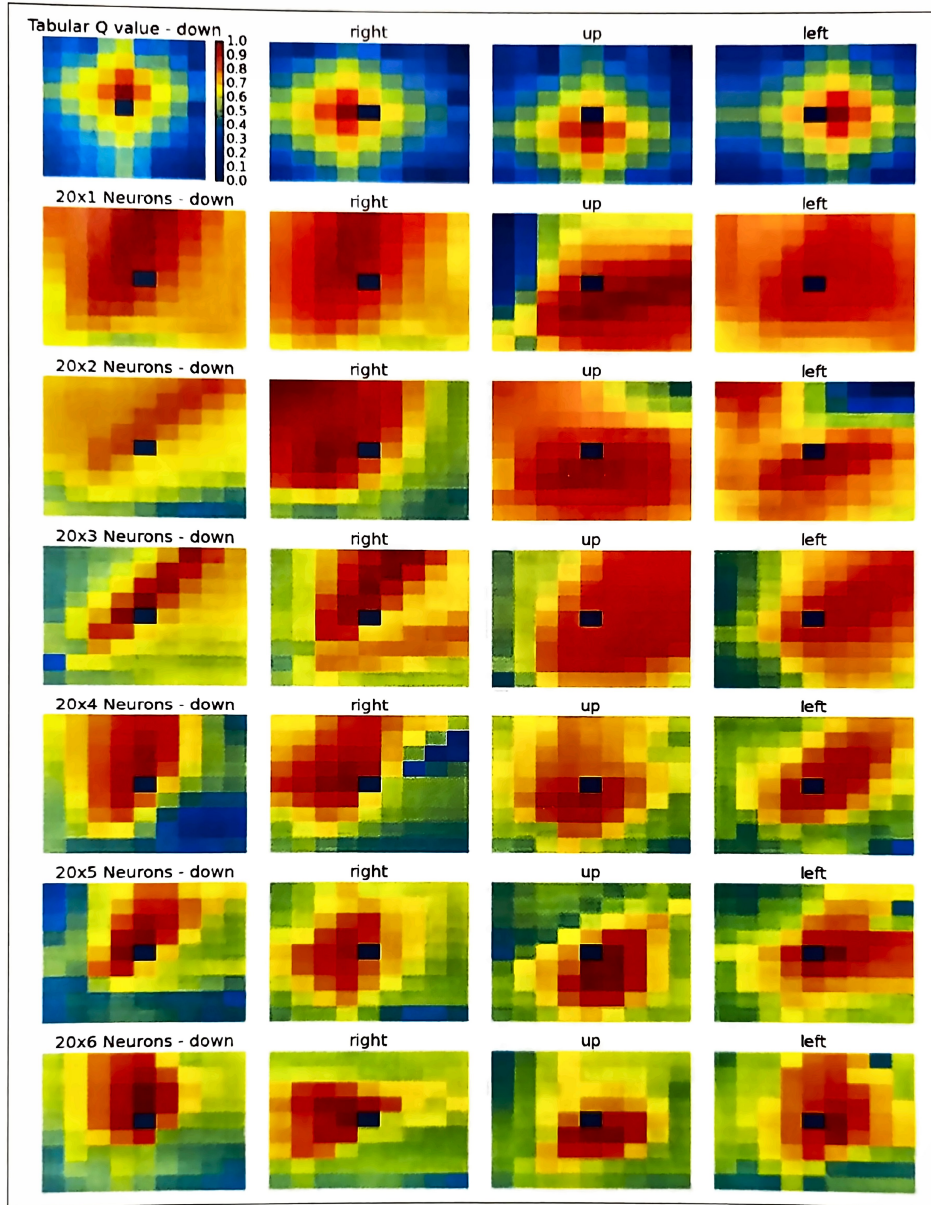


Figure 27- Snap Shot Deep Neural Networks. The Q-Values after 20 times 10000 runs learning by interaction directly with the environment. The top row shows the Q-value results from tabular Q-learning that found the optimal result (See Fig 11). The four columns each represent an action. Line 2 starts with an example run with 20 hidden neurons a single hidden layer. In the final line the agent has 20 hidden neurons stacked in each of its 6 hidden layers. The blue square in the centre represents the fly.

Figure 27 - Snap Shot Deep Neural Networks. The Q-Values after 20 times 10000 runs learning by interaction directly with the environment. The top row shows the Q-value results from tabular Q-learning that found the optimal result (See Fig 11). The four columns each represent an action. Line 2 starts with an example run with 20 hidden neurons a single hidden layer. In the final line the agent has 20 hidden neurons stacked in each of its 6 hidden layers. The blue square in the centre represents the fly.

Figure 27 shows the colour charts for move down action network for increasingly deeper network. As the number of layers in the network increase we can see estimated Q-values looking more and more like the converged tabular results.

## 4.4. Conclusions

In this Chapter we have demonstrated that that deep neural networks are capable of learning online through reinforcement learning for this task. This task took 2500 random moves for a tabular Q-learning agent to master. After 200,000 moves exploring and exploiting its environment deep ANNs agents are almost about to perform to the same level.

We have seen in 4.2 that the task of accurately approximating the landscape for this task is difficult for ANNs. In order for the task to be mastered inputs (grid positions) that are very similar have to have very different outputs. For instance, just below the centre the up action needs the highest output but, very close by, just the left of the fly requires the right action to have the highest output. Both independent ANNs have to learn correctly for the best action to be taken by the agent. 4.2 demonstrate that deep ANNs perform better than wide ANNs because of their ability to fine tune very small weights.

In 4.3 using Q-learning to train ANNs was very successful. The way we adapted the basic Q-learning update function and algorithm to incorporate ANNs worked. When examining deep and wide networks the same patterns occurred as in 4.2. The deep ANNs were the most successful. They were able to learn the best actions through time by slowly updating lots of very small weights in multiple layers as they interacted with their environment.

In Chapter 5 we will use the insights gained from these results to help develop profitable autonomous trading agents using the same components.

## 5. Results 2 : RL using Deep and Wide ANNs for Autonomous Trading

---

We saw in 2.3 that various machine learning and artificial intelligence techniques have been used to develop trading systems for financial markets. For trading to be profitable the system has to find a good time for both opening and closing trades. Traditionally systems have been built to forecast a price at a fixed time in the future [for example in 10]. A trading system is then developed to buy if the current price is below the forecast price and sell if current price is above the forecast price.

The decision to use RL is due to the desire for the system to learn autonomously when to open and close trades. Instead of forecasting future prices and then building trading rules around those forecasts a system built using RL can learn directly from the current situation (state) which trading action is the most profitable one to take.

In this Chapter an autonomous algorithmic trading agent is developed to use on the Betfair betting exchange [see 2.1 for Betfair background]. A Model for greyhound racing markets is developed. These markets were chosen because their low liquidity and high volatility means that the agent can focus on placing market orders rather than adding the complication of having to monitor limit orders. In this Chapter we train agents to learn when to back a greyhound and when to place a lay bet to lock in profit (or loss). The initial bet will always be a back bet. In Chapter 6 when we test agents with real money trading the agents will be able to place either back or lay bets initially.

We start by showing the structure of the RL system and describe the contents of the components used. This is followed by a description of the Q-learning algorithm that was developed to enable the agents to learn. We examine this algorithm using a table to store the Q-values and using a simple ANN to approximate the Q-values. The initial analysis is conducted using a set of ANNs with just two hidden units in a single hidden layer on a small dataset from races from just two days. This is followed by an analysis of using deeper and wider ANNs using a larger dataset from Romford dog track taken from a four month period.

### 5.1. Structure of the Automated Trading System

Figure 28 depicts the structure of the system which was developed. The setup has the same components of agent, state representation, actions, rewards and environment as used in the grid world examples in Chapters 3 and 4. The difference is what the individual components consist of. We will now look at each of the components in turn and then show the Q-learning algorithm that was used to train the agents.

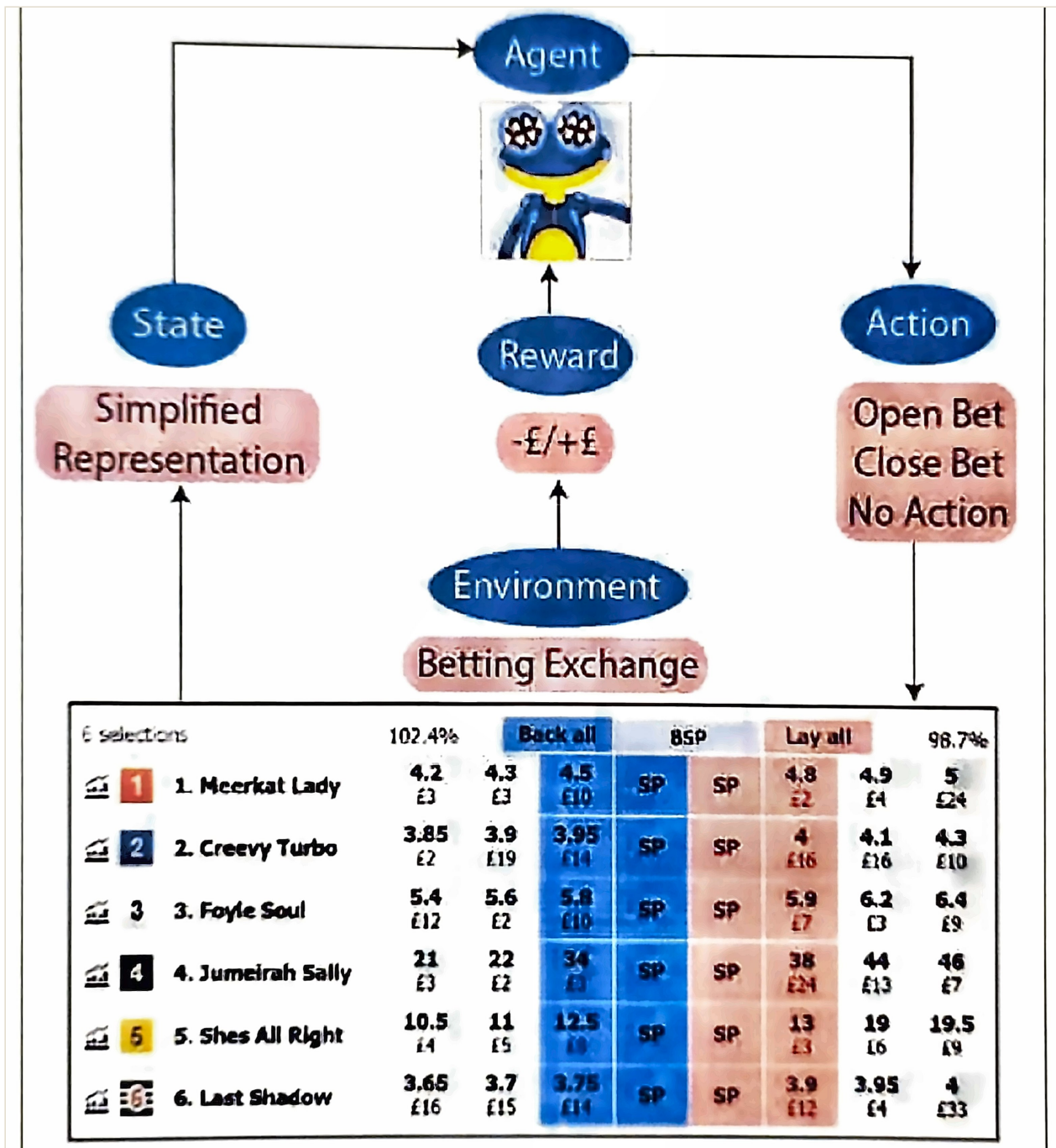


Figure 28 - The components of the proposed trading system. The agent makes an observation of the current order book. The agent then processes this observation to form a simplified representation of the current state. The agent makes a decision as to which action to take given its memory of the rewards associated with the current state. The next time step the agent registers the reward it receives for its actions and updates its state/action memory.

### 5.1.1. State Representation

For the grid world example in the previous chapter the state was represented by the row and column position the agent was currently at. Potentially there are numerous ways the current state of the betting market could be represented. The agent could be given access to the entire order book for every runner at each time step. Using this information the agent could choose to maintain its own history of every runner and build price volume charts. It could also monitor every runner and examine how changes on one selection affect the market for the other runners.

In this thesis we only use the data contained in the order book for the current time step for each individual greyhound. As we are going to test both tabular Q-learning and ANN Q-learning we require two encoding schemes. The tabular scheme requires a number of discrete states whereas the ANN encoding scheme requires a vector of inputs. The two encoding schemes are based the same method. The tabular system rounds the ANN's continuous numbers into a discrete representation. The precise encoding schemes were arrived at by trial and error testing for a commercially sensitive project. In this thesis we will explain the encoding schemes but we will not investigate other schemes or examine why this scheme performed the best.

The starting point for the proposed state encoding method used here is the weight of money (WOM) indicator [11]. The WOM indicator is calculated by dividing the total money available in the three best to lay prices by the total in both the three best to lay and the three best to back prices. For the example in Figure 29 below this would equate to 0.25 (£21+£20+£5 divided by £21+20+£5+£150+£15+£20). A value between 0.33 and 0.66 suggests that the prices will remain stable because there is a fairly equal amount of money on both sides. If the value is below 0.33 this suggests that the price is more likely to drift because there is more money available to back than to lay. The opposite is the case if the number is above 0.66.

The problem with this indicator is that it takes no account which of the three prices contain the highest volumes. For instance, in Figure 29, the biggest volume to back is £150 at 3.05. This will have less influence than if it was at the front of the market at 3.15. The second issue is if there are holes in the market between the best to back and best to lay prices. If there is a gap it is difficult to see any use in the volume information as the money is so far from the mid-point in the market. To overcome these issues we developed a stronger encoding system that takes into account which position in the top three prices the volume occurs and also removes the gap in the market problem.

In order for the learning algorithm to function smoothly if there is a price gap between the best to back price and the best to lay price the agent ignores that time step for the majority of trading. Figure 29 shows a best price to back of 3.15 and a best price to lay at 3.25. The agent would ignore this time step because there is a gap at 3.2. The only time it is taken into account is if a trade needs to be closed just before the race starts.

Figure 29 - Example of gap in the market between top back and top lay price available

| Price   | 3.05 | 3.1 | 3.15 | 3.2 | 3.25 | 3.3 | 3.35 | 3.4  |
|---------|------|-----|------|-----|------|-----|------|------|
| To Back | £150 | £15 | £20  |     |      |     |      |      |
| To Lay  |      |     |      |     | £12  | £20 | £5   | £200 |

The agent's state representation only uses the volume available in the order book at the best back/lay price plus the next two price ticks in both directions. Figure 30 shows an example of the encoding system used by the agent to represent the raw order book data.

Figure 30 (part 1) - order book snapshot

|         | B3  | B2   | B1  | L1   | L2  | L3   |
|---------|-----|------|-----|------|-----|------|
| Price   | 3.1 | 3.15 | 3.2 | 3.25 | 3.3 | 3.35 |
| To Back | £15 | £20  | £55 |      |     |      |
| To Lay  |     |      |     | £12  | £20 | £5   |



Figure 30 (part 2) - encoding transform

|        | B3     | B2     | B1     | L1     | L2     | L3     |
|--------|--------|--------|--------|--------|--------|--------|
| Line 1 | £90    | £75    | £55    | £12    | £32    | £37    |
| Line 2 | 0.00   | 15.00  | 35.00  | 78.00  | 58.00  | 53.00  |
| Line 3 | 0.0000 | 0.0628 | 0.1464 | 0.3264 | 0.2427 | 0.2218 |
| Line 4 | 0      | 1      | 1      | 3      | 2      | 2      |

Table Encoding: 011322

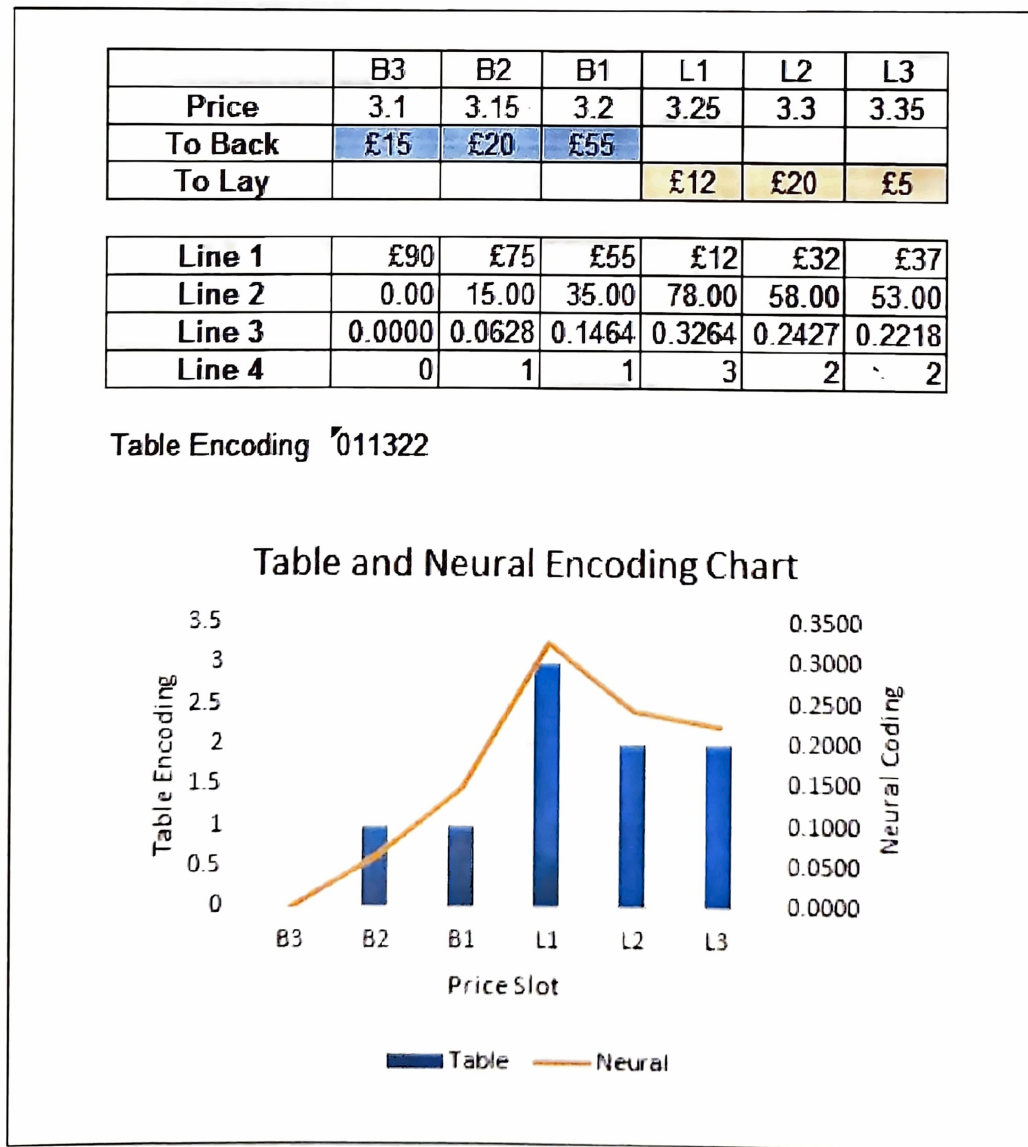


Figure 30 - State encoding system used by the agent

Figure 30 - State encoding system used by the agent (Table and Neural Encoding Chart)

The top table shows the three top back and lay offers currently in the order book. There are no gaps in the market. Lines 1-4 show how this information is transformed by the agent. B1 is the best price to back, L1 is



the best price to lay. B2 and L2 are the second best prices. B3 and L3 are the third best prices. In Line 1 the back and lay volumes are summed. So  $B1 = B1$ ,  $B2 = B2 + B1$  and  $B3 = B1 + B2 + B3$ . Line 2 is the maximum value in Line 1 minus Line 1. Line 3 is the Line 2 divided by the total of Line 2.

Line 3 is a 6 variable (range 0 to 1) representation of the current order book. This is ideal for feeding into an ANN. Line 4 is Line 3 multiplied by 10 and then rounded to the nearest whole number. This outputs 6 integers between 0 and 9. Concatenating the 6 integers gives a 6 digit string that can be used as a unique state representation for tabular Q-learning.

The chart in Figure 30 shows the two encoding types. They are both very similar. The only difference is that the neural encoding contains continuous inputs whereas the table encoder uses discrete inputs. The state representation used here is intuitive but extremely simple. The real values and price levels are lost in the transformation but the encoding gives a good summary of how much money is on each side of the market and where in the market it is. The encoding shows the relative weight of money available at the current time.

The ANN state representation vector contained a 7<sup>th</sup> input to represent the time left before the race starts. This was calculated by dividing the number of seconds left until the race by 600. This gave a continuous variable (range 0 to 1). This variable was not used for the tabular learning as increasing the number of variables for 300 second turns the discrete states from 1000 to 300,000. This put the ANN method at an advantage.

### 5.1.2. Actions

In this Chapter agents are trained to place back bets first. Once a bet has been placed the agent can then place a lay bet to close the trade for a profit or loss. So when the agent currently has no bets on it has two action options:

1. Place a back bet to enter the market
2. Do Nothing

If the agent chooses not to enter the market it still has the action options above. If the agent has entered the market it now has two different options:

1. Place a lay bet to close the back bet and exit the market
2. Do Nothing

If the agent chooses to exit the market it now reverts to the original action options (place back bet or do nothing). If the agent does not exit the market the action options remain the same.

So the agent has four potential actions that are dependent on if it currently has a back bet in the market. For tabular Q-learning this means we have four action tables listing a Q-value for each state. For ANN Q-learning we use four ANNs, one for each action. This is the same method we used in the grid world example.

In Chapter 6 when we develop agents for real world testing we allow the agent to 'lay first' as well as 'back first'. To do this a further four actions are allowed. If no lay bets are currently on the selection the agent can choose to (1) place a lay bet to enter the market or (2) do nothing. If a lay bet is currently on the selection the agent can either (1) place a back bet to close the trade or (2) do nothing. The four extra actions mean that for ANN Q-learning the final agent will have 8 action ANNs.

### 5.1.3. Rewards

The agent receives a positive or negative reward for every time step that it holds a position in the market which is used in the learning algorithm. The reward associated with a state/action combination this time step is the change in profit due to remaining in that position until the next time step.

Figure 31 - Rewards example

| Time/Price | 3.05 | 3.1 | 3.15 | 3.2 | 3.25 | 3.3 | 3.35 | 3.4 | Bet Type | Price | Stake | Reward t+1 | Trade Open |
|------------|------|-----|------|-----|------|-----|------|-----|----------|-------|-------|------------|------------|
| t_0        | 10   | 8   | 6    | 40  | 50   | 250 | 150  | 120 | Back     | 3.25  | £4.44 | -0.0673    | Yes        |
| t_1        | 10   | 8   | 6    | 40  | 46   | 250 | 150  | 120 | None     |       |       | 0.1368     | Yes        |
| t_2        | 10   | 8   | 6    | 30  | 12   | 250 | 150  | 120 | Lay      | 3.2   | £4.51 | 0.0000     | No         |

Figure 31 gives an example of how the rewards are measured by the agent. At time t\_0 the agent places a back bet to win £10 at a price of 3.25. This requires a stake of £4.44. The reward of entering this initial bet is calculated using the following formula:

$$\text{initial reward for opening trade} = (\text{stake} \times \text{BackPriceTaken} / \text{BestLayPrice}_{t+1}) - \text{stake} \quad (2)$$

Unless the market is very fast moving this will generally be a negative value due to the spread between best back and best lay prices. In the case in Figure 31 placing a back bet to win £10 at time t\_0 at odds of 3.25 and having a nominal lay at 3.3 at time t\_1. If the trade was closed out immediately at this lay price to leave equal losses regardless of outcome a loss of around 7 pence would occur.

In time t\_2 the best lay price moves to 3.2. The reward one time step from t\_1 (and for all subsequent time steps that the bet is open) is calculated using formula (3):

$$\text{reward for future timesteps} = (\text{stake} \times \text{BackPriceTaken} / \text{BestLayPrice}_{t+1}) - (\text{stake} \times \text{BackPriceTaken} / \text{BestLayPrice}_t) \quad (3)$$

Using formula (3) for the example in Figure 31 results in a profit of around 14p for holding onto the bet in time step t\_1.

### 5.1.4. Learning Algorithm

Having established the potential actions, rewards and state representations the learning algorithm can be examined. Currently the agent learns by using the algorithm to loop through 'replays' of historic data rather than learning in real time which would potentially cost real money.

The algorithm was initially supplied with data one day at a time to train. This is usually between 80 and 140 greyhound races. Recordings of the current order book were taken every 100 milliseconds for the final five minutes of trading before the official race time resulting in around 6,000 lines of data for every selection every race. The data for the initial testing was recorded from October 2014 to January 2015.

The data was sorted by market, selection and then time. Up to the last 15 seconds before the start of the race lines of data where there was a gap between best lay and best back price were removed. This removed the

opportunity for the agent to trade at low value prices saving learning time. They were left in for the last 15 seconds to allow the agent to trade out regardless of price.

Lines 1-34 below outline the algorithm used by the Agent to learn how to trade using back bets to open and lay bets to close trades. A line by line commentary follows after outline.

(1) For 20 runs through the day's data ( $j=\text{runNumber}$ ):

(2)  $\text{LearningRate} = 0.6/(j+1)$

(3) For 1 run through all the time steps for every runner of the day ( $i=\text{line of data}$ ):

(4) If  $\text{runnerID}(i)$  does NOT equal  $\text{runnerID}(i-1)$ :

(5) Reset  $\text{backPriceTaken} = 0.0$ ;  $\text{stake} = 0.0$ ;  $\text{betOn} = 0$

(6) If  $\text{betOn} = 1$ :

(7)  $Q(\text{doNothing}) = \text{table/neural network state representation action not close trade Q value}$

(8)  $Q(\text{doTrade}) = \text{table/neural network state representation action to close trade Q value}$

(9) If  $\text{betOn} = 0$ :

(10)  $Q(\text{doNothing}) = \text{table/neural network state representation action not open trade Q value}$

(11)  $Q(\text{doTrade}) = \text{table/neural network state representation action to open trade Q value}$

(12) If  $\text{time to race} > 30 \text{ seconds}$ :

(13) If  $\text{betOn} = 0$ :

(14) If  $\text{randomNumber} < \text{openChance}/(j+1)$ :

(15) Place a back bet to win £10 at the current best back price available

(16) Set the reward using equation (2) above and  $\text{betOn} = 1$

(17) Else:

(18) If  $Q(\text{doTrade}) > Q(\text{doNothing})$  and  $\text{randomNumber} > (0.5/(j+1))$ :

(19) Place a back bet to win £10 at the current best back price available

(20) Set the reward using equation (2) above and  $\text{betOn} = 1$

(21) If  $\text{betOn} = 1$ :

(22) If  $\text{randomNumber} < \text{closeChance}/(j+1)$ :

(23) Place a lay bet to close the trade at the current best lay price available

(24) Set the reward to zero above and  $\text{betOn} = 0$

(25) Else:

(26) If  $Q(\text{doTrade}) > Q(\text{doNothing})$  and  $\text{randomNumber} > (0.5/(j+1))$ :

(27) Place a lay bet to close the trade at the current best lay price available

(28) Set the reward to zero and  $\text{betOn} = 0$

(29) If  $\text{betOn} = 1$  AND time to race  $< 30$  seconds AND no gap between best back and best lay price:

(30) Place a lay bet to close the trade at the current best lay price available

(31) If  $\text{betOn} = 1$  AND time to race  $< 15$  seconds:

(32) Place a lay bet to close the trade at the current best lay price available

(33) UPDATE THE Q-VALUES (method for table and NN system explained in Chapters 3 and 4)

(34)  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s, a)]$

The algorithm runs through all the races in the day 20 times (Line 1). The learning rate for the Q updates is initially set at 0.6 (Line 2). This reduces depending on the number of times the data has been passed through until phase 20 when it is  $0.6/20 = 0.03$ . This learning rate is fixed for each run through the day's races. The Agent then runs through the day's races (Line 3). The settings are reset for the agent if the selection changes through the data (Lines 4-5). Each selection in each race is regarded as a separate opportunity to trade. No knowledge about trading on the other selections in the race is taken into account.

The agent then checks if it currently has a back bet on the selection. If it has a back bet on it calculates its current Q value for the action of 'do nothing' and 'do trade' dependent on its current state (Lines 6-8). If no trade is currently on, the agent does the same thing to get Q values for opening a new trade or doing nothing (Lines 9-11).

Lines 12-28 shows how the agent acts from 5 minutes to 30 seconds before the start of the race. Traditionally Q-learning and Deep Q-learning (DQL) algorithms have used a policy that starts with a high amount of random exploration. Slowly it becomes more likely to follow the optimal current policy by performing the action with the highest Q value. With trading, the majority of time steps the agent will do nothing because the spread between back and lay prices makes overtrading costly. Using sequential data to learn to trade more randomness has to be brought into the algorithm otherwise the agent will open the trade in the same place too often. For this reason our algorithm will only choose to randomly open or close a trade with a low probability. These rates decrease through trading as they divided by  $(j+1)$ .

Lines 13-20 are used if the agent does not currently have a back bet on. The agent initially decides whether to open a bet based on pure chance (Line 14). The chance opening a bet here is set initially at 0.009. So initially roughly 1 in 100 times when the agent currently does not have an open bet and the gap between best back and best lay price is at the minimum level a back bet will be placed (Lines 15-16 record this). As the number of times the day is passed through increases the chance of opening a bet by random decreases. On the 20<sup>th</sup> pass the chance is  $0.009/20 = 0.00045$ .

If no bet is currently outstanding and a bet was not placed there is a chance a bet will be opened if the Q-Value for opening a bet exceeds the Q-Value for doing nothing (Line 18). This chance increases as the number of times the data is passed through increases. It starts at a 0.5 chance and ends at a  $(1-(0.5)/20) = 0.975$  chance. The bet is placed and stored Lines 20-21.

Lines 22-28 are used by the Agent if a bet is currently open. A bet can either be closed from a small random chance (Lines 22-24) or through being the best Q-Value (Lines 26-28). The same equations are used as the ones for opening a bet. The only difference is that the initial probability of closing a bet by random was lower than opening one. The level was set at 0.005 to stop the agent overtrading and increasing learning speed.

In the last 30 seconds before the start of the race no more trades are opened. This is designed to ensure that the trades are always closed off to reduce volatility in returns. From 30-15 second before the off the trade is closed only if there is no tick gap between the best back and best lay price (Lines 29-30). For the final 15 seconds before the off the bet is closed at the best lay price regardless of any gap in the market prices (Lines 31-32).

At every time step the agent's Q-Values for the current state and action is updated (Line 34). The exact mechanism for this depends on how the state/action representations are done and if a function approximation method is used. These will be discussed for the table and neural network systems in the next section.

## **5.2. Developing Agents**

The initial agents were trained using 114 dog races from 15/10/2014 and tested on 108 races from 16/10/2014. We tested agents using Q-tables and agents that store Q-value estimates in ANNs.

### **5.2.1. Using Q-Tables**

The first method of learning the Q values was to use a table system. The encoding system described in Figure 22 consisted of a 6 digit integer value that represented the current state of the order book for a particular selection to the agent. A seventh digit was added with '1' representing that a bet was currently open and '0' if no bets were open. This doubled the number of possible states from around 600 to over 1200.

#### **5.2.1.1. Example of the Algorithm Training One Day**

30 agents were trained for a total of 20 passes through the day's data. The random nature of the exploring and exploiting current knowledge meant that each agent learnt a slightly different set of Q-values. The agent starts with two possible actions; place a back bet at the best price currently available to win £10 or do nothing. If a back bet is currently on the agent then has two different options; close the back bet by placing a lay bet at the best lay price available or do nothing.

Figure 32 shows that all the agents achieved slightly different results. Generally the agents performed better the more times they had passed through the data. The relationship between using the best-Q actions for the training and test data were highly correlated. When the agent performed well using the in-sample training data it generally also performed well using the out of sample test data.



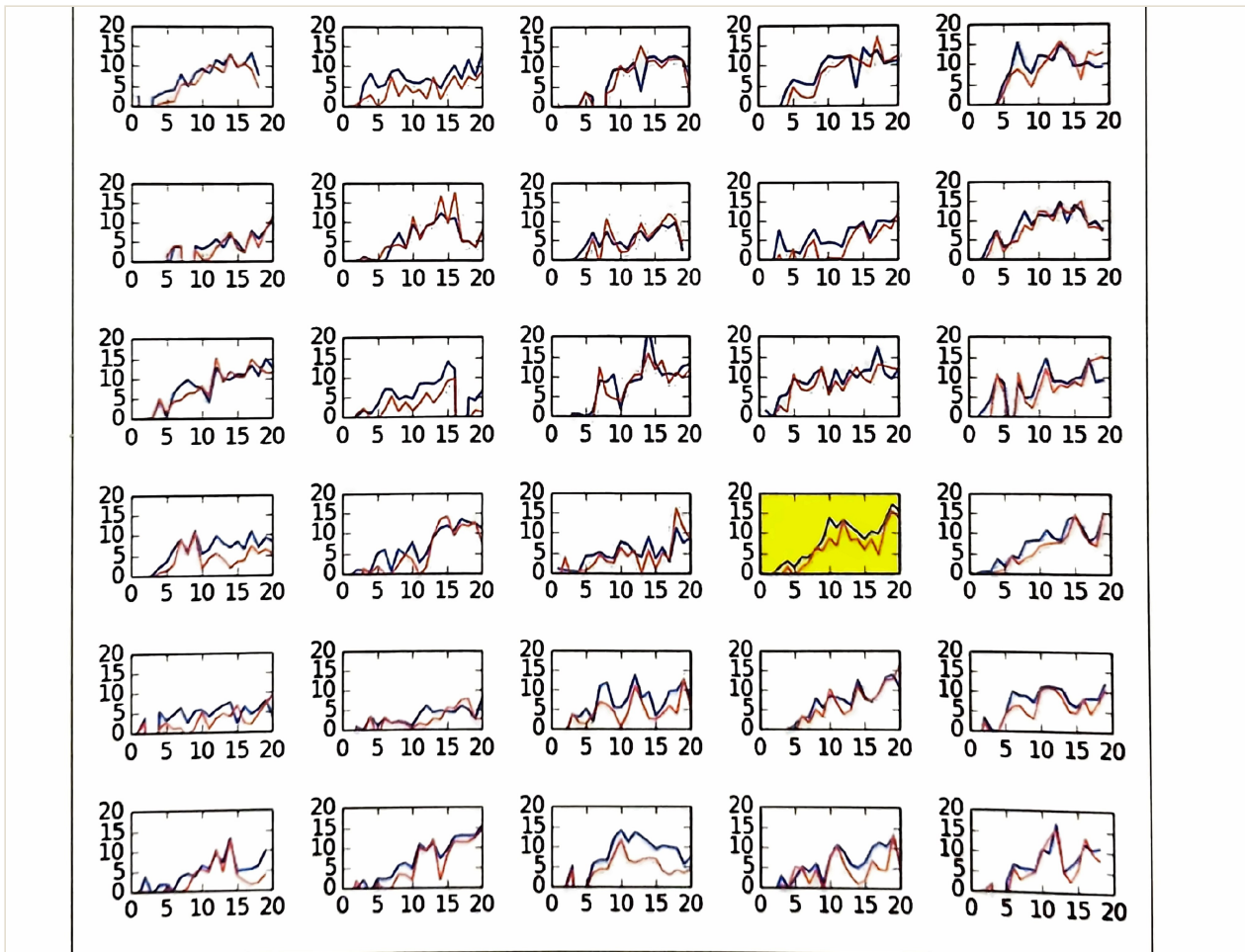


Figure 32 - The 30 mini-charts above each show the performance of a different agent trained for one day using tabular Q-Learning. The X-axis is the number of times the agent has passed through the day's data to learn. At the end of each pass the performance using the best-Q action is measured for the training day (blue) and then the test day (red). The y-axis is the amount of profit returned from following the best Q action betting to win/lose a nominal £10 each bet. The agent depicted in the chart with the yellow background was used for the example below.

Not all 30 agents continued to learn throughout the 20 learning cycles. Some peaked their performance earlier. In order to analyse what the agents are actually doing we developed a charting system to show the current Q-values for a number of different states. This is used throughout the Chapter. Figures 33 and 34 are examples of two tabular agents. Five different states are analysed for when the agent does not currently have a bet in the market. We choose five states where there was more money in the market waiting to back the greyhound than to lay it. We choose these states as the weight of money suggests the price may reduce in the future so an agent is more likely to place a back bet now.

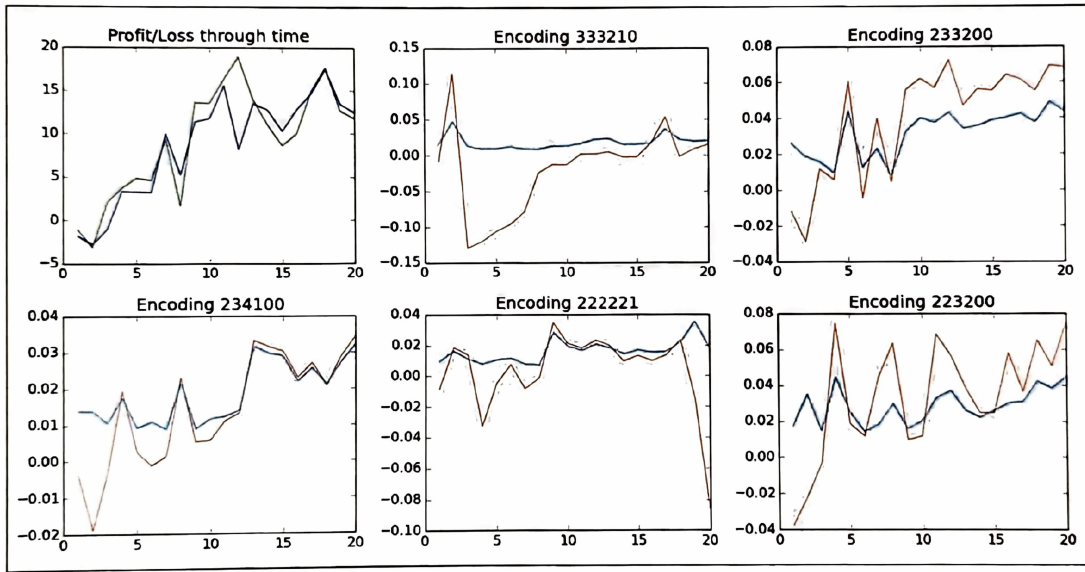


Figure 33 – Q values for open back bet and do nothing for different states during the learning cycle for one agent. The top left chart shows the profit achieved by the agent following the best Q-values (blue training day, green test day). The x-axis is the number of times the agents has been through the races. The remaining five charts show a snap shot of the Q-values for 5 different states when a bet is not currently open. The red lines show the Q-values for opening a back bet. The cyan lines show the Q-values for the 'do nothing' action. If the red line is above the cyan line it means that the agent would choose to place a back bet at that time. (There are a number of these charts in this chapter)

Figure 33 - Q values for open back bet and do nothing for different states during the learning cycle for one agent. The top left chart shows the profit achieved by the agent following the best Q-values (blue training day, green test day). The x-axis is the number of times the agents has been through the races. The remaining five charts show a snap shot of the Q-values for 5 different states when a bet is not currently open. The red lines show the Q-values for opening a back bet. The cyan lines show the Q-values for the 'do nothing' action. If the red line is above the cyan line it means that the agent would choose to place a back bet at that time. (There are a number of these charts in this chapter)

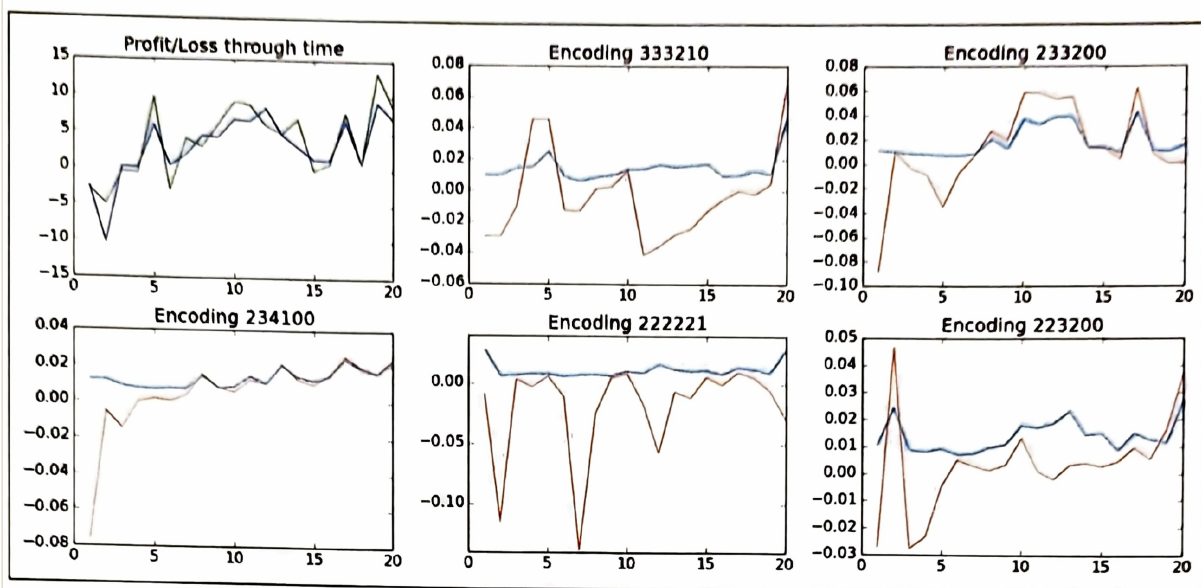


Figure 34 – Example from another tabular agent. (See Figure 33 for full explanation.)

Figures 33 and 34 show two different tabular agents that have been trained using exactly the same procedure. We can see they have both learnt different Q-values through time. For instance, for the Q-values for placing a back bet for state '223200' the agent in Figure 33 oscillates from being under and over the Q-values for doing nothing. This means that the agent is sometimes choosing to place a bet and then changing its mind to not place a bet after looking at the races again. The agent in Figure 34 for the same state decides not to place a bet through most of the training. These charts show how agents trained the same way can learn different Q-values and thus learn to take different actions.

#### **5.2.1.2. Action Example**

The agent with the yellow background in Figure 32 was chosen to provide an example of an agent trading on one individual greyhound. Figure 35 shows the back and lay prices and the action decisions made the agent for this example.

The model assumes that there is zero latency between the agent deciding to bet and the bet being placed; i.e. the price does not change in the time it takes to place the bet. It also assumes that the full and exact bet requested can be placed.

For the price chart in Figure 35 an optimal agent would back the greyhound at the two peaks and close the trades at the two troughs (Open at 3.55 at around 80 seconds and close at 3.05; then open at 3.65 and close at 2.78). Betting to win £10 on the initial backs would have resulted in a profit of £1.82 for the race. This is 67% more than our best agent. However, it is extremely unlikely that an agent could be trained to always pick the top and bottom of every trend. It should also be noted that our agent is making decisions based on the current order book values and not historical prices.

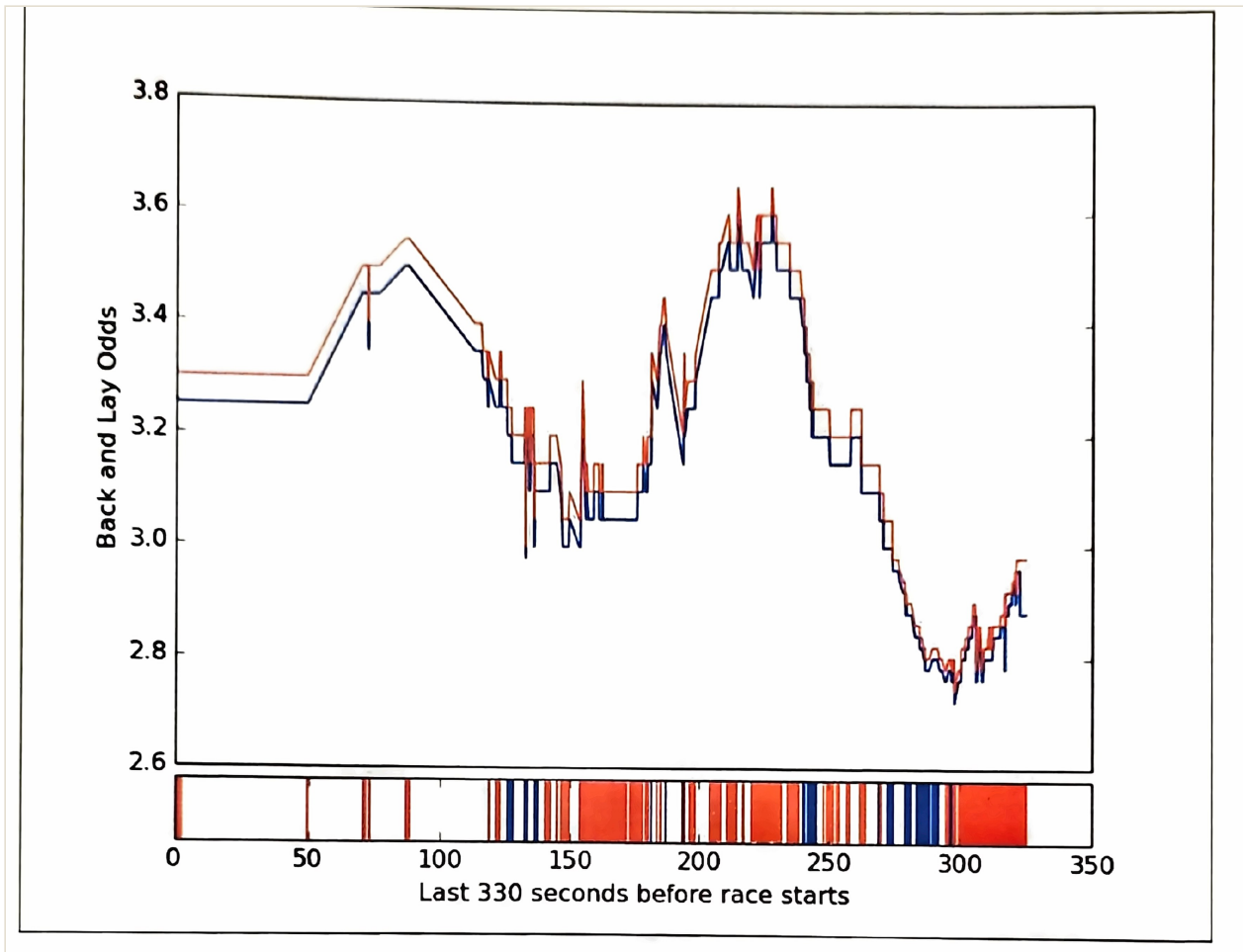


Figure 35 – Trading chart for one greyhound example. The top part of the figure shows the best prices available to back (blue) and lay (red). The colour block below shows the action choices made by the agent following the state/action Q-Value. The blue bars show when the agent's best action is to open a back bet. The red bars are when the agent should close the bet (if one is open). The white areas are where the agent should do nothing.

Figure 36 shows the actual bets that the agent makes. It initially places a back bet after 125.5 seconds at a price of 3.2 because the agent's Q-values to open a bet exceeded the agent's Q-values to do nothing for the state at that time. After 7 seconds this trade is closed out at odds of 3.0 resulting in a notional profit of 30p. The trade was closed because the agents Q-values to close the bet exceeded the agent's Q-values to do nothing for the state at that time. The agent then continues to open and close another 7 trades on this selection before the race starts. The total nominal profit for the agent is £1.09. This process is repeated for every selection every race throughout the whole day.

Figure 36 – Bets placed by the agent on one greyhound. Each line pair shows the back and lay price, amount traded, profit and the time that the trade was on for.

| Time  | Bet Type | Odds | Stake | Profit/Loss | Time Trade Open |
|-------|----------|------|-------|-------------|-----------------|
| 125.5 | back     | 3.2  | £4.55 |             |                 |
| 132.5 | lay      | 3    | £4.85 | £0.30       | 7.0             |
| 132.9 | back     | 3.15 | £4.65 |             |                 |
| 139.5 | lay      | 3.15 | £4.65 | £0.00       | 6.6             |
| 180.8 | back     | 3.3  | £4.35 |             |                 |
| 182.9 | lay      | 3.3  | £4.35 | £0.00       | 2.1             |
| 186.3 | back     | 3.35 | £4.26 |             |                 |
| 193.3 | lay      | 3.2  | £4.45 | £0.19       | 7.0             |
| 193.6 | back     | 3.3  | £4.35 |             |                 |
| 194.0 | lay      | 3.25 | £4.41 | £0.06       | 0.4             |
| 239.8 | back     | 3.35 | £4.26 |             |                 |
| 247.6 | lay      | 3.25 | £4.39 | £0.13       | 7.8             |
| 268.5 | back     | 3.1  | £4.76 |             |                 |
| 268.7 | lay      | 3.1  | £4.76 | £0.00       | 0.2             |
| 271.9 | back     | 3    | £5.00 |             |                 |
| 294.2 | lay      | 2.78 | £5.40 | £0.40       | 22.3            |

The problem with using a table to store Q values is that discrete inputs have to be used to represent the states. If more inputs are added the number of state/action options in the table grows exponentially. For instance, adding another a time to go input with 50 options each would increase the table size from around 1200 to 60,000. The table quickly becomes unmanageable and fails to generalise to new situations. Some of the options are so rare when an agent sees that situation it is unsure what to do. In order to eliminate this curse of dimensionality a function approximation method is often used.

### 5.2.2. Using ANNs

The ANNs Q function approximation method introduced in Chapter 4 is now used instead of Q-tables. Four action neural networks are trained. Two are for when no bet is open; one with the action to open a back bet and one with the action to do nothing. The other two ANNs are for when a bet is on. The actions are to either close the back bet or do nothing.

The state representation consisted of the 6 inputs suggested in Figure 30. In addition to this an extra input is added to provide the agent with information as to how long there is to the start of the race. All the inputs are continuous and scaled to between 0 and 1 to ensure that the neural network can learn without the Q-values exploding. Initially simple ANNs with one hidden layer containing 2 hidden neurons were used. After some initial testing the learning rates for the neural networks were set 0.1. The weights were set initially in a random normal distribution with a mean of zero and a standard deviation of 1. After some testing we reduced the size of the weights by dividing the biases by 10 and other weights by 3 to reduce their sizes.

### 5.2.2.1. Example of the Algorithm Training One Day

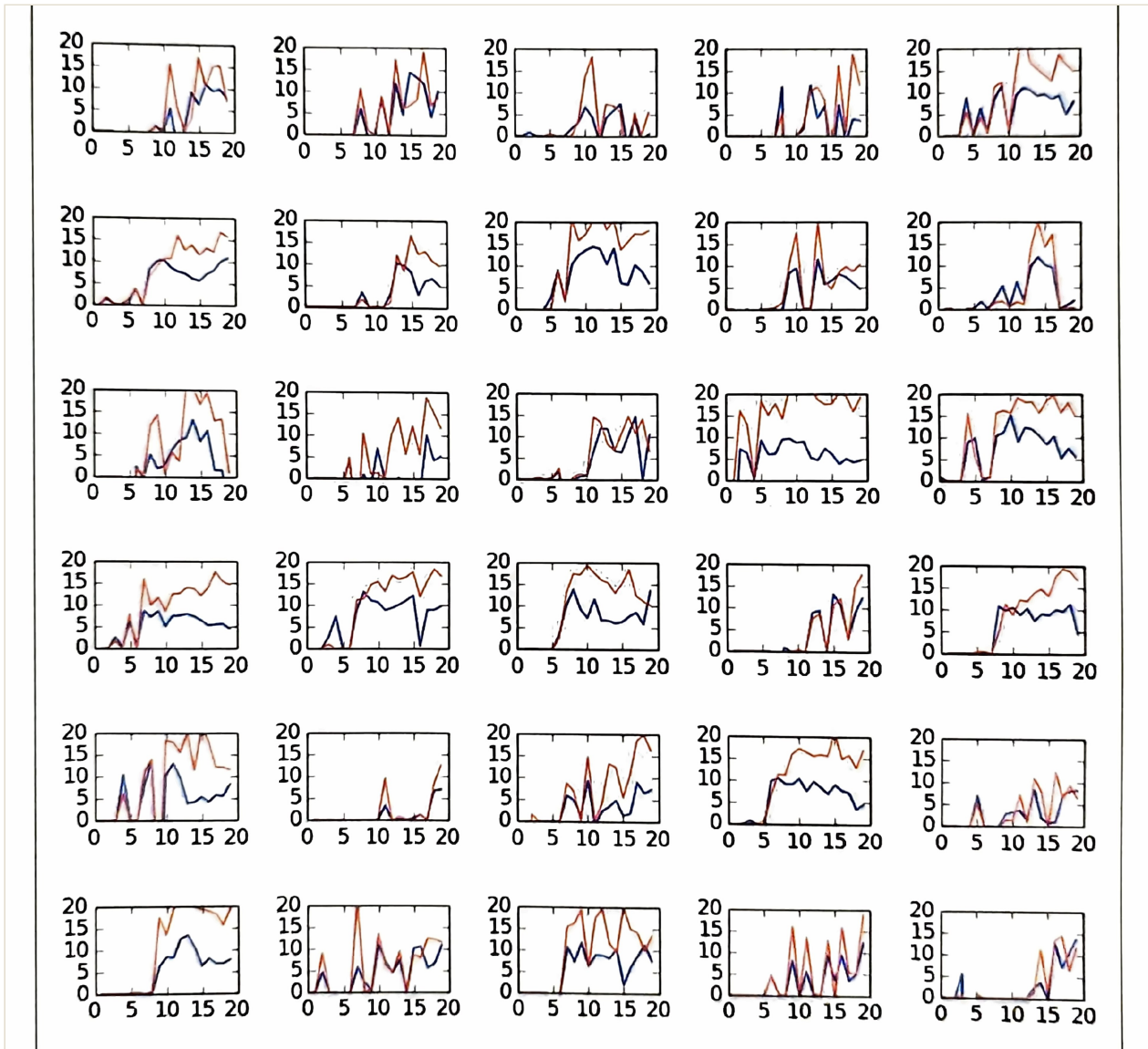


Figure 37 – The 30 mini-charts above are the same representation as in Figure 32 above. The only difference is the Q-values are estimated using a neural network rather than a table.

30 agents were trained in a similar manner to the tests for the tabular Q-Learning above. The only difference is that the ANNs are trained to approximate the Q-values rather than tables. The initial network structure was very simple. Just one hidden layer with two hidden units was used. This was to allow quick testing to see if ANNs agent could learn in this environment. Figure 37 shows the best performances of the 30 agents using the best Q-estimates to make decisions for 20 passes through the day.

The results in Figure 37 show that the agents find it harder to learn using an ANN than using a table for this simple network configuration. Most of the agents have found a profitable system after 20 passes through the data but their performance is varied. It also appears that in most cases the agent performs better on the out of sample test day (red) than the in sample training day (blue). In Figures 38 and 39 we trained the agents using the second day and tested them on the first day. Those agents performed better on the in sample day. This demonstrates that the races on some days might follow the weight of money more closely.



Figures 38 and 39 examine the Q-value outputs from two different ANN agents as they learn for the same states as Figures 33 and 34 above. The agent in Figure 38 shows a wide oscillation through time for the Q-values to open a bet (the red lines) for the different states examined. For most of the states the agent keeps changing its mind as to whether it wished to place a bet or not. This is reflected in the profit/loss chart for the agent which is unstable through time. The agent in Figure 39 also has oscillating Q-values but the values for the actions to open and close a bet do not cross each other so often. This agent changes its mind less after through training for the examined states.

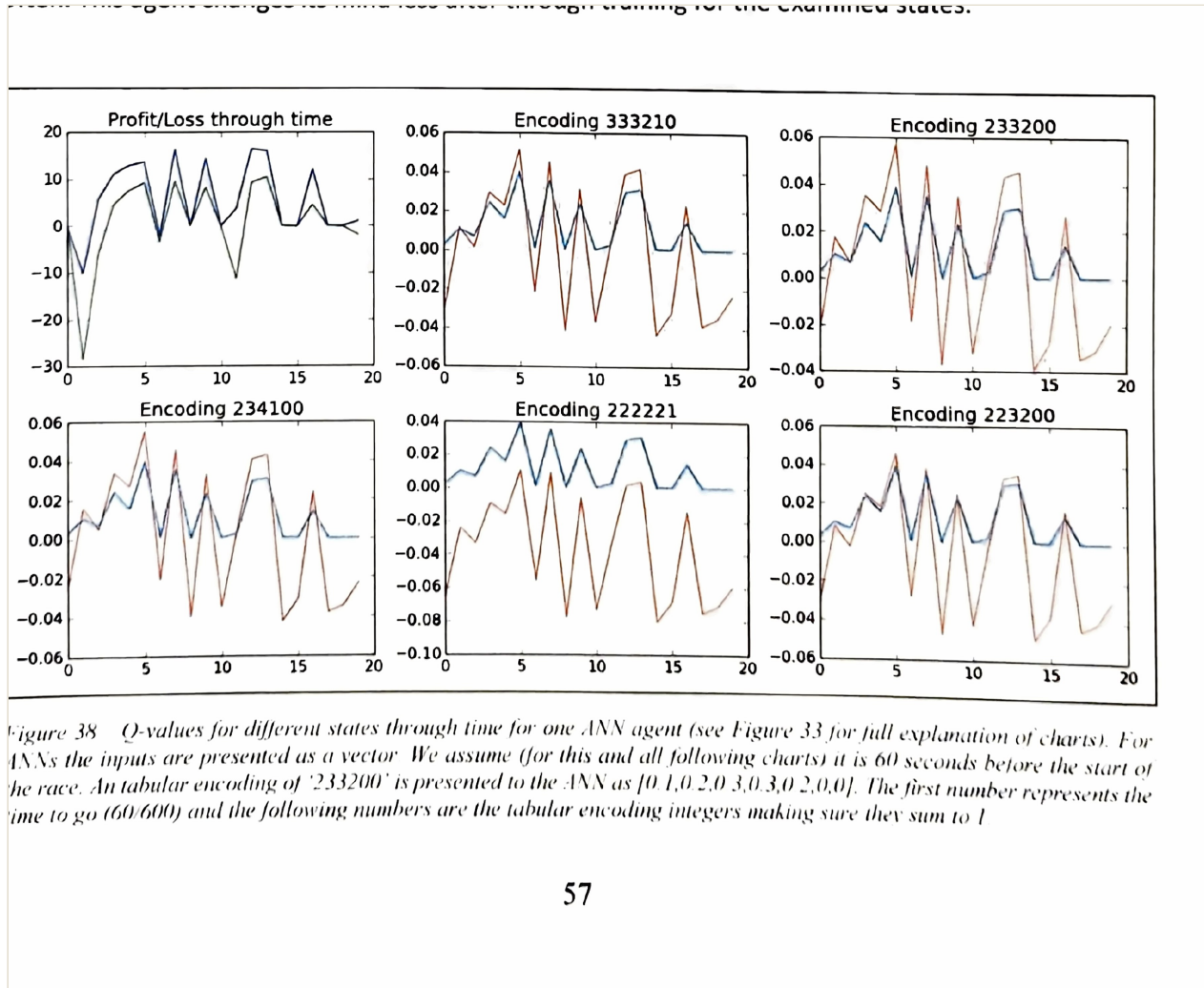


Figure 38 – Q-values for different states through time for one ANN agent (see Figure 33 for full explanation of charts). For ANNs the inputs are presented as a vector. We assume (for this and all following charts) it is 60 seconds before the start of the race. An tabular encoding of '233200' is presented to the ANN as [0.1,0.2,0.3,0.3,0.2,0,0]. The first number represents the time to go (60/600) and the following numbers are the tabular encoding integers making sure they sum to 1.

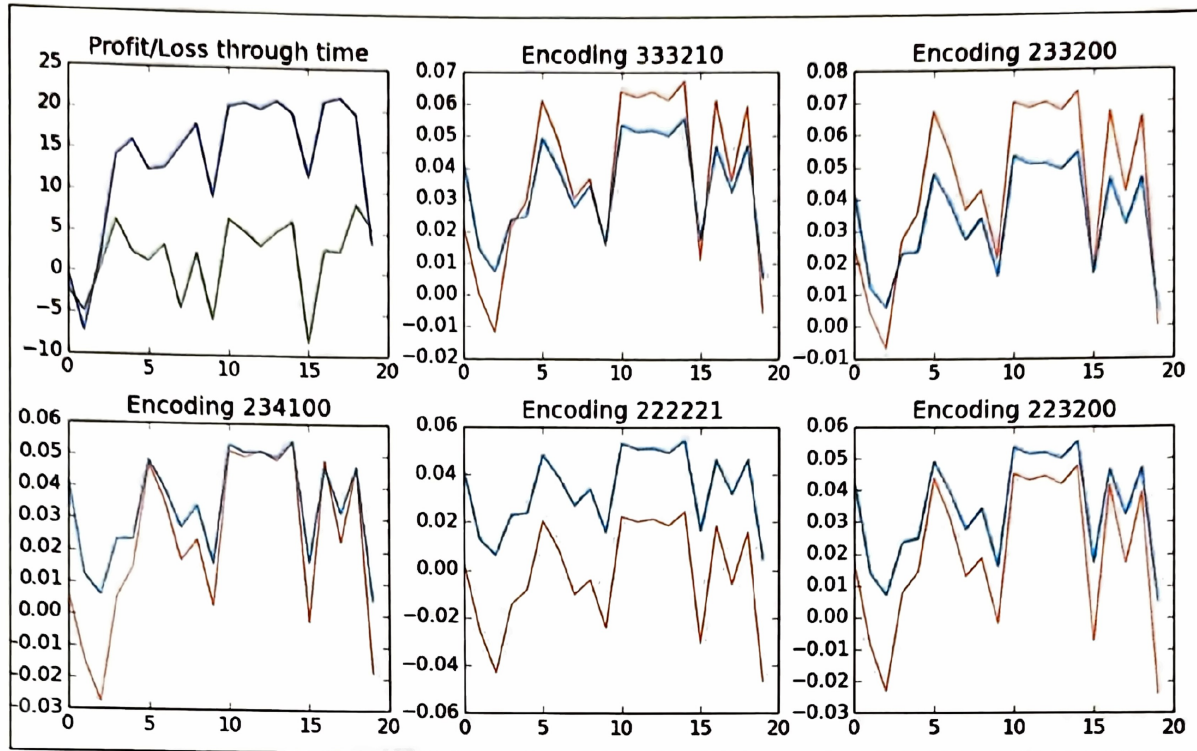


Figure 39 -  $Q$ -values for different states through time for another ANN agent (see Figure 33 for full explanation of charts)

Figure 39 -  $Q$ -values for different states through time for another ANN agent (see Figure 33 for full explanation of charts)

We have seen in this section that agents with only two hidden units in a single layer are unstable. This is similar to the results from the grid world example in the previous Chapter. Two hidden units may not be enough to fully represent the inputs the agent receives. The unstable learning of the agents mean that many agents should be trained before picking the best one for real trading. The high correlation between training and validation races suggest that if a high performing agent can be trained it will probably perform well. In the next section we look at improving and stabilising training using deeper and wider ANNs.

### 5.3. Wider and Deeper ANNs

The neural networks used in the previous section had 7 inputs, 2 neurons in one hidden layer and one output. The datasets consisted of one day for training and one day for validations. Figure 37 demonstrated how unstable learning with these network structures was. In Chapter 4 we also found that ANNs were unstable when they only had two hidden units. For grid world the results showed that increasing by the width and depth of the ANNs improved the performance of the  $Q$ -learning agents.

In this section we will experiment with deeper and wider neural networks for the trading agents to see if their performance can also be improved. We will also use a larger set of races for the training and testing. For the training and testing of deep and wide ANNs we used all the same parameter settings as in 5.2.2. We were only able to test one agent per network structure on the larger dataset as each test took 3-6 hours depending on the network depth and width.

### 5.3.1. Larger Datasets

In 5.2 agents were trained using 114 dog races from 15/10/2014 and tested on 108 races from 16/10/2014. In this section and for training the agents for real world testing we use a larger number of races. The new datasets were split by track. The nine tracks with the most number of races were used. The training races were taken from between 2<sup>nd</sup> October 2014 and 1<sup>st</sup> December 2014. The validation races were from 2<sup>nd</sup> December 2014 and 25<sup>th</sup> January 2015. These new datasets provide more races for training the agents with races spread out over a longer time frame. This should iron out any issues with particular days containing rogue data. This could occur due to problems in data collection, problems with the Betfair API or just different market conditions caused by changes in strategy by one or two big players in the market.

In this Chapter we use the races from Romford for testing deeper and wider ANNs for trading. Romford was chosen because it has the largest number of races over the 4 months the data was collected. 639 races were used for training and agents were tested on 685 races.

Figure 40 - Increased sample size for testing. The most used nine tracks in the UK.

| Track        | Number of Training Races | Number of Validation Races |
|--------------|--------------------------|----------------------------|
| Belle Vue    | 228                      | 278                        |
| Crayford     | 478                      | 451                        |
| Hove         | 541                      | 565                        |
| Monmore      | 446                      | 466                        |
| Newcastle    | 543                      | 511                        |
| Perry Bar    | 264                      | 245                        |
| Romford      | 639                      | 685                        |
| Sheffield    | 394                      | 411                        |
| Sunderland   | 508                      | 439                        |
| <b>Total</b> | <b>4041</b>              | <b>4051</b>                |

### 5.3.2. Wide ANNs for Trading

In 4.3.3 we found that when we increased the number of units in a single layer for grid world agents from 2 to 20 performance improved. Further increasing the number of units resulted in declining performance. For trading, we trained 6 different agents with different numbers of hidden units in a single hidden layer. Figure 41 shows the results of training with 2, 4, 7, 14, 28 and 56 hidden units.

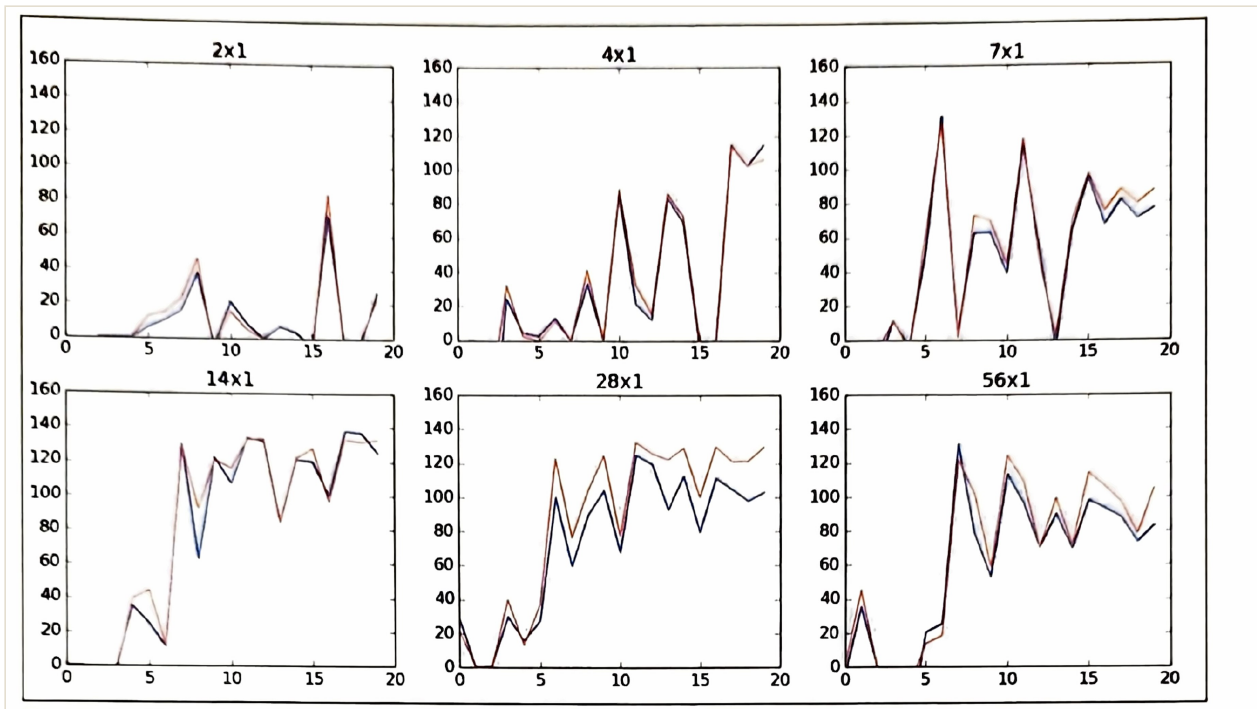


Figure 41 – Shows the performance of single layer ANNs agents of different widths by number of training races learning to back first. The y-axis show the average profit per race when the agent follows the highest action Q-value. The x-axis represents the number of times the agent has gone through the races in the training data. The blue lines are profits on the training races and the red lines are profits on the validation races.

The best performance agent had 14 hidden units. As with grid world we see improvements as the number of hidden increases from 2 to 14 units. There is a slight decline when the number is increased from 28 to 56.

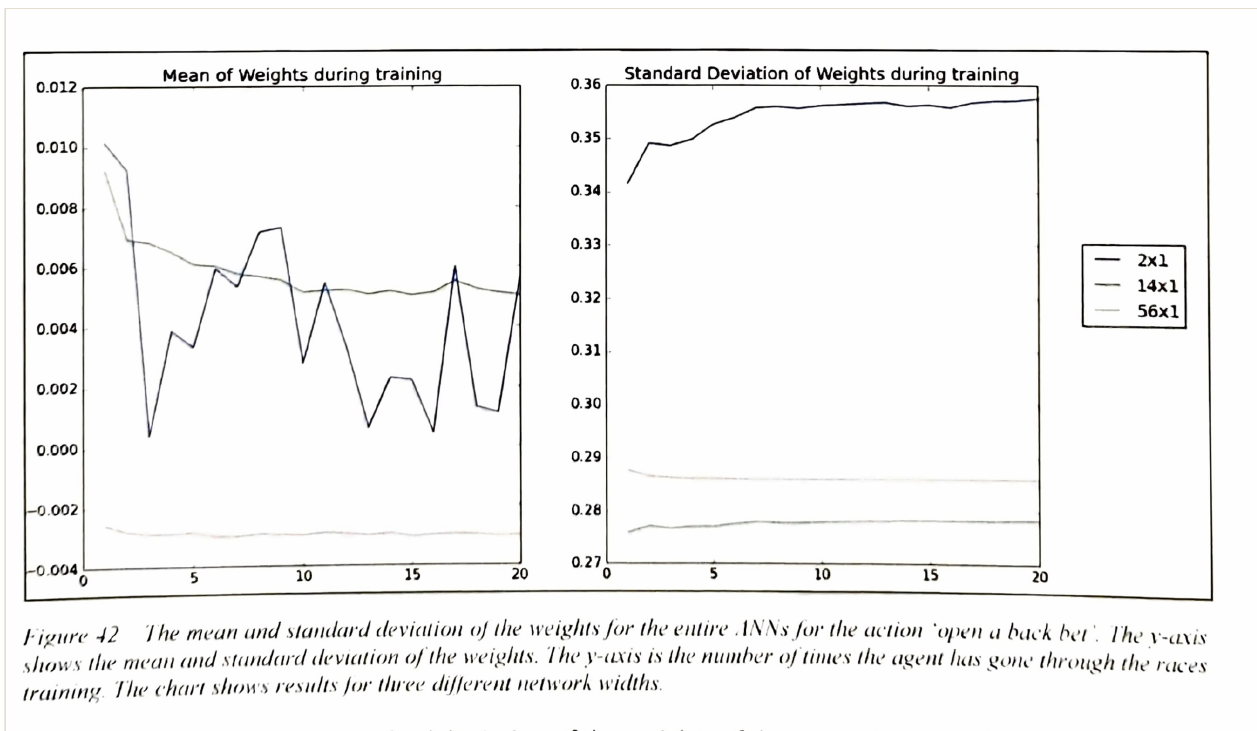


Figure 42 – The mean and standard deviation of the weights for the entire ANNs for the action 'open a back bet'. The y-axis shows the mean and standard deviation of the weights. The x-axis is the number of times the agent has gone through the races training. The chart shows results for three different network widths.

Figure 42 – The mean and standard deviation of the weights for the entire ANNs for the action 'open a back bet'. The y-axis shows the mean and standard deviation of the weights. The x-axis is the number of times the agent has gone through the races training. The chart shows results for three different network widths.

In Figure 42 the mean and standard deviation of the weights of the networks through time is presented for the 'open back bet' action ANN for 2, 14 and 56 hidden units. The average weights for the ANN with 2 hidden units are very variable oscillating up and down much more than the other networks. The standard deviation of the weights also slightly increases. This is reflected in the Q-value outputs. Figure 43 shows the outputs for five encoded states for the 2 hidden unit ANN. For all five states the Q-values for the 'open back bet' action (red lines) oscillated as the weights oscillate. For the states examined the red coloured lines never go over the cyan coloured lines so no bets would be opened.

For the 14 hidden unit ANN the mean of the weights slowly and smoothly decreases through time in Figure 42. This is reflected in Figure 44 where the Q-values change very smoothly. For encoding '333210' there is stable learning with the red line exceeding the cyan line (triggering a bet) after 7 runs through the data and then remaining there for the rest of training. Encodings '233200' and '223200' have very close Q-values for 'open bet' and 'do nothing'. This could be the cause of the slight oscillations in results once the near best performance is found.

For the 56 hidden unit ANN Figure 42 shows small (slightly negative) and unchanging average weights. Performance is slightly lower for 14 hidden units but not greatly. Figure 45 shows the estimated Q-values are relatively stable. This is a contrast from the grid world results where the average weights were bigger with the greater number of hidden units leading to saturated neurons. One reason for this might be that the rewards for trading are both positive and negative rather than just positive for grid world. The agent is also rewarded every time step a bet is open compared to only when the fly is found. This could help stabilise learning.

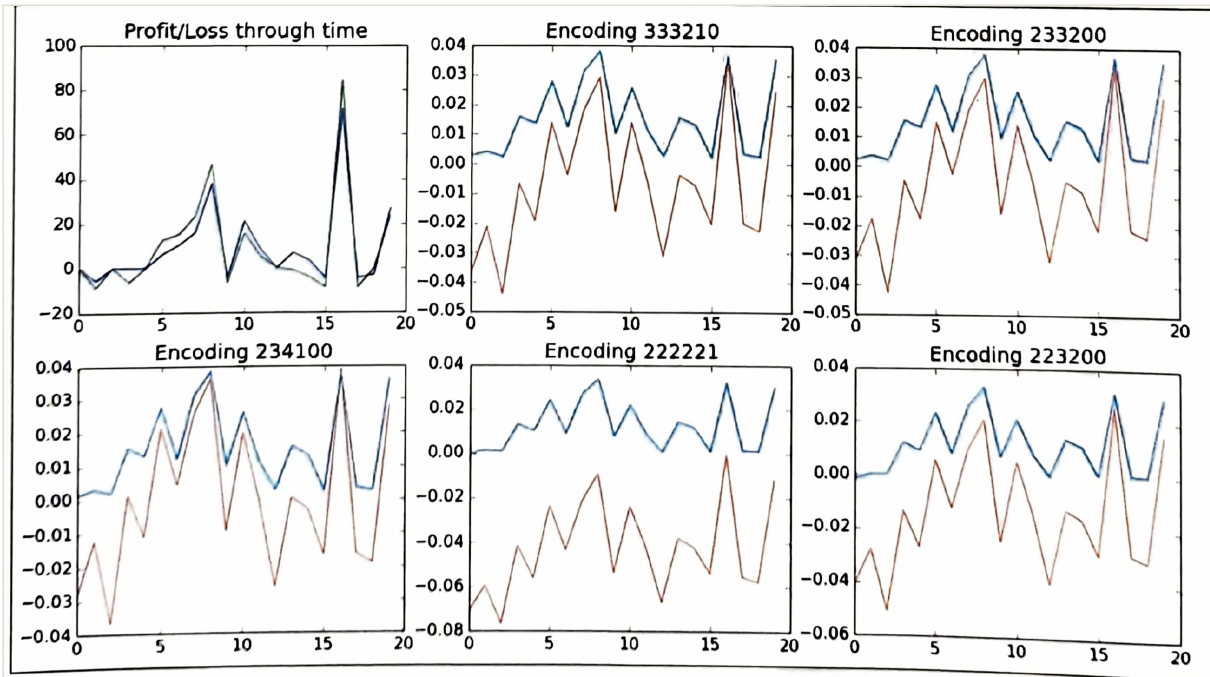


Figure 43 – Q-values for different states through time for an ANN agent with 2 hidden units in a single hidden layer (see Figure 33 for full explanation of charts)

Figure 43 – Q-values for different states through time for an ANN agent with 2 hidden units in a single hidden layer (see Figure 33 for full explanation of charts)



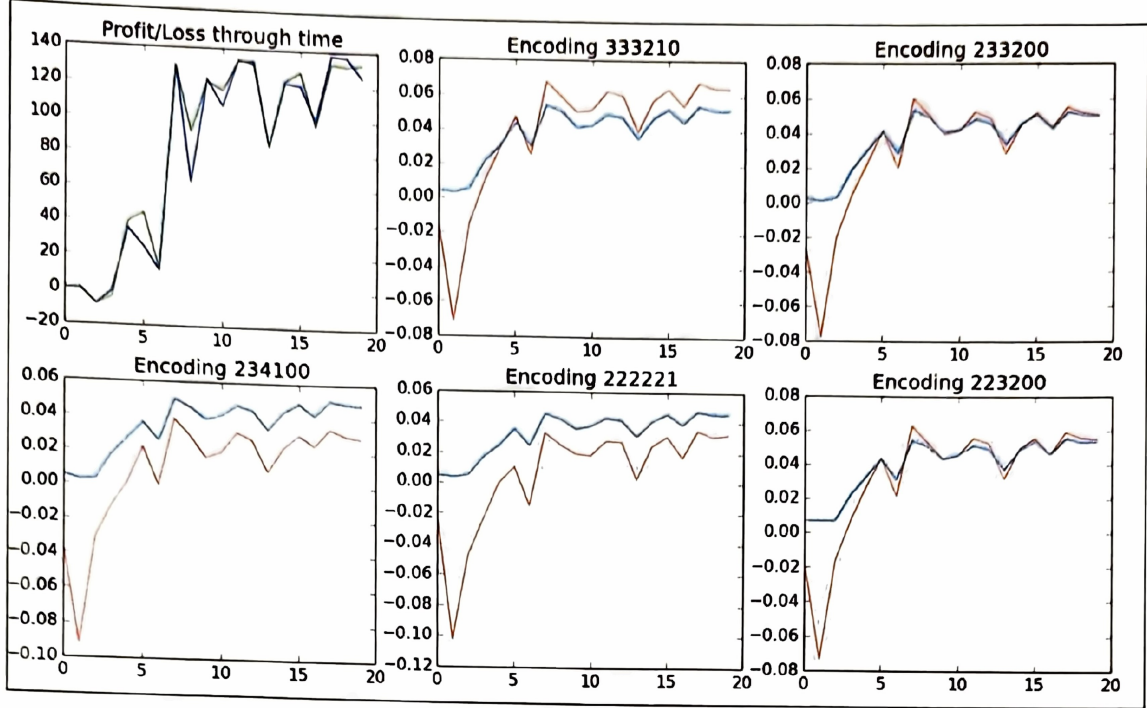


Figure 44 – Q-values for different states through time for an ANN agent with 14 hidden units in a single hidden layer (see Figure 33 for full explanation of charts)

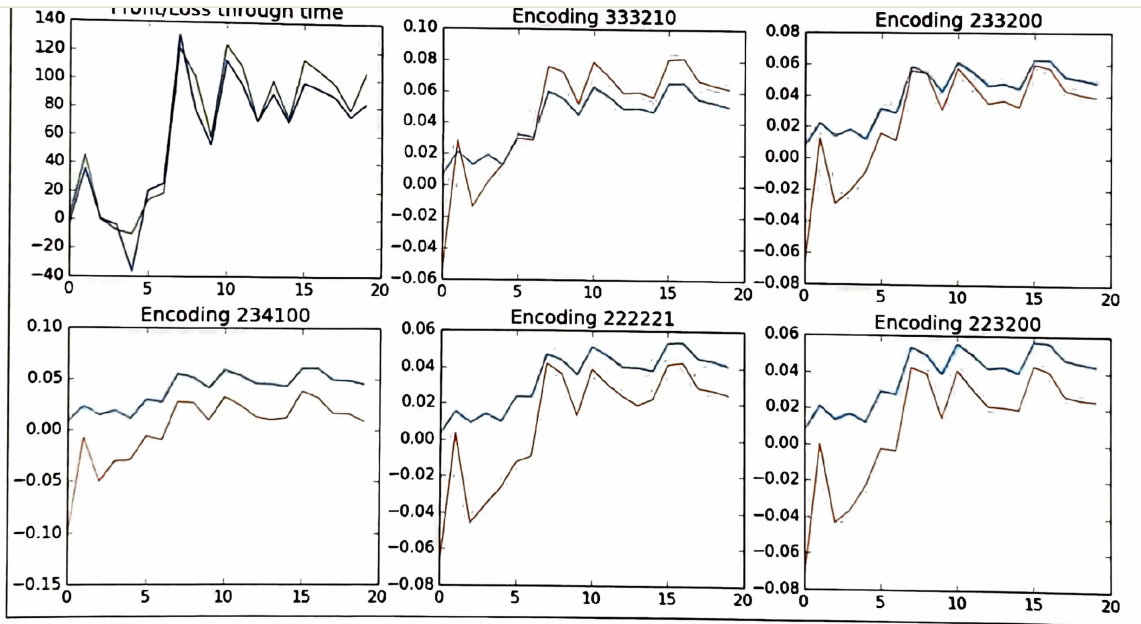


Figure 45 – Q-values for different states through time for an ANN agent with 56 hidden units in a single hidden layer (see Figure 33 for full explanation of charts)

Figure 45 – Q-values for different states through time for an ANN agent with 56 hidden units in a single hidden layer (see Figure 33 for full explanation of charts)

### 5.3.3. Deep ANNs for Trading

In the grid world testing performance increased as the number of hidden layers of the ANNs were increased. Six agents with 14 units per layer were trained with the same parameter settings as 5.3.2.



The agents had 1-6 hidden layers. Figure 46 shows that as the number of hidden layers increase the performance of the agents decline. This is especially the case after four hidden layers.

In the grid world testing the average network weights, and standard deviation of those weights, decreased as the number of layers increased (Figure 26). Figure 47 shows this was not the case for the trading agents we trained. Increasing the number of hidden layers resulted in an increased absolute value of the mean of the weights and an increase in their standard deviation. Figure 48 shows that these higher weight values with higher standard deviation resulted in less stable Q-values during training. This can be seen by comparing Figure 44 for 1 hidden layer and Figure 48 for 6 hidden layers.

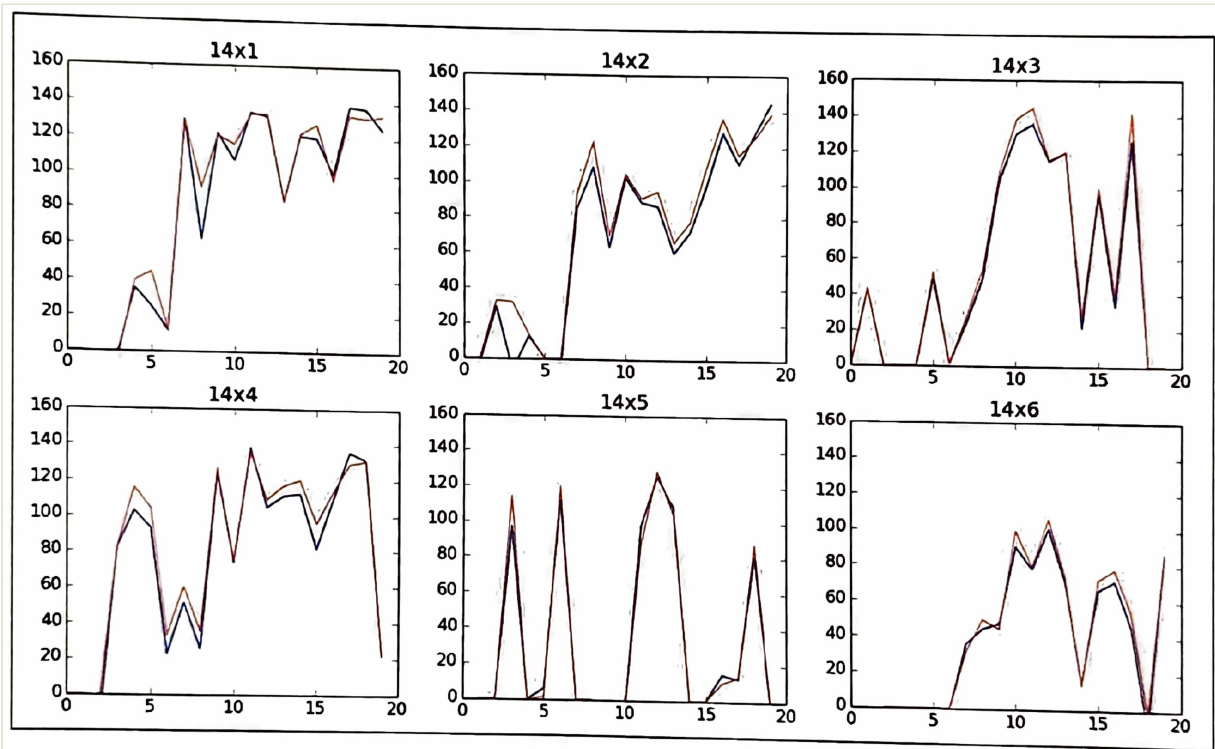


Figure 46 - Shows the performance of ANNs agents of different depths with 14 hidden units in each layer. See Figure 42 for full explanation

Figure 46 - Shows the performance of ANNs agents of different depths with 14 hidden units in each layer. See Figure 42 for full explanation

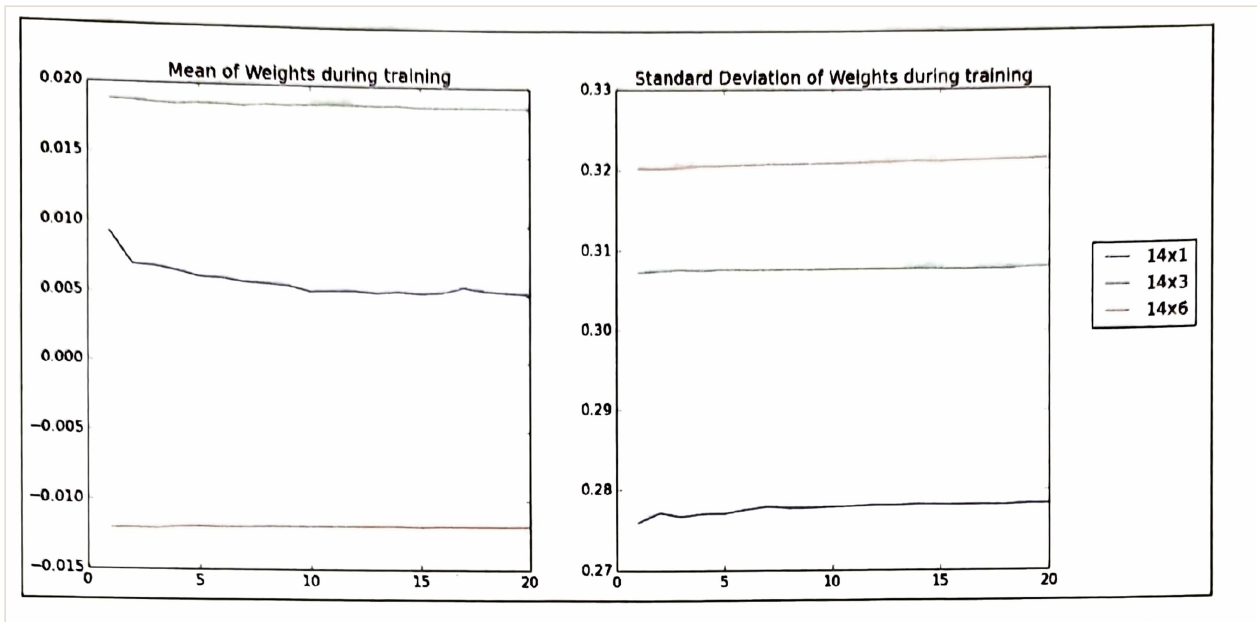


Figure 47 - The chart shows results for three different network depths for the mean and standard deviation of the weights. (See Figure 43 for full explanation)

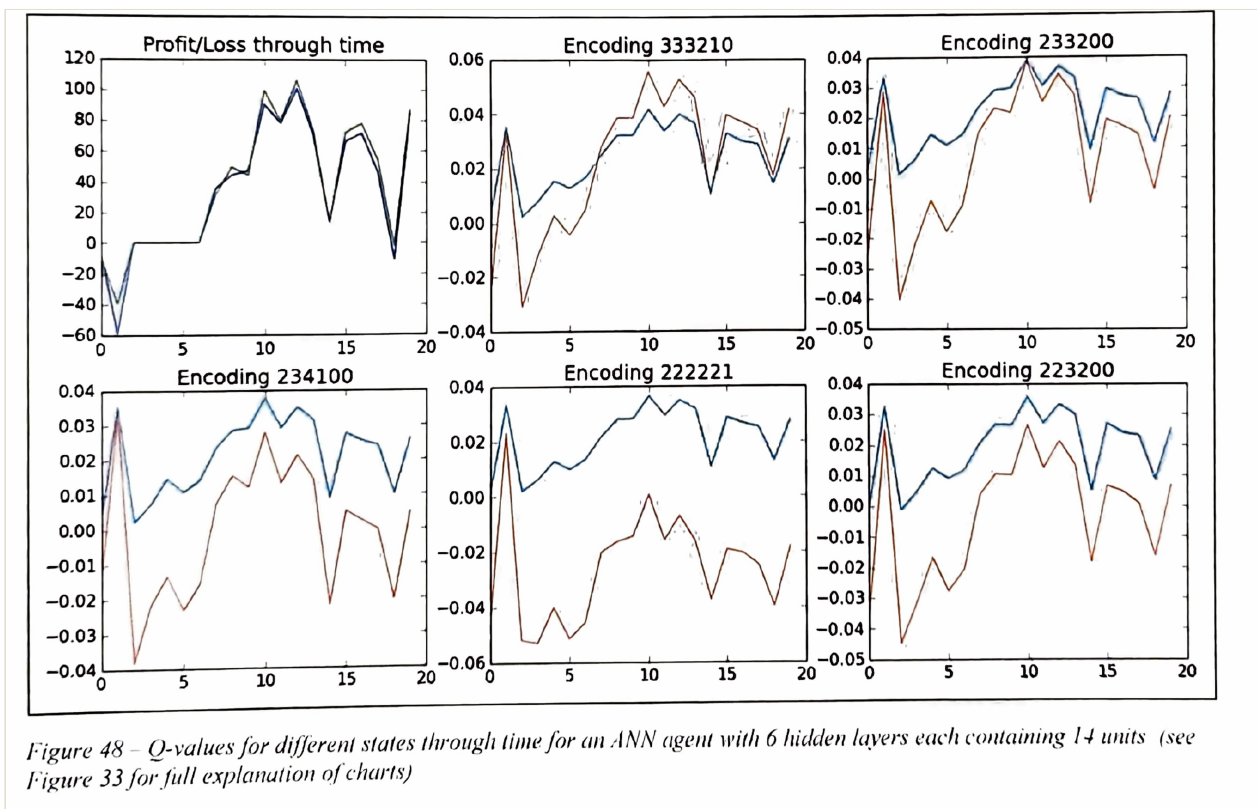


Figure 48 – Q-values for different states through time for an ANN agent with 6 hidden layers each containing 14 units (see Figure 33 for full explanation of charts)

Figure 48 – Q-values for different states through time for an ANN agent with 6 hidden layers each containing 14 units (see Figure 33 for full explanation of charts)

## 5.4. Conclusions

In this Chapter we have designed and tested Q-learning agents that are able to trade profitably in out of sample validation races. Agents using both tabular and ANN were successful. We tested both deep and wide

neural networks. Due to the number of races tested, and the complexity of the algorithm, time meant we could only test one agent for each network structure (in grid world we tested ten agents). The results suggest that a single hidden layer with 14 hidden units is a good network structure for a Q-learning agent for the current trading task. Increasing the depth and width of the ANNs beyond this did not result in increased performance.

Agents have been trained to learn when to take one of four actions. If no bets are currently on the agent decides whether to place a back bet or do nothing. If a back bet is current on the agent decides whether to close the bet or do nothing. In the next Chapter we shall allow the agent to also be able to place lay bets first. We will select agents and test them trading for real money on Betfair.

## 6. Results 3 : Real Money Results

---

In Chapter 5 we trained a number of different agents that made trading profits in out of sample races. In [22-25] RL agents were also trained to make out of sample profits in markets. One of the key goals of this thesis is to demonstrate that RL agents can make profits trading with real money in real time. We could not find any studies in the literature that did this. By making real money we have demonstrated the commercial potential for using Q-learning combined with ANNs in a real world application.

We conducted betting trials on the Betfair exchange for greyhound markets using two different agents. The first trial was for an agent that had just 2 hidden units in a single hidden layer. In February and March 2015 the agent traded on 2632 races and made a profit of £931 after commission. In August 2015 we trialled a second agent that had 14 hidden units in a single layer. This agent traded for 2209 races and made £1222 profit after commission.

In this Chapter we will start by setting out the requirements for the agents. Then we will explain how we selected the agents for the two different trials. This is followed by a detailed look at the results for each of the trials. In this section some of the issues that occur in real world trading will be discussed. The final section contains a summary of the real world results and how they compared to validation results.

### 6.1. Agent Requirements

All the agents trained in Chapter 5 were designed to place back bets first and then close those bets by placing lay bets. This only represents half the opportunities to open trades on Betfair. In order to maximise profits we want our agents to be able to place lay bets first as well. In 5.1.2 the actions required for a back first only agent were explained. Rather than training one agent to place a back or a lay bet first, it is easier to train two agents and then combine them.

The lay first agent's potential actions are very similar to the back first agents. When the agent currently has no bets on it has two action options:

1. Place a lay bet to enter the market
2. Do Nothing

If the agent chooses not to enter the market it still has the action options above. If the agent has entered the market it now has two different options:

1. Place a back bet to close the lay bet and exit the market
2. Do Nothing

The lay agent therefore has 4 action options. As with the back first agent we will train four separate ANNs (one for each action). This will give us a total of 8 ANNs for the combined agent. The combined agent will be able to place both lay first and back first bets. The reward system and the algorithm described in 5.1 had to be adjusted in the code to allow agents to train to lay first. In the next section we will describe how we selected the best lay first and back first agents for our real world testing.

## 6.2. Agent Selection

### 6.2.1. Agent Selection for Real World Betting Part 1

The first agents have two hidden neurons in a single hidden layer. This research was conducted in early 2015. A total of 214 agents were trained once for each day on data collected between 2<sup>nd</sup> October 2014 and 25<sup>th</sup> January 2015. 107 were trained for placing a back bet first and 107 for placing a lay bet first. Each agent was allowed to pass through the day's races 20 times and learned Q-Values using the 7 input neural network method described in Chapter 5. The neural nets for each agent were saved at the end of the 20 passes. Each agent was then tested making decisions using its best-Q value for the 107 days. Figure 49 shows the average results for the top 8 and worst 2 agents for the 107 back and lay days.

Figure 49 – Best and worst agents' performance over 107 days. (Back First Agents)

| Agent Rank | Train Date | Profit |
|------------|------------|--------|
| 1          | 20141205   | £23.04 |
| 2          | 20141226   | £22.96 |
| 3          | 20141214   | £22.82 |
| 4          | 20141116   | £21.80 |
| 5          | 20150107   | £21.63 |
| 6          | 20141126   | £21.62 |
| 7          | 20141217   | £21.55 |
| 8          | 20141130   | £21.34 |
| 106        | 20141208   | -£0.82 |
| 107        | 20141012   | -£1.53 |

Figure 49 – Best and worst agents' performance over 107 days. (Lay First Agents)

| Agent Rank | Train Date | Profit |
|------------|------------|--------|
| 1          | 20150103   | £23.44 |
| 2          | 20141129   | £23.11 |
| 3          | 20141101   | £23.08 |
| 4          | 20141019   | £22.96 |
| 5          | 20141212   | £22.74 |
| 6          | 20141109   | £22.67 |
| 7          | 20141122   | £22.51 |
| 8          | 20141127   | £22.40 |
| 106        | 20141130   | -£4.73 |
| 107        | 20150105   | -£4.97 |

The performance measure for each agent is the average profit made for the 107 days when each opening bet was placed to win (back first) or lose (lay first) a nominal £10. The top ranked 'back first' and the top ranked

'lay first' agents were combined and deployed for real world trading for 28 days from 20<sup>th</sup> February 2015 to the end of 19<sup>th</sup> March 2015. Some of the agents performed particularly badly when tested over the whole 107 days. We saw in Chapters 4 and 5 that ANNs directly using Q-learning can result in very different performances. This is particularly the case when only a single hidden layer with two hidden units is used.

### 6.2.2. Agent Selection for Real World Betting Part 2

After the success of the first real world trial we decided to conduct a second. After further research (Chapters 4 and 5) we decided to use a single hidden layer with 14 hidden units for the second part of the real world testing on Betfair. This was chosen because it is much easier to deploy a single layer neural network for trading and the results from initial testing showed that there was no improvement from adding extra layers. There was, however, a significant improvement from increasing the number of hidden neurons for 2 to 14.

Back first and lay first agents were trained on each of the nine leading greyhound tracks for 20 runs through the races. The weights for the neural networks were stored after each run for each track. In the agent selection process in 6.2.1 we only used the ANNs at the end of training. But having conducted a lot more tests it became apparent that agents sometimes perform their best part way through their training. This resulted in 180 sets of weights being stored for each of the four action networks. We then tested each of these 180 sets of weights for the 4051 validation races (Figure 50).

Figure 50 - Shows the results for the best performing back first and lay first agents on the 4051 validation races. The results highlighted in yellow are the best performing networks average profit per race.

| Track      | Back Profits on Validation | Lay Profits on Validation |
|------------|----------------------------|---------------------------|
| Belle Vue  | 0.1999                     | 0.3209                    |
| Crayford   | 0.2018                     | 0.3735                    |
| Hove       | 0.2339                     | <b>0.3800</b>             |
| Monmore    | 0.1344                     | 0.3544                    |
| Newcastle  | 0.2496                     | 0.3411                    |
| Perry Bar  | <b>0.3055</b>              | 0.2067                    |
| Romford    | 0.2807                     | 0.2978                    |
| Sheffield  | 0.2911                     | 0.3445                    |
| Sunderland | 0.1230                     | 0.3151                    |

The agent with the set of weights that produced the best results in validation races were selected to be used for the real world testing. For backing first the best performing stored weights from the Perry Bar 'back first' agent achieved an average profit per race of 31p when tested on the 4051 validation races. As this was the top scoring agent on the validation races we choose these weights for the back first element of our final agent. This means that the ANNs for the actions to place a back bet or do nothing (if no bet is on) and close the back bet or do nothing (if a back bet is on) were taken from the Perry Barr agent. We did the same for the 'lay first' action ANNs. In the 'lay first' case the best performing agent was trained at Hove with an average of 38p per race on the validation races.

## 6.3. Real Money Results



Bets were placed to back first and lay first using a Betfair account. A maximum risk of £50 per runner was allowed at any time. In the out of sample testing in Chapter 5 we always assumed we could bet to win or lose £10 at the best current price available. This means that in theory our trading should make five times the amount the agents made in validation. In practice we shall see that the full five times multiple is not made. This is mainly due to delays between the agent receiving state information from Betfair and Betfair receiving the agent's choice of action back. If a full bet is not available by the time our trade hit Betfair any unmatched part of the bet is immediately cancelled.

### **6.3.1. Agent NN1**

The combined top ranked 'back first' and 'lay first' agent was deployed for real world trading for 28 days from 20<sup>th</sup> February 2015 to the end of 19<sup>th</sup> March 2015. The agent was allowed to bet up to a maximum to win or lose £50 on any one selection at any one time. Many times smaller bets had to be placed because there was not enough money available in the market. Each day the data was collected the agent was allowed to run through the races betting to win or lose £10 where the bets were guaranteed to be filled at the best price; i.e. trade in the same way the agent was trained.

Figure 51 shows the performance the agent in both the real world (real profits) and how it was trained (model profits). The model profits were very consistent averaging £69.44 a day with a standard deviation of £24.93. There was no degradation of profits throughout the month. The real profits achieved by the agent were far lower and less consistent. Whereas everyday was profitable for the model the real results were loss making on 4 of the 28 days. The average profit betting to win or lose a maximum of £50 a bet was £31.29 with a standard deviation of £38.67. This is before the Betfair commission of between 2% and 5% per winning market was taken into account. Figure 31 shows that once commission is taken in account the profits for the 2,632 markets traded is reduced to just £931.13.

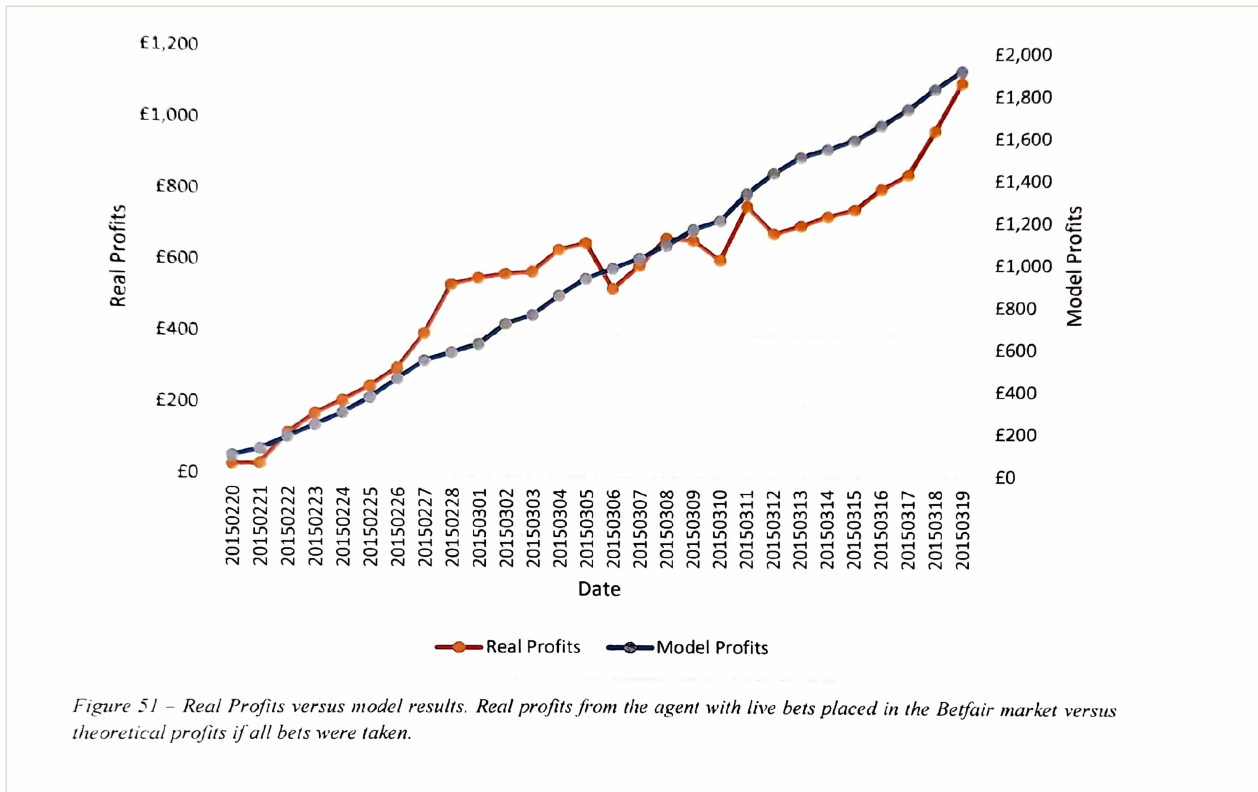


Figure 51 – Real Profits versus model results. Real profits from the agent with live bets placed in the Betfair market versus theoretical profits if all bets were taken.

There are a number of reasons for the difference in theoretical profits and actual profits. The main reason is that not all bets are filled at the best price available. The agent has been trained with this assumption. The agent does extremely well at performing when tested on what it has been trained to do. The environment that the agent is trained would need to be changed with actual amounts in the market used for learning. Another reason for real world profits being less than theoretical profits is latency. We found that 10%-20% of bets attempted by agents were not matched immediately when they hit the market and had to be cancelled. The reason for the less consistent profits in the real world is that occasionally the agent cannot trade out fully its profit or loss to be equal for all runners. This can be because the minimum bet on Betfair is £2. The reason for the big movements can be because the agent has failed to close a trade due to a technical error.

**Betting Profit & Loss**  
**20-Mar-2015 11:19**

Period: Last 30 days  
 (relates to event settlement date)

2015-02-20 00:00 to 2015-03-20 00:00 Get P & L  
 (yyyy-mm-dd hh:mm) (yyyy-mm-dd hh:mm)

Downloaded for Screenshots 7

**Greyhound Racing: £931.13 Total P&L: £931.13**

Greyhound Racing

Market Showing 1 - 20 of 2632 markets

| Market                                     | Start time      | Settled date    | Profit/Loss (£) |
|--------------------------------------------|-----------------|-----------------|-----------------|
| Greyhound Racing / Move 19th Mar : QR 515m | 19-Mar-15 22:26 | 19-Mar-15 22:28 | -2.50           |
| Greyhound Racing / Move 19th Mar : QR 515m | 19-Mar-15 22:11 | 19-Mar-15 22:15 | -2.64           |
| Greyhound Racing / Move 19th Mar : QR 515m | 19-Mar-15 22:03 | 19-Mar-15 22:07 | -0.32           |
| Greyhound Racing / Move 19th Mar : QR 515m | 19-Mar-15 21:56 | 19-Mar-15 22:00 | 2.18            |
| Greyhound Racing / Move 19th Mar : QR 475m | 19-Mar-15 21:48 | 19-Mar-15 21:52 | -1.72           |
| Greyhound Racing / Move 19th Mar : QR 475m | 19-Mar-15 21:40 | 19-Mar-15 21:42 | 5.79            |
| Greyhound Racing / Move 19th Mar : QR 475m | 19-Mar-15 21:32 | 19-Mar-15 21:38 | 1.98            |

Figure 52 – Betfair profit and loss statement for NN1 agents for the 28 days of trading. Profits are shown after commission.

Figure 52 – Betfair profit and loss statement for NN1 agents for the 28 days of trading. Profits are shown after commission.

Figure 53 shows the profits and losses of every race over the 28 days. There are occasions when the graph goes up or down sharply because trades are not properly closed out. The arrow points to an example where £79 (9% of the month's profits) was lost on one race. This shows a potential pitfall of real world trading compared just theoretically trading using historical data. Over the long run the profits and losses from unclosed trades should even out.

profits and losses from unclosed trades should even out.

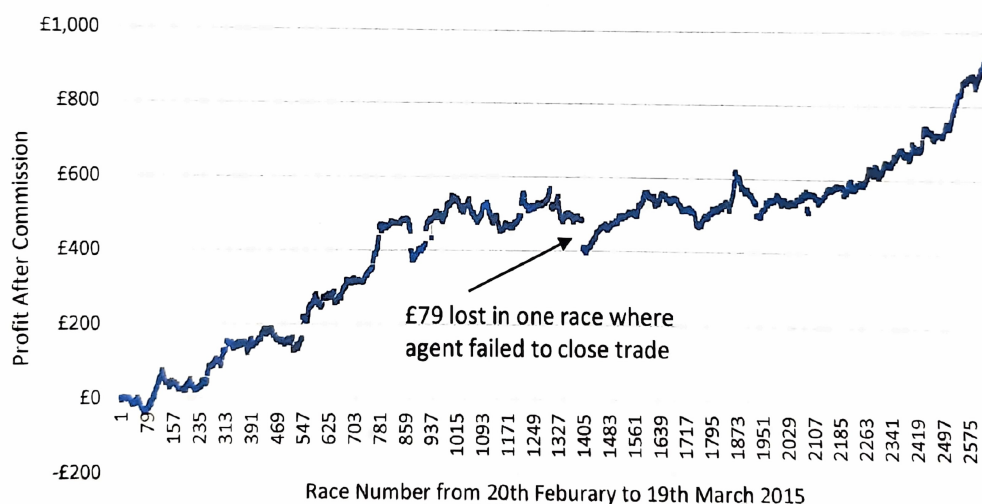


Figure 53 – Chart of real profits for all the bets placed over the 28 days after commission. There are several big movements due to trades not being closed out. The arrow points to one race where £79 was lost due to this.

### 6.3.2. Agent NN2

The selected NN2 agents were tested on Betfair for 24.5 days from 4<sup>th</sup> August 2015 to 29<sup>th</sup> August 2015. The same staking procedure was used as in 6.3.1. Figures 54 and 55 show the profits achieved

Figure 53 – Chart of real profits for all the bets placed over the 28 days after commission. There are several big movements due to trades not being closed out. The arrow points to one race where £79 was lost due to this.

### 6.3.2. Agent NN2

The selected NN2 agents were tested on Betfair for 24.5 days from 4<sup>th</sup> August 2015 to 29<sup>th</sup> August 2015. The same staking procedure was used as in 6.3.1. Figures 54 and 55 show the profits achieved by the agents was consistent averaging £47 per day with a standard deviation of £35.

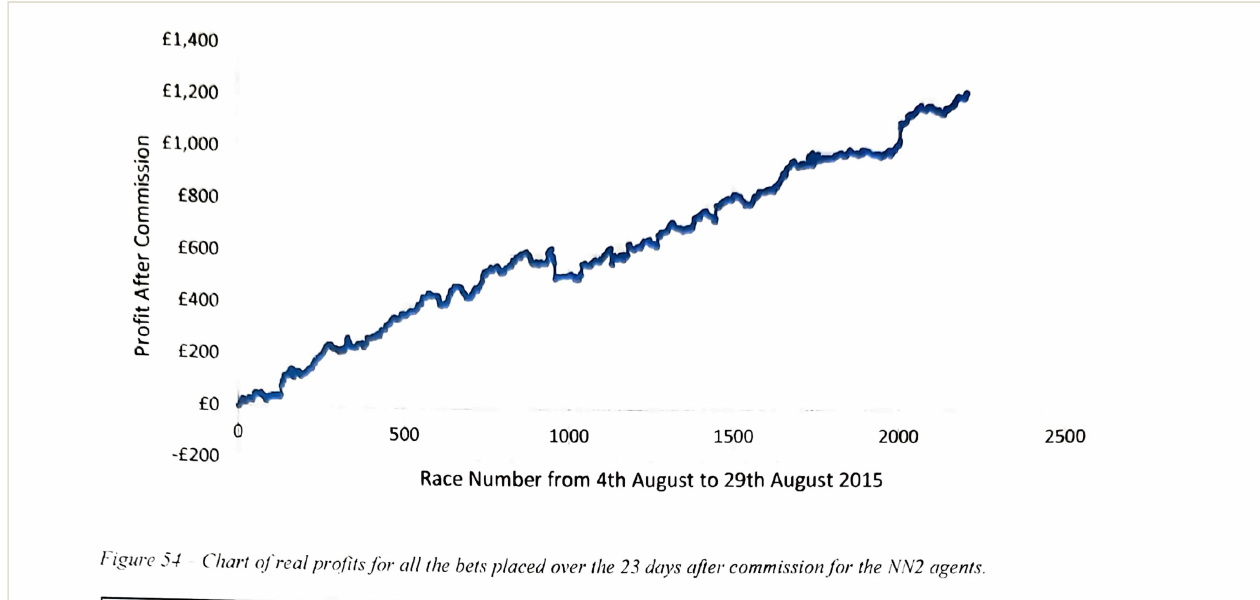
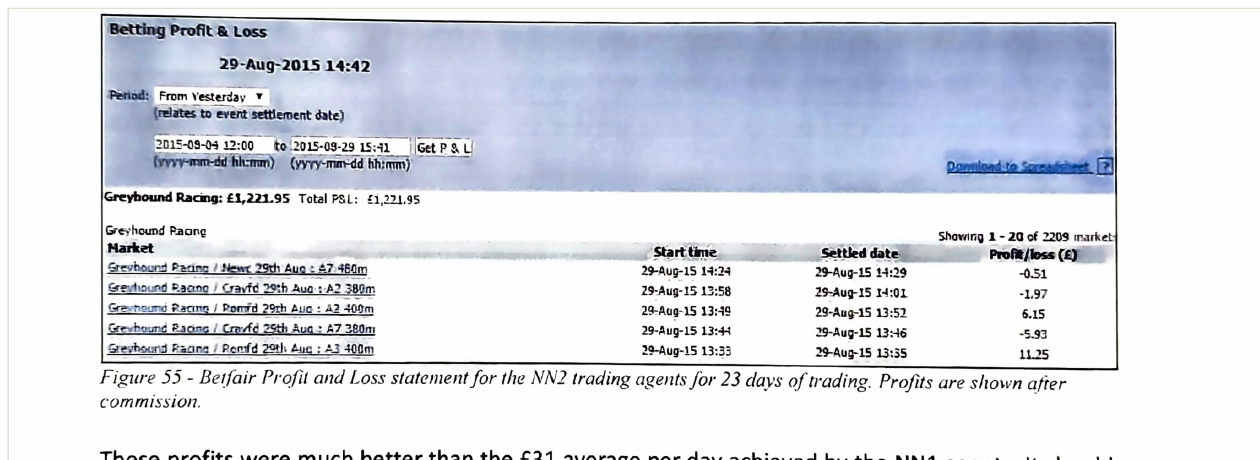


Figure 54 – Chart of real profits for all the bets placed over the 23 days after commission for the NN2 agents.



These profits were much better than the £31 average per day achieved by the NN1 agents. It should

Figure 55 - Betfair Profit and Loss statement for the NN2 trading agents for 23 days of trading. Profits are shown after commission.

These profits were much better than the £31 average per day achieved by the NN1 agents. It should be noted that the NN2 agents were trained using races from October and November 2014 but they were tested in August 2015. Using more up to date races for training might help increase profits in the future.

## 6.4. Summary and Conclusions

In Chapter 5 we developed Q-Table agents and two sets of agents trained using ANNs. The Q-Table agents had a state encoding of the limit order book that consisted of a string of 6 integers. The two ANN agents were

provided with six continuous inputs representing the same information. The ANN agents also had an extra continuous input which informed them on the time remaining before the start of race. The agents trained with the first ANN structure (NN1) and second ANN structure (NN2) had seven inputs, a hidden layer with two neurons and one output.

Figure 56 – Results Summary. The table shows the average profit per race for top agents trained using each of the three methods. The Q-Table method was described in 5.2.1. The results are shown for agents trained to back first and then close bets by laying and agents trained to lay first and close bets by backing. The agent selection process was the same as the one used in 6.3.2 for the 2<sup>nd</sup> neural network model. Q-Table agents were trained using training races for the top 9 tracks. The agents that produced the highest profit per race for the validation races were selected. The NN1 results are the agents selected in 6.2.1. The NN2 model results are the agent selected in 6.2.2. The actual betting results are the profit per race for the actual trading conducted on Betfair using the selected agents for NN1 and NN2. The staking for the real trading was to win a maximum of £50 per bet although often there was not enough liquidity for the full bet to be placed. The Back First and Lay First results are the average profit per race with all bets placed to win £10.

|                | <b>Q-Table</b> | <b>NN1</b> | <b>NN2</b> |
|----------------|----------------|------------|------------|
| Back First     | 0.23           | 0.25       | 0.31       |
| Lay First      | 0.21           | 0.26       | 0.38       |
| Actual Betting | N/A            | 0.35       | 0.55       |

Figure 56 shows the Q-Table agents to be the worst performers. The best performing agents on the validation races produced profits slightly lower than those of NN1. The Q-Table agents have the advantage of being fast to train but they lack the precision of ANNs trained agents which use continuous inputs. The Q-Table agents are also potentially at a disadvantage because they do not have information on how long it is until the race starts.

The NN1 agents perform slightly better than the Q-Table agents for both laying first and backing first. These agents only had 2 hidden neurons to train to form the representation of the inputs. This allowed for faster training than the NN2 agents but restricted the quality of the Q-values it produced. These agents were trained using data one day at a time (around 80-100 races) whereas the data sets for the NN2 agents were split by track (228-639 races). The reason for this was that the testing was done in early 2015 before experiments with larger datasets were conducted.

The best performing agents on the validation data were the NN2 agents. These agents were trained on the larger datasets and had fourteen neurons in a single hidden layer. Increasing the number of neurons in the hidden layer results in increased performance. The NN2 agents produced 40% more profit per race as the NN1 agents (55 pence per race as opposed to 35 pence per race).

The real world betting was conducted at potentially five times higher stakes than the validation testing. If every bet been placed in full to £50 at the best price available the validation profit per race suggests £2.55 ( $5 \times (25p + 26p)$ ) for NN1 and £3.45 ( $5 \times (31p + 38p)$ ) for NN2. The actual results of 35p and 55p were only 14% and 16% of the expected results. Part of the reason for this is lack of money available at the best price available. A second issue is latency which we shall look at in Chapter 7. Another potential issue is changing market conditions. The data used for this thesis was collected from October 2014 to January 2015. The second real world testing was seven months later in August 2015. A final reason is that we choose the best performing agents in the validation dataset. On a different dataset (the real world betting) a different agent may have performed better.

In this Chapter we have successfully proved that Q-learning ANN agents can make real money trading on a market. To our knowledge no one else have published similar results in literature. The results show profits far lower than those archived on the validation races. This shows we should be cautious in future of published validation results unless they are backed by real world profits.



## 7. Results 4 : Extensions

---

In this Chapter we test two trading agent extensions. The first is to build latency into the agent’s environment. The second is to see whether we can use the same agent structure to train agents to trade profitably on markets other than greyhounds.

In Chapter 6 we found that the profits made in real world trading were well below those expected from testing on historical data. We hypothesised that that could be due to latency in the bet placement procedure. Our first extension tests if agents can be improved by taking this into account when training.

Throughout this thesis we have trained agents to trade on greyhound races on Betfair. Betfair runs many different market types for every sporting event imaginable. Every event has the same limit order book information available. We test whether our agents can learn to trade successful on other markets with no changes at all to their settings. We the tested horse racing markets for Australia, USA and the UK/Ireland.

### 7.1. Adding Latency to the Model

So far all the agents have been trained in an environment with the assumption of zero latency. This means that the agent is capable of receiving order book data from Betfair’s servers, processing that data into a simplified state representation, deciding on an action to take and sending details of a bet back to Betfair all in zero time. In reality the process of this roundtrip communication with Betfair plus the processing of the data takes between 20 and 120 milliseconds.

The data points in our test data were collected 100 milliseconds apart. It possible to add this delay to our model setup. The impact of latency can be included by only allowing the agent’s bet to be placed if the price the agent requests is still available (or a better offer is available) in the time frame after it receives the state information. If the price is not available the bet is not placed. We tried using delayed responses from the environment to directly train the agent. We also tested how an agent trained with zero delay performed when validated in a delayed environment.

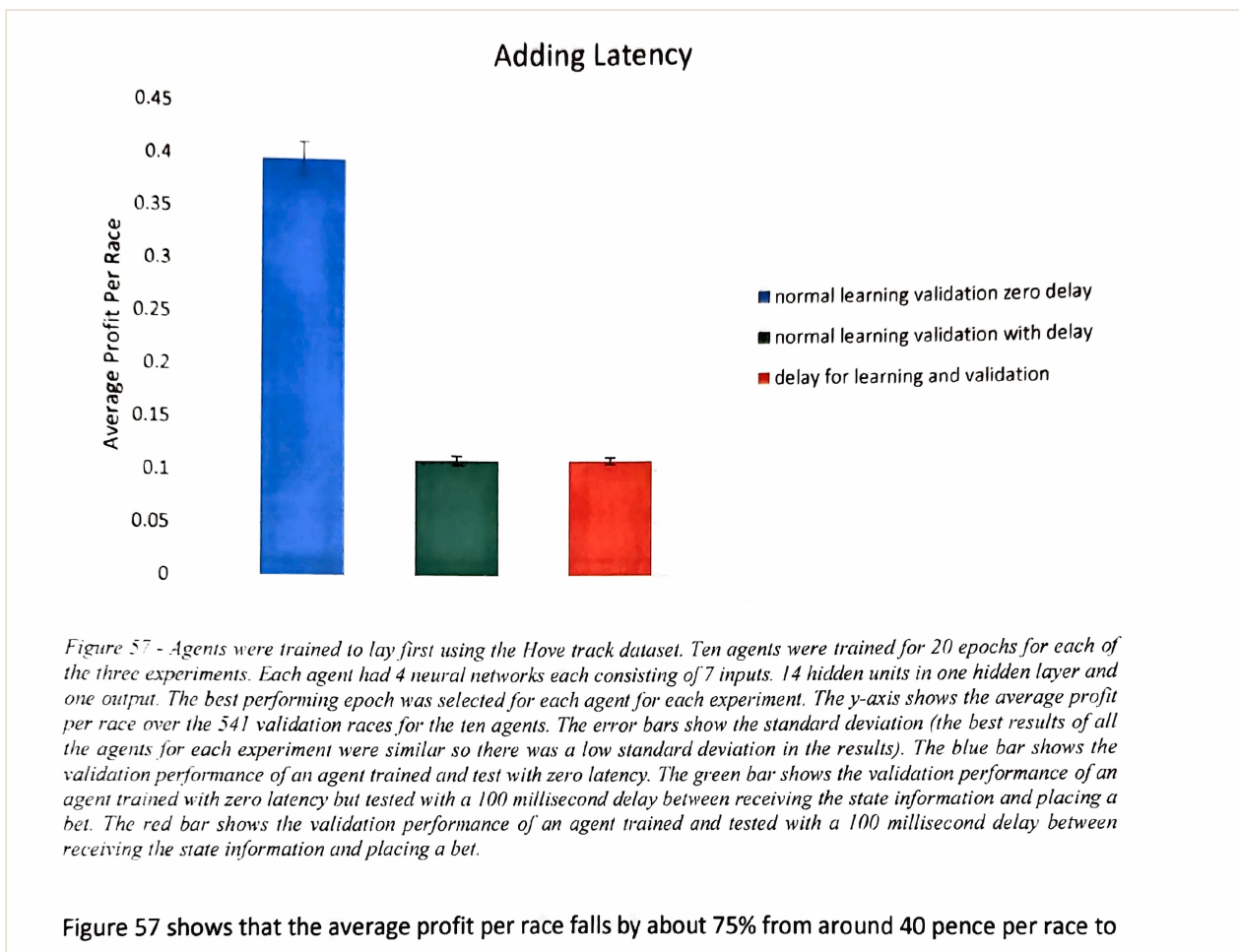


Figure 57 – Agents were trained to lay first using the Hove track dataset. Ten agents were trained for 20 epochs for each of the three experiments. Each agent had 4 neural networks each consisting of 7 inputs, 14 hidden units in one hidden layer and one output. The best performing epoch was selected for each agent for each experiment. The y-axis shows the average profit per race over the 541 validation races for the ten agents. The error bars show the standard deviation (the best results of all the agents for each experiment were similar so there was a low standard deviation in the results). The blue bar shows the validation performance of an agent trained and test with zero latency. The green bar shows the validation performance of an agent trained with zero latency but tested with a 100 millisecond delay between receiving the state information and placing a bet. The red bar shows the validation performance of an agent trained and tested with a 100 millisecond delay between receiving the state information and placing a bet.

Figure 57 shows that the average profit per race falls by about 75% from around 40 pence per race to 10 pence a race when a time delay is introduced for lay first agents for the Hove dataset. The blue bar shows the average profit when the agent is trained with no delay between receiving the state information and the bet being processed. The green bar shows the average profit when the agent is trained with no delay but the validation occurs with a delay. After seeing this large drop in profits we tested to see if the agents could do better if it was trained in an environment with a delay built in. The red bar shows that profitability remained the same regardless of how the agent was trained.

These results demonstrate how important getting the fastest connection possible between the agent’s server and the Betfair server is for gaining profits for real world trading. Any reduction in the time taken will lead to improved profits.

## 7.2. Adapting for Different Types of Markets

We trained ‘back first’ agents with the same 7 input, 14 hidden units and 1 output structure used for the NN2 agents. We made no adjustments to the inputs or the hyper-parameters for the Q-learning algorithm or the ANNs. We tested the win markets for horse racing from the USA, Australia, GB and Ireland. We also tested the place markets for horseracing in GB and Ireland.

Figure 58 shows data from the different race types for the week 17<sup>th</sup>-23<sup>rd</sup> August 2015. Greyhound racing has the highest number of races per week but also the lowest amount matched per race of just £9,276. By far the biggest market is the GB and Ireland win horseracing markets with averaging £519,000 a race. With much higher liquidity we would expect a successful agent to make far higher profits on these races. They would be able to use bigger stakes and not worry about being able to close the trades due to lack of money in the market.

Figure 58 – The average matched per race for different race types on Betfair 17<sup>th</sup>-23<sup>th</sup> August 2015. Data does not include in play betting. (source data.betfair.com)

| Race Type           | Average Matched Per Race | Number of Races |
|---------------------|--------------------------|-----------------|
| GB Dogs             | £9,276                   | 709             |
| USA Horses          | £14,167                  | 466             |
| OZ Horses           | £30,587                  | 530             |
| GB/IRE PLACE Horses | £68,089                  | 285             |
| GB/IRE WIN Horses   | £519,203                 | 286             |

We collected data from Betfair for the tests. The races for the USA, GB and Irish races were collected in January 2015. The races for Australian races were from July 2015. For the USA we trained using 382 races and validated the results on 381 races. For Australia 251 races were used for training and 268 races were used for validation. For the GB and Ireland place markets 110 races were used for training and 93 races were used for validation. For GB and Ireland win markets we could only use 48 races for testing and 42 races for validation. This was because the liquidity in these markets is so high there are barely ever any gaps in the market. This means that every data point is used for testing resulting in very large datasets.

Figure 59 shows the results for one trial using the ‘back first’ agent for each race type. The Australian and American win markets and the GB and Ireland place markets all show profits both in and out of sample. The profits look particularly high for the place markets with a maximum of 60p per race. The results for the UK and Ireland win markets are disappointing. They result in a loss both for the training and validation races. The fully trained agent only recommends an average of two bets a race. This compares with 20 bets a race for the trained place agent.

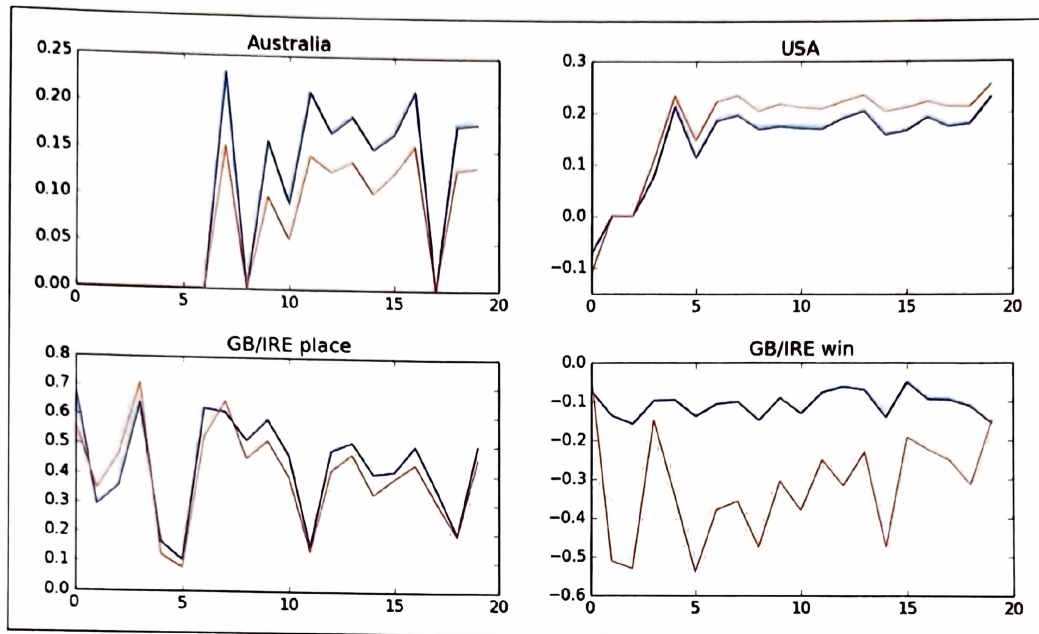


Figure 59 Results from test the 7 input, 14 hidden unit in a single layer agent on different markets. The x-axis shows how many times the agent has gone through the data. The y-axis is the profit/loss that the agent makes following the action with the highest Q-value at the end of each training run through the races. The blue lines show the results of the training data and the red lines are the results from the validation races.

Unfortunately there was not time to run more than one trial on these datasets. The early results are promising for using the same agent structure in new markets. The failure of the agent to learn a

Figure 59 – Results from test the 7 input, 14 hidden unit in a single layer agent on different markets. The x-axis shows how many times the agent has gone through the data. The y-axis is the profit/loss that the agent makes following the action with the highest Q-value at the end of each training run through the races. The blue lines show the results of the training data and the red lines are the results from the validation races.

Unfortunately there was not time to run more than one trial on these datasets. The early results are promising for using the same agent structure in new markets. The failure of the agent to learn a profitable strategy for the GB and Ireland win markets may be because there is simply too much liquidity in those markets causing less volatility in the horses' prices. In this thesis our agents were set to always take a price available on the market rather than leave up a limit order for others to take. In slow moving markets the spread between the best lay and back prices can prove costly.

## 8. Conclusions and Future Work

---

This thesis presented a methodology for profitably automating the decision making process for trading a number of Betfair markets. Q-learning agents were trained using artificial neural networks (ANNs) to learn when to open and when to close trades. These decisions were influenced by the amount of money currently in the market and the amount of time left until the start of the race. The methodology was validated when trained agents were deployed in real markets on the Betfair betting exchange and made £2153 profit. This is the first time, to our knowledge, any machine learning method for trading markets has been validated using real money results.

In Chapter 4, we presented a methodology for using ANNs with Q-learning. We tested this methodology by training agents in a grid world environment. The agent was only given its current grid position and had to learn to move to the middle of a nine by nine square. Whilst this problem was simple using traditional tabular Q-learning (Chapter 3) it was complex for ANNs as grid points that are very close together all require different actions. We found that deeper ANNs performed the best by slowly updating lots of very small weights in multiple layers as they interacted with the environment. If the ANNs got too wide (over 40 hidden units) performance dropped as the neurons got saturated.

In Chapter 5, we used the same ANN Q-learning methodology to train agents to learn to trade on Betfair greyhound races. We presented profitable results on validation races using different network structures. They ranged from a network with two hidden neurons in a single layer to a large network of six hidden layers with fourteen units in each. The best performing agents were those trained with fourteen units in a single hidden layer.

Real world trading results were reported in Chapter 6. The method for selecting agents was described followed by the results themselves. Agents that had fourteen units in a single hidden layer produced 40% greater profits than agents with just two units in a single hidden layer. The actual amount of profit made was only 15% of the amount suggested by results validated on historical data. We found that this was largely due to latency between the agent and the exchange. In Chapter 7, we found that building in 100 milliseconds of latency reduced profits to a quarter of their former levels. The fact that profits were much lower in the real world testing than expected from the historical validation results means we should be cautious in future of system profits that are not validated by real world results.

In Chapter 7, we tested the best network structure for different types of races. The framework and learning algorithms were left unchanged. Initial results showed agents made profits on historical data for Australian and American horseracing win markets. They were also profitable for British and Irish horseracing place markets. For the highly liquid British and Irish horseracing win markets the agents were unable to make a profit.

The promising results in this thesis demonstrate that in future RL has an important role to play in the development of profitable trading systems for betting exchanges and, potentially, the financial markets. The disappointing result in Chapter 7 for the UK and Ireland horseracing win markets show a fresh approach is required to conquer those markets. The agents designed in this thesis were only able to place bets at the

current market price. In future agents could be allowed more potential actions, such as leaving bets up in the market for others to take. For state information price and volume history could be added. Deep-learning methodologies, such as auto-encoders and convolutional neural networks, could be used to help represent the information contained in price charts.

For the grid world problem in Chapters 3 and 4 tabular Q-learning found the optimal result in 2500 moves compared to 200,000 for a deep ANN. The trading results demonstrated that tabular Q-learning is profitable on historical validation races. For this reason tabular Q-learning should be used for fast prototyping of new state inputs and new agent structures. ANNs can be used for added granularity to the inputs to improve performance once the importance new of the inputs or structures are established.

In Chapter 5, we examined how the network weights changed during training and what the Q-value outputs were for different actions (for example Figure 33). In future, tools could be developed to audit exactly what is happening during the training of agents and what actions they are suggesting for different states to see if they make sense. When real money is at risk agents do not have to be followed blindly. The same applies if these techniques are used in fields other than trading.

Deepmind's use of deep reinforcement learning for playing Atari [13] demonstrated how Q-learning agents could learn directly from screen pixels to achieve high game scores. The use of ANNs with RL becomes more complicated when the domain changes to a real world problem. In this thesis expert knowledge was used to present state information to the agent. With current technology it seems inconceivable that complete trading information could be automatically presented to a trading agent. There are so many ways data could be charted and external factors, such as horse bolting on the way to the start, can cause prices to changes.

Complete automation, just to prove complete automation is possible, should not be a goal in itself when we consider using artificial intelligence algorithms for commercial or mission critical reasons. Tesauro found that by adding expert knowledge his backgammon agents went from being the best machine backgammon player to master level play [5-7]. The agents still used ANNs combined with RL but they were given human help. The results in this thesis show similar human help can lead to profitable trading by RL agents.



# Bibliography

---

1. On learning with imperfect representations; Kalyanakrishnan, Shivaram ; Stone, Peter; 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, April 2011, pp.17-24
2. Continuous-action reinforcement learning with fast policy search and adaptive basis function selection; Xu, Xin ; Liu, Chunming ; Hu, Dewen; Soft Computing, 2011, Vol.15(6), pp.1055-1070
3. Learning from Delayed Rewards, Chris Watkins; Phd Thesis, Cambridge, 1989.
4. Reinforcement Learning: Introduction, An Sutton, Richard S. ; Barto, Andrew G; MIT Press 1998
5. TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play; Tesauro, Gerald; Neural Computation, 1994, Vol.6(2), pp.215-219
6. Temporal difference learning and TD-Gammon; Tesauro, Gerald Cohen, Jacques (Editor); Communications of the ACM, March 1995, Vol.38(3), pp.58-68
7. Programming backgammon using self-teaching neural nets; Tesauro, Gerald; Artificial Intelligence, 2002, Vol.134(1), pp.181-199
8. Elements of Artificial Neural Networks; Mehrotra, Mohan, Ranka; 1997; The MIT Press.
9. Multilayer feedforward networks are universal approximators; Hornik, Kurt ; Stinchcombe, Maxwell ; White, Halbert; Neural Networks, 1989, Vol.2(5), pp.359-366
10. From an artificial neural network to a stock market day-trading system: A case study on the BM&F BOVESPA; Martinez, Leonardo C. ; Da Hora, Diego N. ; de M. Palotti, Joao R. ; Meira, Wagner ; Pappa, Gisele L.; 2009 International Joint Conference on Neural Networks, June 2009, pp.2006-2013
11. <http://www.betfairprotrader.co.uk/2010/12/weight-of-money.html>; Betfairprotader blog post, James Butler.
12. Reinforcement learning for optimized trade execution.; Nevmyvaka, Y., Feng, Y., & Kearns, M. (2006). In W. W. Cohen & A. Moore (Eds.), Proceedings of the twenty-third international conference on machine learning (ICML 2006) (pp. 673–680). New York: ACM
13. Human-level control through deep reinforcement learning; Volodymyr Mnih ; Koray Kavukcuoglu ; David Silver ; Andrei A. Rusu ; Joel Veness ; Marc G. Bellemare ; Alex Graves ; Martin Riedmiller ; Andreas K. Fidjeland ; Georg Ostrovski ; Stig Petersen ; Charles Beattie ; Amir Sadik ; Ioannis Antonoglou ; Helen King ; Dharshan Kumaran ; Daan Wierstra ; Shane Legg ; Demis Hassabis; Nature, 2015, Vol.518(7540), p.529
14. Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations.; Wendelin Böhmer, Jost Tobias Springenberg, Joschka Boedecker, Martin Riedmiller, and Klaus Obermayer. KI-Künstliche Intelligenz, pages 1–10, 2015.

15. Modelling high-frequency limit order book dynamics with support vector machines; Kercheval, Alec N. ; Zhang, Yuan; Quantitative Finance, 03 August 2015, Vol.15(8), p.1315-1329
16. Applying Deep Learning to Enhance Momentum Trading Strategies in Stocks; Lawrence, Takeuchi, Yu-Ying (Albert Lee); December 2013; CS229 Coursework for Andrew Ng Machine Learning course.
17. An autonomous trader agent for the stock market based on online sequential extreme learning machine ensemble. Cavalcante, R. C., & Oliveira, A. L. I. (2014). In International joint conference on neural networks IJCNN'2014 (pp. 1424–1431).
18. Machine Learning Techniques and Use of Event Information for Stock Market Prediction: A Survey and Evaluation; Yoo, P.D. ; Kim, M.H. ; Jan, T. International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce 2005, Vol.2, pp.835-841
19. Utilizing artificial neural networks and genetic algorithms to build an algo-trading model for intra-day foreign exchange speculation; Evans, Cain ; Pappas, Konstantinos ; Xhafa, Fatos; Mathematical and Computer Modelling, 2013, Vol.58(5-6), pp.1249-1266
20. Deep learning; Yann Lecun ; Yoshua Bengio ; Geoffrey Hinton; Nature, 2015, Vol.521(7553), p.436
21. Intelligent trading of seasonal effects: A decision support algorithm based on reinforcement learning. Eilers, Dennis ; Dunis, Christian L. ; von Mettenheim, Hans-Jörg ; Breitner, Michael H. Decision Support Systems, 2014, Vol.64, pp.100-108
22. A Multiagent Approach to Q-Learning for Daily Stock Trading; Jae Won Lee ; Jonghun Park ; Jangmin O ; Jongwoo Lee ; Euysseok Hong; IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, Nov. 2007, Vol.37(6), pp.864-877
23. An automated FX trading system using adaptive reinforcement learning; Dempster, M.A.H. ; Leemans, V. ; Expert Systems With Applications, 2006, Vol.30(3), pp.543-552
24. Adaptive systems for foreign exchange trading; Austin, Mark ; Bates, Graham ; Dempster, Michael ; Leemans, Vasco ; Williams, Stacy; Quantitative Finance, 2004, Vol.4(4), pp.37-45
25. Evolutionary reinforcement learning in FX order book and order flow analysis; Bates, R.G. ; Dempster, M.A.H. ; Romahi, Y.S. ; 2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003 2003, pp.355-362
26. Testing different Reinforcement Learning configurations for financial trading: Introduction and applications.; F. Bertoluzzo and M. Corazza: Procedia. Economics and Finance, 3, 68-77, 2012
27. Bet Angel trading software for Betfair; www.betangel.com; Bet Angel Limited.
28. Geeks Toy trading software for betting exchanges; www.geekstoy.com; Talented Mavericks Limited.
29. Betfair                                      Api                                      Reference                                      Guide;  
<https://api.developer.betfair.com/services/webapps/docs/download/attachments/4392320/API->

ReferenceGuide-OfflineVersion-22ndJuly2015.pdf?version=1&modificationDate=1437555641000;  
Betfair Group Plc.

30. Autonomous reinforcement learning on raw visual input data in a real world application; Lange, S. ; Riedmiller, M. ; Voigtlander, A. The 2012 International Joint Conference on Neural Networks, June 2012, pp.1-8
31. Improving Elevator Performance Using Reinforcement Learning; R.H. Crites and A.G. Barto; Advances in Neural Information Processing Systems 8; MIT Press, 1996, pp. 1017-1023
32. Autonomous helicopter flight via reinforcement learning, A.Y. Ng, H.K. Kim, M.I. Jordon and S. Sastry, Advances in Neural Information Processing Systems 16. MIT Press, 2004.
33. Efficient Capital Markets: A Review of Theory and Empirical Work; Malkiel, Burton G; Fama, Eugene F.; The Journal of Finance, 1970, Vol.25(2), pp.383-417.
34. Identifying winners of competitive events: A SVM-based classification model for horserace prediction; Lessmann, Stefan ; Sung, Ming-Chien ; Johnson, Johnnie E.V. European Journal of Operational Research, 2009, Vol.196(2), pp.569-577
35. Wall Street a history. Charles R. Geisst; Oxford University Press. New York : Oxford University Press 1999
36. Studies in Tape Reading by D. Richard Wyckoff, Rollo Tape, Richard D. Wyckoff (ISBN: 9781614271840)
37. Flash Boys: Cracking the Money Code, by Michael Lewis, W.W. Norton & Company (2014). ISBN 978-0393244663.