Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Evaluation of Hierarchical Temporal Memory in algorithmic trading

by

Fredrik Åslin

LIU-IDA/LITH-EX-G--10/005--SE

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Examensarbete

Evaluation of Hierarchical Temporal Memory in algorithmic trading

av

Fredrik Åslin

LIU-IDA/LITH-EX-G--10/005--SE

2010-02-08

Handledare: Fredrik Heintz

Examinator: Peter Dalenius

Abstract

This thesis looks into how one could use Hierarchal Temporal Memory (HTM) networks to generate models that could be used as trading algorithms. The thesis begins with a brief introduction to algorithmic trading and commonly used concepts when developing trading algorithms. The thesis then proceeds to explain what an HTM is and how it works. To explore whether an HTM could be used to generate models that could be used as trading algorithms, the thesis conducts a series of experiments. The goal of the experiments is to iteratively optimize the settings for an HTM and try to generate a model that when used as a trading algorithm would have more profitable trades than losing trades. The setup of the experiments is to train an HTM to predict if it is a good time to buy some shares in a security and hold them for a fixed time before selling them again. A fair amount of the models generated during the experiments was profitable on data the model have never seen before, therefore the author concludes that it is possible to train an HTM so it can be used as a profitable trading algorithm.

Contents

Table of Figures

1. Introduction

This section serves as an introduction to the thesis. It starts with presenting the objective of this thesis in section 1.1, then goes on to explain the motivation for doing this thesis in section 1.2. The problem statements answered in this thesis are stated in section 1.3 and the target audience of this thesis is presented in section 1.4.

1.1. Objective

The goal of this study is to evaluate whether a machine learning approach called Hierarchical Temporal Memory (HTM) can be used to generate profitable strategies for algorithmic trading. A profitable trading algorithm is defined as an algorithm that makes at least 5% more profitable trades than losing trades. The HTM algorithms will also be compared to an existing approach based on the genetic programming tool Discipulus.

1.2. Motivation

There are two main reasons for investigating new approaches to algorithmic trading: To see if the new approach is vastly superior to the current and to create an ensemble of different approaches which together works better than any individual approach.

The current approach used by the company 8bit exclusively use genetic programming (GP) to generate programs for algorithmic trading. The exclusive GP approach often generates algorithms that are to some degree correlated with each other. Two correlating algorithms will buy and sell at about the same time, doubling the exposure to the market and increasing the risk. On the other hand, two uncorrelated algorithms will trade at different times leading to less risk and exposure to the market.

It is possible that other technologies could generate more profitable and uncorrelated algorithms. With this in mind, 8bit has been looking for a way to incorporate other technologies such as HTM into their system.

1.3. Problem formulation

The question this thesis tries to answer is:

- Will training an HTM on the same input data as Discipulus result in equally good models when used as trading algorithms?
- How do HTM trading strategies compare to strategies generated by Discipulus.

1.4. Target group

This thesis addresses mainly the employees at 8bit and my fellow computer science students. It will give them an insight to how to train and develop an HTM, as well as how one might formulate a problem so the trained HTM can be used for algorithmic trading.

1.5. Used tools

The experiments in this thesis all used the NuPIC framework version 1.7.1 available from Numenta at http://numenta.com/. The framework is bundled with Python 2.5.4 which were used to build and train the HTM networks in the experiments.

2. Algorithmic trading

This chapter starts with a brief introduction to algorithmic trading, presenting a commonly used trading algorithm and the typical way of developing trading algorithms. Some common concepts in algorithmic trading are explained in section 2.1.

In electronic financial markets, algorithmic trading is the use of computer programs for deciding when to enter trading orders into the market considering aspects of the order such as the timing, quantity, and price. Some algorithms operate completely without any human interaction. (The Economist, 2006)

Algorithmic trading has been steadily increasing and as of 2009 account for 73% of all US equity trading volume (Iati, 2009). The larger part of algorithmic trading is concerned with reducing transaction costs and executing large orders on the market without anyone noticing (Wikinvest). For example, if one would directly buy 500000 shares in Sony Ericsson the price of the stock would increase since one would buy more or less every share on the market to any price. To try to keep the average price of the trade down, it is common to use trading algorithms to divide the trade into a set of smaller ones, e.g. 5000 shares 100 times, and execute those trades over a time interval.

The common way of developing an algorithm for trading is to implement a strategy a human has already developed for himself while trading on the financial market into an executable program. Many day-traders[1] sit in front of computers following the price changes and financial news concerning some security[2], then buy respectively sell when they believe they know where the market is headed. Many of their decisions could equally well be made by a computer.

One algorithm that is used heavily in algorithmic trading is called arbitrage which is the practice of taking advantage of a price differential of a security between two or more exchanges (Iati, 2009). In theory every exchange should sell or buy a security at the same price but in practice, price differences occur which the arbitrage algorithm uses. For example, one can buy S&P 500 future security[3] both at the GLOBEX exchange and at the ISIS exchange. The security is the same regardless where you buy it. Thereby if the price in GLOBEX is 100 when the price in ISIS is only 99.75, then it is possible to buy shares in ISIS and then sell an equal amount of shares in GLOBEX to make a small profit. This algorithm is not risk-free since if not both trades are executed simultaneously the prices might change unfavorable before the remaining trade takes place. In practice, it is the one who can react and execute trades the fastest that can take advantage of this algorithm.

2.1. Common terms in algorithmic trading

2.1.1. Price data of securities

The value of a security at any point in time is the amount that someone is prepared to buy it for and someone else is prepared to sell it for. There are securities that at times that do not have anyone willing to buy them, and it can be argued that they do not have any price or value at that specific time. The price of a security usually fluctuates in time, as can be seen in the Figure 1 representing the

[1] Day-trader is a person who repeatedly buys and sells securities within the same trading day and usually close all open positions before the market closes.

[2] A security is an instrument representing a financial value. For example a common stock as a share in Sony Ericsson

[3] Standard and Poor 500 future

OMX stock price from 2009-12-05 to 1010-01-05. One OMX stock cost 13.10 dollars in the beginning of December 2009, and rose to 13.40 dollars on the fifth of January 2010.

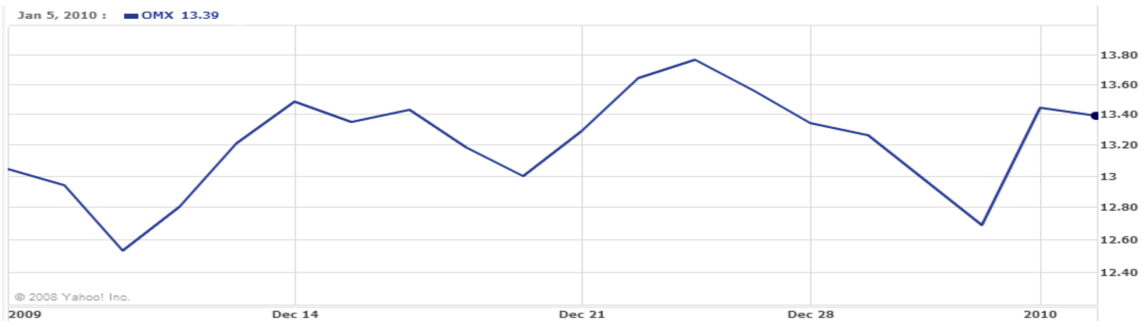

Figure 1 – OMX stock price movement during December 2009

There are a lot of factors that affect the market price of a security. For example; the Sony Ericson might secure a large order to build a new mobile network in India. Upon hearing that news, people might come to the conclusion that it is going well for Sony Ericson and buy some stocks in the company.

Independent of the reason, the consequence is that either more people want to sell, or buy the security. If there are more people wanting to buy rather than sell a security, the price usually goes up.

Example, the market for a security is usually divided into two sides called the buy and sellside. Figure 2 shows two examples of the sides for a figurative security. In the above example in Figure 2, the buyside consists of orders to buy 5 shares for $100, and 10 shares for $99.75. The sellside consists of orders to sell 10 shares for $100.25 and 50 shares for $100.5. The current price is then said to be either 100 or 100.25 since usually one always tries to buy the shares with the lowest price, and vice versa.

The price will not change until someone has bought all the shares at the lowest price, in this example 100.25, or someone sells all the shares at the highest price, in this example 100. One way to raise the price in this example, is to offer to buy 15 shares at 100.25, which will change the market into the lower table in Figure 2. The price will then be said to be either 100.25 or 100.50 and the price of the security has increased 0.25.

	buyside	sellside	
shares	price	price	shares
		100.50	50
		100.25	10
5	100.00		
10	99.75		

	buyside	sellside	
shares	price	price	shares
		100.50	50
5	100.25		
5	100.00		
10	99.75		

Figure 2 - Buy/Sell example

2.1.2. Charts

Financial price movements over time are commonly illustrated by plotting it in an Open-High-Low-Close (OHLC) chart. Each vertically line, data point, on the chart is typically referred to as a bar and shows the price range (highest and lowest value) over either a unit of time, e.g. one day or one hour, or a fixed amount of trades, e.g. 50 trades. Tick marks projected from each side of the line indicates the opening price (e.g. the price on the first trade in the bar) on the left, and the closing price on the right. The top indicates the highest price in the bar, and the bottom shows respectively the lowest price in the bar.

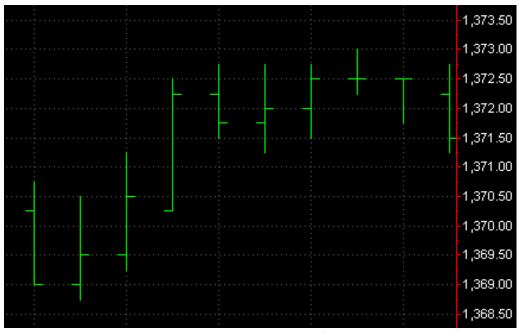

Figure 3 - Tick chart example, ES[4] built from 729 ticks / bar

A tick is a representation of a trade and consists of the execution price together with the amount of shares bought. The Standard and Poor future security had in January an average of 212 ticks per minute.

2.1.3. Indicators

Another technique commonly used by algorithmic traders is to use so called indicators, which is an algorithm that is applied to a chart to try to bring out some useful information about the price movements. One example of an indicator is to use the value of a moving average on the closing price. Each bar it calculates the average of the closing price for the last X bars. The number of bars back it uses is usually called the length, such as a moving average with length 5 to refer to an indicator which returns the average of the last 5 bars. Figure 4 shows a tick chart with two moving average indicators superimposed, the blue line has a length of 5 bars and follows the closing price much closer than the pink line which uses a length of 15 bars. The idea behind this indicator is that when looking at the average closing price instead of the current closing price, one gets a better trend line, a smother curve.

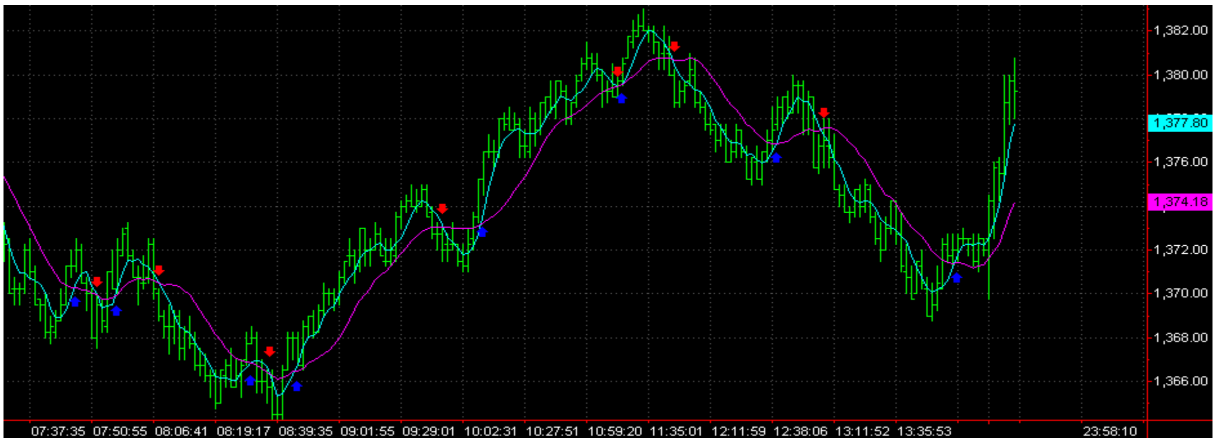

Figure 4 - Tick chart on ES[4] with two moving average indicators superimposed

[4] ES – The symbol for E-mini Standard and Poor 500 security

3. The process used by 8bit

This chapter introduces Genetic Programming (GP) and the program Discipulus, which 8bit is currently using to generate trading algorithms. It starts with a brief introduction to how GP works and then moves on to show an example output from Discipulus in section 3.2.

8bit was founded 2006 with the goal of creating a fund which bought and sold various securities based solely on decisions taken by computers. 8bit concluded that there were two ways of achieving their goal, to develop an algorithm by hand and have a computer execute it or developing a system which generates trading algorithms by itself. 8bit choose the later alternative.

8bit has developed a system which uses genetic programming (GP) to generate algorithms that is capable of deciding when to enter and exit the market either by buying or shorting[5] the security it operates on. 8bit is currently using Discipulus as their main GP engine.

The process of generating an algorithm is composed of four steps as illustrated in Figure 5. As the GP engine works with numbers, the first step is to generate "interesting" data from the market in a format suitable for the GP engine. This is done by applying different indicators on price data for some security. The next step is to run the GP engine and let it work to find good algorithms by using the supplied input data. It is then necessary to analyze whether the generated algorithms is good enough to be used on the market.

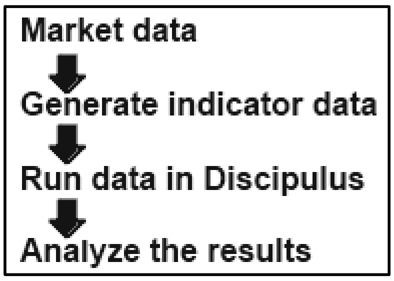

Figure 5 - Overview of 8bit's GP process

One problem with the current system is that it has a tendency to generate trading algorithms that are highly correlated with each other. The goal is to have different kinds of algorithms that trade on different market conditions to diversify the risk.

8bit is currently trying to use other technologies when generating trading algorithms to try to solve this. One technology they are interested in is Hierarchical Temporal Memory (HTM).

3.1. Discipulus and Genetic Programming (GP)

Genetic Programming is one branch in the field of machine learning, which is the field of study where one tries to give the computer the ability to learn. The term itself was coined by Samuel in 1959, the first person who made a computer perform a serious learning task (Samuel, 1959). A good modern definition of machine learning is the study of computer algorithms that improve automatically through experience (Wolfgang, Nordin, Keller, & Francone, 1997).

GP tries to do that. GP is the automatic, computerized creation of computer algorithms to perform a select task. The GP uses an evolutionary algorithm-based methodology inspired by biological evolution to find programs that performs the given task well enough. How good a program performs

[5] Shorting is the act of selling assets, usually securities, that has been borrowed from a third party with the intent of buying identical assets back at a later date to return them to the lender. The person shorting a security is hoping to make a profit from a decline in the price of the borrowed assets.

the task is evaluated by a fitness function[6] that takes a program and returns a value indicating how good the program is. (F.D.Francone--Chalmers University of Technology and RML Technologies, Inc)

The GP used in this study is Discipulus™. The algorithm used to generate programs in Discipulus works as follows. It starts with a population of randomly generated computer programs, step 1 in Figure 66. These programs are the base for the evolution process which hopefully leads to a program that performs the task well. Then, Discipulus randomly chooses four programs, step 2 in Figure 66, from the base and let them compete against each other for in a tournament. The two best programs, of the four picked, are copied and also transformed by either mutation or crossover transformations into two new programs. This is represented in step 3 in Figure 6 with a green respectively a red program originating from the two winning programs. Then the two new programs from step 3 replace the two losing programs from step 2. Discipulus continues to repeat step 2 and 3 until it has generated a program that performs the task. (F.D.Francone--Chalmers University of Technology and RML Technologies, Inc)

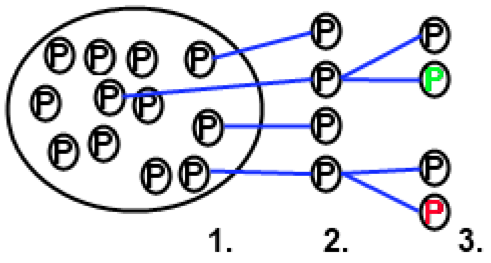

Figure 6 - Discipulus working process

The two transformations mutation and crossover, mentioned above, takes inspiration from biology. Mutation transforms a program by randomly changing an instruction into something else, for example; a subtraction might be replaced by an addition instruction. Crossover transformation picks a set of instructions from one of the two winning programs to be changed with a set from the other, e.g. an addition from one program might be replaced by a subtraction from the other program.

As explained above, Discipulus writes computer programs from examples given to it. It is a supervised learning system, which means one has to tell Discipulus the correct output for each input. The correct input is then used in the fitness function when evaluating how good the program performs the task.

A common practice is to use two different data sets when evolving programs. These data sets are commonly called “training data” and “validation data” which need to be supplied to Discipulus before any program generation can be done. Training data is the data set which Discipulus uses to learn and build models. The validation data set is used to provide information about how well the generated programs do on data they haven’t been trained on. It is important to note that no data from the validation set is ever used when generating programs. There is also a third data set which Discipulus calls “applied data”, used to see how already generated programs do on completely new data. For example, in the trading algorithm case one might want to try the programs on data that wasn’t available at the time they were generated them.

[6] A function that gives each program a value that represents how well it performed the task

3.2. An example of program generated by Discipulus

Discipulus can output generated programs in various programming language such as C or Assembler. Because the C language has a simple syntax and can commonly be understood without extensive programming knowledge, this thesis will restrict itself to only use C when presenting generated programs from Discipulus.

One example of a program generated by Discipulus can be seen in Figure 77. The programs always come in the form of a function that takes an array of floats as input and returns a float as the result of the program. The input array, float v[], represents one row in the input data, and thus v[0] refers to the first data column, v[1] to the second column and so on.

The other declared array, long double f[8], is used by Discipulus as a temporary storage of values between calculations. This can been observed in Figure 7 on the last label L5, where f[0] is divided by f[1] which was calculated before in labels L1 and L2.

```
float DiscipulusCFunctionSubC0(float v[])
{
  long double f[8];
  long double tmp = 0;
  int cflag = 0;

  f[0]=f[1]=f[2]=f[3]=f[4]=f[5]=f[6]=f[7]=0;

  L0:     f[0]+=v[4];
  L1:     f[1]+=v[1];
  L2:     f[1]*=f[0];
  L3:     tmp=f[3]; f[3]=f[0]; f[0]=tmp;
  L4:     f[0]-=v[2];
  L5:     f[0]/=f[1];

  return f[0];
}
```

Figure 7 - Example program generated from Discipulus

4. Hierarchical Temporal Memory (HTM)

This chapter introduces the theory behind the HTM technology in section 4.1, briefly explains how the HTM works in section 4.2 with a short example in section 4.3.

HTM is a computing framework developed at Numenta[7] that replicates the structure and function of the human neocortex. It has been applied to various problems, such as tracking people and objects in videos[8]to an HTM that recognize and play tick-tack-to[9].

[7] www.numenta.com
[8] http://www.numenta.com/about-numenta/demoapps.php
[9] http://www.numenta.com/vision/webservices-embed.php?startpage=netinfo&nid=22d4f3c5888b594e8bf0

4.1. The theory behind HTM

In 2004 Jeff Hawkins published the book “On Intelligence” laying out his proposal of how the brain works and how the intelligence of humans can be mimicked by computers.

The basic theory is simple and can be summarized in these two points.

1. The structure of an HTM network mirrors the hierarchical structure of objects in the world. Larger objects are made up of smaller objects, Songs are composed of verses, books contain pages and the page themselves contains paragraphs, sentences, words and letters.
2. Each node in the HTM network implements a common algorithm that identifies commonly occurring patterns, both spatial and temporal.

The first point was derived from biology by looking at how neocortex works and processes information. The second point originates from a theory by Mountcastle that the brains different regions all work in nearly the same way.

The neocortex is a part of the brains of mammals and is a thin sheet of neural tissue that wraps around the brain and handles all higher level thoughts such as perception, language, imagination, mathematics, arts, and planning (Campbell, 2002).

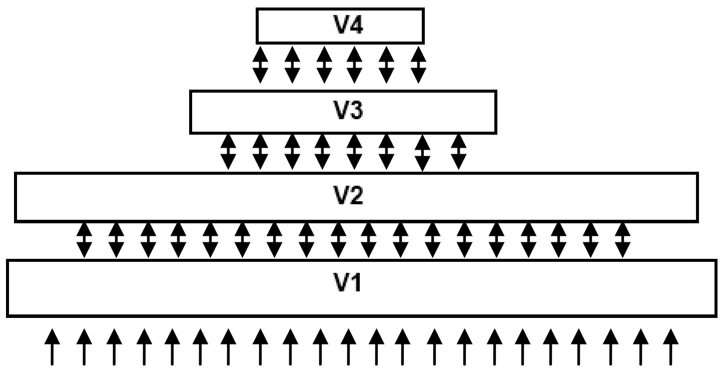

Figure 8 – Illustration of the hierarchy in the visual region

The neocortex has six horizontal layers, where the lowest layer receives sensory data such as vision and sound. For example, visual information enters the cortex through the primary visual area, called V1 for short. V1 sends information upwards to other layers, such as V2, V4, and V5. Each layer is also connected to the layers below it, so as V1 sends information up to V2, V2 at the same time sends information down to V1. It is also known that the lowest region in the visual cortex looks for very small things, for example a single cell on the retina responds to a fixed color. If one goes up in the hierarchy one finds cells that responds for more complex things, for example a cell in V3 might respond to star shaped symbols. (Hawkin & Blakeslee, 2004)

In 1978 Vernon Mountcastle, a neuroscientist at John Hopkins University in Baltimore, published a paper titled “An organizing Principle for Cerebral Function”. In this paper Mountcastle points out that the brain is remarkably uniform in its structure, and that most regions look the same even though they handle completely different things. Mountcastle then proceeds to suggest that if most regions are practically the same, then most regions most also process information and work the same way. (Mountcastle, 1978)

The structure with nodes separated into interconnected layers sets a restriction if one wants to build a framework mimicking the brain. The framework would have to follow the structure of the neocortex and have layers of nodes where each layer is connected both to the layers above it and also to the layers below it. In accordance to the theory of Mountcastle, all nodes share a common algorithm.

4.2. How an HTM works

HTMs perform the following two basic functions regardless of the problem they are applied to:

1) Discover causes in the input
2) Infer causes of novel input

The world for an HTM consists of objects and their relationships where objects could be cars, songs, or people. A necessary property of the world being modeled is that the objects persist over time, for example, an apple sitting in our field of vision will show up as an object hitting the retina. Unless one moves the attention elsewhere so the apple leaves the field of vision, the apple will persist over time. In HTM world, the objects that persist over time and cause patterns to emerge in the input data are called “causes”. (Numenta, Inc, 2007)

All HTM networks have some type of sensory input data which is fed into the lower layer in the HTM to allow it to learn the causes and patterns in the world modeled by the data. There are two criteria for the sensory data for it to be a good choice for an HTM. First, the data has to measure the world in a way so the causes in it directly or indirectly affect the sensory data, i.e. to make a model to predict the weather one would want to use data such as air temperature and pressure as these are directly affected by the weather conditions. Second, the sensory data has to change with time while the underlying causes remain relatively the same.

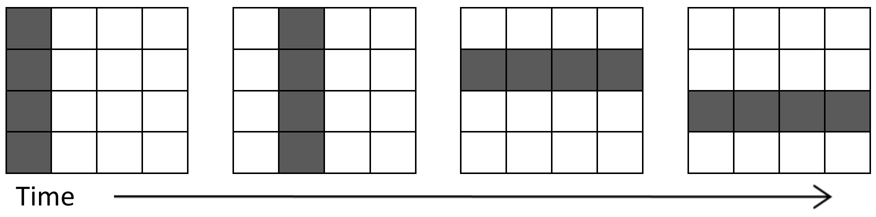

Figure 9 Spatial and temporal patterns

Suppose the input to a node is a 4 x 4 bit array, the number of possible patterns is 2^16. Some patterns tend to recur such as those in Figure 10, which looks like horizontal and vertical lines. If each array represents a snapshot of the input data in time, than the node might infer that the horizontal lines moves to the left and the vertical lines moves down for each time unit. The node identifies the spatial patterns and the sequences of them as a cause, i.e. it infers that an external cause most be responsible for the occurring patterns.

After an HTM has learned what the causes in the world are, and how to represent them, it can perform inference. Given a sensory input data stream, an HTM will “infer” what the causes are likely to be present in the world at that moment.

4.2.1. How an HTM discovers and infers causes

HTMs are structured as tree-shaped hierarchies of nodes, Figure 11 shows a simple HTM hierarchy with 6 nodes in the bottom layer, 3 in the middle and one at the top. Each node looks for spatial and temporal patterns, explained further below, of its input and learns to assign causes to these input patterns. As stated above, each node discovers the causes of its input.

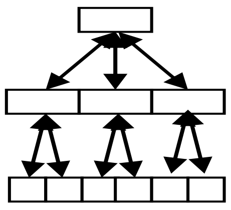

Figure 10 - Simple HTM hierarchy

Each node starts with a fixed set of possible causes. In the beginning, there is no meaning to any of the possible causes, but as the HTM trains on the sensory data, it assigns meaning to them. This can changed during the training session and it isn't guaranteed that a cause will have the same meaning during the whole training. (Numenta, Inc, 2007)

The basic node in an HTM is divided into two operations, (the latest version of the framework has the node split into two nodes that perform one of these operations each). Each node has a set of quantization points representing some spatial patterns which, as stated above, are unknown until the network learns them during training. During training, the node decides how close the current input is to each of its quantization points and assigns a probability to each quantization point. (Numenta, Inc, 2007)

Spatial patterns are patterns within the current sensory input vector. Suppose a node has 10 inputs, and two of those inputs, I_1 and I_2, are active at the same time often enough one could assume they have a common cause. This follows common sense, if two things often follow each other we can assume they have a common cause somewhere. If a node has learned a set of spatial patterns, the next time it receives novel input, it looks at how close the input is to each spatial pattern and assigns a probability that the input matches each of the known spatial patterns. These spatial patterns are the quantization points discussed earlier. (Numenta, Inc, 2007)

Temporal patterns are patterns on how the spatial patterns occur in time. If the node has learned 10 spatial patterns, sp_1 thru sp_{10}, and then suppose that the node observes that sp_2 often follows sp_8, and does it more often than if it occurred by chance, one could assume the $sp_2 – sp_8$ has a pattern in time – temporal pattern. The output of the node to the nodes in the higher hierarchies is the probability that the current input belongs to each learned temporal pattern. (Numenta, Inc, 2007)

4.3. Bitworm example application

This section introduces a simple example called Bitworm that is available when downloading the 1.7.1 version of the HTM framework from Numenta.

The input data are 16-bit vectors. There are solid bitworms which consists of consecutive 1-bits (left part in Figure 11), and textured bitworms consisting of consecutive alternating 1 and 0 bits (right part in Figure 11). The bits not representing a bitworm consists of 0-bits.

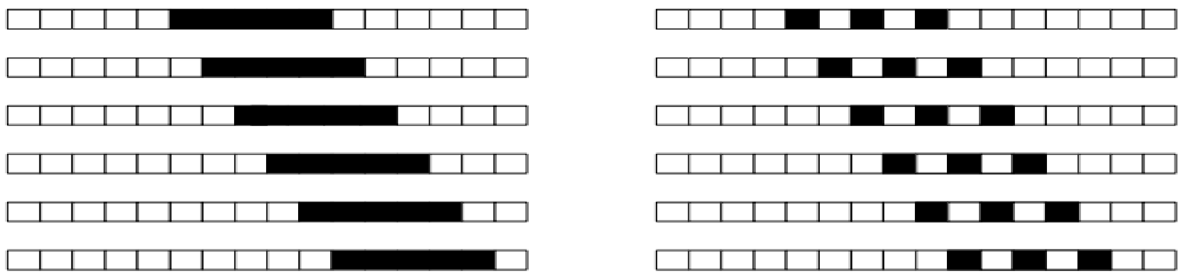
Figure 11 - Solid and textured bitworms

4.3.1. Data representation

The example trains an HTM to model the world of bitworms and then infer if there is either a solid bitworm or textured bitworm in the input.

0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0
0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0
0	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0

Figure 12 - Bitworm example data without noise

Figure 11 shows a representation of the input data for the bitworm example. Each row represents a single input vector with 16 bit values. The output is either class0 – a solid bitworm exists in the input vector, or class1 – a textured bitworm exists. The first six rows shows a solid bitworm moving for each row one step closer the left edge, the six last rows shows a textured bitworm moving from the left edge towards the right edge with one step for each input row.

4.3.2. Running without noise

Running the bitworm example without noise will yield a 100% correct prediction in the training data, and 97.86% correct prediction, 411 correct out of 420 input vectors, on the applied data.

The HTM found 40 spatial patterns and 7 temporal patterns.

Group 1															
0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

Group 2															
0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0
0	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0
0	0	0	1	0	1	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0
0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	0
0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0

Figure 13 – Causes learned from data without noise

Figure 12 shows some of the causes found by the HTM when it was trained on the training data. Each row represents a spatial pattern found by the HTM, and the rows together form a sequence of spatial patterns over time. That sequence is the temporal patterns found by the HTM.

Looking at Group1, one can clearly see a solid bitworm moving first a step to the right, then moving steadily to the right. Similar, looking at Group2 shows a textured bitworm moving steadily to the right with the exception of the last rows which one can expect is actually the first spatial pattern in the sequence.

4.3.2. Running with noise

The noise will be represented as a bit-flip, where each bit in the 16 input vector has a 15% chance of flipping to 0 if it was 1, and 1 if it was 0.

Running the bitworm example with noise will yield a 100% correct prediction in the training data, and 74.76% correct prediction, 314 correct out of 420 input vectors, on the applied data.

The HTM found 360 spatial patterns and 18 temporal patterns. The number of spatial patterns found are almost as many as the total amount of input data rows used in training (420 input vectors). The temporal patterns more than doubled and the size of each group are more than twice the size of the groups when running the example without noise. More temporal patterns indicate that the HTM didn't any longer find any clear sequences of spatial patterns but it could still remember most of the spatial patterns and temporal sequences to overfitt the model. This can be observed by looking at the drop in correct prediction from 100% to 74.76%.

5. The process of developing an HTM

This chapter discusses the process of developing an HTM. It starts with a short introduction in section 5.1, and then discusses what problems are suitable for an HTM in section 5.2. Data generation for an HTM is discussed in section 5.3. The development process for developing a model is discussed in section 5.4.

5.1. Introduction

The process of developing an HTM application differs from a traditional software engineering process. Traditional software is programmed and the behavior of the program is controlled directly, an HTM, like neural networks, is trained rather than programmed and the behavior is the result of the configuration of the network, node parameters, the input data and so on. (Hawkin & Blakeslee, 2004)

Because the nature of HTMs is that it can be a bit hard to predict the consequences when one change a parameter slightly, often resulting in an iterative process, where one starts with an idea of how one thinks the HTM would work best, try it out and gradually change different parameters and configurations as one inspects the results from these tests. This process could be implemented as a hill climbing process where one restricts the parameters and values involved and let the process optimize each parameter in turn, keeping the values that result in the best models.

5.2. Defining the problem

According to Numenta a suitable problem for an HTM network should have these two properties.

1. Spatial hierarchy. Data generated by common sub-causes are often correlated. In any problem domain, input that is affected by the same cause will have some correlation. One example of a problem that has a spatial hierarchy is a model of a car. A simple model could include the temperature of the engine, the speed which the wheels are turning, and so. The spatial hierarchy in this example is for example is that an increase in temperature in the engine usually means that the speed increased in the wheels as well. (Numenta, Inc, 2007)
2. Temporal hierarchy. Higher-level causes vary more slowly compared to lower-level causes. In the above example model of a car, a temporal hierarchy could for example be when the car accelerates. When accelerating the engine temperature will increase since it will start to burn more fuel in an effort to increase the speed, the tires will reflect this and rotate faster. When the car reaches the target speed, the tires will keep their rotation speed while the engine will cool down a little since it is easier to maintain than accelerate. Acceleration can only be observed by looking at the temperature and tire speed through time.(Numenta, Inc, 2007)

Algorithmic trading commonly uses the financial price data over time as the source for input data when testing and developing trading algorithms. Because of this the input data for the HTM model in the financial case should be able to find temporal hierarchies, sequences of indicator values over a time interval. Commonly, indicators used when developing a trading algorithm all use the same market data source. Because all input data originally comes from the same source, any change in the source should be reflected over all input data. This should bring up some spatial patterns, patterns of indicators which in some way are correlated with each other.

5.3. Representing the data

The input data for both Discipulus and HTM is a vector with floating point numbers, where the number of input variables decides the length of the input data vector. Both can handle arbitrary large and small numbers with the only restriction that the values most be represented as a floating points.

Both Discipulus and HTM work on floating point data but there is a fundamental difference in how they process the data. Discipulus uses only a small set of linear mathematical functions, such as addition, subtraction, multiplication, division, etc to process the input.

HTM process the data rather differently. As described in section 4.2, each node finds common spatial patterns and then finds sequences of the spatial patterns found. Because a node has to find spatial patterns it treats each input value uniquely, e.g. the value 3.003 is different than 3.0. This creates a problem when dealing with data working on very wide or precise data ranges. (Hawkin & Blakeslee, 2004)

To counter this problem there is a way to cluster data points near each other into the same class. The parameter MaxDistance sets the maximum Euclidean distance at which two input vectors are considered the same. For example, in figure 11, points A and C are quite close to each other. If the MaxDistance parameter is large enough so the circle would include point C, both point A and C would be considered to be occurrences of the same point. If MaxDistance would be large enough to include all three points, then all three points would be considered as occurrences of the same data point. (Numenta, Inc)

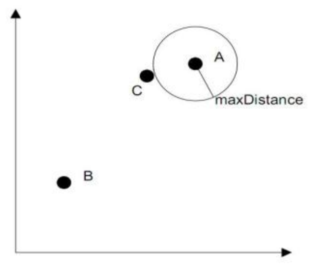

Figure 13 - MaxDistance example

5.4. Development and training of the HTM network

As mentioned before, the development of an HTM is an iterative process and it is hard to know before what are the best parameters without some experimenting. When repeatedly experimenting with the parameters and network settings on the same data set, one run into the problem that one might optimize the settings for that particular data set. This is commonly called to over-fit a model. This problem could be avoided by using cross validation. This is a technique for assessing how the results of an analysis will generalize to an independent data set. One round of cross-validation involves partitioning a sample of data into subsets, performing the analysis on one subset (called training data) and validating the results on the other subset (called validation set).

This problem could be avoided if one had unlimited amount of data to train on, since it would be possible to use completely unused input data for each experiment.

Another way to be able to use old data would be to reverse the data series, going from the future to the past. This way one would be able to use the available data twice. In trading algorithm development, there is always a notion of time in the data series. Based on the past price movements, one want to predict the future movements and if possible, act upon it and make some profit. By reversing the data series, one would start in the future and try to predict the past which isn't the goal when trading (one can't buy now and sell it yesterday).

To avoid this problem, the experiments in this thesis all follow the following procedure.

1. Define the problem the HTM and GP should solve
2. Start with a basic idea of what the network structure and parameters should be for the HTM for this problem.
3. Construct an HTM with the above settings and parameters.
4. Divide the data into two smaller data sets, training[10] and validation[11], and run the above mentioned HTM multiple times to optimize the settings until the HTM can't be improved.

[10] Training data set – the data used for training the model

5. Divide the data into two medium data sets, training and validation. Then running the HTM from step 4 on the validation set, noticing its performance on the bigger data set compared to the smaller one. This gives an indication whether the smaller dataset was representative for this bigger data set.
6. Run the HTM multiple times with the data sets from step 5 to tweak the HTM once again. This step might not be relevant if no noticeable change could be seen when comparing the results from the smaller data set to the bigger set.
7. Divide the data into two large data sets, training and validation, and run the HTM resulting from step 6 over the validation data. Once again, comparing the results from the data sets in step 5 to the large data set generated in this step.

The parameters experimented upon is the following

- The topology of the network, represented as a vector with the number of nodes in each layer. For example, [2,1] would generate a network with two nodes in the bottom level and one node in the top level. Each level in the topology is constrained so it can be evenly divided by the lower level. For example, if the bottom level is four the top level can only be 4, 2 or 1 node.
- MaxDistance parameter, as explained in section 5.3 it is important to cluster the input data into smaller values if the input data range is too large.
- Coincidences – the number of spatial patterns a node can maximal learn during training
- Groups – the number of temporal patterns a node can maximal learn during training

6. Experimental Setup

6.1. Problem definition

One of the objectives in this thesis is to study whether an HTM could replace Discipulus in the current system, so the problem in the following experiments has to be as close as possible to the problem Discipulus solves today.

An HTM handles only classification problems, that is, to say what class the input belongs to. For example, if the classification function searched for is: “class P if all inputs is greater than 0, class N if otherwise”, then the input {1,0} belongs to class N and input {1,1} belongs to class P. Discipulus can generate programs with almost equal output to mathematical functions such as F(x) = x^2 given only a set of input values and its respective output for F(x). Discipulus can also handle binary classifaction problems. HTM on the other hand can handle an arbitrary numer of classes.

With this constraint in mind, the problem for the following experiments is defined as:

To predict if the price will increase with at least 2 ticks[12] compared to the current bars closing price after 10 bars.

[11] Validation data set – the data used for validation that the model works on data it hasn’t trained upon.
[12] Tick – the smallest valid price unit on a security. For example, OMX stock exchange trades with 0.01 as the smallest difference between two prices.

If one would have a model that could predict that the price will increase with at least 2 ticks compared to the current price, one could buy some shares at the beginning at the next bar and then sell them at the predicted time. If the prediction is true, and price doesn't change more than 1 tick before one can execute the first order, the trade will be profitable. Because the price is either on the buy or sell side, see chapter 2.1.1, it could very likely increase or decrease depending on if the last executed trade. If the last trade was on the sell side, and the next trade is on the buy side, the price will very likely increase with 1 tick making it very unlikely to be able to buy at the predicted price. To avoid these complications and increasing the chances of a successful trade, the definition predicts an increase with at least 2 ticks.

The classes used are;

0 – The price will not increase with more than 1 tick after 10 bars.
1 – The price will increase with at least 2 ticks after 10 bars.

6.2. Prediction results

This problem definition should capture how one would use the generated model as a trading algorithm, and the meaning of class 1 is that the algorithm buys some shares and holds it for the next 10 bars. The class 0 on the other hand means that the algorithm does nothing. It is impossible to lose any money while doing nothing, so it doesn't matter if the algorithm predicts the input to be in class 0 when the correct answer is class 1. In the reversed case, when the model predicts a class 1 when the correct prediction would be class 0. The resulting action would be to buy some shares while the price will decrease, leading to losing money. This isn't entirely true either, because the definition is that the market has to increase at least 2 ticks, it is possible that the market will only increase with 1 tick. If so, depending on if it is possible to execute the trade before the price change, the trade might still make a small profit.

With that in mind, the results from the experiments are presented into four groups.

- False positive (FP) – predicts class 1 when correct is class 0
- True positive (TP) – correctly predicts class 1
- False negatives (FN) – predicts class 0 when the correct is class 1
- True negatives (TN) – correctly predicts class 0

Because of the above reasoning, it is important to the success of the algorithm to have a high TP while having a low FP. The other two groups, FN and TN, are included for inclusiveness but don't affect the result of the trading algorithm since they would not lead to any action.

6.3. Experimental layout

The input data for Discipulus in the current system used by 8bit is a random set of indicators; see chapter 3, which outputs values from Booleans to Decimals. As described in chapter 5, an HTM is sensitive about what kind of input it trains on and needs spatial and temporal patterns to be successful. If using a random set of indicators as input, it might be possible that the indicators picked doesn't inhibit any easily inferred patterns making it harder for the HTM. 8bit has used Discipulus for a while and noticed that the result from pure Boolean input is much weaker than decimal input. Given this, the following tests will use two input data sets, one with only Boolean values and one with only decimal values.

6.4. Specification of the experiments

The experiments will use two financial market data sources, Standard and Poor 500 E-Mini[13](ES). 8bit is currently using 249 tick-bar[14] charts to generate input data. The development of the HTM will follow the outlined procedure in section 5.3, and use two data sets, one for training and one for verifying that the model performs about the same on data it hasn't been trained on. These data sets are called training data and validation data. The sizes for the small, medium and large data sets mention in the procedure are listed in the table below.

The experiments will vary three different parameters, the topology of the network, the MaxDistance parameter for each node and the amount of coincidences and groups each node is able to learn.

The MaxDistance parameter decides the granularity of the input data, an optimal value for MaxDistance should cluster the input data in a way that makes it easy for the nodes to detect spatial and temporal patterns. This should mean that the optimal value for MaxDistance is independent from the topology, coincidences and groups setting. The experiments will then start by analyzing the input data by using different MaxDistance settings for a simple HTM.

The topology should affect how many coincidences and groups a node has to learn to be able to perform well. For example, Figure 14 shows an HTM with four nodes in the bottom layer and two nodes in the top layer (blue squares). The input is visualized as an 8x8 bit array. The left node in the bottom layer receives its inputs from the top 4x4 bit square as visualized by the lines in Figure 14. The number of possible patterns in the input is 2^16. If the left node would instead receive its input from the top 4x8 part of the array, the number of possible patterns increases to 2^32. The value of coincidences and groups states how many patterns a node can learn and if it can only learn a small subset of the possible patterns it can't model complex behaviors and sequences. Therefore one should first find a good topology, then optimize the number of coincidences and groups a node can learn.

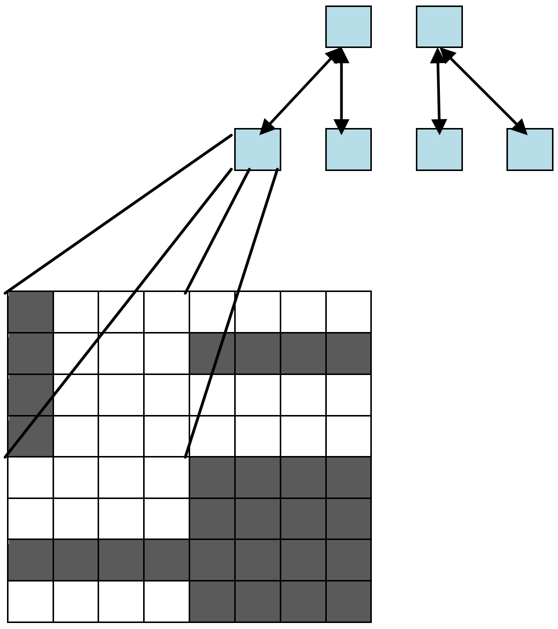
Figure 14 - HTM with topology of [4,2]

[13] http://www.cmegroup.com/trading/equity-index/us-index/e-mini-sandp500.html
[14] 249 Tick bar chart – each data point in the chart represents 249 ticks

6.5. Experiments on the decimal data set

Because of the reasoning that the optimal value for MaxDistance should be independent of both the topology, coincidences and groups settings, MaxDistance is optimized first. The input values in this data set ranges from [2,0.00001], therefore it would not make any sense to increase MaxDistance higher than 2 because that would cluster all input data into the same value. The experiement will start by training models that have MaxDistance values in the range of [0.00001,0.5], where 0.5 is deemed large enough to cluster most values into very few values.

As stated in section 5.4, each level in the topology has to be evenly divided by the preceding level. The experiments uses 30 indicators as input and thus the bottom level can only be 30, 15, 10, 5, 3, or 1.

Another constrain is that using a topology together with a large value for coincidences and groups caused the Numenta framework to run out of memory. The out of memory exception is raised when a node is trying to allocate a too large memory block causing the python framework to raise an exception. No solution was found to this problem so the range of topologies used in this experiment is constrained to use no more than 4 levels.

Experiments

1. The first experiment trained 13 models, M1-13, with various MaxDistances between [0.00001,0.5]. The goal was to find a suitable value to use for the next set of experiments. Model M6 was picked as one of the winners because it had the highest TP/FP ratio in the validation data set. The second winner was model M10 since it had the highest profit as well as the second highest TP/FP ratio in the validation data set.
2. The second experiment trained 9 models, M6.1-9, with various topologies with the goal to find the topology resulting in the highest TP/FP ratio in the validation set.
3. The third experiment trained 8 models, M6.4.1- 8, from the best model in experiment 2, M6.4, with various settings for coincidences and groups (CaG). The settings ranged from 64-768, where the memory constraint disabled the usage of any higher values than 768. The model, M6.4.7, from the trained set was deemed the winner for the small data set because it had the highest TP/FP ratio (even compared to the best model in experiment 5 below). The model then ran on the medium data set to see whether it would perform equally on larger data set.
4. The forth experiment trained 8 models, M10.1- 8, with various topologies using the base model MaxDistance of [0.05,0,0]. The goal was the same as experiment 2 above, to find the topology resulting in the highest TP/FP ratio.
5. The fifth experiment trained 6 models, M10.1.1- 6, with various settings for coincidences and groups. The model M10.1 was designated as the winner because even though it had slightly worse TP/FP ratio than model M10.2, it almost earned twice as much. The MaxDistance settings for this branch of experiments were chosen because it earned the most in experiment 1.
6. The sixth experiment increased the bar used in training and validation data sets to 25002, trained 9 models, M6.4.7.1- 9, using the MaxDistance and CaG values from the base model M6.4.7. The goal was to see if the best topology changed when training on a larger data set.
7. The seventh experiment trained 5 models, M6.4.7.4.1- 5, with the new topology from the best model in experiment 6, M6.4.7.4, on various CaG values. Then ran the best performing

model, M6.7.4.3, on the largest training data used in the experiments to see if the performance changed when using a large set of data not yet used in training or validation.
8. The eighth experiment trained 7 models, M10.1.6.1-7 tried various topologies using the MaxDistance value from M10 and the CaG value from the winning model, M10.1.6, on the medium data set to see if the best topology would be different compared to what was found in experiment 6.
9. The ninth experiment increased the data used in training and validation to 50002 bars. Trained 3 models, M6.4.7.3.1-3, with three different topologies that had been proved to be good throughout the experiments.
10. The tenth and final experiment trained four models, M6.4.7.3.2.1-4, with various CaG values using the topology from the best model, M6.4.7.4.3.2, in the previous experiment.

6.6. Experiments on boolean data set

Experiments

1. The first experiment trained 13 models, M1-13, with various topologies with the goal to find the best topology to use when training on the small data set. The CaG parameter was chosen to use 128 for every level as the best model on the largest data set used this value as well. The best topology was found to be [30, 15, 5] as it had the highest TP/FP ratio and lost the least amount of money.
2. The second experiment trained 16 models based on M5, M5.1-16, with various CaG settings. The previous experiments winning models topology was used in every model. The goal was to find the best CaG setting. The best model in the experiment was M5.10 as it had both the highest TP/FP ratio as well as lost the least amount of money. As the winner for the small data set, M5.10 also ran on the medium data set to see how it performed on a larger set of unseen data.
3. The third experiment increased the data set in training and validation data to the medium data set. Then trained 10 models,M5.10.1-10, based on the winning model in experiment 2, M10, with various topologies to try to find the best topology for the larger data set.
4. The forth experiment trained 7 models, M5.10.9.1-7, with the winning topology in experiment 3, M5.10.9, and various CaG settings to try to find the best CaG for best topology. The winning model, M5.10.9.3, also ran on the large data set to see if it lost any performance when running on a much larger set of not yet seen data.

7. Results

This chapter presents the results from the experiments and discusses the results say in section 7.4.

7.1 Results for Decipulus

		Validation				
market	market	TP	FP	TP/FP	TN	FN
Decimal	5002	351	1905	0.184	2340	406
Decimal	50002	6021	17319	0.348	19858	6805
Boolean	5002	753	1459	0.516	1884	906
Boolean	50002	4671	18736	0.249	21463	5132

7.2 HTM results for the Decimal data set

			Training data		Validation data			Coincidences and	
Experiment	Models	DS[15]	TP/FP	Profit	TP/FP	profit	Topology[16]	Groups[17]	MaxDistance
E1	M0	S	3.417	104287.5	0.840	3925	[10, 5]	[512, 256]	[1.0e-005, 0]
E1	M1	S	3.375	104550.0	0.830	2375	[10, 5]	[512, 256]	[0.0001, 0]
E1	M2	S	3.375	104550.0	0.830	2375	[10, 5]	[512, 256]	[0.0001, 0.0001]
E1	M3	S	3.454	104175.0	0.845	4613	[10, 5]	[512, 256]	[0.001, 0.001]
E1	M4	S	3.308	102812.5	0.825	2950	[10, 5]	[512, 256]	[0.01, 0.01]
E1	M5	S	3.449	101575.0	0.837	4000	[10, 5]	[512, 256]	[0.05, 0.05]
E1-WR[18]	M6	S	2.311	82475.0	0.873	5400	[10, 5]	[512, 256]	[0.1, 0.1]
E1	M7	S	1.707	58937.5	0.826	-563	[10, 5]	[512, 256]	[0.2, 0.2]
E1	M8	S	1.381	35212.5	0.813	4150	[10, 5]	[512, 256]	[0.5, 0.5]
E1	M9	S	3.425	102625.0	0.823	713	[10, 5]	[512, 256]	[0.01, 0]
E1-WP[19]	M10	S	3.268	101650.0	0.860	7113	[10, 5]	[512, 256]	[0.05, 0]
E1	M11	S	2.132	77987.5	0.858	4075	[10, 5]	[512, 256]	[0.1, 0]
E1	M12	S	1.694	58300.0	0.829	1100	[10, 5]	[512, 256]	[0.2, 0]
E1	M13	S	1.365	40175.0	0.804	3150	[10, 5]	[512, 256]	[0.5, 0]
E2	M6.1	S	1.697	62225.0	0.884	7875	[30, 10, 5]	[512, 256, 128]	[0.1, 0.1, 0.1]
E2	M6.2	S	1.494	49225.0	0.886	7413	[30, 15, 5]	[512, 256, 128]	[0.1, 0.1, 0.1]
E2	M6.3	S	1.199	26650.0	0.881	7600	[15, 5, 1]	[512, 256, 128]	[0.1, 0.1, 0.1]
E2	M6.4	S	1.222	24862.5	0.905	8938	[10, 5, 1]	[512, 256, 128]	[0.1, 0.1, 0.1]
E2	M6.5	S	1.153	17662.5	0.868	3375	[10, 2, 1]	[512, 256, 128]	[0.1, 0.1, 0.1]
E2	M6.6	S	1.682	55462.5	0.830	1750	[15, 5]	[512, 256]	[0.1, 0.1]
E2	M6.7	S	1.646	54325.0	0.833	1825	[15, 3]	[512, 256]	[0.1, 0.1]
E2	M6.8	S	2.311	82475.0	0.873	5400	[10, 5]	[512, 256]	[0.1, 0.1]
E2	M6.9	S	1.797	62500.0	0.874	8725	[10, 2]	[512, 256]	[0.1, 0.1]
E3	M6.4.1	S	4.177	123612.5	0.886	7738	[10, 5, 1]	[256, 256, 256]	[0.1, 0.1, 0.1]
E3	M6.4.2	S	2.153	77287.5	0.883	10038	[10, 5, 1]	[256, 128, 128]	[0.1, 0.1, 0.1]
E3	M6.4.3	S	1.683	57975.0	0.872	4125	[10, 5, 1]	[128, 128, 128]	[0.1, 0.1, 0.1]
E3	M6.4.4	S	1.263	39725.0	0.878	7725	[10, 5, 1]	[64, 64, 64]	[0.1, 0.1, 0.1]
E3	M6.4.5	S	40.373	171512.5	0.892	6850	[10, 5, 1]	[512, 512, 512]	[0.1, 0.1, 0.1]
E3	M6.4.6	S	204.333	177687.5	0.898	9725	[10, 5, 1]	[768, 768, 768]	[0.1, 0.1, 0.1]
E3-WR	M6.4.7	S	14.664	155000.0	0.925	14250	[10, 5, 1]	[768, 512, 256]	[0.1, 0.1, 0.1]
E3	M6.4.7	M*			0.854	3550	[10, 5, 1]	[768, 512, 256]	[0.1, 0.1, 0.1]
E3	M6.4.8	S	4.423	113050.0	0.890	6950	[10, 5, 1]	[368, 256, 128]	[0.1, 0.1, 0.1]
E6	M6.4.7.1	M	1.180	23725.0	0.956	10712.5	[30, 10, 5, 1]	[768, 512, 256, 256]	[0.1, 0.1, 0.1, 0.1]
E6	M6.4.7.2	M	1.222	11012.5	0.931	4625	[30, 15, 5, 1]	[768, 512, 256, 256]	[0.1, 0.1, 0.1, 0.1]

[15] The data set size used in training and validation. S = 5002, M=25002 and L = 50002 bars.
[16] Topology – the network layout, for example; [2,1] is a network with two nodes in the bottom, and one node in the top.
[17] Coincidences and groups – the maximal number of spatial patterns (coincidences) and temporal patterns (groups) that a node can learn. For example, [16,8] sets the networks so the nodes in the bottom layer can learn 16 spatial and temporal patterns, and the nodes in the top layer can learn 8 spatial and temporal patterns.
[18] The model that has the highest TP/FP ratio in the experiment
[19] The model that has the highest profit in the experiment.

			Training data		Validation data			Coincidences and	
Experiment	Models	DS	TP/FP	Profit	TP/FP	profit	Topology	Groups	MaxDistance
E6	M6.4.7.3	M	2.521	107000.0	0.963	5150.00	[30, 10, 5]	[768, 512, 256]	[0.1, 0.1, 0.1]
E6-WR	M6.4.7.4	M	4.172	44550.0	1.074	9262.50	[30, 15, 5]	[768, 512, 256]	[0.1, 0.1, 0.1]
E6	M6.4.7.5	M	1.056	10137.5	0.963	-1187.50	[15, 5, 1]	[768, 512, 256]	[0.1, 0.1, 0.1]
E6	M6.4.7.6	M	1.231	33650.0	1.001	40087.50	[10, 5, 1]	[768, 512, 256]	[0.1, 0.1, 0.1]
E6	M6.4.7.7	M	0.889	-7237.5	0.944	13287.50	[10, 2, 1]	[768, 512, 256]	[0.1, 0.1, 0.1]
E6	M6.4.7.8	M	2.891	293112.5	0.909	10825.00	[10, 5]	[768, 512]	[0.1, 0.1]
E6	M6.4.7.9	M	1.831	177675.0	0.893	-6000.00	[10, 2]	[768, 512]	[0.1, 0.1]
E7	M6.4.7.4.1	M	3.314	57425.0	1.038	12463	[30, 15, 5]	[256, 256, 256]	[0.1, 0.1, 0.1]
E7	M6.4.7.4.2	M	3.653	23837.5	1.190	6038	[30, 15, 5]	[256, 128, 128]	[0.1, 0.1, 0.1]
E7-WR	M6.4.7.4.3	M	3.835	28925.0	1.239	9025	[30, 15, 5]	[128, 128, 128]	[0.1, 0.1, 0.1]
E7	M6.4.7.4.3	L*			1.124	2550	[30, 15, 5]	[128, 128, 128]	[0.1, 0.1, 0.1]
E7	M6.4.7.4.4	M	3.429	9062.5	1.091	950	[30, 15, 5]	[64, 64, 64]	[0.1, 0.1, 0.1]
E7	M6.4.7.4.5	M	3.514	106150.0	0.955	27775	[30, 15, 5]	[512, 512, 512]	[0.1, 0.1, 0.1]
E9	M6.4.7.4.3.1	L	1.807	98962.5	0.941	17338	[30, 10, 5]	[128, 128, 128]	[0.1, 0.1, 0.1]
E9-WR	M6.4.7.4.3.2	L	2.372	73175.0	1.022	363	[30, 15, 5]	[128, 128, 128]	[0.1, 0.1, 0.1]
E9	M6.4.7.4.3.3	L	1.002	-3562.5	0.979	-2938	[10, 5, 1]	[128, 128, 128]	[0.1, 0.1, 0.1]
E10	M6.4.7.4.3.2.1	L	2.477	132462.5	1.017	-10825	[30, 15, 5]	[256, 256, 256]	[0.1, 0.1, 0.1]
E10	M6.4.7.4.3.2.2	L	2.044	54987.5	0.975	-13075	[30, 15, 5]	[256, 128, 128]	[0.1, 0.1, 0.1]
E10-WR	M6.4.7.4.3.2.3	L	2.372	73175.0	1.022	363	[30, 15, 5]	[128, 128, 128]	[0.1, 0.1, 0.1]
E10	M6.4.7.4.3.2.4	L	1.708	25950.0	0.856	-15788	[30, 15, 5]	[64, 64, 64]	[0.1, 0.1, 0.1]
E4-WP	M10.1	S	2.173	74787.5	0.908	12587.5	[30, 15, 5]	[512, 256, 128]	[0.05, 0, 0]
E4	M10.2	S	1.246	28225.0	0.888	7675	[15, 5, 1]	[512, 256, 128]	[0.05, 0, 0]
E4	M10.3	S	1.361	31450.0	0.910	12437.5	[10, 5, 1]	[512, 256, 128]	[0.05, 0, 0]
E4	M10.4	S	1.168	16837.5	0.884	6337.5	[10, 2, 1]	[512, 256, 128]	[0.05, 0, 0]
E4	M10.5	S	1.898	67162.5	0.824	1187.5	[15, 5]	[512, 256]	[0.05, 0]
E4	M10.6	S	1.817	64625.0	0.819	1700	[15, 3]	[512, 256]	[0.05, 0]
E4	M10.7	S	3.268	101650.0	0.860	7112.5	[10, 5]	[512, 256]	[0.05, 0]
E4	M10.8	S	2.131	70762.5	0.859	3637.5	[10, 2]	[512, 256]	[0.05, 0]
E5	M10.1.1	S	2.153	76312.5	0.849	8925	[30, 15, 5]	[256, 256, 256]	[0.05, 0, 0]
E5	M10.1.2	S	1.558	46737.5	0.877	7738	[30, 15, 5]	[256, 128, 128]	[0.05, 0, 0]
E5	M10.1.3	S	1.566	50925.0	0.877	5725	[30, 15, 5]	[128, 128, 128]	[0.05, 0, 0]
E5	M10.1.4	S	1.426	41200.0	0.878	5538	[30, 15, 5]	[64, 64, 64]	[0.05, 0, 0]
E5	M10.1.5	S	5.372	118675.0	0.865	6175	[30, 15, 5]	[512, 512, 512]	[0.05, 0, 0]
E5-WP	M10.1.6	S	2.420	76575.0	0.898	13300	[30, 15, 5]	[368, 256, 128]	[0.05, 0, 0]
E8-WP	M10.1.6.1	M	1.825	82512.5	1.040	43713	[30, 10, 5]	[368, 256, 128]	[0.05, 0, 0]
E8	M10.1.6.2	M	3.264	144662.5	0.996	12025	[30, 15, 5]	[368, 256, 128]	[0.05, 0, 0]
E8	M10.1.6.3	M	1.185	11800.0	0.934	6875	[15, 5, 1]	[368, 256, 128]	[0.05, 0, 0]
E8	M10.1.6.4	M	1.130	-2537.5	1.093	-13338	[10, 5, 1]	[368, 256, 128]	[0.05, 0, 0]
E8	M10.1.6.5	M	0.886	-14775.0	0.902	4650	[10, 2, 1]	[368, 256, 128]	[0.05, 0, 0]
E8	M10.1.6.6	M	2.337	183187.5	0.899	4438	[10, 5]	[368, 256]	[0.05, 0]
E8	M10.1.6.7	M	1.395	93387.5	0.934	8400	[10, 2]	[368, 256]	[0.05, 0]

7.3 HTM results for the Boolean data set

			Training		Validation			Coincidences
Experiment	Models	DS	TP/FP	profit	TP/FP	profit	Topology	Groups
E1	M1	S	1.094	12737.5	0.816	-33737.5	[30, 10, 5, 1]	[128, 128, 128, 128]
E1	M2	S	1.018	15000	0.813	-25887.5	[30, 15, 5, 1]	[128, 128, 128, 128]
E1	M3	S	1.063	21375	0.823	-29212.5	[30, 15, 3, 1]	[128, 128, 128, 128]
E1	M4	S	1.268	29287.5	0.864	-23700	[30, 10, 5]	[128, 128, 128]
E1-WR	M5	S	1.242	18312.5	0.882	-18687.5	[30, 15, 5]	[128, 128, 128]
E1	M6	S	1.228	33325	0.869	-17062.5	[30, 15, 3]	[128, 128, 128]
E1	M7	S	1.141	23175	0.796	-25887.5	[10, 5, 1]	[128, 128, 128]
E1	M8	S	1.080	19862.5	0.818	-24937.5	[15, 5, 1]	[128, 128, 128]
E1	M9	S	1.181	27987.5	0.828	-18337.5	[15, 3, 1]	[128, 128, 128]
E1	M10	S	1.141	23175	0.796	-25887.5	[10, 5, 1]	[128, 128, 128]
E1	M11	S	1.531	52712.5	0.835	-26387.5	[15, 5]	[128, 128]
E1	M12	S	1.399	59350	0.797	-33737.5	[15, 3]	[128, 128]
E1	M13	S	1.163	31750	0.814	-48737.5	[10, 2]	[128, 128]
E2	M5.1	S	0.953	4875	0.811	-32225	[30, 15, 5]	[16, 16, 16]
E2	M5.2	S	1.041	5225	0.815	-32737.5	[30, 15, 5]	[16, 32, 32]
E2	M5.3	S	1.041	5225	0.815	-32737.5	[30, 15, 5]	[32, 32, 32]
E2	M5.4	S	1.086	1062.5	0.799	-11287.5	[30, 15, 5]	[64, 32, 16]
E2	M5.5	S	1.097	8612.5	0.842	-20087.5	[30, 15, 5]	[64, 64, 64]
E2	M5.6	S	1.097	8612.5	0.842	-20087.5	[30, 15, 5]	[128, 64, 64]
E2	M5.7	S	1.097	9262.5	0.819	-23837.5	[30, 15, 5]	[128, 128, 64]
E2	M5.8	S	1.242	18312.5	0.882	-18687.5	[30, 15, 5]	[128, 128, 128]
E2	M5.9	S	1.242	18312.5	0.882	-18687.5	[30, 15, 5]	[256, 128, 128]
E2-WR	M5.10	S	1.427	17212.5	0.894	-5512.5	[30, 15, 5]	[256, 256, 128]
E2	M5.10	M*			1.005	31037.5	[30, 15, 5]	[256, 256, 128]
E2	M5.11	S	1.300	18050	0.970	-2950	[30, 15, 5]	[256, 256, 256]
E2	M5.12	S	1.356	26575	0.955	-5450	[30, 15, 5]	[368, 368, 368]
E2	M5.13	S	1.364	25337.5	0.964	-4312.5	[30, 15, 5]	[512, 512, 512]
E2	M5.14	S	1.364	25337.5	0.964	-4312.5	[30, 15, 5]	[768, 512, 512]
E2	M5.15	S	1.345	26050	0.949	-5700	[30, 15, 5]	[768, 768, 768]
E2	M5.16	S	1.300	18050	0.970	-2950	[30, 15, 5]	[368, 256, 256]
E3	M5.10.1	M	0.000	2537.5	1.326	687.5	[30, 10, 5, 1]	[256, 256, 128, 128]
E3	M5.10.2	M	0.000	-2387.5	1.016	-1925	[30, 15, 5, 1]	[256, 256, 128, 128]
E3	M5.10.3	M	0.000	2175	0.982	-1937.5	[30, 15, 3, 1]	[256, 256, 128, 128]
E3	M5.10.4	M	0.000	675	0.444	-1100	[30, 10, 5]	[256, 256, 128]
E3	M5.10.5	M	0.000	0	0.000	0	[30, 15, 5]	[256, 256, 128]
E3	M5.10.6	M	0.000	0	0.000	0	[30, 15, 3]	[256, 256, 128]
E3	M5.10.7	M	0.000	3000	1.093	-1500	[15, 5, 1]	[256, 256, 128]
E3	M5.10.8	M	0.000	13100	0.983	-6825	[15, 3, 1]	[256, 256, 128]
E3-WR	M5.10.9	M	0.000	3750	1.364	2625	[15, 5]	[256, 256]
E3	M5.10.10	M			0.957	-6425	[10, 2]	[256, 256]
E4	M5.10.9.1	M	0.000	8525	1.075	-987.5	[15, 5]	[768, 512, 512]
E4	M5.10.9.2	M	0.000	9625	1.000	-750	[15, 5]	[768, 768, 768]

			Training		Validation			Coincidences
Experiment	Models	DS	TP/FP	profit	TP/FP	profit	Topology	Groups
E4	M5.10.9.3	M	0.000	3750	1.364	2625	[15, 5]	[368, 256, 256]
E4-WR	M5.10.9.3	L*			1.105	1687	[15, 5]	[368, 256, 256]
E4	M5.10.9.4	M	0.000	-28100	0.942	-59200	[15, 5]	[16, 16, 16]
E4	M5.10.9.5	M	0.000	7862.5	1.069	9037.5	[15, 5]	[32, 32, 32]
E4	M5.10.9.6	M	0.000	14387.5	1.072	3175	[15, 5]	[64, 64, 64]
E4	M5.10.9.7	M	0.000	11050	0.990	-2812.5	[15, 5]	[128, 128, 128]

7.4. Discussion

This section discusses some of the observations from the experiments. The models in the Decimal input data case mostly outperformed the models in the Boolean input data case both in training and in validation data.

Comparing the TP/FP results between training and applied data, one observes a quite large drop in the ratio between TP/FP (which is the important output since it will cause the trading algorithm to buy a security). This is generally called overfitting, and is caused by the model learning a behavior specific for the data it trains on and not a generalized pattern that also holds in the validation data. A common way of dealing with overfitting is giving the model more data to work on to disable it to learn too specific information of the domain and force it to rely on more generalized behavior. As can be observed from the results, the ratio drops to around the same as the validation data when going from 5002 to 25002 bars in training and validation input data without increasing the coincidences and groups parameters.

7.4.1. Results for the decimal data set

The two trained models, M6.4.7 and M6.4.7.4.3, both ran on a large amount of data not included in training. The TP/FP ratio and profit was lower compared to the small validation set but both algorithms made a profit. This indicates that the trained model found patterns in the input data that also occurs when running on data further in the future and is a necessary property for trading algorithms. This property is commonly called robust trading algorithm in algorithmic trading.

The decimal data set used two different MaxDistance settings,M6 with [0.1,0.1,0.1] and M10 with [0.5,0,0]. The experiments based on M6 and M10 often concluded that the best topology was either [30, 15, 5] or [10, 5, 1]. This supports the assumption made in section 6.5 that MaxDistance should not affect the optimal topology for the problem.

7.4.2. Results for the Boolean data set.

Most models trained on the Boolean data set lost money in the validation data set even though some of them have a TP/FP ratio over 1. This is further discussed in section 7.4.4 but a summary is that the average profit made from each TP is lower than the average lost made from each FP.

Models trained on the Boolean data set predicted class 1 more rarely than compared to the models from the decimal case. The models also predicted fewer class 1 when trained on larger data sets compared to when trained on small. In the decimal case, increasing the data set size used in training and validation also meant an increase in the TP/FP ratio.

7.4.3. Discipulus results

Discipulus and HTM generated completely different results under the same conditions. While the model generated by Discipulus only predicted class 0 for almost every bar, as seen by the high TN and low FN ratio (easiest to achieve if one only predicts class 0 for every input), HTM models made a fairly high ratio of predicted class 1. Discipulus behavior could be explained by looking at how it evaluates a model.

Discipulus evaluates a model by looking at the average hit rate of positives and negatives. This is calculated as the average of FN and TN. The simplest model only predicts class 0, which gives it an average hit rate of around 50% (because the distribution of the classes in the data sets is about 50%). Then, when trying to increase the average hit rate by prediction more input to belong to class 1, it is a high probability that a TN will be replaced by either FP or TP. If a new model predicts mostly class 0 but some class 1, it is a high probability that what was previously a TN gets turned into a FP and lowering the average hit rate.

HTM doesn't have the same problem of only generating models that mostly predicts input as belonging to class 0. The top node in an HTM network looks at the input and tries to learn what the probability of the current input belonging to either class 0 or class 1. HTM doesn't care about the hit rate or anything similar when learning so the simple solution of only predict the class 0 doesn't occur as easily.

The underlying problem here is that the only relevant values for evaluating how good a model is, if used as a trading algorithm, to look at the ratio between TP and FP. A solution to the problem with Discipulus would be to implement a new fitness function that returns total TP / total FP for fitness.

7.4.4. Profit results

The profit the model would theoretical have made if used as a trading algorithm is presented for each HTM model. The profit and number of trades is calculated as stated in the problem definition, if the model predicts the input to belong to class 1, it represents that the model buys 1 share of a security and sells it 10 bars later. The profit for one trade is then the price difference, price 10 bars later minus the current close price. The security used in the experiments is Standard and Poor E-mini 500, where the big point value is 50. The big point value represents the amount of dollar one will make if the price goes up by one. The profit for each trade will be multiplied by the big point value to show how many dollars each trade earns or loses.

Another point to keep in mind when looking at the profit price is that a TP always is a profitable trade which earns at least 2 ticks times big point value[20], in these experiments the smallest earning would then be 12.5 dollars. FP on the other hand can have an arbitrary price differential, theoretically, a price could fall with 20% during those 10 bars.

Because of this, even though the model has about the same percentage of TP as FP, it can still lose money if the FPs loses more money than the TPs earns. This is what happens in the Boolean data set where almost all models make a negative profit while having about the same TP/FP ratio as the models from the decimal data set.

[20] 1 tick on ES is worth 0.25 dollars

The goal of the model is to find some patterns in the training data that are general enough to also work well in the validation data. It is a sign that the model is overfitted to the training data when it makes a large profit in training and then makes a negative profit in the applied data.

7.4.5. Coincidences and groups

A good setting for the maximal coincidences and groups a node could learn was found by running over the same data repeatedly while increasing the amount CaGs node could learn.

A pattern that emerged in both data sets was that if one increased the maximal coincidences and groups a node could learn over some value (most likely the value most suited for the current data and settings), the resulting model performs better in the training data while performing worse in the validation data. Allowing a node to learn too many spatial and temporal patterns will lead to an overfitted model as discussed in the beginning of the chapter. This indicates that this parameter greatly affects whether an HTM model will overfit itself or not, if it can, it will remember all possible spatial and temporal patterns because that would enable it to perfectly predict the class1 for each input (it already have stored the current pattern and thus only needs to look it up to see what input will follow it).

8. Conclusions

The company 8bit has developed a system that exclusively uses genetic programming (GP) to generate algorithms for trading. One of the problems they discovered was that the current approach generated algorithms that were to some degree correlated with each other. Two correlating algorithms will buy and sell at about the same time, doubling the exposure to the market and increasing the risk. On the other hand, two uncorrelated algorithms will trade at different times leading to less risk and exposure to the market. An approach to generate uncorrelated algorithms could be to use another tool, such as HTMs, to generate the algorithms.

In line with this, the goal of the thesis was to try to answer whether training an HTM on similar conditions as Discipulus, the GP tool used in 8bits process, would result in equally good models when used as trading algorithms.

The goal of the experiments in this thesis was to develop an HTM model that would be profitable when used as a trading algorithm. An HTM has various settings that affect the training and the resulting model. The experiments looked into three parameters, MaxDistance, network topology and coincidences and groups (CaG).

The experiments all followed a hill climbing process when searching for good values for these parameters. With the assumptions that the optimal MaxDistance parameter would be independent from the rest of the parameters, see section 6.5, the experiments all started with training a set of HTMS to see what value resulted in a model that performed well. After a good value had been found the experiments trained a set of HTMs with various network topologies. The assumption was that the topology decides how many CaGs are needed in each layer, see section 6.5 for further explanations. The third step in the experiments was to train a set of HTMs with CaGs.

A profitable trading algorithm was defined as an algorithm that had at least 5% more profitable trades than loosing trading. The experiments in this thesis generated a few models that passed this criterion. For example, model 6.4.7.4.3 had 23.9% more profitable trades and a profit of 9025 dollars when running the model the validation data. Even more, running the model on a larger set of not yet seen data resulted in a trading algorithm that though it had more loosing trades than profitable trades, made a profit of 2550 dollars.

Four trained models, M5.10.9.3, M5.10, M6.4.7.4.3 and M6.4.7, was extensively tested on, compared to the size of the training data set, large amount of not yet seen data. The four models got a small reduction in performance when they ran on new novel data. This is an attractive property of any trading algorithm as it proves that the algorithm is robust and found some patterns in the input data that exists in newer data.

A fair amount of the generated models reached the goal of at least a 1.05 TP/FP ratio, and several models also made a profit. Since there is more room for improvements through either larger models or a more through parameter optimization the conclusion is that it seems possible to generate profitable models for algorithmic trading using HTMs.

8.1 Future work

The process outlined by the experiments all followed a hill-climbing process by optimizing each parameter iteratively. The order of which parameter to optimize used in this thesis also proved to work fairly well, as seen in the experiments on the decimal data set. The experiments used two different values for MaxDistance parameter, [0.1,0.1,0.1] and [0.05,0,0], and both parameters lead to the same topology [30,15,5].

There are more settings than the three looked into in this thesis, and some of them might be relevant when developing an HTM for use in algorithmic trading. A broader study where one optimizes more parameters might be a suitable continuation of this thesis.

The process of optimizing the parameters for an HTM could easily be done automatically by a computer and incorporated into the current process used by 8bit. The system would then follow the same procedure as the experiments in this thesis to explore what setting results in the best HTM for the given input data.

The experiments in this thesis only trained an HTM on two different data sets, Boolean and Decimal, and as section 6.4 explains, an HTM is sensitive to how the input data is presented to it. There might be better ways of organizing and normalizing the input data than used by in this thesis.

Bibliography

Campbell, N. R. (2002). *Biology.* Benjamin Cummings.

Crosby, J. L. (1973). *Computer Simulation in Genetics.* John Wiley & Sons: London.

Discipulus. *Owners manual.*

F.D.Francone--Chalmers University of Technology and RML Technologies, Inc. (n.d.). Discipulus™ Linear-Genetic-Programming Software: How it Works.

Hawkin, J., & Blakeslee, S. (2004). *On intelligence.* Times book.

Iati, R. (2009, July 10). The Real Story of Trading Software Espionage. *AdvancedTrading.com* .

Mountcastle, V. (1978). *An organizing Principle for Cerebral Function: The Unit Model and the Distributed System.*

Numenta, Inc. (n.d.). Advance Nupic Programming.

Numenta, Inc. (2007, 3, 27). Hierarchical Temporal Memory - Concepts, theory and terminology.

Numenta, Inc. (2007). Problems that fit HTM.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* , 3.

The Economist. (2006, Feb 2). Moving markets Shifts in trading patterns are making technology ever more important.

Wikinvest. (n.d.). High-Frequency Trading.

Wolfgang, B., Nordin, P., Keller, R. E., & Francone, F. D. (1997). *Genetic Programming: An Introduction.* Morgan Kaufmann Publishers.