Creating Algorithmic Traders with Hierarchical Reinforcement Learning

MSc Dissertation

Thomas Elder

s0789506

Master of Science

School of Informatics

University of Edinburgh

2008

Abstract

There has recently been a considerable amount of research into algorithmic traders that learn [7, 27, 21, 19]. A variety of machine learning techniques have been used, including reinforcement learning [20, 11, 19, 5, 21]. We propose a reinforcement learning agent that can adapt to underlying market regimes by observing the market through signals generated at short and long timescales, and by using the CHQ algorithm [23], a hierarchical method which allows the agent to change its strategies after observing certain signals. We hypothesise that reinforcement learning agents using hierarchical reinforcement learning are superior to standard reinforcement learning agents in markets with regime change. This was tested through a market simulation based on data from the Russell 2000 index [4]. A significant difference was only found in the trivial case, and we concluded that a difference does not exist for our agent design. It was also observed and empirically verified that our standard agent learns different strategies depending on how much information it is given and whether it is charged a commission cost for trading. We therefore provide a novel example of an adaptive algorithmic trader.

Acknowledgements

First and foremost, I must thank Dr. Subramanian Ramamoorthy for supervising and motivating this research, and for providing sensible suggestions when I found myself low on ideas. I would also like to thank my family and friends for cheering me up and giving me moral support.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Thomas Elder*
s0789506)

Table of Contents

Chapter 1

Introduction

The shortcomings of human traders have been demonstrated countless times. In most cases, this does not have a far reaching effect: somebody might lose their savings, somebody else's promising career as an investment banker comes to an abrupt end. But at times, the results can be catastrophic, such as the Wall Street Crash of 1929, or the collapse of Barings Bank in 1992.

Human traders are poorly suited to the fast moving, highly numerical domain of trading. Moreover, people are beset by a number of psychological quirks that can result in poor trading decisions, such as greed, fear or even just carelessness. It is no secret that computers are immune to these problems, and they are rapidly replacing humans in all aspects of trading. In most countries, the days where traders yelled at each other across exchange room floors are long gone.

However, computers do not have a clean track record either. The 'Black Monday' Stock Market Crash of 1989 is widely believed to have been caused or exacerbated by program trading [30]. However, these program traders were merely implementing strategies prescribed by humans. This suggests a problem; though we can replace humans with computers, somebody still needs to program those computers. So long as algorithmic traders use rigid predetermined strategies, people will be needed to create and update those strategies, and human weakness will remain in the marketplace. The reason we need people, is because for all their weaknesses, humans do have a huge advantage: they are good at adapting and learning.

It is reasonable to ask whether the numerical advantages of algorithmic traders could be combined with the adaptability of humans. After all, there are well established machine learning techniques that allow computers to learn. In fact, there has recently been considerable interest in algorithmic traders than learn [7, 27, 21, 19]. However, we feel that most of this research fails to replicate the adaptability of humans. Though some have created agents that learn strategies, few have asked why they these strategies were learnt or how different strategies might be learnt in different situations. Moreover, few have attempted to create agents that learn to adapt to market conditions or regimes,

In this dissertation, we propose an algorithmic trading agent which addresses these issues. When designing our agent, we have made decisions that aim to minimise the human influence on resulting strategies. For this reason we have chosen to design a reinforcement learning agent, since reinforcement learning has been used in other domains to learn strategies that humans would not find.

Other researchers have created algorithmic traders through reinforcement learning [20, 11, 19, 5, 21]. However, unlike in these approaches, our agent is designed to adapt to underlying market regimes. We argue that identifying these regimes makes the market environment better suited to reinforcement learning, which should improve the performance of resulting strategies. We propose two ways in which the agent can identify these regimes. Firstly, the agent can make observations which reveal long term market trends. Secondly, the agent uses the CHQ algorithm [23], a hierarchical learning method which allows it to change its strategy after making certain observations. The latter is the focus of our research, which explores the hypothesis that an agent with hierarchical reinforcement learning can outperform a standard reinforcement learning agent in a market with regime change.

Both standard and hierarchical agents were tested on a simulated market constructed using data from the Russell 2000 Index [4]. We show that our hypothesis is true in a very trivial setup where neither agent makes any market observations. Unfortunately, we could not show any significant difference between the standard and hierarchical agent in any other experiment. We conclude that our hypothesis cannot be supported given our agent design, but reason that a difference might exist for an improved agent design.

While testing our agent, it was noted that the agent would learn considerably different strategies when given more information about the market or charged a cost for trading. Formal experiments were undertaken to investigate these effects, and it was found that the agent shows some interesting adaptive behaviour. If the agent has more information about the market, it either makes better trades or trades more frequently. If the agent is charged a cost for trading, then it trades less frequently. These are novel results and provide an interesting example of an agent that replicates human adaptivity.

1.1 Outline

The remainder of this dissertation is structured as follows: Chapter 2 provides the background to our research, and contains a brief overview of trading and reinforcement learning. Previous research is discussed and used to motivate our hypothesis. Chapter 3 describes the market and broker components of the simulated environment which our agent operates in. Chapter 4 contains detailed explanations and justifications for each element of our agent design. Chapter 5 begins by describing our experimental methodology, after which the results of a wide range of tests and experiments are presented and discussed in depth. Chapter 6 concludes our research by tying together our results, discussing the shortcomings of our method, proposing possible solutions, and suggesting other avenues for future research.

Chapter 2

Background

In this chapter we give an overview of trading and reinforcement learning while simultaneously motivating our research. We review previous research involving machine learning and algorithmic trading, focusing on approaches that have used reinforcement learning.

2.1 Trading

A financial trader must interpret large amounts of information and make split second decisions. Clearly, traders can use computers to gain an advantage, and so it is not surprising that computers have had a prominent role in trading for decades [14]. Most traders will use computational techniques on some level, even just as tools to assist in decision making. Others allow computers to make decisions about specific aspects of the trade, such as a system where a human trader decides what to trade, but a computer decides when and how much to trade. Some companies use fully autonomous traders.

In order to motivate our research, we begin with a brief overview of the methods used by human and algorithmic traders, and how they differ. We define a human trader as a trader who makes all their own decisions, such as when and what to trade, but may use computers to assist in the process. We define an algorithmic trading system as one where a computer makes some decision about the trade. Algorithmic traders are referred to as *agents*.

2.1.0.1 Human trading

Techniques used by human traders fall under one of two broad approaches: fundamental or technical analysis. Though a trader may prefer one approach, they are likely to incorporate elements from the other.

2.1.0.1.1 Fundamental analysis Fundamental analysis views security prices as a noisy function of the value of the underlying asset, such as performance of a company or demand for a commodity [25]. If the underlying value can be estimated, then it can be determined whether the security is under or overvalued. This information can be used to predict how the security price will change. Similarly, if it is possible to predict the effect that news has on the underlying value of an asset, then price changes can be predicted. For instance, if a company announces record losses, then it is very likely that both the underlying value and price of its stock will fall.

2.1.0.1.2 Technical analysis Technical analysis is based on the assumption that future prices of a security can be predicted from past prices [25]. This view is justified by noting that human traders are somewhat predictable, and claiming this results in repeated patterns in security prices. However, this contradicts the Efficient Market Hypothesis, which states that future prices are independent of all past and present prices. Consequently, technical analysis has traditionally been dismissed by academics [9], despite remaining popular among actual traders [29]. However, there is evidence that technical analysis does work [17, 18], and many are beginning to suspect that the market is not as efficient as assumed [15].

2.1.0.2 Algorithmic trading

The algorithms used by algorithmic traders are diverse, but generally based on methods used by human traders [14]. However, agents are less likely to use fundamental analysis due to the difficulty of interpreting news articles. On the contrary, many use technical analysis type rules, that humans struggle to follow due to slow reaction times or psychological factors, such as panicking when the value of a holding falls rapidly. Some agents implement highly mathematical strategies that have no parallel in human trading due to the extreme amount of computation required.

2.1.1 Machine learning and algorithmic trading

The machine learning community has recently shown much interest in the construction of agents that learn strategies, as opposed to the standard approach of agents with a ‘hand-coded’ strategy. The methods used by these agents need to be learn-able, which rules out the mathematically derived strategies used by some algorithmic traders. As such, research has focused on agents that learn technical analysis type strategies. Some approaches involve agents creating their own rules, while others have involved agents combining rules to create strategies. Early approaches involved neural networks [24]. These are an obvious choice for learning technical analysis type rules, with their ability to map patterns to discreet signals. However, the opaque rules learnt by neural networks means it is hard to characterise their performance.[26].

Dempster & Jones [7] criticised early research for focusing on developing a single trading rule, unlike actual human traders who generally use a combination of rules. A genetic algorithm was used to produce an agent that combined rules and outperformed agents exclusively using one rule. Subramanian et al. [27] showed that this method can be extended to balance risk and profit, and developed an agent that could adapt to different market conditions or *regimes*.

Economic regimes are periods of differing market behaviour that may be part of the natural market cycle, such as bull and bear markets, or may be caused by unpredictable events, such as a change of government, bad news or natural disasters [12]. Although human traders are certainly conscious of such regimes, most researchers have ignored their presence.

Reinforcement learning has been used to train automated traders which map market observations to actions, and though these methods have enjoyed success, none have addressed regime change. Some approaches use technical analysis indicators as market observations [9], while others use the historical price series [20, 11] or a neural network trained on the price series [19, 8, 10, 5].

We argue that the performance of reinforcement learning algorithms in a market environment can be improved if economic regime change is taken into account and modelled as a hidden market state. We therefore propose to use a reinforcement learning algorithm designed for environments with hidden states: hierarchical reinforcement learning. To motivate this argument, we begin with a review of reinforcement learning.

2.2 Reinforcement learning

Reinforcement learning is an approach to machine learning where agents explore their environment and learn to take actions which maximise rewards through trial and error. Sutton & Barto [28] provide a comprehensive introduction to the field, which we follow in this section.

The agent always exists in some state s. It may choose some action a which takes the agent to a new state s' according to some transition probability distribution $P(s'|s,a)$. After taking an action, the agent receives a reward $R^a_{ss'}$. The agent has a policy $\pi(s,a)$ which gives the probability of taking action a at state s. The goal of the agent is to learn an optimal policy π^* which maximises expected reward.

Agents maintain a *state value function* $V(s)$ which estimates the reward expected after visiting s, or a *state-action value function* $Q(s,a)$ which estimates the reward expected after taking action a in state s. Algorithms featuring state-action value functions are known as *Q-learning algorithms*. Given V or Q, a deterministic *greedy* policy π^* can be generated by making $\pi^*(s,a) = 1$ for the action with the highest expected reward. Calculating these rewards from a state value function V requires knowledge of the transition probability distribution P and reward function R, whereas an agent with a state-action value function Q can directly estimate these rewards. Q-learning is therefore better suited to learning in environments where P or R are unknown.

The greedy policy π^* is the optimal policy, provided that the value function from which it is generated is optimal. However, in order to ensure that the value functions converge to optimal values, the agent must continue to sample all states or state-action pairs. The agent must therefore explore, and so the policy used for learning must contain random non-greedy actions.

However, this poses a problem; if the policy contains exploration, then the value function will reflect this policy and not the greedy policy. There are two approaches to reinforcement learning which handle this problem in different ways. In *on-policy* reinforcement learning, the amount of exploration is reduced over time, so that the policy converges to the optimal policy. In *off-policy* reinforcement learning the agent uses a sub-optimal learning policy to generate experience, but uses a greedy policy to update the value function.

2.2.0.0.1 The Markov property An environment satisfies the Markov property if the state transition probabilities P depend only on the current state and action. The learning task is a Markov decision process if the environment satisfies the Markov property. Convergence proofs for popular reinforcement learning algorithms assume that the learning task is a Markov decision process [31, 6].

2.2.0.0.2 Partially observable Markov decision orocesses (POMDPs) A learning task is a partially observable Markov decision process (POMDP) if the environment satisfies the Markov property, but the agent's observations o do not always inform the agent which environment state it is in. This occurs when the agent makes the same observation o in distinct states $s_1 \neq s_2, \ldots, s_n$. This is known as *perceptual aliasing*. The card games Poker and Bridge are POMDPs, as are many robot navigation tasks where the robot's sensors do not uniquely specify its location.

Though standard reinforcement learning is unable to learn optimal policies for POMDPs, it is reasonable to ask whether some modified form of reinforcement learning can. As Hasinoff [13] remarks, humans are adept at solving POMDPs. If humans can learn to play Poker and find their way out of labyrinths, then machines should be able to do the same. Indeed, there has been a considerable amount of research into using reinforcement learning to solve POMDPs [13]. Though the methods used vary, the general idea is to *restore the Markov property* by 'revealing' the underlying Markov decision process.

2.2.0.1 Hierarchical reinforcement learning

HQ-learning is a hierarchical Q-learning algorithm developed by Wiering and Schmidhuber [32] for learning policies for certain POMDPs. The algorithm features a number of separate Q-learning *subagents*. Only one subagent is active at once, and its policy is used for control. When that subagent reaches its goal state, control is transferred to the next subagent. Unlike standard reinforcement learning, the subagents have HQ-tables, which estimate the expected reward of choosing particular goal states. The agents use this to choose their goal state. If there are underlying hidden states which require different policies, then a system with a subagent which is fitted to each hidden state should obtain optimal performance. It follows that subagents should learn to pick goal states that mark transitions between hidden states, as this will maximise expected reward. HQ-learning can be viewed as an elegant way of incorporating memory into reinforcement learning. Though the subagents do not directly store memories, if a subagent is currently active it follows that certain states must have been observed in the past.

CHQ-learning is an extension of HQ-learning by Osada & Fujita [23] which allows the sequence of subagents to be learnt and subagents to be reused. The HQ-tables are modified to estimate the expected reward of choosing a particular goal-subagent combination.

2.2.1 Reinforcement learning and algorithmic trading

The use of Q-learning to develop a simple but effective currency trading agent was demonstrated by Neuneier [20]. Gao & Chan [11] showed that the performance of the agent can be improved by using the weighted Sharpe ratio as a performance indicator instead of returns.

Dempster & Romahi [9] used Q-learning to learn a policy mapping combinations of standard technical analysis indicators to actions. This method was improved by generalising the indicator combinations [10] and using order book statistics in place of indicators [5].

Moody & Saffell [19] argued that reinforcement learning trading agents learn better through a recurrent reinforcement learning (RRL) algorithm. This algorithm uses a neural network to directly map market observations to actions, training the neural network on experience. This bypasses the need for the state-action value function used by Q-learning. Dempster et al. [8] used the actions from this method as 'suggestions' for a higher level agent which takes actions after evaluating risk.

Lee & O [16] designed a portfolio trading system with four Q-learning agents controlling different aspects of trading, A complicated matrix was used to quantify the price series. This method was later expanded on [21] in a system where multiple local agents make recommendations about what to buy, which is used as the state space for a reinforcement learning agent which outputs the amount to allocate to each agent.

2.2.1.1 Restoring the Markov property in a market

While some previous researchers have acknowledged the non-Markov nature of markets, there has been no substantial discussion on how to restore the Markov property in a market. Before attempting to restore the Markov property in a market, we must ask whether the market actually has an underlying Markov decision process. Consider the technical analysis assumption; that future prices can be predicted from past prices. Clearly, then, the prices do not follow a Markov decision process. However, instead of individual prices, consider a short series of historical prices. We can assume that there is some limit on the effect prices have on the future. For example, we might assume that future prices are independent of prices more than two weeks in the past. Then we can say that *series of prices* follow a Markov decision process.

However, since prices are continuous, there will be infinitely many price series, so we cannot directly use the price series as a state. One possible solution is to use a neural network to map price series directly to values or even actions, bypassing the need for a discreet value function and circumventing the state space issue. This is the approach used by Gao & Chan [11], Moody & Saffell [19] and in the later work of Dempster et al. [8]. An alternative solution is to assume that it is possible to fully capture the behaviour of a price series in certain discreet indicators, such as technical analysis indicators. Then instead of using all the prices as the state, discreet indicators can be combined to get a discreet state. This is the approach used by Neuneier [20], Lee & O [16, 21] and in the earlier work of Dempster et al. [9, 10, 5],

However, if we take regime changes into account, then restoring the Markov property becomes more complicated. We assume that, under different regimes, price series transition to other price series with different probabilities. Then an agent that has some knowledge of the underlying regime will perform better than an agent that does not. Identifying market regimes requires price series at a lower frequency than needed to predict short term movements (e.g. daily prices instead of hourly prices). If a neural network is being used to restore the Markov property, then a lower frequency price series could be presented to the network alongside the usual high frequency series. Similarly, if indicators are being used, equivalent indicators for the lower frequency price series can be included alongside the standard indicators.

However, traders may need to remember which states they have seen in the past in order to identify the underlying regime. For example, when some signal appears, a trader might take it to mean that the market has now turned bearish and adjust its strategy accordingly. If an agent can only see the current market observation, it is not capable of this behaviour. The significance of this weakness is highlighted by noting that some signals may only appear for a single time step. Conversely, a hierarchical reinforcement learning agent using the HQ-learning algorithm outlined above would be capable of making these distinctions. For instance, if a subagent tailored to bullish markets is active but observes the bearish signal, it would be capable of switching to a subagent tailored to bearish markets.

Thus we propose to restore the Markov property in three steps. First, by using technical analysis indicators to compress the price series into distinct observations. Then, by adding long term technical analysis indicators to assist in the identification of underlying regimes. Finally, we use hierarchical reinforcement learning so that the agent can disambiguate underlying regimes based on signals. Though there is evidence that neural network based approaches are superior [8], we use technical analysis indicators to facilitate hierarchical reinforcement learning. This leads us to hypothesise that *re-*

inforcement learning agents using hierarchical reinforcement learning are superior to standard reinforcement learning agents in markets with regime change.

Chapter 3

The simulation

In this chapter we describe the market and broker simulator. The market simulator provides the continuous price series that the agent compresses into discreet observations. The broker simulator establishes a framework which determines the effects of the agent's actions. As such, it is necessary to outline the simulation before describing the agent design.

3.1 The market simulator

Real data from the Russell 2000 Index [4] was used to simulate a market with a single asset. The Russell 2000 index is related to the Russell 3000 index, which measures the performance of the 3000 largest companies in the US. However, the Russell 2000 index only considers the 2000 smallest securities from the Russell 3000 index, resulting in an index which is free of the noise introduced by the largest companies. This has made it popular among technical analysis traders.

Tick by tick data was preprocessed to create a price vector for each fifteen minute interval. Each price vector contains the following information:

Open Price The price of the first trade in the interval

Low Price The lowest price of any trade in the interval

High Price The highest price of any trade in the interval

Close Price The price of the last trade in the interval

Ask Price The current ask price at the start of the interval

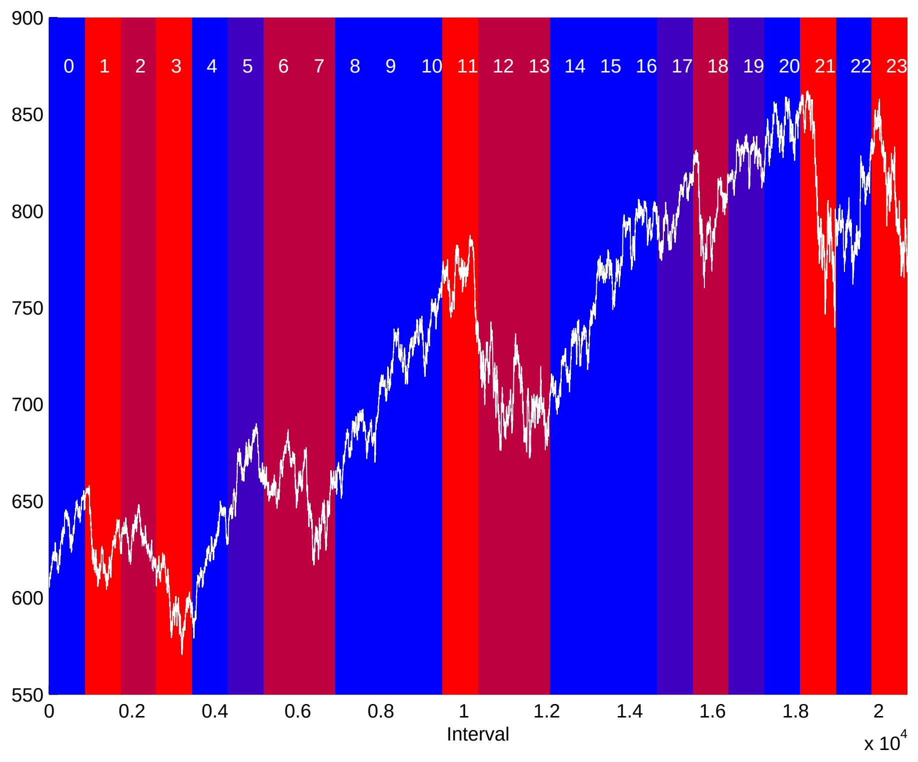


Figure 3.1: Russell 2000 Index (2004-11-10 to 2007-11-15) 15 Minute Intervals

Bid Price The current bid price at the start of the interval

The open, low, high and close prices are *market prices*: prices at which trades actually took place. The ask price is the lowest price at which some trader is willing to sell stock, and the bid price is the highest price at which some trader is willing to buy stock.

The close prices are plotted in figure 3.1. The market has been split into 24 datasets, each which has been given a colour according to the following scheme:

Bright Blue Rapidly rising segment: Close price is above open price and average price

Soft Blue Rising segment: Close price is above open price but below average price

Soft Red Falling segment: Close price is below open price but above average price

Bright Red Rapidly falling segment: Close price is below open price and average price

The colours are later used to investigate how *robust* the agents strategies are: how well they perform under different market conditions. The scheme is designed so that bright segments have very definite trends, while the soft segments have a noisier sideways trend. For the purposes of our research, we take these different segments to represent different regimes. This is a simple approach; more sophisticated ways of segmenting the market could certainly be used.

The first eight segments are assigned to the training dataset, the second eight to the validation dataset and the final eight to the test dataset. The training dataset is used for agent learning, while performance on the validation dataset is used to tweak parameters. The test set is used to test performance on completely unseen data. The Russell 2000 index between 2004 and 2007 is well suited to this sort of partition, since each dataset contains diverse market conditions. A market segment of each colour is found in each dataset, with the exception of soft blue and the validation set. This is far more diverse than the FTSE100 index, which was originally investigated.

3.2 The broker simulator

The broker simulator determines how the agent can trade on the simulated market. For the sake of realism, the broker simulates a contract for difference (CFD) service which is offered by many real brokers [1, 3]. Traders take out a contract with the broker, stating that the seller will pay the buyer the difference between the current price of asset (e.g. a number of shares) and the price of the asset at some future time. The trader can either be the buyer or seller in the contract, these are known respectively as long and short positions. The price of the contract does not necessarily reflect the price of the asset; one of the big attractions of CFDs is that they allow traders to take positions on stock they could not actually afford. However, the trader will ultimately end up paying or receiving the full increase or loss for the asset.

Traders must pay the broker commission for both opening and closing a position, as well as a funding cost every day. A realistic commission price would be 0.25% of the opening asset value [1]. Long positions accumulate funding costs on a day by day basis, whereas short positions accumulate a funding dividend. Funding costs are based on bank interest rates. This is because the broker has to borrow the money used to buy the asset when the trader takes a long position, but can invest the money from selling the asset when the trader takes a short position. Since we do not include bank interest rates in our simulation, funding costs are ignored. We do however experiment with comission prices.

We choose to use CFDs for two reasons. Firstly, they do not discriminate between long positions, which profit from rising prices, and short positions, which profit from falling prices. Our agent therefore has the potential to make money in both rising and falling markets. Secondly, CFDs can be closed at any time, which means we can impose less restrictions on how the agent acts.

When an agent opens a long position, the ask price from the market simulator is used as the current price, and when the position is closed, the bid price is used. Conversely, short positions are opened at the bid price and closed at the ask price. The agent's indicators are based on close prices, so it has no knowledge of the bid or ask price when it makes a trade. This difference between the price a trader sees when it decides to open or close a position and the actual price the position is opened or closed at is known as *slippage*.

Our simulation is not entirely realistic. It is assumed that the agent has no effect on the market and can always open or close a position. These assumptions are partially justified by restricting the agent to small trades. Creating a fully realistic simulator is not trivial and is beyond the scope of this investigation. However, our assumptions are not unprecedented and the use of ask and bid prices means our simulation is more realistic than simulations which only use market prices.

Chapter 4

Agent design

In this chapter we describe each aspect of the agent design: the observation space, actions, reward function and algorithm.

4.1 Observation space

The agent's observation space combines information about the agent's current holdings with information about the market. Note that we use the term 'observation space' instead of 'state space' as in standard reinforcement learning. This is a technical formality. In standard reinforcement learning, observations map to unique states, so there is no need for the distinction. However, we cannot guarantee this will be the case in our market environment, though we hope that observations map to certain states with high probability after the Markov property has been restored.

At each time step t the agent has an observation vector $o_t = (h_t, m_t, e_t)$ where h_t is the agent's holding state, m_t is a market observation and e_t is a binary variable indicating whether t is the last time step in an an episode. This binary variable is included because the agent is forced to close its position at the last time step in an episode, and so the available actions are different when $e_t = 1$.

4.1.1 Holding state

The holding state h_t is a discreet variable taking values $h_t \in \{0, \ldots 8\}$. The value informs the agent whether it has a long or short position and how well this position is performing. The values are explained in figure 4.1 where profit is defined as open price $+$ close price $-(($ open price $\times$ commission $) + ($ close price $\times$commission$))$.

	State	**Actions**
0	Long: Profit over upper threshold	Close Position, Hold Position
1	Long: Profit over zero	Close Position, Hold Position
2	Long: Loss over lower threshold	Close Position, Hold Position
3	Long: Loss below lower threshold	Close Position, Hold Position
4	Short: Profit over upper threshold	Close Position, Hold Position
5	Short: Profit over zero	Close Position, Hold Position
6	Short: Loss over lower threshold	Close Position, Hold Position
7	Short: Loss below lower threshold	Close Position, Hold Position
8	No Position	Open Long Position, Open Short Position

Figure 4.1: Holding States

The upper and lower thresholds are respectively positive and negative, and are included to give the agent the potential to ‘wait’ until the profit moves outside these thresholds before making a decision. Without these thresholds the agent tended to trade too often. The thresholds are fixed parameters which must be chosen independently of the learning algorithm; a good choice would depend on the market.

The distinction between profitable and lossy positions is also included to improve learning. Even if the profits made vary wildly, closing a position when it is profitable should usually result in a positive reward, and vice versa. Thus, even if the agent is unable to converge on accurate values, it should be able to converge on values of the correct *sign*. Without this distinction (i.e. just the upper and lower thresholds) the agent had difficulty learning what to do immediately after opening a position.

Lee and O [16] use a similar threshold based representation, but most prior approaches simply inform the agent what it is holding.

4.1.2 Market observation

Three popular technical analysis indicators are used to generate discreet signals from the historical price series: Bollinger Bands, RSI and the Fast Stochastic Oscillator (KD). Each indicator is discussed in detail below, where we follow the descriptions of Schwager [25]. Signals may be generated at both short and long time scales. The short signals capture the behaviour of prices from the last day, while long signals capture behaviour from the last week. The vectors $m_s \subseteq \{BB_t^s, RSI_t^s, KD_t^s\}$ and

$m_l \subseteq \{BB_t^l, RSI_t^l, KD_t^l\}$ give complete long and short term signals for a given time step. Note that m_s and m_l may be any subset of these indicators; the agent does not necessarily use all of them.

The complete market observation is an vector m_t of short and long signals. Several combinations are experimented with:

S0L0 $m_t = \emptyset$ (no indicators)

S1L0 $m_t = (m_t^s)$

S2L0 $m_t = \left(m_{t-1}^s, m_t^s\right)$

S0L1 $m_t = \left(m_t^l\right)$

S1L1 $m_t = \left(m_t^s, m_t^l\right)$

S2L1 $m_t = \left(m_{t-1}^s, m_t^s, m_t^l\right)$

Agents using combinations with a long term signal are expected to perform well, since they should be better at identifying underlying market regimes. In some combinations the short term predictions for both the current and previous time step are used. Agents using these combinations are also expected to perform well, since a number of technical analysis rules are based on changing signals rather than the signals themselves. Such rules exist for all three chosen indicators. Note that we have not specified these rules, but have given the agent the potential to discover them.

We restrict the agent so that each short or long term signal in its combination is formed using the same technical analysis indicators (e.g. $m_t^s = \{BB_t^s, RSI_t^s\}$ and $m_t^l = \{BB_t^l, RSI_t^l\}$). This gives our agent 46 different ways of generating market observations. We investigate the performance of agents using many of the combinations, though some are infeasible due to the resulting size of the observation space.

4.1.2.1 Bollinger Bands

Bollinger Bands are based on an n-period simple moving average, which is simply the average value of some indicator over the last n-periods (15 minute intervals in our case). Moving averages smooth out short term price fluctuations and are typically used to reveal long term price trends. The Bollinger Bands are two bands which are constructed by plotting two standard deviations above and below the moving average. The movement of the price in and out of the bands is thought to be informative.

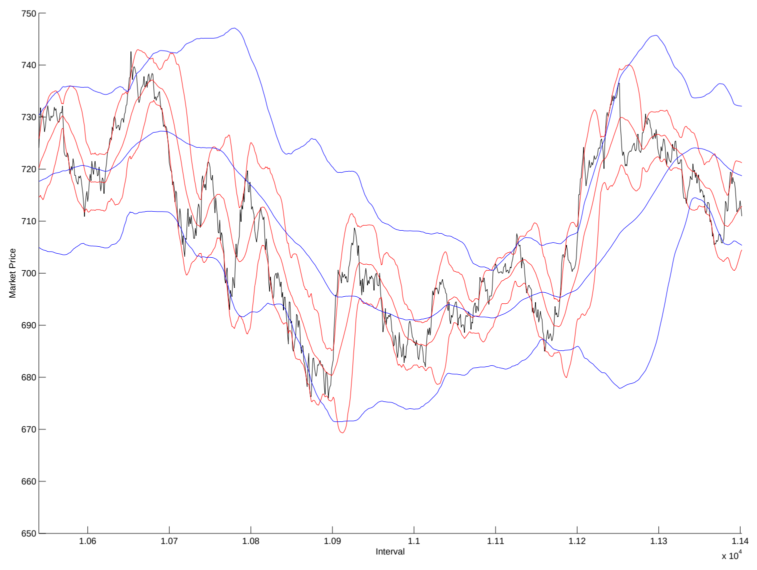


Figure 4.2: Bollinger Bands for the closing price from dataset segment 12 (Red: Short, Blue: Long)

Signal	**Description**
0	Market price above upper band
1	Market price between moving average and upper band
2	Market price between lower band and moving average
3	Market price below lower band

Figure 4.3: Scheme used to generate discreet signals from Bollinger Bands

A 20-period moving average is usually used by to plot Bollinger Bands for day by day data. For our short indicator, we have used 28-periods, which is the median number of intervals in a day for our dataset. For our long indicator, we used 140-periods, which is 5×28: the median number of intervals in a week. Figure 4.2 plots the short and long Bollinger Bands for the closing price from segment 12 of the dataset.

Figure 4.3 shows the scheme used to generate discreet signals from the Bollinger Bands. The discretization is based on two popular technical analysis signals that involve Bollinger Bands. Firstly, the market price moving above the upper band and below the lower band are respectively considered to signal a fall or rise in price. Secondly, the market price moving above and below the moving average respectively indicate up and down trends. This should make it clearer why we have included signal

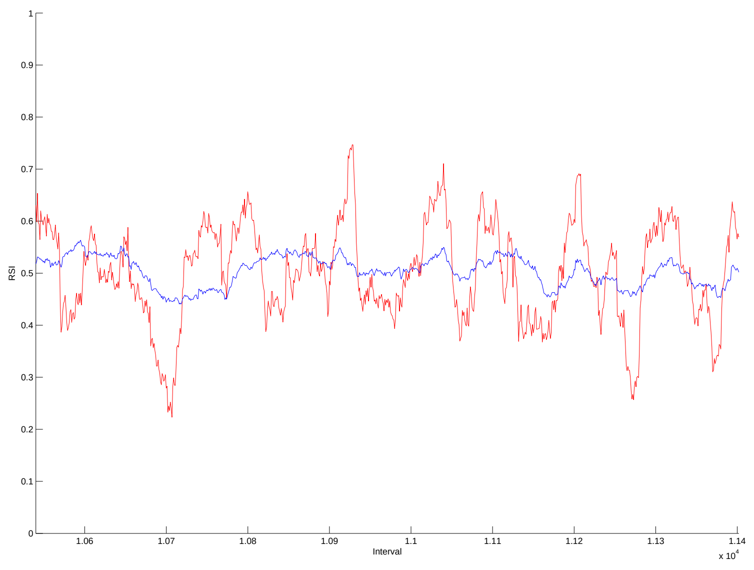


Figure 4.4: RSI for the closing price from dataset segment 12 (Red: Short, Blue: Long)

combinations using observations from two consecutive time steps. It allows agents to act when signals change, which indicates that one of the bands or the moving average has been crossed.

4.1.2.2 RSI

The RSI is a popular directional indicator which was introduced by J. Welles Wilder in 1978. RSI stands for Relative Strength Index, but the long form is seldom used to avoid confusion with other similarly named indicators. RSI measures the proportion of price increases to overall price movements over the last n periods. It is given by

$$RSI = \frac{\bar{g}}{\bar{g} + \bar{l}}$$

where $\bar{g}$ and $\bar{l}$ are the average gain and loss for the n periods. The average gain is computed by ignoring periods where the price fell, and average loss is computed analogously. Note that we are expressing RSI as a ratio, which differs from the standard practice of expressing it as a percentage.

As with the Bollinger Bands, 28-periods are used for the short indicator and 140-periods are used for the long indicator. Figure 4.4 plots the short and long RSI for the closing price from segment 12 of the dataset.

Wilder recommended using 0.7 and 0.3 as overbought and oversold indicators. The

Signal	Short RSI Indicator	Long RSI Indicator
0	$RSI \geq 0.65$	$RSI \geq 0.55$
1	$0.65 \geq RSI \geq 0.5$	$0.55 \geq RSI \geq 0.5$
2	$0.5 \geq RSI \geq 0.35$	$0.5 \geq RSI \geq 0.45$
3	$0.35 \geq RSI$	$0.45 \geq RSI$

Figure 4.5: Scheme used to generate discreet signals from RSI

RSI falling below 0.7 is then considered to signal a falling market, while the RSI rising above 0.3 is considered to signal a rising market. However, the variance of the RSI depends on the period used, which can be clearly seen in the difference between the curves in figure 4.4. Wilder's recommendations of 0.7 and 0.3 were for an RSI with 14-periods. In order to generate reasonable signals, different levels were used for our 28 and 140 period indicators: 0.65 and 0.35, and 0.5 and 0.45 respectively. These were chosen so that the RSI moving outside these bounds was significant, but still likely to occur at least once during a given dataset segment. Figure 4.5 shows the resulting scheme used to generate discreet signals. We also use 0.5 to partition the values since it provides a useful directional indicator.

4.1.2.3 Fast Stochastic Oscillator (KD)

The Fast Stochastic Oscillator (KD) was developed by George C. Lane and indicates how the current price of a security compares to its highest and lowest price in the last n-periods. It is derived in two steps. Let p_{max} and p_{min} be the highest and lowest prices in the last n-periods and let p_t be the current price. Then

$$K = \frac{p_t - p_{min}}{p_{max} - p_{min}}$$

Again, a ratio is used instead of the standard percentage. Note that if $p_t = p_{min}$ then $K = 0$, while if $p_t = p_{max}$ then $K = 1$. K can be thought of as representing 'how far' the current price is between the n-period high and low. The Fast Stochastic Oscillator KD is simply a smoothed version of K, and is defined as the m-period moving average of K, where $m \leq n$. The indicator has the effect of preserving the peaks and dips in the price series, while compressing it between 0 and 1.

For our short indicator, 28-period K and a 4-period (one hour) moving average are used. For our long indicator, 140-period K and a 28-period moving average are used.

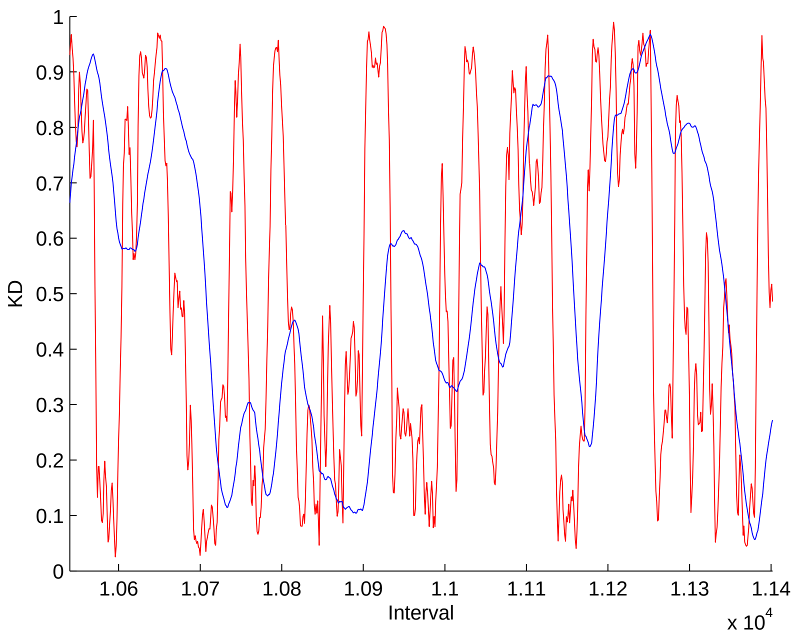


Figure 4.6: KD for the closing price from dataset segment 12 (Red: Short, Blue: Long)

Signal	**Description**
0	$KD \geq 0.8$
1	$0.8 \geq KD \geq 0.2$
2	$0.2 \geq KD$

Figure 4.7: Scheme used to generate discreet signals from KD

Figure 4.6 plots the short and long KD for the closing price from segment 12 of the dataset.

Lane recommended that 0.2 and 0.8 are used as oversold and overbought levels. These levels are used in our scheme to generate discreet signals from KD, shown in figure 4.7.

4.2 Actions

The actions available to an agent at time t depend on its current holding state and whether t is the final interval in the episode. If the latter is true, the agent is forced to close any position it has. Otherwise, if the agent has a position it may hold it or 'invert' it by closing that position and opening the opposite position (e.g. long if the original

Final Time Step?	**Holding States**	**Actions**
Yes	$\{0,...,7\}$	Close Position
No	$\{0,...,7\}$	Invert Position, Hold Position
No	$\{8\}$ (No position)	Open Long Position, Open Short Position

Figure 4.8: Actions

position was short). If it has no position it may open either a long or short position. This is shown formally in figure 4.8.

The agent begins in the ‘no position’ state, but cannot return there; the agent must always hold a position. This is partly due to learning issues, but also reflects the idea that either going long or short must be as good or better than doing nothing. Taking an action causes the agent to move forward one time interval. Cash is completely ignored except for measuring performance; it is assumed that the agent always has enough cash to open a position or cover its losses.

The agent simply opens and closes positions on the minimum amount of shares (1). This is done to justify two assumptions that were made to simplify the simulation: that the agent does not effect the market, and that the agent can always open and close its position. Trading multiple stocks would also introduce a new problem; how to best time the trades. We want to ignore this issue and focus on basic performance. However, it is not unreasonable to say that a strategy that is profitable trading on one stock can be adapted to trading multiple stocks and remain profitable. Other research has typically concerned similar small positions, with the exception of the work of Lee & O [16, 21].

4.3 Reward functions

The agent only receives reward for closing a position, which means the agent must have enough lookahead so that it learns to associate opening a position with the reward it later receives for closing it. The algorithm is designed with this in mind. We experimented with three different reward functions.

4.3.1 Profit

Profit is the obvious reward function, as it is generally a good idea to reward reinforcement learning agents with whatever quantity is being maximised. Additive profits are used, since the fixed ‘smallest position’ structure is being used and so the agent is not reinvesting any profits into additional positions.

4.3.2 Profit Made Good

The Profit Made Good indicator is based on the Sharpe Ratio indicator. Using only profit as a reward function is often criticised, since practical trading systems need to take risk into account. A popular alternative to profit is the Sharpe Ratio which measures the returns per unit of risk. It is defined as

$$SR = \frac{E(R - R_f)}{\sqrt{var(R)}}$$

where R is returns and R_f is the return of some risk free asset. Since we ignore the existence of risk free assets, the Sharpe ratio becomes.

$$SR = \frac{E(R)}{\sqrt{var(R)}}$$

However, our agent can potentially make very rapid trades, even switching positions in consecutive intervals. The Sharpe Ratio is not defined for a single return, since variance is zero. Moreover, the Sharpe Ratio tends to be unnaturally high or low for a small number of returns. While it is a good measure for long term performance, it is a poor choice for providing immediate reward for small trades. This motivated us to design the Profit Made Good indicator.

$$PMG = \frac{E(R)}{E(|R|)} = \frac{\sum R}{\sum |R|}$$

Intuitively this can be thought of as the ratio of returns to return movement. If all returns are negative, then $PMG = -1$, if the sum of returns is zero then $PMG = 0$ and if all returns are positive then $PMG = 1$. It is also well defined for a single return. It still suffers from unnaturally high or low values for a short number of returns, but since it is bounded between -1 and 1, this is less of an issue. It also requires less computation.

PMG is related to the Sharpe Ratio, as both express some relationship between returns and asset volatility. It was informally observed that they appear to approach an approximately linear relationship for a large number of returns.

4.3.3 Composite

The composite reward function is defined as the product of profit and Profit Made Good:

```
1. For each episode
    (a) Make observation and choose starting action
    (b) Do
        i. Take action, get reward and new observation
        ii. Choose the next action and find the optimal action
        iii. Compute optimal Q-value using the optimal action
        iv. Set eligibility trace for current observation-action pair
        v. Use the optimal Q-value to update the Q-values of eligible state-action
           pairs
        until episode terminates
```

Figure 4.9: Sketch of Watkins-Q algorithm

$$profit \times PMG$$

This is intended to provide a compromise between the two indicators.

4.4 The algorithm

The algorithm is a hybrid of the Watkins-Q algorithm given by Sutton & Barto [28] and the CHQ algorithm given by Osada & Fujita [23].

4.4.1 The Watkins-Q algorithm

The Watkins-Q algorithm is an off-policy Q-learning algorithm with eligibility traces. Eligibility traces are a way of dealing with temporal credit assignment. A sketch of the algorithm is given in figure 4.9 and a pseudo code version with more detail is given in figure 4.10. Visited states are given a trace which is decayed at every time step. The trace indicates how eligible a state is for receiving rewards at the current time step. Effectively, eligibility traces allow state-action pairs to receive reward for future actions. This sort of ‘lookahead’ is important in our agent design; recall that the agent only receives reward for closing trades. Thus, it needs to be able to link the reward received when the trade is closed to the action of opening it.

```
Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s, a$
Repeat (for each episode):
    Initialize $s$, $a$
    Repeat (for each step of episode):
        Take action $a$, observe $r$, $s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $a^* \leftarrow \arg\max_b Q(s', b)$ (if $a'$ ties for the max, then $a^* \leftarrow a'$)
        $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s,a)$
        $e(s,a) \leftarrow e(s,a) + 1$
        For all $s, a$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
            If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                              else $e(s,a) \leftarrow 0$
        $s \leftarrow s'$; $a \leftarrow a'$
    until $s$ is terminal
```

Figure 4.10: Watkins-Q algorithm in pseudo code (from [28])

4.4.2 The CHQ algorithm

The CHQ algorithm is a modification of the HQ-learning algorithm proposed by Wiering and Schmidhuber [32]. HQ-learning uses a system with M subagents $C_1, C_2, \ldots, C_M$ that learn through conventional Q-learning. However, each subagent also has a HQ-table HQ_i .

Learning is performed on episodes with $t = 1, 2, \ldots, T$ discreet time steps. Subagent C_1 is activated at the beginning of the episode and greedily chooses a subgoal g from its HQ-table. It follows a policy based on its Q-value function Q_i until subgoal g is reached, at which point control is transferred to the next agent C_2. This continues until the episode terminates when $t = T$, then Q and HQ-values are updated. Only one subagent is active at a given time step.

The CHQ algorithm proposed by Osada & Fujita [23] simply modifies the HQ-learning algorithm so that subagents pick the next subagent in addition to the goal. Consequently, CHQ-learning can deal with repetitive tasks that HQ-learning cannot. A sketch of the algorithm is given in figure 4.11. Unlike the Watkins-Q algorithm, the agent does not immediately learn from experience. This takes place through a form of batch updating at the end of each episode. These two learning paradigms are known as online and offline updating. This should not be confused with on and off policy algorithms; both algorithms are off-policy. A pseudo code version of the algorithm is given in figure 4.12. The following conventions are used: t = time step, i = ith active subagent, o = observation, a = action, s = subagent, g = goal, n = next subagent and r

```
1. For each episode
   (a) Make observation and choose starting action, goal and next subagent
   (b) Do
         i. Take action, get reward and new observation
        ii. If the new observation matches the goal state, swap to new subagent
       iii. Choose the next action
       until episode terminates
   (c) Update Q-values
         i. Calculate optimal Q-value at each time step
        ii. Update Q-values according to these optimal Q-values
   (d) Update HQ-values
         i. Calculate optimal HQ-value for each subagent used
        ii. Update HQ-values according to these optimal HQ-values
```

Figure 4.11: Sketch of CHQ-algorithm

```
1. Initialise all values in $Q_i$ and $HQ_i$ to zero for each $i \in \{1, \dots M\}$
2. Repeat for each episode
   (a) Let $t = i = 1$
   (b) Let $s_i = 1$
   (c) Choose $g_i$ and $n_i$ greedily from $HQ_{s_i}(g,n)$.
   (d) While $t \leq T$
       i. Select action $a$ greedily from $Q_{s_i}(o_t, a)$.
       ii. Do $a$, get reward $r_t$ and new observation $o_{t+1}$
       iii. If $o_t = g_i$
            • $s_{i+1} \leftarrow n_i$
            • $i \leftarrow i+1$
            • Choose $g_i$ and $n_i$ greedily from $HQ_{s_i}(g,n)$
       iv. $t \leftarrow t+1$
   (e) Update Q-values
       i. Find the ideal Q-values $Q^t$
          A. $Q^T \leftarrow r_t$
          B. For each $t = 1, \dots, T-1$
             • $Q^t \leftarrow r_t + \gamma((1-\lambda)\max_a(Q_{s_t}(o_{t+1})) + \lambda Q^{t+1})$
       ii. Then $Q_{s_t}(o_t, a_t) \leftarrow (1-\alpha^Q)Q_{s_t}(o_t, a_t) + \alpha^Q Q^t$
   (f) Update HQ-values Let $s_N$ be the last subagent in the episode
       i. Find the ideal HQ-values $HQ^i$. Let $R_i = \sum_{t=t_i}^{t_{i+1}-1} \gamma^{t-t_i} r_t$.
          A. $HQ^N \leftarrow R_N$
          B. For each $i = N-1, \dots, 1$
             • $HQ^i \leftarrow R_i + \gamma^{t_{i-1}-t_i}((1-\lambda)\max_{g,n}(HQ_{s_i}(g,n)) + \lambda HQ^{i+1})$
       ii. Then $HQ_{s_i}(g_i, n_i) \leftarrow (1-\alpha^{HQ})HQ_{s_i}(g_i, n_i) + \alpha^{HQ} HQ^i$
```

Figure 4.12: CHQ algorithm in pseudo code

```
1. For each episode
   (a) Make observation and choose starting action, goal and next subagent
   (b) Do
       i. Take action, get reward and new observation
       ii. Choose the next subagent and find the optimal next subagent
       iii. Choose the next action and find the optimal action
       iv. Compute optimal Q-value using the optimal action and subagent
       v. Set eligibility trace for current observation-action-subagent triple
       vi. Use the optimal Q-value to update the Q-values of eligible state-
           action-subagent triples
       vii. If the subagent has changed, choose new goal and next subagent
       until episode terminates
   (c) Update HQ-values
       i. Calculate optimal HQ-values for each subagent used
       ii. Update HQ-values according to these optimal HQ-values
```

Figure 4.13: Hybrid Algorithm Sketch

= reward. We abuse the notation and use both s_i and s_t, where the former means the ith active subagent and the latter means the subagent active at time t.

4.4.3 Hybrid algorithm

In early informal experiments, both the Watkins-Q and CHQ algorithms were used to train non-hierarchical agents. CHQ was used with a single subagent. The aim of this experimentation was to gain some idea of the suitability of the algorithms in our market environment.

Agents trained with the Watkins-Q algorithm learnt much faster than those trained with CHQ, which we assumed to be a result of the online learning in Watkins-Q. This motivated us to develop an online version of the CHQ algorithm. This is a reasonable endeavour; Wiering & Schmidhuber [32] remarked that online versions of HQ-learning should work. A fully online algorithm would be impractically complicated, so we developed an algorithm where Q-learning is online, but HQ-learning is offline. This is a reasonable compromise, since Q-values are used more frequently, and thus are more likely to benefit from faster, online learning. A sketch of the algorithm is given in 4.13. Note that the eligibility traces function is changed to map to state-action-*subagent* triples.

The algorithm was also modified to make it better suited to our market simulation. Instead of goal observations g, the agent maintains some partition of the observation space $\{G_1, \ldots, G_n\}$. The agent reaches its goal when it observes any observation g in its chosen *goal set* G. The partitions ensure that there is a good chance of goals being reached in a given dataset segment, which would not be the case if single observations were used as goals. Three partitions are used in our experiments:

- G_i contains all observations where holding state $h_t = i$
- G_i contains all observations where short market signal $m_t^s = i$
- G_i contains all observations where long market observation $m_t^l = i$

Another modification was made to the algorithm, so that observing a goal state $g \in G$ does not 'count' as reaching the goal unless some other observation $o \notin G$ has been observed since the subagent assumed control. This prevents two subagents with the same goal alternating at every time step.

We also found it necessary to decay the learning parameter α^Q in order to ensure convergence. Convergence is otherwise poor, which we assume to be caused by the non-Markov nature of the market, even after our attempts at restoring the Markov property. The downside to decaying the learning parameter is that early experience tends to have a disproportionate effect on the final policy learnt by the agent. However, the variation caused by this is limited, so it is possible to pick combinations of parameters that cause the agent to converge to good policies *on average*. The parameter α^{decay} controls the rate of decay. The separate learning rate for the HQ-values, α^{HQ} is not directly decayed, but instead tied to α^Q by the parameter α^H:

$$\alpha^{HQ} = \alpha^{Q^{\alpha^H}}$$

A pseudo code version of the complete hybrid algorithm is given in figure 4.14. The hierarchical structure significantly complicates the selection of the optimal action. A list of active eligibility traces $\mathbf{e}$ is maintained to reduce computation time.

```
1. Initialise all values in $Q_i$ and $HQ_i$ to zero for each $i \in \{1, \ldots M\}$
2. Repeat for each episode
   (a) Let $t = i = 1$
   (b) Let $s_t = 1$
   (c) Choose $G_i$ and $n_i$ greedily from $HQ_{s_t}(g,n)$
   (d) Let $G^*$ and $n^*$ be the goal-set and next subagent maximising $HQ_{s_t}(g,n)$
   (e) Let $a_1$ be the action maximising $Q_{s_1}(o_1, a)$.
   (f) While $t \leq T$
       i. Do $a_t$, get reward $r_t$ and new observation $o_{t+1}$
       ii. If $o_{t+1} \in G_i$ and $o_t \notin g$ then $s_{t+1} = n_i$ otherwise $s_{t+1} = s_t$.
       iii. If $o_{t+1} \in G_i^*$ and $o_t \notin g^*$ then $Q^* = Q_{n^*}$ and $s^* = n^*$, otherwise $Q^* = Q_t$
            and $s^* = s_t$.
       iv. Choose $a_{t+1}$ greedily from $Q_{s_{t+1}}(o_{t+1}, a)$
       v. Let $a^*$ be the action maximising $Q^*(o_{t+1}, a)$.
       vi. If $t = t_{max}$, $\delta \leftarrow r - Q_t(o_t, a_t)$ otherwise $\delta \leftarrow r + \gamma Q^*(o_{t+1}, a^*) -$
           $Q_{s_t}(o_t, a_t)$.
       vii. Set $e(o_t, a_t, s_t) \leftarrow 1$ and place $o_t, a_t, s_t$ on $\mathbf{e}_t$.
       viii. For all $\hat{o}, \hat{a}, \hat{s}$ on $\mathbf{e}_t$.
             A. $Q_{\hat{s}} \leftarrow Q_{\hat{s}}(\hat{o}, \hat{a}) + \alpha\delta e(\hat{o}, \hat{a}, \hat{s})$.
             B. If $a_{t+1} = a^*$ and $s_{t+1} = s^*$, then
                • $e(\hat{o}, \hat{a}, \hat{s}) \leftarrow \gamma\lambda e(\hat{o}, \hat{a}, \hat{s})$.
                • If $e(\hat{o}, \hat{a}, \hat{s}) > 0.01$, place $\hat{o}, \hat{a}, \hat{s}$ on $\mathbf{e}_{t+1}$.
       ix. If $o_{t+1} \in G_i$ and $o_t \notin G_i$
           A. $i \leftarrow i + 1$
           B. Choose $G_i$ and $n_i$ greedily from $HQ_{s_{t+1}}(g,n)$
           C. Let $G^*$ and $n^*$ be the goal-set and next subagent maximising
              $HQ_{s_{t+1}}(g,n)$.
       x. $\alpha \leftarrow \alpha \cdot \alpha^{decay}$
       xi. $t \leftarrow t + 1$
   (g) Update HQ-values (As in CHQ algorithm)
```

Figure 4.14: Hybrid Algorithm

4.4.4 Training procedure

In the context of our trading simulator, an episode is a complete sweep through a segment of the dataset. During training, the agent cycles through segments 0-7 of the complete dataset, then repeats the process. Multiple sweeps over the same epsiodes are necessary to provide the agent with sufficient experience. The possibility of the agent over-fitting to the training dataset is hoped to have been reduced by splitting it into segments with different market conditions.

Chapter 5

Experiments

We begin this chapter by outlining the methodology used in our experiments. We explain how performance is measured and describe which statistical considerations are made. The remainder of the chapter covers a large number of tests that are split into two sections: *parameter tests* and *experiments*. The former are used to identify values for parameters that were required for the algorithm, but are neither interesting or relevant to the hypothesis. These tests involve looking at a broad range of parameters in little depth.

The experiments are used to investigate aspects of our agent that produce results which are interesting or relevant to the hypothesis. There are five in total, looking at the performance of the hierarchical agent, the combination of technical analysis indicators, the combination of long and short signals, commission rates and performance on unseen data. The presentation and discussion of results are interleaved for both parameter tests and experiments, and so there is no separate 'results' discussion.

5.1 Methodology

5.1.0.0.1 The Buy and Hold Strategy We sometimes refer to the *buy and hold strategy* and use it as a control in some of our experiments. This is a simple strategy where a long position is taken at the start of the trading period and closed at the end. This tends to make money because market prices tend to rise over time. A good trading strategy should at least be able to do better than the buy and hold strategy.

5.1.1 Measuring performance

The performance of an agent with given parameters is measured through several indicators. To evaluate the effect that parameters have on agent performance, a number of agents are trained with the same parameters, and the values of indicators are averaged over all the agents.

5.1.1.1 Indicators

5.1.1.1.1 Profit Profit is our primary indicator, since profit is what any trader ultimately seeks to maximise. In the majority of parameter tests, this is the only indicator used.

5.1.1.1.2 Number of trades The number of trades made by an agent gives us a rough idea of what sort of strategy it is using. It if is only trading once per market segment, then it must be using some sort of ‘buy and hold’ or ‘sell and hold’ strategy. The value of other indicators should be questioned if the number of trades is low. If the agent only trades a few times per market segment, then its strategy has not been tested enough to say much about its performance.

5.1.1.1.3 Long/short profit Long/short profit compares the profit made from long and short positions. The numbers should at least both be positive, but it is expected that profit made from short positions is smaller. This is because our price series is not symmetrical and rises more than it falls. Making money off short positions is therefore harder since the timing of closing the position is critical. Long positions should ideally be closed at ‘peaks’ in the market. It is not critical if a peak is missed, because the price generally increases over time and the missed peak will likely be surpassed. Short positions should ideally be closed at ‘troughs‘. However, if a trough is missed, there is less chance that the market will return to this low point in the future.

5.1.1.1.4 Maximum down-draft Maximum down-draft measures the biggest potential loss made by the agent on any single position. Potential loss is calculated at every time step, and measures how much of a loss (or profit) would be made if the agent closes its position at that time step. Note that this is different from *actual* loss; the agent probably didn’t close the position at that time step. Maximum down-draft is therefore ‘worst case’ loss and provides an idea of how risky the strategy is. As a rule of thumb, the absolute maximum down-draft should be less than the profit that the agent ultimately makes.

5.1.1.1.5 Number of subagents used The number of subagents used is only relevant to the hierarchical agent, and tells us how often the subagents are switching control. It is particularly significant if a single subagent is used; this means that the hierarchical agent is using a standard reinforcement learning policy.

5.1.1.2 Convergent policy vs. best policy

In section 4.4.3 we explained how we artificially decay the learning parameter α to ensure that the agent converges on a policy; we call this the *convergent policy*. However, there is no guarantee that the agent converges on the optimal policy, or even the best policy it finds while it is learning. When choosing parameters in the parameter tests, we measure the performance of the convergent policies. This is because the parameter tests are intended to optimise the complete learning algorithm. However, when running experiments, we focus on the performance of the best policies. This is because we are interested in how well the agent can perform with different parameters, so we cannot justify ignoring the best policies simply because the algorithm fails to converge on them. In some experiments, we compare the performance of the convergent and best policies to provide an idea of how convergence could be improved.

5.1.1.3 Performance on training, validation and test sets

In practice, the performance of agents on the training and validation datasets was often found to be better on the validation dataset. This is due to more favourable market conditions in the validation dataset. This indicates that the agent is not over-fitting to the training dataset, and since it is desirable to evaluate the agent on as wide a range of market conditions as possible, we usually give indicators that measure performance on a combined training and validation dataset.

Performance on the test dataset is only measured in an experiment which is used to investigate the performance of various policies on unseen data.

5.1.1.4 Commission costs

The introduction of commission costs introduced a lot of noise into the agents' performance which made it difficult to compare the effects of different agent designs and parameters. For this reason, commission has been ignored outside of a single experiment which directly investigates its effect.

5.1.2 Statistical considerations

Several agents with the same parameters are not guaranteed to have the same best or convergent policy. In fact, the performance of policies found by identical agents is considerably noisy. However, we can show that, on average, certain parameters produce different levels of performance. A number of agents with the same parameters are trained using different random seeds. The policy learnt by each agent can be seen as a sample of the population of all policies that agents with these parameters can learn. Thus, by taking the mean of a given performance indicator for each agent, we can obtain a mean value for agents with the design and parameters. We use standard error to estimate error in the mean. This is defined as

$$\frac{\sigma}{\sqrt{n}}$$

where σ and n are standard deviation and number of samples. Since we use sample sizes of at least 32, we can be 95% confident that the true value of the mean lies within approximately two standard errors of the sample mean [2]. As such, error bars showing two standard errors are included on many of our graph. If the error bars for two populations of agents with different designs and parameters do not overlap, there is a statistically significant difference between the two populations. If the error bars do overlap, we cannot be sure that there is no statistically significant difference. In our discussion we use the word ‘significant’ to mean statistically significant.

5.2 Parameter tests

In this section we identify good values for the agent’s parameters so that we can be sure it is achieving reasonable performance in our experiments. The agent requires so many parameters that an exhaustive search through all combinations would be impractical. As such, the parameters are broken into groups such that the effect of each parameter is roughly dependent on the other parameters in its group and independent of parameters in other groups.

Convergence parameters (Note that α^{decay} is set so that $\alpha \approx \alpha^{target}$ at the end of training.)

α_1 Starting value of alpha

α^{target} Target value of alpha at end of training

α^{H} (Hierarchical agent only) Hierarchical alpha factor

ε Percentage of exploratory actions

Iterations

Iteration parameter Number of iterations when $\alpha = \alpha^{target}$

Learning parameters

γ Discount factor

λ Degree of eligibility

Behaviour parameters

Reward function Profit, Profit Made Good or Composite

Trading parameters

Lower threshold Lower threshold for determining holding state

Upper threshold Upper threshold for determining holding state

These groupings are by no means rigorous. Good parameters were found for the first group, then used to find parameters for the second group, and so forth. Placeholder parameters were used for parameters that had not yet been found; these were ‘guesses’ based on observations in informal experiments. Two sets of parameters were obtained: one for the standard reinforcement learning agent, and the other for a hierarchical agent. We cannot claim that the resulting parameters are perfect, but they should be sufficient for meaningful comparisons.

The initial ‘placeholder’ parameters were set as follows:

- $\gamma = 0.5$
- $\lambda = 0.75$
- Iterations: 64
- Reward Function: Profit Made Good

α_1	**0.5**	**0.3**	**0.1**
	58.82	65.18	58.05
$\alpha^{\mathbf{target}}$	**0.1**	**0.001**	**0.00001**
	41.13	69.58	71.34
ε	**0.5**	**0.3**	**0.1**
	32.10	53.33	96.93

Figure 5.1: Average profit of agents trained with various convergence parameter values

- Upper Threshold: 1
- Lower Threshold: 1

Note that the convergence parameters are absent since they were the first to be chosen. The market observation space was fixed so that the agent had two short and one long Bollinger Band signals. We wanted to test parameters on a large state space, but not so large that the experiments would require too much computation time. When testing hierarchical agents, two identical subagents were used, since this matches the setup used in our experiments.

5.2.1 Convergence tests

A sample of 32 agents were trained for each combination of the following parameters.

- $\alpha_1 \in \{1.0, 0.5, 0, 25\}$
- $\alpha^{target} \in \{0.1, 0.001, 0, 00001\}$
- $\alpha^H \in \{0.125, 0.25, 0.5, 1, 2, 4, 8\}$ (Hierarchical agent only)
- $\varepsilon \in \{0.5, 0.3, 0.1\}$

5.2.1.0.1 Standard agent performance The mean profit for each parameter value was averaged over the values of the other parameters. These averages are shown in figure 5.1. We can pick out good parameter values by looking at these averages.

The initial value of alpha α_1 appears to have little effect, though the value of 0.3 stands out as the best choice. We can see that the target value α^{target} should not be 0.1, but there is little difference between 0.001 and 0.00001. The lowest value for epsilon $\varepsilon = 0.1$ is clearly the best choice. This makes sense intuitively, since agents that explore

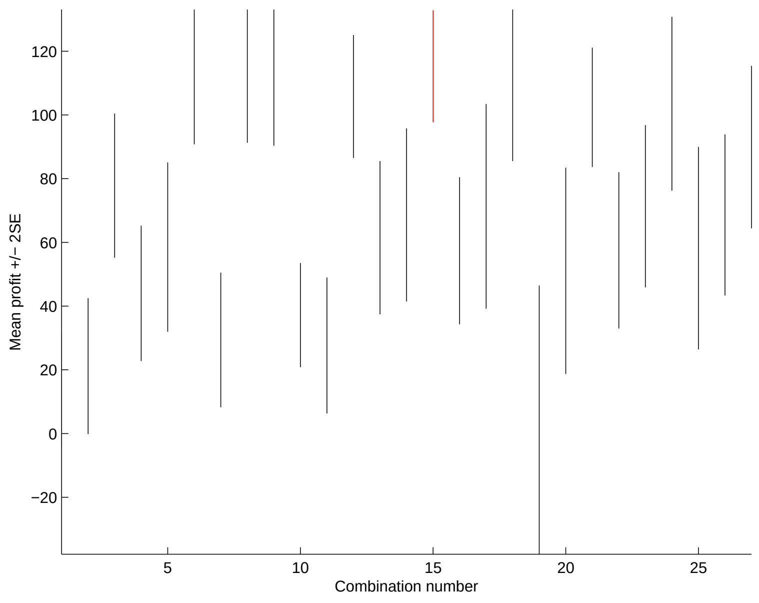


Figure 5.2: Performance of chosen parameter combination (red) compared to other parameter combinations

too much may never hold positions long enough to experience holding states above or below the thresholds. Thus, $\alpha_1 = 0.3$, $\alpha^{target} = 0.001$ and $\varepsilon = 0.1$ would be a good choice of parameters.

However, we have been treating the effects of the parameters as independent, so we should also consider the performance of the parameters in combination. Figure 5.2 shows the profit made by all parameter combinations $\pm$ two standard errors. The selected combination is marked in red. This gives us a general idea of how the performance of our chosen parameter combination compares to other potential combinations. It is acceptable; it overlaps with the other 'top' combinations and not with any of the worst combinations.

5.2.1.0.2 Hierarchical agent performance The same approach was used to pick convergence parameters for the hierarchical agent. We expected that good convergence parameters for the hierarchical agent may differ from the standard agent, since they also have a role in the hierarchical part of the algorithm. The average profit for each parameter value is shown in figure 5.3.

Indeed, the effect of the parameters seems to differ for the hierarchical agent. It appears that 0.5 is the best choice for the initial learning rate α_1 while 0.001 stands out as the best target learning rate α^{target}. The hierarchical learning rate factor does not appear to have much effect. Finally, the low value of epsilon ε, 0.1 is clearly the best

α_1	**0.5**	**0.3**	**0.1**				
	69.07	64.64	52.32				
$\alpha^{\mathbf{target}}$	**0.1**	**0.001**	**0.00001**				
	58.37	67.52	58.14				
$\alpha^{\mathbf{h}}$	**0.125**	**0.25**	**0.5**	**1**	**2**	**4**	**8**
	61.60	59.57	65.39	65.16	62.86	56.86	57.97
ε	**0.5**	**0.3**	**0.1**				
	26.56	56.55	100.92				

Figure 5.3: Average profit of hierarchical agents trained with various convergence parameter values

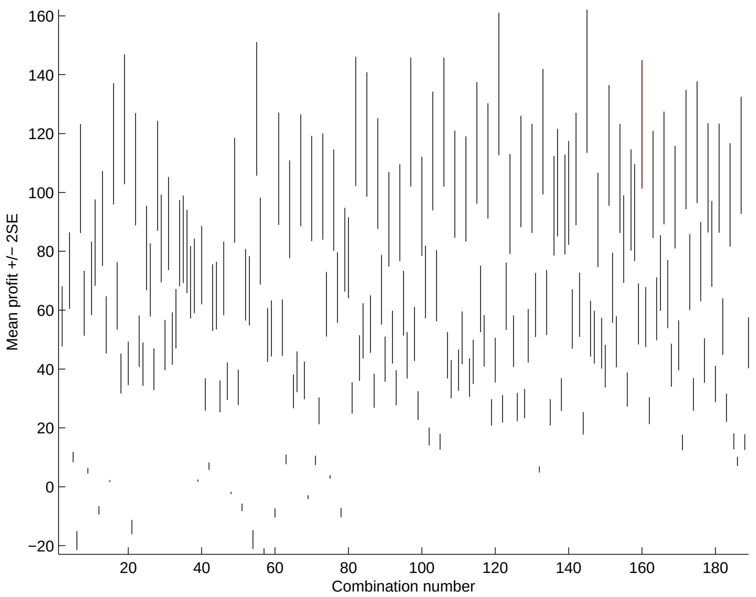


Figure 5.4: Performance of chosen parameter combination (red) compared to other parameter combinations

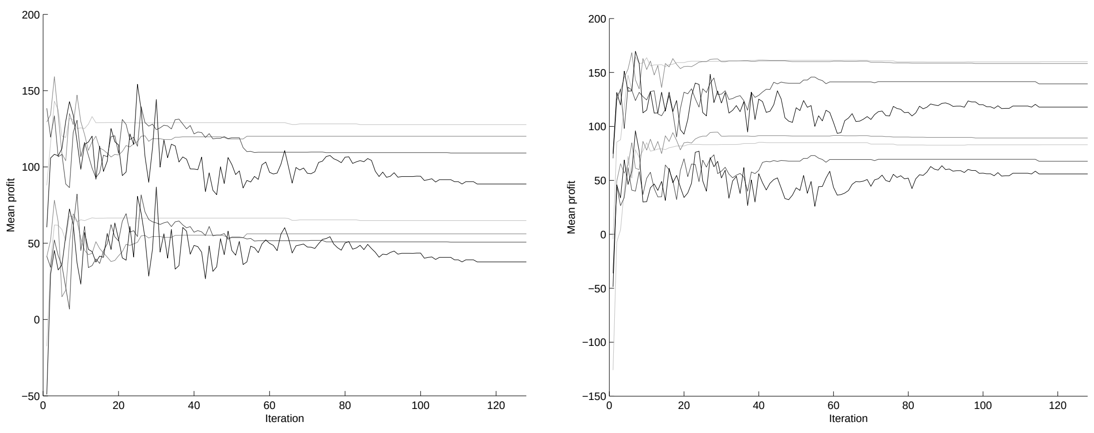


Figure 5.5: Mean profit against iterations for iteration parameter $\in \{64, 128, 256, 512\}$ (lightest: 64, darkest: 512)

choice, as with the standard agent. Based on this, $\alpha_1 = 0.5$, $\alpha^{target} = 0.001$, $\alpha^H = 2$ and $\varepsilon = 0.1$ would be a good choice of parameters. Again, we do a rough comparison again other combinations in figure 5.4. Our chosen combination looks acceptable.

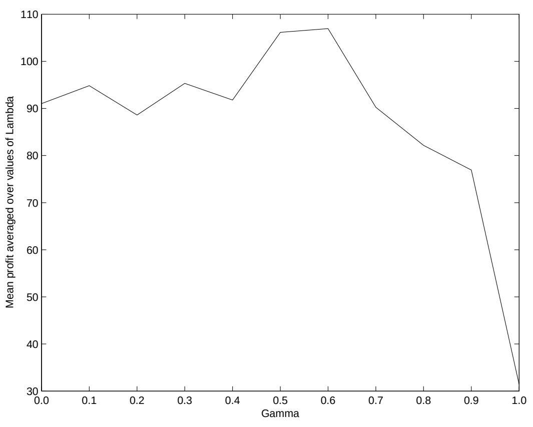
5.2.2 Iteration tests

The iteration parameter controls the rate at which the learning rate decays so that the target value of alpha α^{target} is reached at the final iteration. This is independent of the actual number of iterations used for training. A sample of 32 agents was trained for each of the following values of the iteration parameter.

- Iteration parameter $\in \{64, 128, 256, 512\}$

We used 512 iterations for training regardless of the iteration parameter.

The purpose of this test is threefold. Firstly, it gives us some idea of how the agent converges on a good policy. Secondly, it lets us know if it worth letting the learning rate decay slower so that it reaches α^{target} after more iterations. Finally, it lets us know if there is any benefit to letting the agent continue learning after the target learning rate has been reached. Figure 5.5 plots the mean profit on the training and validation sets against the number of iterations. The lightest lines correspond to an iteration parameter of 64 while the darkest lines correspond to 512. The upper and lower lines respectively show the mean performance $\pm$ two standard errors; the actual means are not plotted. The graphs are truncated to 128 iterations since performance did not change substantially after this point.



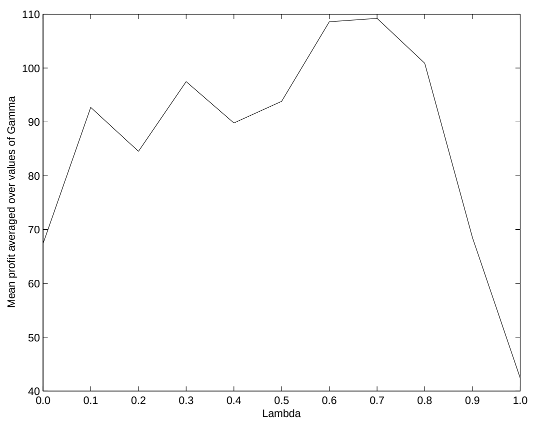


Figure 5.6: Mean profit for values of γ (left) and λ (right)

The noisy performance early in the plots shows that the agents have difficulty learning, which suggests that our environment is not approximating a Markov process. If using hierarchical reinforcement learning helps restore the Markov property, we would expect to see less noise in the right graph, which is the case. However, we have not investigated other factors that may be causing the difference.

In both graphs, the number of iterations seems inversely proportion to the performance at convergence. However, the standard error lines overlap considerably, so we cannot say that there is any significant difference between the performance of agents with different iteration parameters.

In the absence of any significant difference, it is best to stick with an iteration parameter of 64 to minimise computation time. Since the lines for an iteration parameter of 64 reach their convergent behaviour very quickly, we can also keep the number of iterations at 64.

5.2.3 Learning parameter tests

A sample of 32 agents was trained for each combination of the following values of γ and λ.

- $\gamma \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
- $\lambda \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$

5.2.3.0.3 Standard agent performance The mean profit for each value of γ was averaged over the values of λ and plotted in the plot on the left of figure 5.6. The

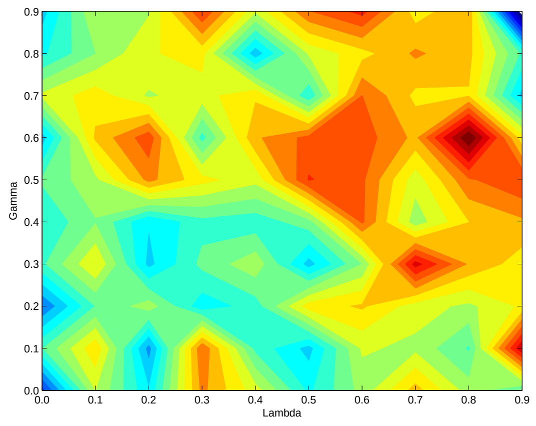


Figure 5.7: Mean performance of γ and λ combinations

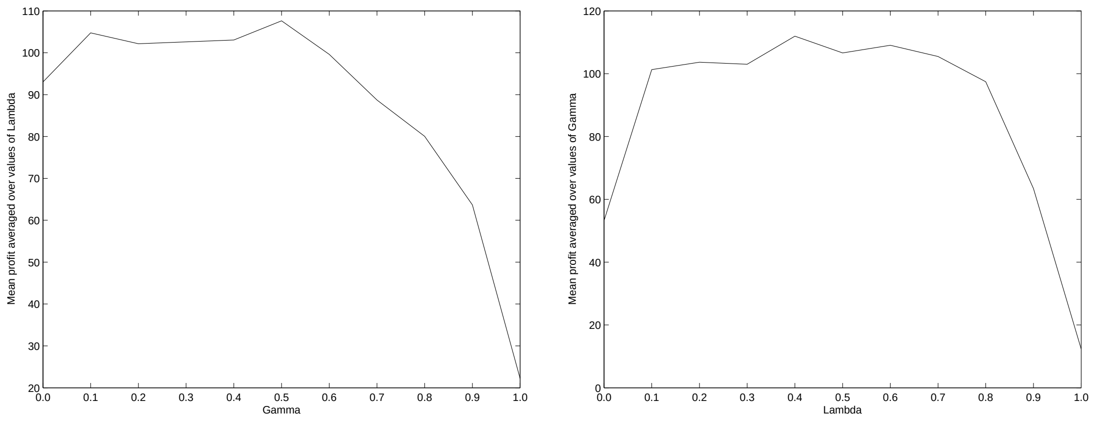


Figure 5.8: Mean performance for values of γ (left) and λ (right)

right plot is the equivalent plot for Lambda. We can get an idea of which values of these parameters are good by looking at these plots. There is a definite ‘peaked’ structure in both plots, with γ peaking at 0.6 and λ peaking at 0.7. This reflects our intuition that both parameters should be high to provide the agent with the ability to ‘lookahead’. Note that performance is poor for $\lambda = 0$, which justifies our decision to use eligibility traces. Based on these plots, $\gamma = 0.6$ and $\lambda = 0.7$ would be a good parameter combination.

Since we are only comparing two parameters, we can produce the contour plot seen in 5.6. Warm and cold colours respectively indicate regions of good and poor performance. If we look where $\gamma = 0.6$ and $\lambda = 0.7$ we see a low point between two regions of better performance. Increasing λ to 0.8 brings us on to one of these better regions. Therefore $\gamma = 0.6$ and $\lambda = 0.8$ are our chosen learning parameters.

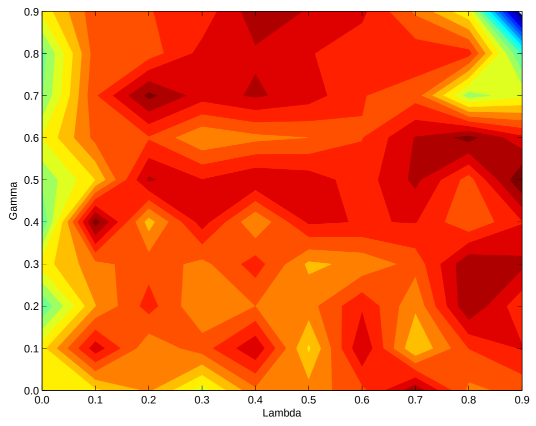


Figure 5.9: Mean performance of γ and λ combinations

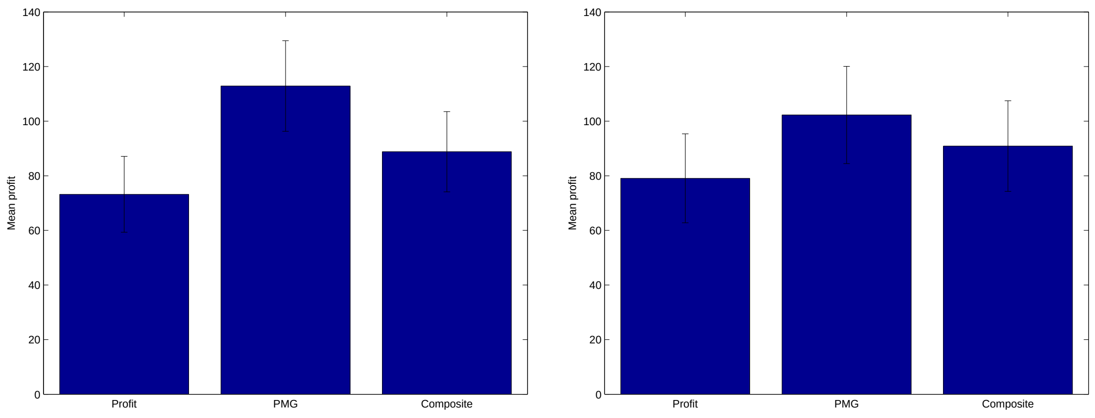


Figure 5.10: Mean performance of reward functions for standard (left) and hierarchical (right) agents

5.2.3.0.4 Hierarchical agent performance Equivalent plots were created for the hierarchical agent. As with the convergence parameters, we expected that good parameter values would differ for the hierarchical agent, since both γ and λ have a role in the hierarchical component of the algorithm. Indeed, the plots in figure 5.8 have less pronounced peaks than in figure 5.6. The peak value of γ is at 0.5, while λ appears to peak somewhere between 0.4 and 0.7.

The contour plot in figure 5.8 allows us to pick a more precise value for λ. Assuming we want γ to be 0.5, then we can choose $\lambda = 0.7$ so that our parameter combination falls in the region of good performance on the right of the plot.

5.2.4 Reward tests

A sample of 128 agents was trained for each reward function described in section 4.3: profit, profit made good (PMG) and composite. The bar graphs in figure 5.10 show the profit made by the standard and hierarchical agents on the training and validation sets. It appears that PMG is the best reward function, and profit is the worst. However, if we take error bars into account, the only thing we can be certain of is that PMG is better than profit for the standard agent. This may be because PMG takes risk into account while profit does not. Gao & Chan [11] noted that the Sharpe ratio is a better reward function than profit for this reason. Alternatively, PMG may be better because it is bounded between -1 and 1, and therefore less noisy than either profit ot the composite function. Whatever the case, we chose to use PMG in our other experiments, since it is the only reward function that we can say has an advantage over another reward function,

5.2.5 Threshold tests

A sample of 32 agents was trained for each combination of the following lower and upper thresholds:

- Lower Threshold $\in \{0, 1/256, 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$
- Upper Threshold $\in \{0, 1/256, 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$

5.2.5.0.5 Standard agent performance The same approach used to select values for γ and λ was used to select thresholds. The mean profit for each lower threshold was averaged over the upper thresholds and plotted in the plot on the left of figure 5.11. The right plot is the equivalent plot for the upper thresholds.

Both plots reveal interesting patterns. In both cases, thresholds of zero give sub-par performance, which justifies using thresholds. Low lower thresholds give bad performance, while moderate thresholds perform best and higher thresholds give average performance. Upper thresholds between 1 and 64 result in the best performance, with the exception of 16, which results in extremely poor performance.

The presence of this ‘valley’ is unexpected. It might be that there exist fundamentally different strategies that agents will only learn with certain upper thresholds. To

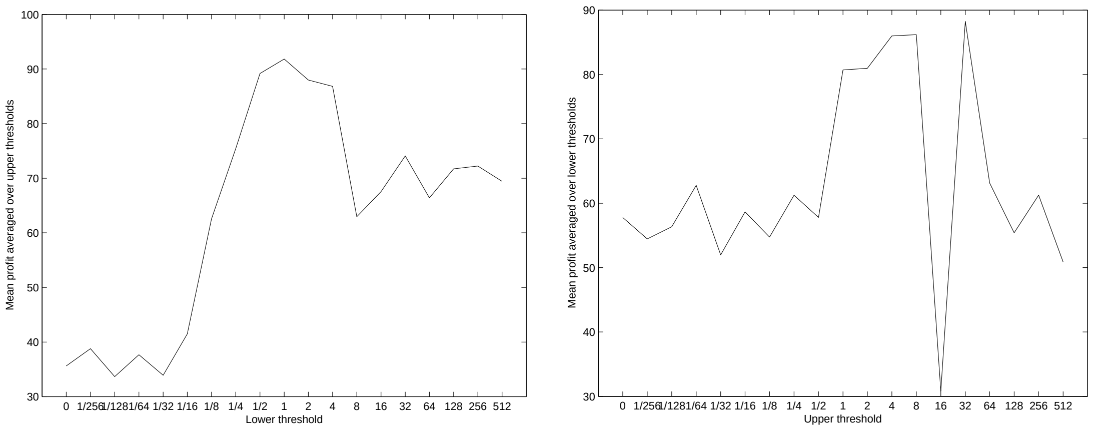


Figure 5.11: Mean profit for lower (left) and upper (right) thresholds

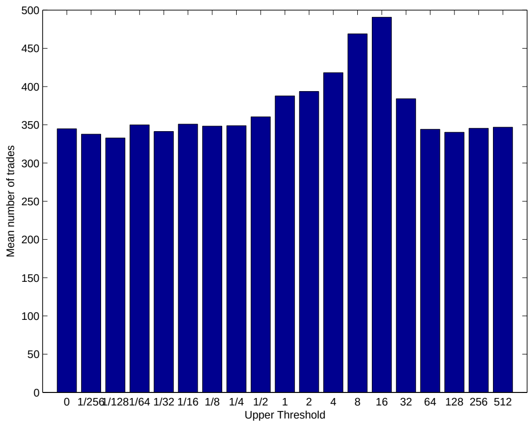


Figure 5.12: Mean number of trades for differ upper thresholds

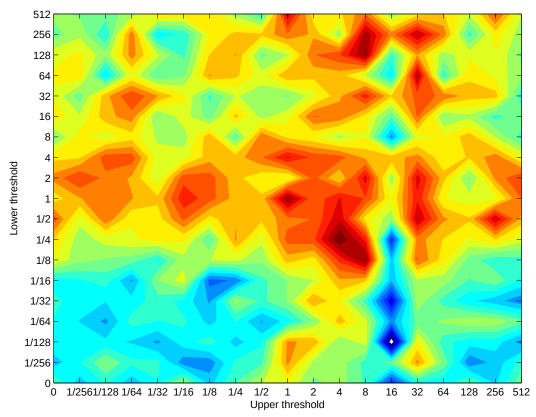


Figure 5.13: Performance of threshold combinations

test this idea, the average number of trades made for each upper threshold was plotted in figure 5.12. Indeed, the number of trades depends on the upper threshold, peaking when it is 16. Though the number of trades is not much higher than for 8, it appears to have a dramatic effect. This also makes 8 an attractive choice for the upper threshold, since strategies which trade often are likely to be more robust.

The contour plot in 5.13 shows how thresholds perform in combination. We have already determined that 8 is a good choice for the upper threshold. However, before we pick a combination, we should consider how the thresholds effect the hierarchical agent. Unlike the other parameters, the thresholds do not effect how the agent learns, but rather what strategy it learns. Thus, there is no reason to choose different thresholds for the standard and hierarchical agent; this will simply make comparing their performance difficult at a later stage. Ideally we want a combination that performs well for both agents.

The average plots for the hierarchical agent are shown in figure 5.14 and are similar to those for the standard agent, though performance with high lower thresholds is notably better. The peak at 8 and the valley at 16 in the performance of the upper thresholds is still present. We can therefore justify using an upper threshold of 8 for both values.

Finally, let us consider the contour plot for the hierarchical agent seen in figure 5.15. Given that we want to use an upper threshold of 8, a lower threshold of 2 would be a good choice for both agents.

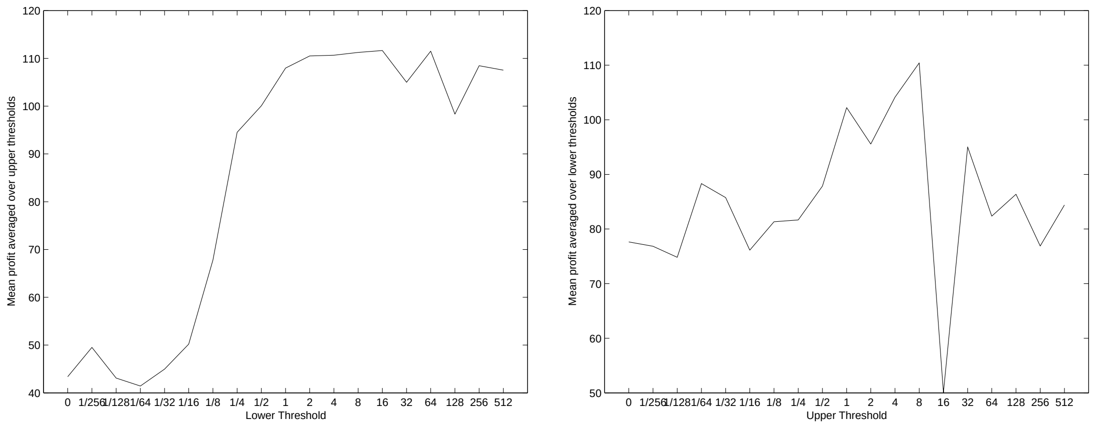


Figure 5.14: Mean performance for values of upper threshold (left) and lower threshold (right)

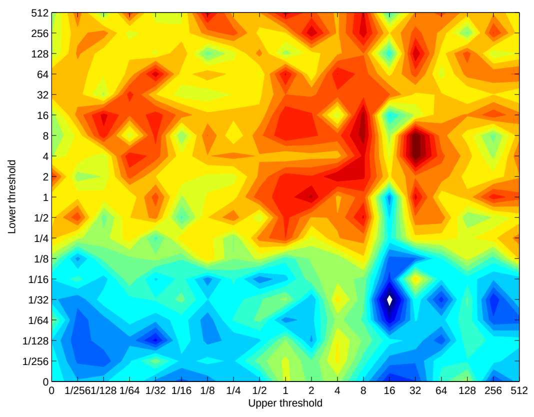


Figure 5.15: Performance of threshold combinations

5.3 Experiments

5.3.1 Hierarchical experiment

This experiment was designed to test our hypothesis by comparing the performance of the standard agent to the hierarchical agent. We began by investigating which goal partition scheme results in the best performance but failed to find any significant difference between the schemes. We then compared standard and hierarchical agents using a wide range of indicator and signal combinations. A significant difference in performance was noted in the trivial case with no indicators. Moreover, in this case, we showed that the hierarchical agent can find a strategy that outperforms any strategy the standard agent can find. However, differences in performance were not found outside of the trivial case.

Only two subagents were used by our hierarchical agent. We assumed that if no difference in performance can be found using two subagents, then there is unlikely to be any difference when more subagents are used. Other parameters were fixed at the values found for the hierarchical agent in the parameter tests.

5.3.1.0.6 Goal partition schemes As explained in section 4.4.3, the hierarchical agent chooses groups of observations as goals instead of single observations. There are three ways in which the observation space can be partitioned into groups: based on holding states, short signals or long signals. The best partition scheme must be determined before the hierarchical agent can be compared to the standard agent.

To do this, a sample of 128 agents were trained for each partition scheme and fourteen different indicator setups. Of these indicator setups, six had at least one short and long signal, so any goal partition scheme could be used. The remaining eight did not have a long signal, so only the holding state or short signal partition schemes were applicable. When discussing these experiments, the indicator setups have been grouped accordingly:

Long Signal Group = { B11,R11,K11,B21,R21,K21 }

No Long Signal Group = { -,B10,R10,K10,BR10,BK10,RK10,BRK10 }

The letters indicate which technical analysis indicators were present while the numbers respectively denote the number of short and long signals. Note that only one technical analysis indicator is used for all setups with a long indicator. This is because using too many indicators and signals results in an infeasible number of possible

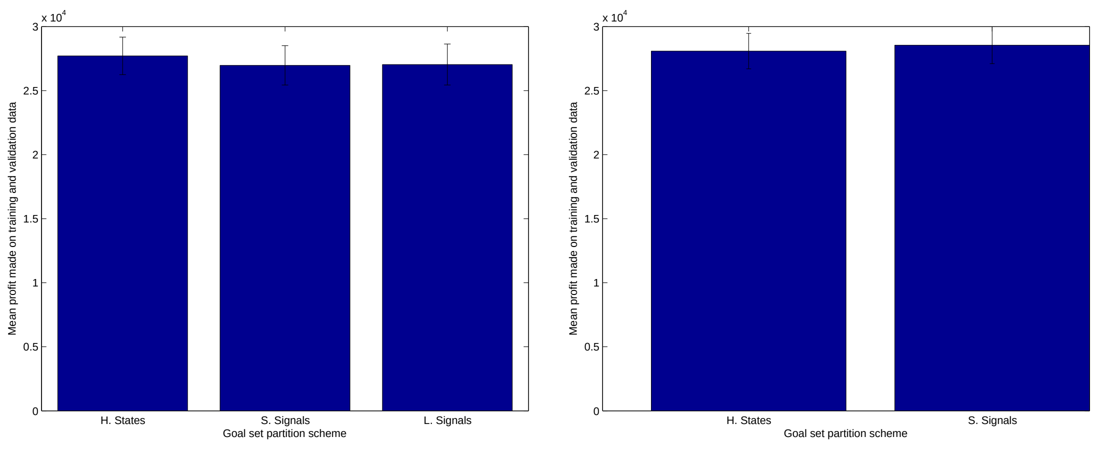


Figure 5.16: Mean profit made by each goal partition scheme

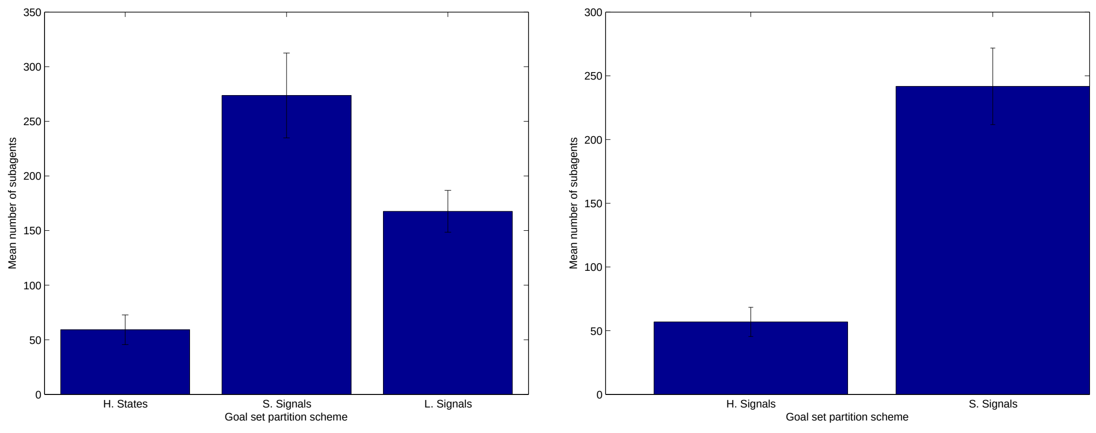


Figure 5.17: Mean subagents used for each goal partition scheme

observations. Consider an agent using all three indicators to generate two short and one long signal. This gives $4^3 \cdot 4^3 \cdot 3^3 \cdot 9 \cdot 2 = 1990656$ possible observations, orders of magnitude higher than the number of points in the dataset. The fourteen indicator setups used here are the only ones with a feasible number of observations.

Figure 5.16 shows the mean profit on the training and validation set for each partition scheme, with the Long Signal Group on the left and No Long Signal Group on the right. The values were obtained by averaging over the results of all the indicator setups in each group. It is apparent that there is no significance difference between the different partition schemes. Plotting short/long profit, number of trades and maximum down-draft yield similar uniform plots. The only indicator which reveals any difference is the number of subagents used by the agents, shown in figure 5.17.

It is likely that agents using the holding space partition use fewer subagents on average because many of them are picking the initial holding state as a goal. Since this initial state is only visited once per market segment, agents using this goal will never change subagents. Similarly, some agents using the long signal partition might be picking unusual signals as goals. Conversely, there are no unusual short signals, so agents with the short signal partition cannot choose unusual goals and change subagents often no matter which goals they choose. However, the number of subagents used has no effect on the resulting policies, as seen in 5.16.

Since no significant difference could be found between the partition schemes, we choose to use the holding state partition in subsequent hierarchical experiments, since it can be used for any indicator setup.

5.3.1.0.7 Comparison between standard and hierarchical agent A sample of 128 standard and hierarchical agents were trained using the same fourteen indicator setups used above. In fact, the relevant results from above were simply reused for the comparison. The indicator groupings used above are retained for the purpose of plotting graphs.

Since the different goalset partitions had little effect on performance, this suggests that the hierarchical structure has little effect on performance in general. Thus, we did not expect to see much difference between the standard and hierarchical agents. However, some initial informal experiments found that the hierarchical agent appeared to be performing better for indicator setup KD21. It was speculated that this might be due to the different learning and convergence parameters used in the standard and hierarchical agent. Indeed. when the standard agent was retrained using the learning parameters from the hierarchical agent, the difference disappeared. In order to make a fair comparison, the hierarchical parameters were used for both agents in these experiments.

Figure 5.18 compares the mean profit made by the standard and hierarchical agents. Each pairing of bars corresponds to an indicator setup, with the blue and red bars respectively corresponding to the standard and hierarchical agent. There is no significant difference for any indicator setup other than the trivial setup with no indicators.

In fact, for an agent with no indicators, we can show that the hierarchical agent is capable of finding a strategy that makes more profit on the training and validation sets than any strategy the standard agent can find. This is because there are only $2^9 = 512$ possible strategies for a standard agent with no indicators, so it is feasible to iterate through all of them and find the highest profit made. This turns out to be 30550, from 64 trades. However, in our experiments, the hierarchical agent found a strategy that

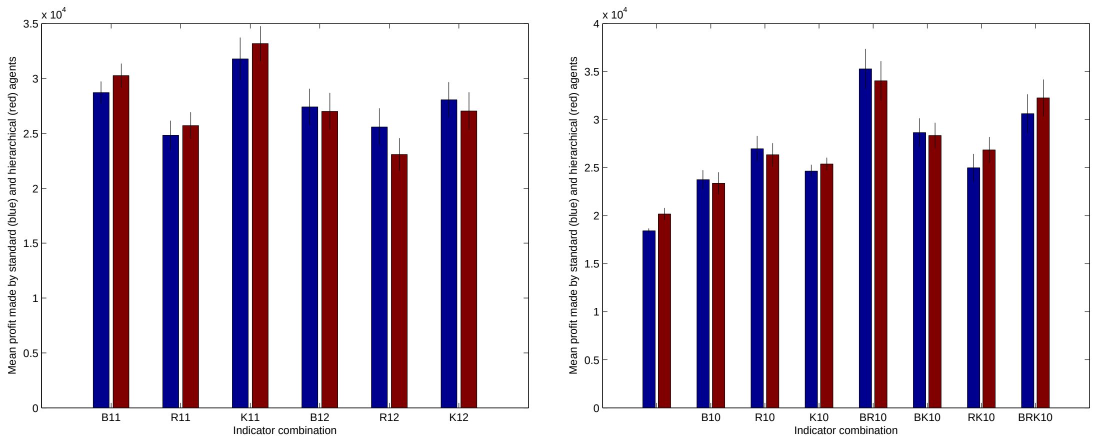


Figure 5.18: Mean profit made by standard and hierarchical agent

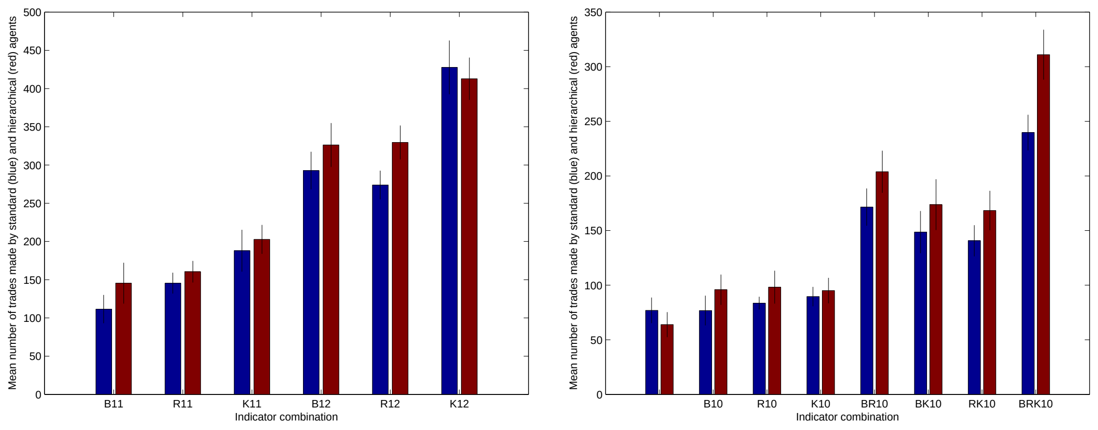


Figure 5.19: Mean number of trades made by standard and hierarchical agent

makes 32510 from 335 trades. Moreover, there are likely to be better strategies, since we did not iterate through all the hierarchical strategies (there are over 42 million). This is not a particularly practical observation, but proves that the hierarchical structure can provide an advantage in at least one trading environment.

Looking at the number of trades made by the agents suggests there could be a small difference between the standard and hierarchical agent. Figure 5.19 has the same format as figure 5.18, but shows the average number of trades made. Here we see that the hierarchical agents often make more trades. However, a paired t-test gives a *p*-value of of 0.1895, which is not considered significant. However, a significant difference is seen if setups R12 and BRK10 are considered in isolation. It might be that the hierarchical agent finds more robust strategies for these combinations. This is tested later by looking at how these strategies perform on unseen data.

Investigating the short profit and maximum down-draft does not illuminate our

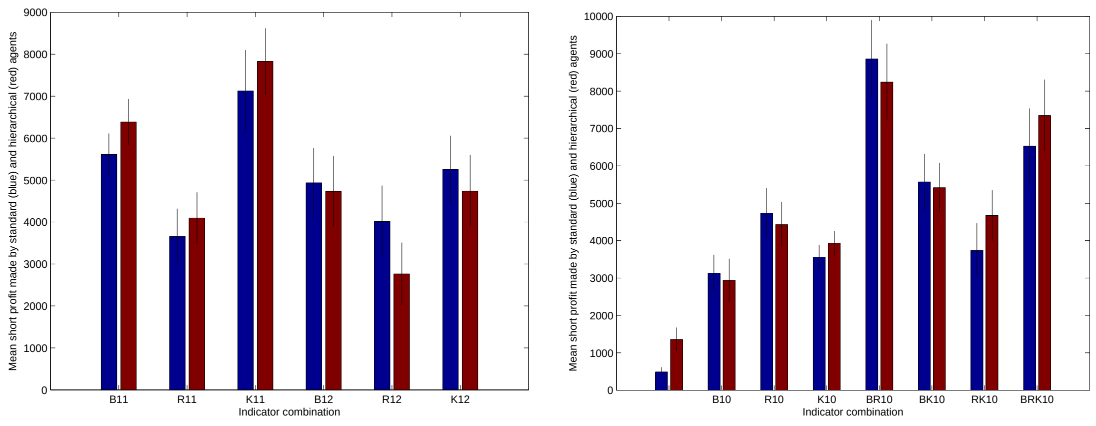


Figure 5.20: Mean short profit made by standard and hierarchical agent

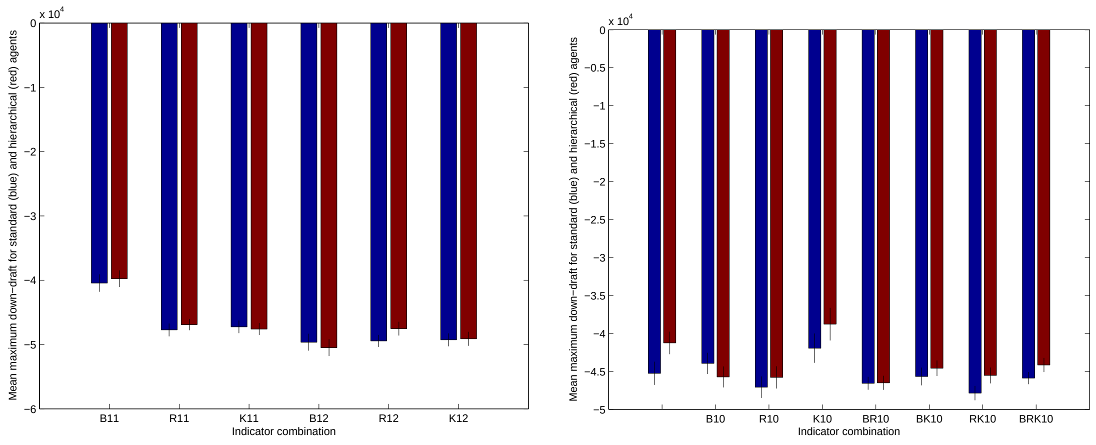


Figure 5.21: Mean down-draft for standard and hierarchical agent

results. The mean short profit is shown in figure 5.20 and simply appears to mirror what was seen in figure 5.18. The mean maximum down-draft is shown in figure 5.21 and does not reveal any consistent patterns. For some indicator setups, the standard agent has significantly less mean down-draft, whereas the reverse is true in other setups. The difference appears to depend on the particular setup, and so no general statements can be made.

The poor performance of the hierarchical agent is assumed to be due to noise in reward signals used to update the HQ-values. It may be that the hierarchical 'memory' does simply not help agents disambiguate market regime any more than their direct observations. However, we consider this unlikely, since we found an example of a trading problem where the hierarchical agent does have an advantage. It follows that our design is preventing the hierarchical agent from obtaining an advantage. Informal observations found that reward recieved for choosing the same goal set and next sub-agent varied wildly. As a consequence, the hierarchical agents simply seemed to be converging on random goal sets and next subagents. If the algorithm was altered so that these reward signals are more consistent, the hierarchical agent might outperform the standard agent. It stands that some choices of goal set and next subagent must be better, so it should be possible to design a reward function that reflects this.

5.3.2 Indicator combination experiment

Our agent is capable of using any combination of the three technical analysis indicators described in section 4.1.2: Bollinger Bands, RSI or the Fast Stochastic Oscillator (KD). In this experiment we investigated which of these combinations are of value to the agent. This experiment does not relate directly to the hypothesis, but highlights some interesting behaviours that emerged from the standard reinforcement learning agent. The hierarchical agent was ignored for this experiment. We found that agents with multiple indicators could make more profit than agents with one or no indicators. Dempster & Jones [7] made a similar observation. We also found that agents with multiple indicators found more robust strategies than agents with one or no indicators; these strategies made significantly more trades and made a profit on a wider range of market conditions. This is a novel result, and provides an example of an agent learning strategies appropriate to the information it is given.

For each combination of the three indicators, we trained 128 different agents. For each technical analysis indicator present, a single short signal was generated. All other

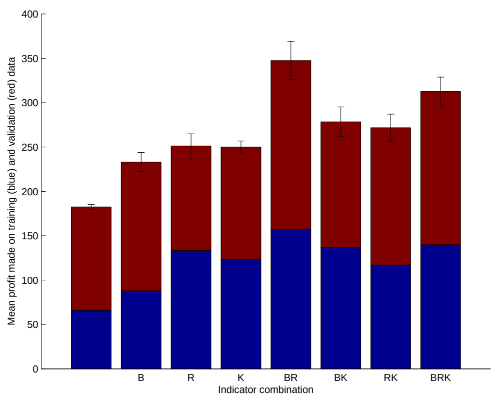


Figure 5.22: Mean profit made on training (blue) and validation (red) sets

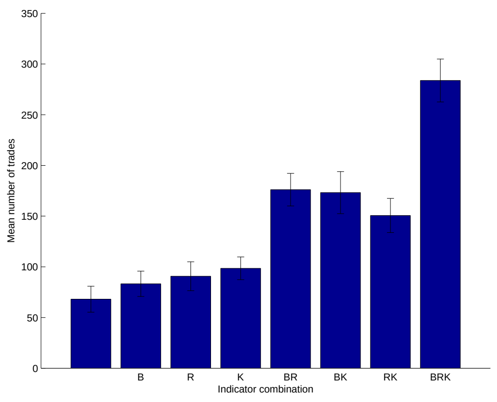


Figure 5.23: Mean number of trades

parameters were fixed at the values found during parameter testing.

A single short signal was used to keep the number of possible market observations low. If we limit the agent to a single short signal, then the agent with all three indicators only has $4 \cdot 4 \cdot 3 \cdot 9 \cdot 2 = 648$ indicators, which is a much more manageable number. Limiting the agent in this way therefore ensures that we are seeing the effects of giving the agent more information, rather than the effects of 'overwhelming' the agent. The value of combinations of short and long signals is investigated in another experiment, where the number of possible states is reduced by limiting the number of technical analysis indicators.

The mean profit for each combination is shown in figure 5.22. The letters B, R and K respectively refer to Bollinger Bands, RSI and the Fast Stochastic Oscillator (KD). Note that the agents made more profit on the validation set, as opposed to the training set, which would usually be the case. This is because the validation set contains more rising market segments, and so is more favourable to our agent design regardless of which indicators are used. It also means that the distinction between the training and validation set is meaningless here, so other indicators use the combined performance on the training and validation sets.

It appears that agents using any two indicators in combination performed better than agents using any single indicator, although this is only significant for the agents using Bollinger Bands and RSI or the agent using all three indicators. Figure 5.23 shows the average number of trades made by agents and reveals a significant difference between agents using different numbers of indicators.

A clear relationship is seen between the number of indicators and the number of trades made; the agents using one or no indicators made between 50 to 100 trades

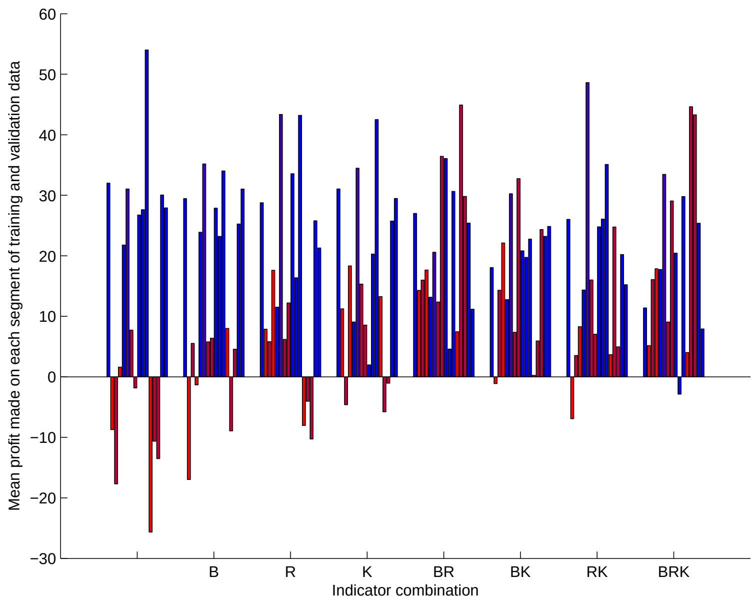


Figure 5.24: Mean profit made on each training and validation market segment

on average, while the agents using two made between 150 and 200, and the agents using all three indicators make almost 300. Note that this is the number of trades made over all the training and validation market segments, and since the agents are forced to close their position at the end of a segment, they must trade at least 16 times. Thus, the agents with no indicators made on average $68.08-16=52.08$ voluntary trades, and $52.08/16=3.26$ voluntary trades per market segment. By comparison, the agents with all three indicators made on average $283.75-16=267.75$ voluntary trades; 16.73 per market segment. The 'logic' of strategies found by agents using more indicators was therefore checked more often, and so we expect that these strategies are more likely to work on unseen data. This indicates some level of robustness.

Figure 5.24 provides a clearer picture of how robust agents' strategies are. Each cluster of bars shows performance on the 16 market segments from the training and validation datasets. The colours of the bars match the colours in figure 3.1; blue and red respectively denote rising and falling market segments. The agents with no indicators clearly struggle to make profit on falling markets. The agents with one indicator do a little better, but still make the body of their profit on rising markets. Conversely, the agents with two or more indicators do not appear to favour falling or rising markets.

Figures 5.25 and 5.26 reveal weaknesses common to all agents. Figure 5.25 compares profit made on long and short positions. Clearly, all agents make less profit from short positions. This is not entirely unexpected, as discussed in section 5.1.1.1. How-

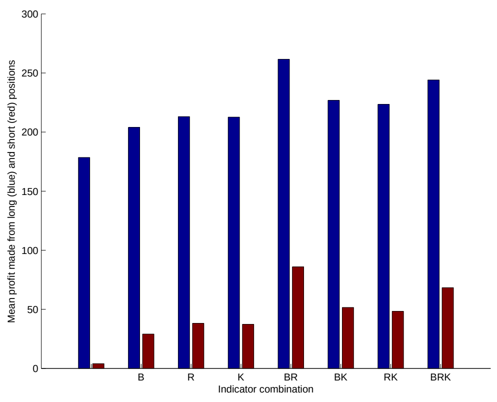


Figure 5.25: Mean profit made on long (blue) and short (red) positions

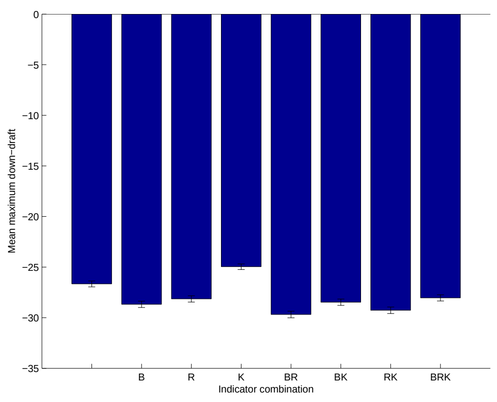


Figure 5.26: Mean maximum down-draft

ever, note that the agents using Bollinger Bands and RSI or all three indicators made almost twice as much profit from short positions as agents using any one indicator. By contrast, the profit made on long positions increases very little. This suggests that the primary advantage of having more indicators, or information, is the ability to make money on short positions. Figure 5.26 shows mean maximum down-draft which is fairly large, but does not seem to correlate to the number of indicators used. However, the mean maximum down-draftft values reflect worse on the agents with one or no indicators since they ultimately make less profit.

Finally, figure 5.27 compares the mean profit made by convergent strategies to that made by the best strategies, which was used to produce the other figures. The convergent policies are significantly worse than the best policies, suggesting that the convergence of the algorithm could be improved considerably. Moreover, the difference in performance seems worse for the agents using more indicators. For instance, the convergent strategies of agents using just Bollinger Bands or RSI make more profit than the strategies of agents using both.

5.3.3 Signal combination experiment

Our agent is capable of using six combinations of short and long signals as detailed in section 4.1.2: S0L0, S1L0, S2L0, S0L1, S1L1, S2L1. In this experiment we investigate which of these six combinations are of value to the agent. Again, this experiment does not relate directly to the hypothesis and ignores the hierarchical agent. We found that giving agents long signals allowed them to make better trades and increase their profits, but did not seem to alter their fundamental strategies. Giving agents an additional

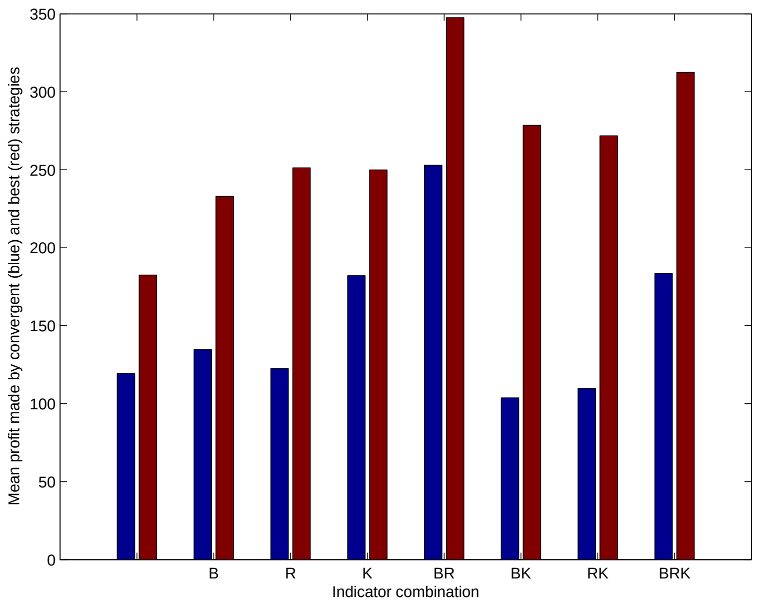


Figure 5.27: Mean profit made by convergent (blue) and best (red) strategies

short signal dramatically changed the strategies they found and increased the number of trades they made. These are both novel results, and reinforces the idea that the agent learns strategies appropriate to the information it is given.

For each combination of short and long signals, we trained 128 different agents. A single technical analysis indicator was used to generate the signals: the Fast Stochastic Oscillator (KD). All other parameters were fixed at the values found during parameter testing.

A single technical analysis indicator is used for the same reason that a single short signal was used in the indicator combination experiment: to limit the size of the observation space. We chose KD over BB or RSI simply because it has three discreet states instead of four, which further reduces the observation space. Otherwise, the indicator combination provided little reason for choosing one indicator over another. Using just KD, we have a maximum of $3 \cdot 3 \cdot 3 \cdot 9 \cdot 2 = 486$ states when two short and one long signal are used.

Figure 5.28 shows the mean profit made by agents on the training and validation sets. The numbers next to S and L beneath each bar show how many short and long signals were used. Recall that we expected that increasing either the number of short or long term signals would improve performance. Figure 5.28 reveals that this is true for the long signals. The three rightmost bars correspond to agents trained with the long signal. The first two of these are signficicantly higher than the corresponding bars

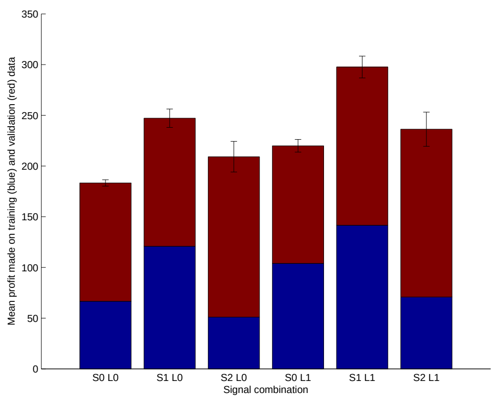


Figure 5.28: Mean profit made on training (blue) and validation (red) sets

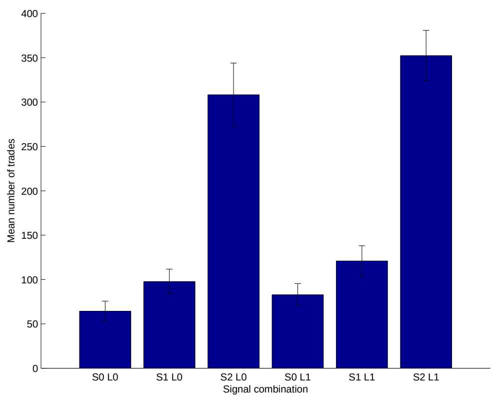


Figure 5.29: Mean number of trades

on the less. The last bar is not, but these agents were using more signals than any other, and so we may be seeing the effects of ‘overwhelming’. The two bars for the agents using two short signals are lower than the corresponding bars for agents using a single short signal, which does not meet our expectations.

However, it is important to take the number of trades made by agents into account when evaluating performance. This is shown in figure 5.29. Here was see a similar effect to what was seen in the indicator combination experiment; increasing the amount of information causes the agents to make more trades. However, the effect of increasing the number of short signals from one to two is far more dramatic than the effect of adding the long signal. In fact, the effect of doing the latter is insignificant.

The results support our reasons for including the additional signals. We reasoned that adding the second short term indicator would allow agents to learn a number of technical analysis rules that involve changing signals. This should allow then to develop more complex strategies, which seems to be the case here; the number of trades increase dramatically when the second signal is added. On the contrary, we reasoned that adding the long term indicator would enable the agent to partially uncover the underlying regime. While this should not fundamentally change the agents’ strategies, it should give them a better idea of how the market transitions between observations. This allows better trades to be made, which results in increased profit, as seen in figure 5.28.

Figure 5.30 shows how the agents perform on each segment of the market, and provides further support for our view. Consider the performance of the agents with one short signal in the second and fifth clusters. Notice how the fundamental behaviour of the agent changes when another short signal is added (third and sixth clusters). The

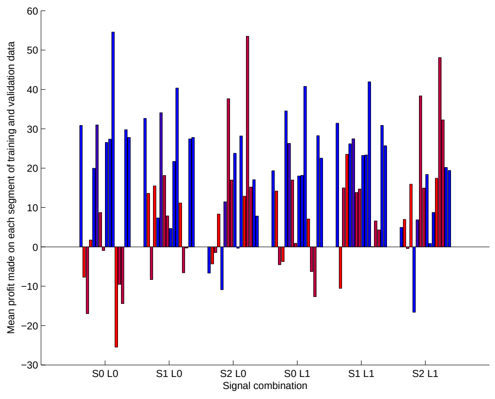


Figure 5.30: Mean profit made on each training and validation market segment

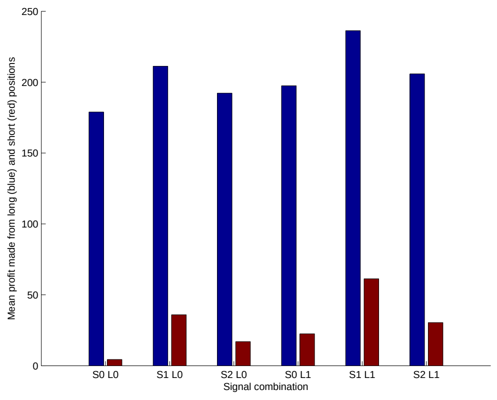


Figure 5.31: Mean profit made from long (blue) and short (red) positions

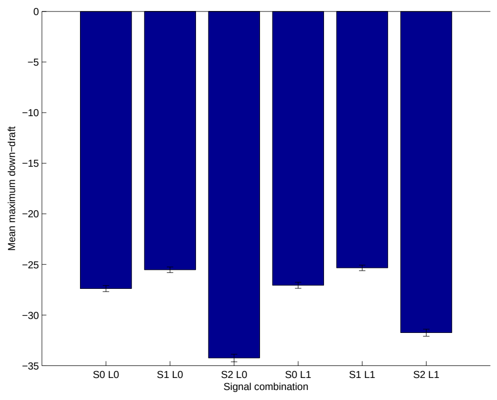


Figure 5.32: Mean maximum down-draft

longest bars are now red instead of blue. Conversely, the difference between the two clusters with one short signal, which shows the effect of using the long signal, is less pronounced. There are more taller bars, but the overall appearance of the cluster is similar.

We might take figure 5.30 to mean that the agents with two short signals are making more profit from short positions, since their clusters contain many tall red bars. Figure 5.31 shows that this is not the case; all agents are still make significantly less profit from short positions. Figure 5.30 therefore reveals that the agents with two short positions are good at making money off long positions in falling markets. This is supported by noting that the longer red bars are soft red which are the gently decreasing segments with lots of ups and downs. This suggests a fairly complex strategy with good timing. As with the indicator combination experiment, figure 5.31 shows that increasing information has a greater effect on short profits. However, figure 5.32 has

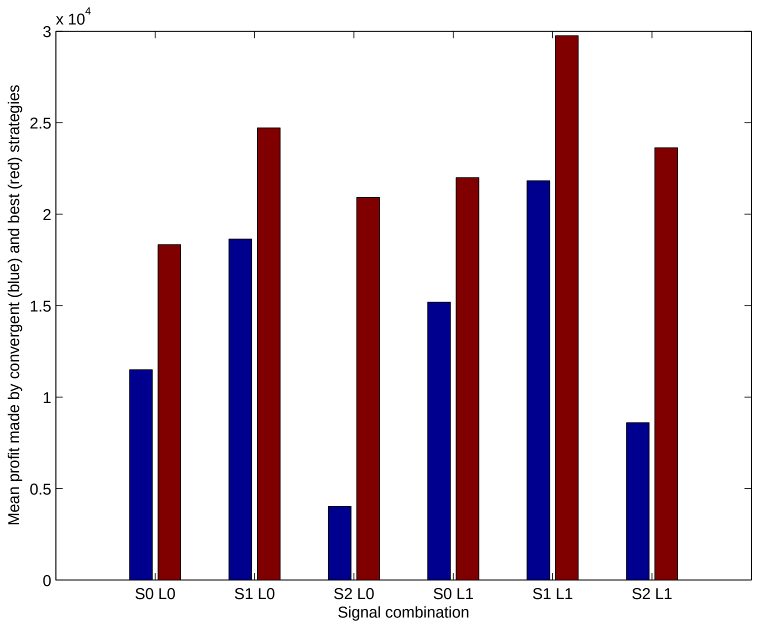


Figure 5.33: Mean profit made by convergent (blue) and best (red) strategies

at least one significant feature not seen before. The agents with two short signals have a large maximum down-draft. This suggests that the increased number of trades made by these agents results in strategies with higher risk.

Finally, let us consider how the convergent policies compare to the best policies we have been considering. Figure 5.33 again confirms that the convergent policies are significantly worse than the best policies, particularly for the agents with two short signals. In the previous experiment we speculated that the difference in performance increased with the amount of information given to the agent. This suggests that the difference may be related to the number of trades the agents are making, rather than the amount of information.

5.3.4 Commission experiment

In this experiment, we investigate the effects of charging the agent a realistic commission rate of 0.25% every time a position is opened or closed. Comission rates were ignored in previous experiments since they obscure some of the interesting results noted above. We found that comission rates had a dramatic negative effect on performance, such that agents could not consitently outperform the 'buy and hold' strategy. This problem is common in novel algorithmic traders. However, we also found that agents made fewer trades when comission rates were introduced, providing a further example

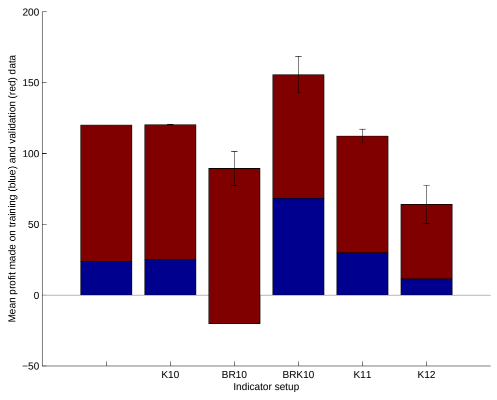


Figure 5.34: Mean profit made on training (blue) and validation (red) sets

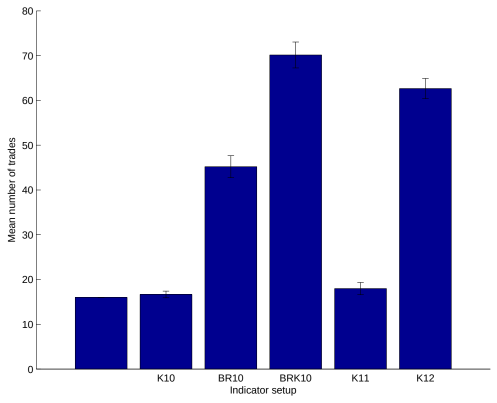


Figure 5.35: Mean number of trades

of the agent adapting to its situation.

The broker simulator was set to charge commission, and performance with six different indicator setups was investigated: { -, K10, BR10, BRK10, K11, K12 }. Each of these setups is intended to represent a number of setups that were found to produce similar performance in previous experiments.

Figure 5.34 shows the mean profit made by agents on the training and validation sets. Observe that the performance of agents trained with no indicators and just the Fast Stochastic Oscillator (KD) are similar. In fact, these agents are almost exclusively finding the buy and hold strategy. Thus, the performance level set by these agents can be taken as a target level for the other agents. Unforunately, this target is only surpassed by the agent with all three indicators. Figure 5.35 is somewhat more encouraging. The number of trades made by these agents is significantly less than the equivalent agents without commission (see sections 5.3.2 and 5.3.3). For instance, without commission, the agents with all three technical analysis indicators made on average 283.75 trades; here they make just over 70. This indicates that the agents are adopting sensible strategies and trading less to try and reduce commission costs. Unfortunately, this is not enough to offset the loss.

Figure 5.36 shows the mean profit made by agents on each market segment from the training and validation sets. Here we see that the agents with long and short KD indicators are failing to make money from declining markets. This stands in contrast to the equivalent agents in the indicator scope experiment which were able to average a profit on all but one declining market segment. The profiles for the other agents show a similar drop in performance.

Figure 5.37 suggests why this might be. None of the agents are able to average a

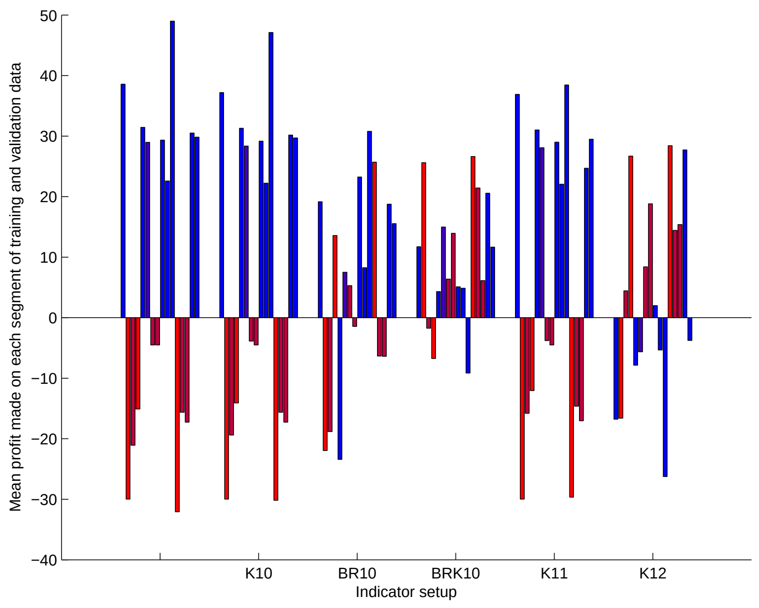


Figure 5.36: Mean profit made on each training and validation market segment

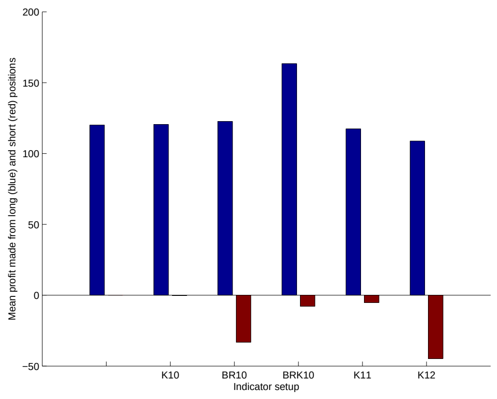


Figure 5.37: Mean profit made from long (blue) and short (red) positions

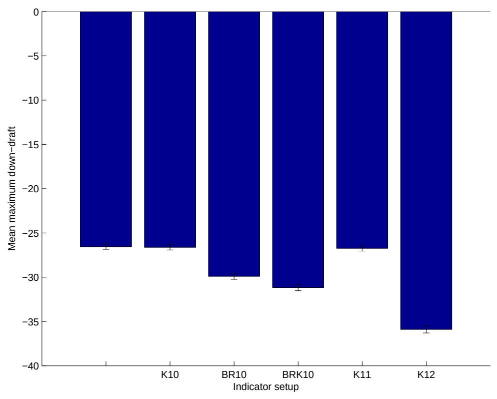


Figure 5.38: Mean maximum down-draft

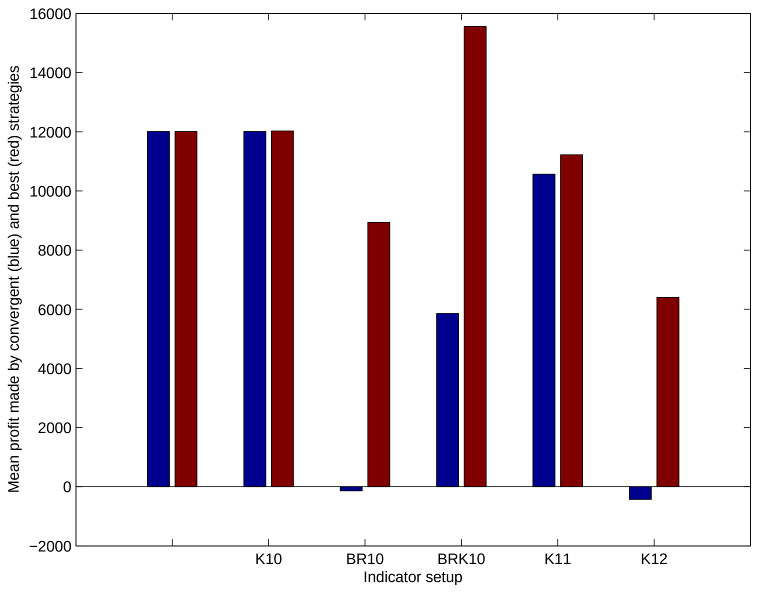


Figure 5.39: Mean profit made by convergent (blue) and best (red) strategies

profit from short positions. As explained in paragraph 5.1.1.1.3, making profit from short positions requires good timing which requires reliable market signals. Therefore it may be that the indicators provided to the agents are informative enough for profit to be made when commission is ignored, but not when it is included. Here it would be useful to look at the agents' performance with multiple indicators at multiple time scales. Unfortunately, this is infeasible with our current agent design, due to the blow-up of the observation space. An agent that could generalise the observation space might be able to overcome this problem. Figure 5.38 shows that maximum down-drafts are not significantly worse that those seen in previous experiments, though they are worse when considered in relation to the ultimate profit.

Finally, figure 5.39 shows how the performance of the convergent policies compares to the performance of the best policies. For three of the indicator setups, the convergent policies perform similarly to the best policies. However, this is not particularly significant, since these agents are mainly finding the trivial 'buy and hold' strategy. For the remaining three indicators, the convergent policies are significantly worse, as in the previous experiments, Moreover, some of indicators are causing the agents to converge on policies that make a loss. This reaffirms our assertion that the learning algorithm could be improved significantly. As noted in section 5.3.3, the difference seems to be greater when the agents are making more trades.

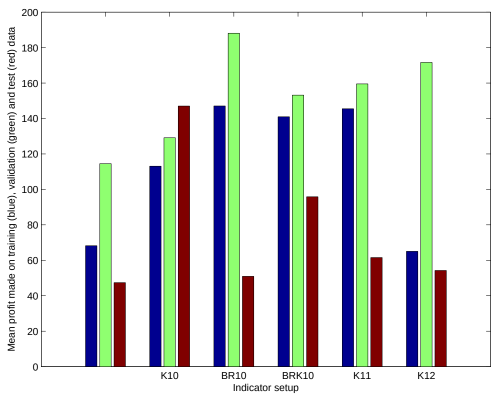


Figure 5.40: Mean profit made on training (blue), validation (green) and test (red) sets

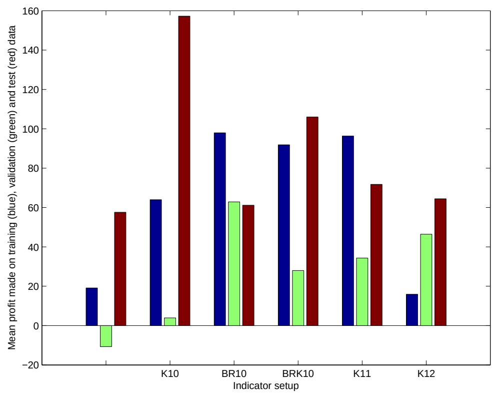


Figure 5.41: Adjusted mean profit made on training (blue), validation (green) and test (red) sets

5.3.5 Unseen data experiment

In this experiment we tested the agent's performance on unseen data. In our previous tests and experiments, the agents were trained on the training dataset and then tested on both the training and validation datasets. To truly evaluate the performance of our agent, we must investigate its performance on unseen data. While the validation dataset is unseen to some degree, it was used for choosing parameters in the parameter test. The test dataset, however, has not been used for anything and so is genuinely unseen data. We found that the agents without a commission charge could make a profit on unseen data and agents trained with more indicators gave a more balanced performance across the training, validation and test set. Agents with a commission cost performed poorly across all datasets, and failed to outperform the buy and hold strategy on any unseen data. Finally, we found no difference between the performance of standard and hiearchical agents on unseen data.

A sample of 128 agents was trained with the same six indicator setups used in the commisison experiment: $\{$ -, K10, BR10, BRK10, K11, K12 $\}$. We then recorded the mean profit made by agents on the training, validation and test dataset. Other indicators were ignored in this experiment, since the number of trades was observed to remain consistent over different datasets, and other indicators were found to be proportional to profit.

Figure 5.40 shows the mean profit on the training, validation and test datasets. The agents are making less profit on the test datasets which suggests that they are performing worse on unseen data. However, we should acknowledge that our agent

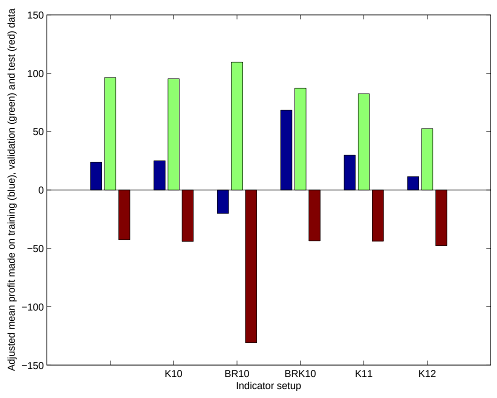


Figure 5.42: Mean profit made training (blue), validation (green) and test (red) sets

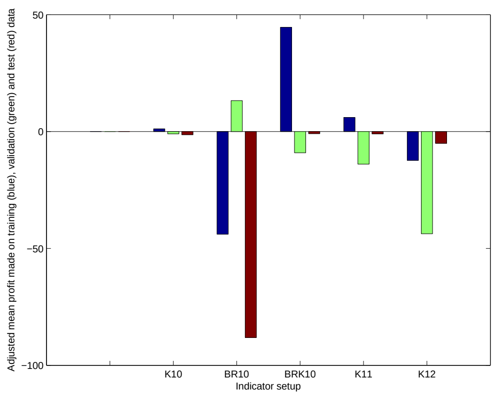


Figure 5.43: Adjusted mean profit made on training (blue), validation (green) and test (red) sets

design tends to do better in increasing markets, regardless of strategy. This is accounted for by subtracting the profit made by the buy and hold strategy from our agents' mean profit. Since the buy and hold strategy only makes a profit on increasing markets, it indicates how much the market increased in each dataset. The buy and hold strategy respectively makes 49.1, 125.2 and -10.2 profit on the training, validation and test datasets.

Figure 5.41 shows the adjusted profit and reveals a different picture. Note that the agents with the right four indicator setups are all able to do substaintially better than the buy and hold strategy on all three datasets. In previous experiments we speculated that agents trained with more information find more robust strategies. That appears to be the case here. The agents with the left two indicator setups have less information, and appear to have difficulty outperforming buy and hold on the validaiton set.

Figures 5.42 and 5.43 are the equivalent of figures 5.40 and 5.41 for agents that were charged a 0.25% commission cost for trading. The consistently poor performance on the test dataset in figure 5.42 is again a consequence of more favourable conditions in the validation set. As above, we compensate for this by subtracting the profit made by the buy and hold strategy. When comission is charged, the buy and hold strategy respectively makes 23.81, 96.28 and -42.72 profit on the training, validation and test datasets. The adjusted profit is shown in figure 5.43. None of the agents were able to outperform the buy and hold strategy on all datasets. This supports our idea that the agents do not have enough information to deal with commission costs.

Finally, in the hierarchical experiment, we noted that some of the hierarchical agents tended to make more trades, and speculated that this might translate to more

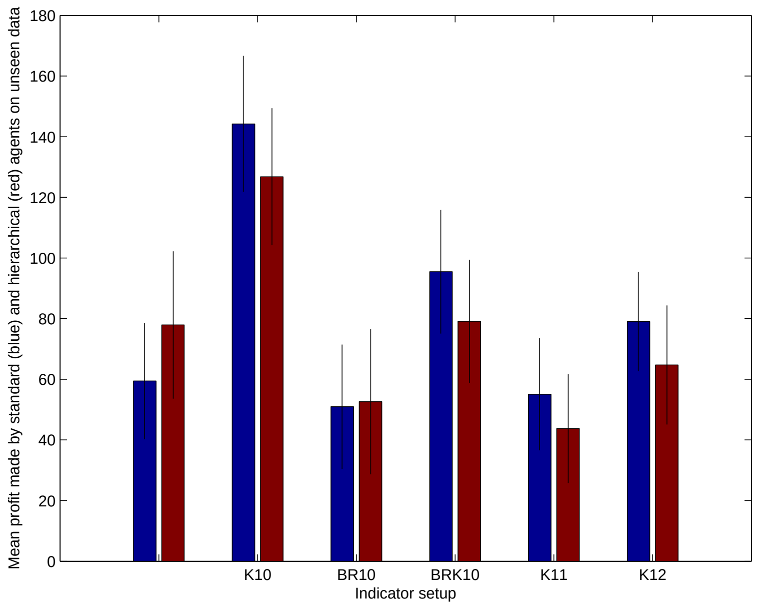


Figure 5.44: Mean profit made by standard (blue) and hierarchical (red) agents on unseen data

robust strategies. If this is the case, then we should see the hierarchical agent making significantly higher profits than the standard agent on the test dataset. However, this is not seen in figure 5.44.

Chapter 6

Conclusion

In this dissertation we were motivated by the question of whether hierarchical reinforcement learning can outperform standard reinforcement learning on markets with regime change. We showed that in a very simple scenario where an agent has no indicators, a hierarchical agent can learn a policy that makes more profit than any policy the standard agent can find. The robustness of a policy that uses no indicators whatsoever is likely to be poor, so this is not a practical result. However, it does show that there do exist trading problems where the hierarchical structure gives the agent an advantage.

Unfortunately, our other comparisons between the profit made by the standard and hierarchical agent did not find any significant difference. The hierarchical agent found strategies that made slightly more trades under certain conditions. It was speculated that these strategies may be more robust, and thereby perform better on unseen data. However, no such effect was observed when the performance of standard and hierarchical agents on unseen data was compared.

We can therefore conclude that, given our agent design, there is no significant difference between performance of the standard and hierarchical agent. The poor performance of the hierarchical agent was assumed to be caused by the inconsistent nature of the reward signals used by agents to select a goal and next subagent. Thus, a design that reduces the noisiness of these reward signals might give the hierarchical agent an advantage. A possible way of doing this might be to use subagents with different reward functions. This might cause the subagents to learn more distinctive policies which gives them a clear advantage in different regimes. This would require substantial alterations to the current algorithm.

Our research also found some interesting results that were unrelated to the hypothesis. Agents provided with two or more technical analysis indicators were able to achieve greater profits than agents with one or none, confirming the results of Dempster & Jones [7]. These agents had found strategies that made more trades, which we speculated might be more robust. Indeed, agents with more indicators could make a substantial profit on all three of our datasets.

Our agent design allows the agent to use its technical analysis indicators to generate signals at different time scales. A long signal was included so that the agent could identify the underlying market regime. The option of using two short signals from consecutive time steps was included so that the agent could potentially learn complex technical analysis rules. Both of these design decisions were justified by our results. Giving agents a long signal allowed them to make more profit without making significantly more trades. We speculated that the long signal helped restore the Markov property, so the agent had a better idea of how the market transitioned, allowing it to make better trades. On the contrary, giving agents two short signals caused them to find complex strategies which made many trades and could profit on noisy market segments.

Charging agents a commission charge for trading caused them to make fewer trades. However, this was not enough to offset the charge, and so these agents generally failed to outperform the simple ‘buy and hold’ strategy on both seen and unseen data. As realistic traders must always pay a commission cost, this indicates that our strategies would have little merit in real world applications. It was noted that the agents which were charged commission had difficulty making money from short positions. This led us to reason that the indicators provided to the agents were not informative enough to facilitate the complex trading strategies needed to make money in the face of commission charges. Thus, giving the agent an even greater number of indicators might overcome this problem.

Unfortunately, our design meant that we could only give the agents a limited number of indicators without causing the agents observation space to become infeasibly large. This meant that we did not investigate the most complicated indicator combinations possible under our design, such as using all three technical analysis indicators to generate two short and one long signal. A modification of our design that allowed agents to generalise large observation spaces would allow such combinations to be investigated. Moreover, indicators other than Bollinger Bands, RSI and the Stochastic Oscillator could also be investigated. Indicators could also be used to generate signals at a greater number of time scales, possibly longer than the ‘long’ week-scale indicator used here.

Another significant weakness in our design was the poor performance of convergent policies, which was demonstrated several times. The difference between the performance of the best and convergent policies seemed to be greater for agents that traded more often. This was assumed to be a result of the market not following a true Markov process. It was hoped that the hierarchical structure would help restore the Markov property. Thus, if the design of our agent was changed so that the hierarchical structure did have an advantage, then convergence might improve. Alternatively, there are also many methods of improving convergence in non-Markov processes in the existing literature [19, 22] that could be incorporated.

The root of most of the design faults stems from the learning algorithm, rather than the chosen observation space. The use of thresholds to form the holding state is unusual and seems promising. It was shown that agents without these thresholds performed worse than agents with them. There were also unexpected results in the informal threshold test where the agent learns dramatically different policies for different upper thresholds. We did not examine this phenomenon further. Our observation space can be uncoupled from the reinforcement learning algorithm, so other approaches could be used to investigate it. Since our strategies can be represented by a binary bit string, some sort of genetic algorithm could be a good alternative to reinforcement learning.

In light of our hypothesis, the key finding of this work is that a hierarchical agent can outperform a standard agent in a trivial case. It is regrettable that we could could not produce more results which illuminate our hypothesis, but this work has also produced several results that are interesting from other perspectives. Our results showing how the agent can tailor its strategies to the information it is given provide a convincing example of an agent that learns to trade, rather than being told how to trade. Some of these results reaffirm what has been noted by other researchers, but others, particularly our examples of agents adapting their strategies, are novel results for reinforcement learning.

Bibliography

[1] Cfd trading from gni touch, August 2008. http://www.gnitouchcfds.com/.

[2] Do the error bars overlap? not a very helpful question., August 2008. http://www.graphpad.com/ library/ BiostatsSimple/ article_55.htm.

[3] Ig markets - cfds, forex, indices, August 2008. http://www.igmarkets.co.uk/.

[4] Russell indexes (russell.com), August 2008. http://www.russell.com/ indexes/.

[5] R. G. Bates, M. A. H. Dempster, and Y. Romahi. Evolutionary reinforcement learning in fx order book and order flow analysis. In *Proceedings 2003 IEEEE International Conference on Computation Intelligence for Financial Engineering*, pages 355–362, 2003.

[6] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362, 1992.

[7] M. A. H. Dempster and M. C. Jones. A real-time adaptive trading system using genetic programming. *Quantitative Finance*, pages 397–413, 2001.

[8] M. A. H. Dempster and V. Leemans. An automated fx trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30:543–552, 2006.

[9] M. A. H. Dempster, T. Payne, Y. Romahi, and G. Thompson. Computational learning techniques for intraday fx trading using popular technical indicators. *IEEE Transactions on Neural Networks*, 12(4), 2001.

[10] M. A. H. Dempster and Y. Romahi. Intraday fx trading: an evolutionary reinforcement learning approach. *IDEAL 2002 (Lecture Notes in Computer Science 2412)*, pages 347–358, 2002.

[11] X. Gao and L. Chan. An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization. In *Proceedings of International Conference on Neural Information Processing, ICONIP 2000*, pages 832–837, 2000.

[12] J. Hamilton. Regime-switching models. In *New Palgrave Dictionary of Economics and Law, 2nd Edition*. Palgrave, 2008.

[13] S. Hasinoff. Reinforcement learning for problems with hidden state. Technical report, University of Toronto, 2002.

[14] K. Kim. *Electronic and Algorithmic Trading Technology*. Academic Press (Elsevier), 2007.

[15] B. Lebaron. Technical trading rule profitability and foreign exchange intervention. *Journal of International Economics*, 49(1):125–43, 1999.

[16] J. Lee and J. O. A multi-agent q-learning framework for optimizing stock trading systems. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, volume 2453, pages 153–162, 2002.

[17] A. Lo and A. MacKinlay. Stock market prices do not follow random walks: Evidence from a simple specification test. *Review of Financial Studies*, 1:41–66, 1988.

[18] A. Lo and A. MacKinlay. *A Non-Random Walk Down Wall Street*. Princeton University Press, Princeton, NJ, 1999.

[19] J. Moody and M. Saffell. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889, 2001.

[20] R. Neuneier. Optimal asset allocation using adaptive dynamic programming. *Advances in Neural Information Processing Systems*, 8:952–958, 1996.

[21] J. O, J. Lee, W. J. Lee, and B. Zhang. Adaptive stock trading with dynamic asset allocation using reinforcement learning. *Information Sciences*, 176:2121–2147, 2006.

[22] D. Ormoneit and P. Glynn. Kernel-based reinforcement learning in average-cost problems: An application to optimal portfolio choice. *Advances in Neural Information Processing Systems*, 13:1068–1074, 2000.

[23] H. Osada and S. Fujita. CHQ: A multi-agent reinforcement learning scheme for partially observable markov decision processes. *IEICE - Transactions on Information and Systems*, E88-D(5), 2005.

[24] P. Refenes. *Neural Networks in the Capital Markets.* John Wiley & Sons, 1995.

[25] J. Schwager. *Getting Started In Technical Analysis*. John Wiley and Sons, 1999.

[26] H. Subramanian. Evolutionary algorithms in optimization of technical rules for automated stock trading. Master's thesis, The University of Texas at Austin, 2004.

[27] H. Subramanian, S. Ramamoorthy, P. Stone, and B. Kuipers. Designing safe, profitable auotmated stock trading agents using evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2006.

[28] S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[29] M. Taylor and H. Allen. The use of technical analysis in foreign exchange markets. *Journal of International Money Finance*, 11:304–314, 1992.

[30] M. Waldrop. Computers amplify black monday. *Science*, 238(4827):602–604, 1987.

[31] C. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 1992.

[32] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1997.