5-11-2022

Using an Agent Based Model and Deep Learning to Simulate and Infer Market Behavior on Networks

Benjamin Weir
baw3143@rit.edu

Using an Agent Based Model and Deep Learning to Simulate and Infer Market Behavior on Networks

by

Benjamin Weir

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Applied and Computational Mathematics
School of Mathematical Sciences, College of Science

Rochester Institute of Technology
Rochester, NY

Date

May 11th, 2022

Committee Approval:

Bernard Brooks, Ph.D. Date

School of Mathematical Sciences

Thesis Advisor

Kara Maki, Ph.D. Date

School of Mathematical Sciences

Committee Member

Nathan Cahill, D.Phil. Date

School of Mathematical Sciences

Committee Member

Michael Cromer, Ph.D. Date

School of Mathematical Sciences

Director of M.S. Program in Applied and

Computational Mathematics

Abstract

The financial markets have been affected by the recent advancements in social media. Retail traders often use social media as a source of financial information. This thesis provides a mathematical proof of concept for how retail traders on social media can cause a large change in market price and that it is possible to identify the factors that drive the price change before it is seen in the market. An Agent Based Model is used to simulate a self-contained artificial market on a network. Traders gather pricing information from social media posts created by traders on the network and their own pricing techniques. Trader demographics and network structures are varied to study their results. A network with trader demographics of predominately momentum traders has a threshold corresponding to a large change in market price. The threshold, first discovered in this thesis, is denoted as the Herd Threshold. There is a negative relationship between network clustering and the magnitude of the price change. A convolutional neural network is used to predict trader demographics and network structure. The convolutional neural network has greater success when classifying trader demographics than network structure. This thesis is a foundation for multiple avenues of future work.

Contents

1 Introduction **1**

2 Simulating the Market **2**

2.1 Market Dynamics 2

2.2 Agents 3

2.3 Perceived Price 4

2.4 Fundamental Trader 6

2.5 Momentum Trader 8

2.6 Network Structures 10

3 Example Simulation **12**

4 The Spread of Pricing Information **19**

4.1 Agent Activation and Interaction 19

4.2 Perceived Price Spread 20

5 Impact of Trader Demographics **23**

6 Impact of Network Structure **26**

7 Inferring Market Conditions **30**

7.1 The Convolutional Neural Network 30

7.2 Training and Results 32

8 Conclusion **38**

9 Future Work **39**

10 Glossary **i**

11 Appendix **ii**

11.1 Model Summary ii

11.2 Code iii

1 Introduction

In recent years, the financial markets have been influenced by the rise of social media platforms [13]. The increase in social media use offers a wider access to information for all traders [13]. Institutional investors have many ways to attain financial information that are not always accessible to retail traders [3]. Social media offers the retail investor opportunities to access financial information [3]. Risks in trading financial securities exist to all investors and may increase for those not accredited but the U.S. Securities Exchange Commission. (Refer to the glossary in the appendix for definitions of technical financial terms.) According to the Securities Exchange Commission, there are numerous criteria that qualify an individual or entity as an accredited trader, including but not limited to sufficient financial net worth or sufficient financial knowledge gained through accreditation [12]. The criteria will not be thoroughly described here. This thesis refers the reader to the U.S. Securities Exchange website for an in depth definition. The retail trader is an individual trader who is not associated with a financial entity and often times a non-accredited. In fact, retail traders often confuse speculation with investing and are ignorant to their own pitfalls. Retail traders are more susceptible to cognitive bias and which causes them to make irrational financial decisions [1]. Most, if not all, financial models and theories assume that investors behave rationally. However, retail traders are not always rational market participants but exist in financial markets. As retail traders become more prevalent, reliance on fundamental financial models and theories decline and risk increases [4]. Social media can be utilized by retail traders for information and investment decision making. A motivator for this thesis was the GameStop Corp. ($\tilde{}$ticker GME) stock during the month of January 2021. Its price increased from \$16 to \$347 per share during this period. A network of traders communicated on the social media platform, Reddit, to placed similar trades coinciding with the increase in share price [13]. The retail traders were trying to prove to the large institutional investors the "power of the crowd" [13]. Prior to the network activities, GameStop was considered to be a financially weak company as reflected by stock price relative to its financial fundamentals. Daily volume and share price levels were low. However, traders on Reddit demonstrated that share price can increase beyond traditional financial fundamentals. Since GameStop, there have been other securities that have followed a similar pattern including AMC $\tilde{}$(ticker AMC) and Bed Bath and Beyond $\tilde{}$(ticker BBBY).

This thesis provides a mathematical proof of concept for how retail traders on social media can cause a large change in price and that it is possible to identify the factors that drive the price change before it is reflected in the market. An Agent Based Model is used to simulate a market environment. Section 2 describes the forward model for market dynamics along with two types of traders (agents) and their trading behavior. Agents are placed on two different network structures and allowed to interact, i.e., read each other's postings. Section 3 provides a step-by-step example simulation to demonstrate the market dynamics. Section 4 provides analysis on the flow of information. In sections 5 and 6, the impact of trader demographics and network structure are explored. Section 7 describes the inverse model using a convolutional neural network to predict the trader demographics and network structure.

2 Simulating the Market

The model developed in this thesis has two areas of focus. First, it captures the impact of trader demographics on a market. Second, this model examines how network structure impacts a market. An Agent Based Model is implemented to simulate an artificial trading environment. The python library Mesa is used a framework for the Agent Based Model [8]. The three main components of this model are the market price formation, traders, and network structures. Market price and trader attributes are kept consistent as the network structures and trader demographics are varied to study their consequences. Among various definitions of traders, this model is limited to two types; a fundamental trader and momentum trader. In this section, the market dynamics, traders, and network structures are defined and explained.

2.1 Market Dynamics

This model assumes the market to have adequate liquidity provided by an undefined market maker allowing buy and sell orders to be filled instantaneously. The model also assumes no transaction costs, consistent with most online brokerages' zero transaction fees for market orders on stocks. Noise traders are likely to be retail traders acting independently through an online broker. The only trade orders that are made are market orders, there are no limit orders, stop-loss orders, etc.. This model also assumes fractional shares can be traded. This model works with discrete indexing as traders do not act continuously and the market is not updated continuously. The index variable, t, should not be thought of in standard units of time. The indexing represents a discrete interval indicating event execution, i.e., from t to $t+1$ agents are activated to execute their tasks and the market is updated. Thus, this is an event-based simulation.

The price of an asset S_t is determined by the linear price formation rule [11]. The idea behind linear price formation is that the market price changes based on the excess demand with respect to the amount of liquidity in the market. The market price is given by,

$$S_{t+1} = S_t + \frac{\Theta_t}{\lambda}, \tag{1}$$

where S_t is the market price at time t, Θ_t is the order imbalance, λ is a market liquidity constant [11]. Price has units of dollars per share, the order imbalance has units of dollars, and market liquidity is in shares.

At each time step agents make the decision to place a market order. All the agents' market orders are then summed up to produce the excess market demand. For the ith agent that decides to trade at time step, t, the excess market demand is,

$$\Theta_t = \sum_{i=1}^{N} \theta_{i,t}, \tag{2}$$

where θ_i is the ith agent's market order and N is the total number of agents. Each market order is defined by a quantity of shares bought at the current market price. The units for market orders are in dollars.

$$\theta_{i,t} = \pm q_{i,t} S_t, \tag{3}$$

The quantity of shares the ith agent trades is proportional to their wealth, $\omega_{i,t-1}$. The wealth is then normalized by the market price to give a quantity of shares that can be bought [6].

$$q_{i,t} = \alpha_{i,t} \frac{\omega_{i,t-1}}{S_t}, \tag{4}$$

The value, $\alpha_{i,t}$, is the absolute relative difference between the trader's perceived price and the current market price. The perceived price will be explained in greater detail in section 2.3.

$$\alpha_{i,t} = \frac{|p_{i,t} - S_t|}{S_t}, \tag{5}$$

Wealth, $\omega_{i,t}$, has units of dollars, market price has units of dollars per share, and $\alpha_{i,t}$ is unit-less. Therefore, an agent's quantity has units of shares. It is now evident that the units of the liquidity constant, λ, are in shares to ensure that equation (1) is a valid equation. The liquidity constant is set to the number of traders on the network [5]. Note that Equation (3) can be simplified and written as, $\theta_{i,t} = \alpha_{i,t}\omega_{i,t}$ but this simplification will not be made. Traders first calculate a quantity of shares based on Equation (4) which is then sent to the market maker as a market order. The quantity of shares must be calculated before the market order is calculated or else the market orders are incorrect. See Appendix for all model equations. Section 2 demonstrates an example simulation where all dynamics are explicitly explained.

2.2 Agents

Agents are traders and will be referred to interchangeably throughout this thesis. Agents are placed on a network, one agent per node, and allowed to communicate with any adjacent nodes. For this model, communication means a one way conversation, e.g., a person on a social media platform like Reddit or Twitter reading a post. The person who tweets or makes the Reddit post does not gain any information from the person who reads the post therefore, it is a one way conversation. Each each trader has a portfolio, perceived stock price, and trading rule. Every trader has a position, $x_{i,t}$, that is the value of the number of shares the agent bought or shorted in units of dollars. Their portfolio's consist of their cash, position, and wealth. Giving each trader a portfolio allows for explicit bookkeeping and market price updates. A trader's wealth is the sum of his cash, $c_{i,t}$, and the value of their position

$$w_{i,t} = c_{i,t} + |x_{i,t}|. \tag{6}$$

Each agent starts with an identical cash amount. At each time step, each agent has a probability of being activated, i.e., read a post and/or place a trade. If an agent wants to place a trade, but does not have enough cash to afford that trade then no trade will be placed. In other words, there is no trading on margin. If an agent's wealth goes to zero, they are taken out of the market and not allowed to talk or trade.

Each agent's activation is a stochastic process that can be written

$$\{A_{i,t}, t \in T\}$$

for the ith agent with t being time and activation variable $A_{i,t}$. This random variable, $A_{i,t}$, is binary with $A_{i,t} = 1$ indicating the agent is activated and $A_{i,t} = 0$ indicating the agent is not activated. An agent is activated with a probability from 0 to 1. There exists a set that contains all agents, $I = \{a_1, a_2, ..., a_N\}$. This set is referred to as the scheduler. If $A_{i,t} = 1$ then the ith agent is pulled from the set I to be activated.

Each agent, a_i, has a set of adjacent nodes which indicates who a_i is connected to. Each agent's set of adjacent nodes can be defined as

$$J_i = \{a_k, a_l, a_m, ...\}$$

where $k, l, m \in \mathbb{N}$. All agents have an equal chance of being chosen for activation. Once agent a_i is activated, another agent, a_j, is randomly selected from I. If this randomly selected agent is in the set, J_i, then, a_j is a neighbor and a_i reads a_j's online post.

2.3 Perceived Price

Traders on a network communicate about the price of a stock. The way in which traders interpret postings about stock prices can be modeled similarly to rumor propagation. There are two factors that influence how agents interpret postings about stock prices; their group membership and their wealth. In rumor propagation, a person's group membership influences their credibility [2]. People of the same group place more trust in members of their own group compared to members of another group [2]. This idea from rumor propagation is adopted in a trader's perceived price. Group membership is denoted, g_i. Group membership is defined by the agent's trading strategy. This thesis models two trading strategies, the fundamental strategy and the momentum strategy. These strategies will be further explained in subsequent subsections. With two groups of traders, there are three possible online interactions; (i) momentum and fundamental, (ii) momentum and momentum, and (iii) fundamental and fundamental. The activated agent, a_i, is the only person who updates their perceived price during an interaction. They actively seek a post made by some other trader, a_j, while the other trader, a_j, does not know a_i is reading the post. Trader, a_i, changes their perceived price value by by gaining the pricing information posted by trader, a_j. This leads to four perceived price combinations that determine how trader a_i's perceived price is updated.

1. Neither agent has heard of the stock so no pricing information is passed and perceived prices remain zero. Agent, a_i, has no opinion on the stock and reads a post made by agent, a_j, that has nothing to do with the stock

   $$(p_{i,t} = 0 \wedge p_{j,t} = 0) \Rightarrow p_{i,t+1} = 0.$$

2. Agent a_j has heard of the stock but agent a_i has not therefore, agent a_i takes agent a_j's perceived price

$(p_{i,t} = 0 \wedge p_{j,t} \neq 0) \Rightarrow p_{i,t+1} = p_{j,t}$.

3. Agent a_i has heard of the stock but agent a_j has not. Agent, a_i, gains no pricing information from reading agent, a_j's post. Agent a_i's perceived price is updated based on their pricing analysis, $E_{i,t}$

   $(p_{i,t} \neq 0 \wedge p_{j,t} = 0) \Rightarrow p_{i,t+1} = E_{i,t+1}$.

4. Both agents have heard of the stock. The perceived price of agent a_i at time step $t+1$, is a weighted linear combination of agents a_i and a_j's perceived price at the previous time step plus agent a_i's own pricing analysis, E_{t+1}. Agent a_i must decide how much they want to incorporate agent a_j's perceived price. This is determined by a confidence probability explained below

   $(p_{i,t} \neq 0 \wedge p_{j,t} \neq 0) \Rightarrow p_{i,t+1} = \nu_i \left(\gamma_{i,t} p_{i,t} + (1 - \gamma_{i,t}) p_{j,t}\right) + (1 - \nu_i) E_{i,t+1}$.

The parameter, ν_i, is a weighting that determines how much the information heard on the network impacts the perceived price, $p_{i,t+1}$.

The confidence probability, $\gamma_{i,t}$, is dependent on group membership which forces two cases, one for intragroup interactions and one for intergroup interactions. The first case is when traders belong to the same trading group. If agents are of the same group then, $\gamma_{i,t}$ is the relative difference between both agents' wealth. In rumor propagation, credibility influences the belief of the rumor [2]. In a trading environment, a trader's wealth is a measure of their credibility. A trader places trades determined by their perceived price. If a trader's perceived price is correct then their wealth will increase. Therefore, a wealthy trader will have an accurate perceived price and a strong credibility. Agents' interactions are random which means that there is a chance that they will keep their perceived price or take another agent's perceived price. The odds that an agent a_i, keeps their perceived price are equal to the relative difference between a_i's wealth and a_j's wealth. The log odds that an agent will keep their perceived price is equal to and the relative difference in wealth, π, times a scaling factor β_1, plus a constant β_0. The log is taken to ensure that the odds cannot be negative.

$$\begin{aligned}
\ln\left(\frac{\gamma_{i,t}}{1-\gamma_{i,t}}\right) &= \beta_0 + \beta_1 \pi \\
\frac{\gamma_{i,t}}{1-\gamma_{i,t}} &= e^{\beta_0 + \beta_1 \pi} \\
\gamma_{i,t} &= e^{\beta_o + \beta_1 \pi} - \gamma_{i,t} e^{\beta_0 + \beta_1 \pi} \\
\gamma_{i,t} &= \frac{e^{\beta_0 + \beta_1 \pi}}{1 + e^{\beta_0 + \beta_1 \pi}}
\end{aligned}$$

Traders of different groups have less trust in one another's investment strategies. This means traders have little to no confidence that their counter-group's wealth is a reflection of their investment strategy. In heterogenous group interactions the confidence probability, $\gamma_{i,t}$, should not be dependent on wealth. The confidence probability takes on a fixed value from 0.5 to 1 since the trader believes their own perceived price is more accurate than their counter-group trader's. When $g_i \neq g_j$, the confidence probability is chosen to be 0.75 because it is greater than 0.5 and halfway between 0.5 and 1. Throughout this paper, $\beta_0 = 0$ and $\beta_1 = 1$.

The perceived price and confidence probability can be written as two piecewise functions,

$$p_{i,t+1} = \begin{cases} 0 & (p_{i,t} = 0 \wedge p_{j,t} = 0) \\ p_{j,t} & (p_{i,t} = 0 \wedge p_{j,t} \neq 0) \\ E_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} = 0) \\ \nu_i \left(\gamma_{i,t} p_{i,t} + (1 - \gamma_{i,t}) p_{j,t}\right) + (1 - \nu_i) E_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} \neq 0) \end{cases} \tag{7}$$

$$\gamma_{i,t} = \begin{cases} \frac{\exp(\beta_0 + \beta_1 \pi)}{1 + \exp(\beta_0 + \beta_1 \pi)} & g_i = g_j \\ 0.75 & g_i \neq g_j \end{cases} \tag{8}$$

2.4 Fundamental Trader

The trader's group determines how they place trades and update their perceived price. Fundamental traders use fundamental financial analysis to gain information about stock value. For the purpose of this model the means by which the fundamental trader determines stock value is given by an exogenous signal [5]. The fundamental value is modeled by a random walk,

$$V_{i,t+1} = V_{i,t} + \zeta_{i,t+1}, \qquad V_{i,0} = v_i \tag{9}$$

where $V_{i,t}$ is the fundamental value at time, t, v_i is a constant that is specific to each fundamentalist to account for variability in fundamental value, and $\zeta_{i,t+1}$ is sampled from a normal distribution with a mean of zero and constant variance, σ_v. The agent specific value constant, v_i, is randomly chosen from a uniform distribution, $v_i \sim U[-10, 10]$. The fundamentalist's perceived stock price is given as,

$$p_{i,t+1} = \begin{cases} 0 & (p_{i,t} = 0 \wedge p_{j,t} = 0) \\ p_{j,t} & (p_{i,t} = 0 \wedge p_{j,t} \neq 0) \\ V_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} = 0) \\ 0.25 \left(\gamma_{i,t} p_{i,t} + (1 - \gamma_{i,t}) p_{j,t}\right) + 0.75 V_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} \neq 0) \end{cases} \tag{10}$$

The parameter, ν_i, allows the perceived price to be weighted more heavily on online postings or their financial analysis. A fundamental trader is less likely to be influenced by online postings therefore, their perceived

price should have less emphasis on those postings. For a fundamental agent the parameter, ν_i, is set to 0.25 so that the agent's perceived price is mostly reliant on their fundamental analysis. Each agent has a position, $x_{i,t}$. This position indicates if they bought or sold shares and how much it cost, $x_{i,t} = \pm q_{i,t} S_t$. A fundamental agent's position can be in three different states: long, short, or empty. The agent will change their position if the stock is under or overvalued. If the stock is undervalued, they will enter a long position and if the stock is overvalued the agent will enter a short position [5]. In other words, if their perceived price is greater than the stock price, they will buy. If their perceived price is less than the stock price, they will sell.

Agents can only move one state at a time each time step. To get from a long position to a short position, the agent must pass through the empty state. This means if they have a long position and the stock price is greater than their perceived price, they will sell their shares and return to the empty state before short selling to get to a short position.

$$p_{i,t} > S_t \;\; \& \;\; x_{i,t-1} = 0 \Rightarrow x_{i,t} = +q_{i,t} S_t$$
$$p_{i,t} < S_t \;\; \& \;\; x_{i,t-1} = 0 \Rightarrow x_{i,t} = -q_{i,t} S_t$$

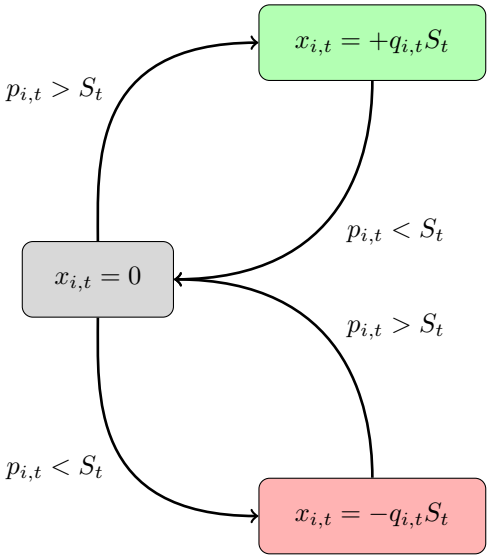


Figure 1: The fundamental trader's position state diagram.

2.5 Momentum Trader

Momentum agents behave differently than fundamental agents with respect to their trading rule. Instead of using fundamental financial analysis, momentum traders follow trends in the stock price [5]. A momentum agent believes that the stock price at the current time step is dependent on a window of previous stock prices. For example, if the price has been rising in the past five days then, they believe it will continue to rise on the sixth day. A common and basic tool that momentum traders use is a moving average over a specified number of time steps. The moving average is given as,

$$M_t = \frac{1}{T}\sum_{k=1}^{T} S_{t-(T-k)} \tag{11}$$

with $T = 20$ as the window of previous stock prices. The perceived price then becomes a weighted linear combination of the moving average and interactions on the social network.

$$p_{i,t+1} = \begin{cases} 0 & (p_{i,t} = 0 \wedge p_{j,t} = 0) \\ p_{j,t} & (p_{i,t} = 0 \wedge p_{j,t} \neq 0) \\ M_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} = 0) \\ 0.5\left(\gamma_{i,t} p_{i,t} + (1-\gamma_{i,t}) p_{j,t}\right) + 0.5 M_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} \neq 0) \end{cases} \tag{12}$$

A momentum agent's position can also be in three different states: long, short, or empty. They will take a long position when the stock price is greater than their perceived price, and a short position when the stock price is less than the perceived price [5]. Momentum agents can only move one state per time step, just like the fundamental agents.

$$p_{i,t} < S_t \;\; \& \;\; x_{i,t-1} = 0 \Rightarrow x_{i,t} = +q_{i,t} S_t$$
$$p_{i,t} > S_t \;\; \& \;\; x_{i,t-1} = 0 \Rightarrow x_{i,t} = -q_{i,t} S_t$$

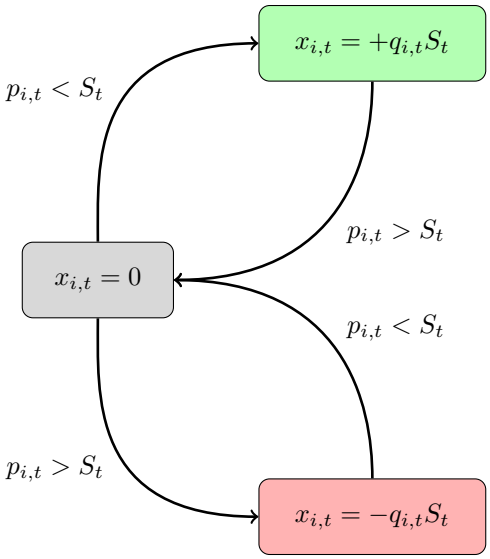
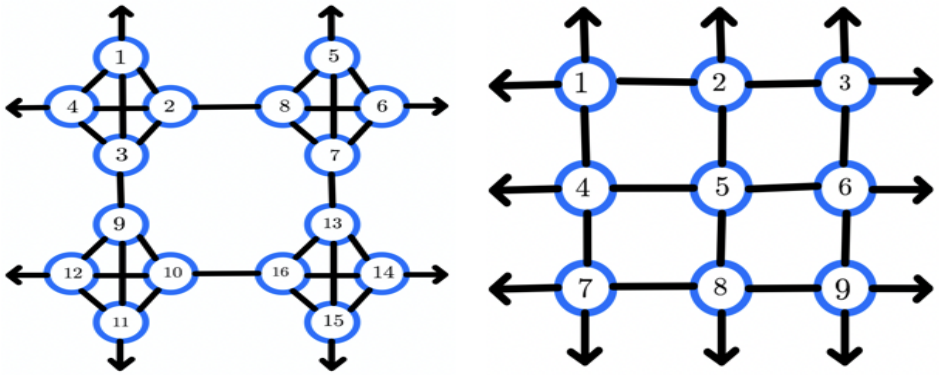


Figure 2: The momentum trader's position state diagram.

2.6 Network Structures

Two network structures are used to model social media networks. Nodes represent traders and edges represent the communication relationships between traders, (i.e., who follows whom). The two network structures used are the Family network and the Torus network. Both networks are regular networks, which means that each node has exactly the same number of edges. Regular networks are used so that it does not matter where on the network the traders are placed as long as the distribution of nodes is consistent. A Torus network is a 2D Lattice with the boundaries wrapped around and connected forming a 3D Torus or "donut". The Family network is a collection of connected complete subgraphs of four nodes. Similar to the Torus, the boundaries are wrapped around and connected forming a 3D "donut". Each family, or complete subgraph, has four nodes with each node connected to all other nodes in the family. Each node in the family also has one edge to another family. The Family network can be thought of as a 2D Lattice but there are complete subgraphs at each node of the 2D Lattice.

Both networks are regular with connected boundaries to ensure that information flows uniformly on each network. In both networks all nodes have degree four. The Torus network has a clustering coefficient of zero while the Family network has a clustering coefficient of 0.5. The clustering coefficient can be defined as the probability that two neighboring nodes are connected. Clustering is the main difference between the networks. Agents are placed in such a way that all of the momentum traders are grouped together and all of the fundamental traders are grouped together. This means there are two boundaries where the two groups meet with heterogeneous group edges. The Torus network has four times more heterogenous group connections than the Family network. These networks are unrealistic representations of a real trading environment but offer a controlled environment to understand how network attributes impact the market.



(a) Family Network Structure (b) Torus Network Structure

Figure 3: Small scale two dimensional visual representations of the Torus and Family networks. The boundary nodes wrap around and are connected to the nodes on the other side. For example, looking at the Torus network, nodes 1, 4, and 7 connect to nodes 3, 6, and 9, respectively. Likewise, nodes 1, 2, and 3 connect to nodes 7, 8, and 9, respectively. This forms the three dimensional Torus.

3 Example Simulation

A small scale simulation is run to see how the agent based market works. A network of four traders is created each with two connections. There are two fundamental traders and two momentum traders. Four traders are a part of an online social community such that each trader "follows" two other traders. One trader starts to post about stock XYZ and the other agents on the network take interest.

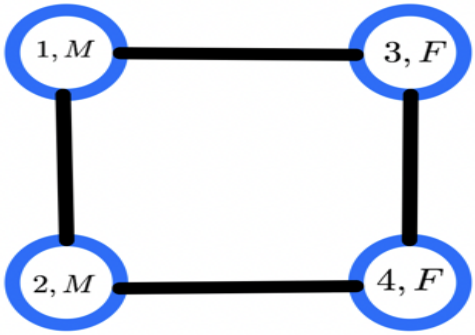


Figure 4: The network structure described above with two momentum and two fundamental traders.

Nodes represent traders and edges represent whether two traders follow each other. The momentum traders are at nodes 1 and 2 while fundamental traders are at nodes 3 and 4. To start the simulation, one trader is seeded with a nonzero perceived price. This means at $t = 0$ all traders except one have never heard of the stock and as the simulation runs the perceived price is spread around the network. Here, trader 1 takes their perceived price to be the market price of \$100 and the other three traders have perceived prices of zero. If agent a_i has not yet heard of the stock which means a perceived price of zero, and they read agent a_j's online post who has heard of the stock which means a nonzero perceived price, agent a_i will take the perceived price of agent a_j.

Each agent's set of adjacent nodes are given below.

$$J_1 = \{a_2, a_3\}$$
$$J_2 = \{a_1, a_4\}$$
$$J_3 = \{a_1, a_4\}$$
$$J_4 = \{a_2, a_3\}$$

The parameters used for this simulation are listed below in table 1: the market price, S_0, liquidity constant, λ, each trader's starting cash, $c_{i,0}$, and the 20 day moving average, M_t.

S_0	λ	$c_{i,0}$	M_t
\$100	4	\$1000	\$101

Table 1: Parameters used in the example market simulation.

At t = 0.

Network and market are initialized. Agent 1 starts with a perceived value of 100 and all other agents have perceived prices of 0.

No trades are placed therefore, $S_0 = 100$.

Agent 1 (M)	Agent 2 (M)	Agent 3 (F)	Agent 4 (F)
$p_{1,0} = 100$	$p_{2,0} = 0$	$p_{3,0} = 0$	$p_{4,0} = 0$
$\theta_{1,0} = 0$	$\theta_{2,0} = 0$	$\theta_{3,0} = 0$	$\theta_{4,0} = 0$
$\omega_{1,0} = 1000$	$\omega_{2,0} = 1000$	$\omega_{3,0} = 1000$	$\omega_{4,0} = 1000$

t = 1

Below are the agent attributes at the beginning of step one. N.B., they are the same as the the attributes from the end of step zero.

Agent 1 (M)	Agent 2 (M)	Agent 3 (F)	Agent 4 (F)
$p_{1,0} = 100$	$p_{2,0} = 0$	$p_{3,0} = 0$	$p_{4,0} = 0$
$\theta_{1,0} = 0$	$\theta_{2,0} = 0$	$\theta_{3,0} = 0$	$\theta_{4,0} = 0$
$\omega_{1,0} = 1000$	$\omega_{2,0} = 1000$	$\omega_{3,0} = 1000$	$\omega_{4,0} = 1000$

The market orders from the last time step get filled and the market price is updated, (no trades were made therefore, the market price doesn't change).

$$S_1 = 100$$

$$A_{1,1} = 1, \quad A_{2,1} = 1, \quad A_{3,1} = 0, \quad A_{4,1} = 0$$

Agent 1 is randomly chosen to be activated, $A_{1,1} = 1$, and agent 1 follows agents 2 and 3 on a social media site, $J_1 = \{a_2, a_3\}$. Agents 2 and 3 have an equally likely chance of being picked from the set J_1. Agent 2 is picked and agent 1 reads agent 2's post.

$$p_{1,0} = 100, \qquad p_{2,0} = 0$$

Since agent 2 has not yet heard of the stock their post didn't contain any information about the stock. This means nothing about the traders' interaction impacted agent 2's perceived price. Agent 1 will only use their trading edge to update the perceived price. Agent 1 is a momentum agent and the moving average, (which was initialized before the simulation started), is at 101. Agent 1's perceived price becomes the moving average.

$$\begin{aligned} p_{1,1} &= M_0 \\ &= 101 \end{aligned}$$

Agent 1 sees that their perceived price is greater than the market price and, being a momentum trader, the agent shorts the stock. First agent 1 needs to calculate the amount of shares they want to short.

$$q_{1,1} = \alpha_{1,1}\frac{\omega_{1,0}}{S_1}, \qquad \alpha_{1,1} = \frac{p_{1,1} - S_1}{S_1}$$

$$q_{1,1} = 0.01\frac{1000}{100} = 0.1$$

Now the agent sends a market order to sell 0.1 shares at \$100 per share. Since the agent is selling the shares the market order, $\theta_{1,1}$, must be negative.

$$\begin{aligned} \theta_{1,1} &= \pm q_{1,1} S_1 \\ &= -0.1(100) \\ \theta_{1,1} &= -10 \end{aligned}$$

Since agent 1 shorted 0.1 shares their position will be, $x_{1,1} = -10$. The agent's wealth then becomes $w_{1,1} = 1000 + 10$, (the agent's cash plus the *value* of the agent's assets). When a trader short sells a stock, they are selling stock that they don't yet own. They first get the cash amount for the sell and then at some time in the future they need to buy back those shares to actually give to the buyer.

Now consider agent 2's activation during this time step.

$$A_{2,1} = 1, \qquad J_2 = \{n_1, n_4\}$$

Agent 2 reads agent 1's online post.

$$p_{2,0} = 0, \qquad p_{1,0} = 100$$

Then agent 2's perceived price becomes agents 1's at the previous time step.

$$p_{2,1} = 100$$

Agent 2 places no trade. Trader attributes at the end of $t = 1$ are listed below.

Agent 1 (M)	Agent 2 (M)	Agent 3 (F)	Agent 4 (F)
$p_{1,1} = 101$	$p_{2,1} = 100$	$p_{3,1} = 0$	$p_{4,1} = 0$
$\theta_{1,1} = -10$	$\theta_{2,1} = 0$	$\theta_{3,1} = 0$	$\theta_{4,1} = 0$
$\omega_{1,0} = 1010$	$\omega_{2,0} = 1000$	$\omega_{3,0} = 1000$	$\omega_{4,0} = 1000$

t = 2

Agent 1 (M)	Agent 2 (M)	Agent 3 (F)	Agent 4 (F)
$p_{1,1} = 101$	$p_{2,1} = 100$	$p_{3,1} = 0$	$p_{4,1} = 0$
$\theta_{1,1} = -10$	$\theta_{2,1} = 0$	$\theta_{3,1} = 0$	$\theta_{4,1} = 0$
$\omega_{1,1} = 1010$	$\omega_{2,1} = 1000$	$\omega_{3,1} = 1000$	$\omega_{4,1} = 1000$

The market orders from the last time step get filled and the market price is updated.

$$\begin{aligned} S_2 &= S_1 + \frac{\Theta_1}{\lambda} \\ S_2 &= 100 - \frac{10}{4} \\ S_2 &= 97.5 \end{aligned}$$

During this time step, only agent 3 decides to check online postings.

$$A_{1,2} = 0, \quad A_{2,2} = 0, \quad A_{3,2} = 1, \quad A_{4,2} = 0$$

Agent 3 is activated, $A_{3,2} = 1$, with adjacent nodes $J_4 = \{n_1, n_4\}$.

$$A_{3,2} = 1, \qquad J_4 = \{n_1, n_4\}$$

Agent 3 reads a post about the stock made by agent 1. Agent 3 has never heard of the stock so they don't know any better and take agent 1's perceived price as their own.

$$\begin{aligned} p_{3,2} &= p_{1,1} \\ &= 101 \end{aligned}$$

Agent 3 is a fundamental trader and sees that the stock is undervalued, $S_2 < p_{3,2}$, therefore, agent 3 enters a long position.

$$q_{3,2} = \alpha_{3,2}\frac{\omega_{3,1}}{S_2}, \qquad \alpha_{3,2} = \frac{p_{3,2} - S_2}{S_2}, \qquad \theta_{1,1} = \pm q_{3,2} S_2$$
$$q_{3,2} = \left(\frac{101 - 97.5}{97.5}\right)\frac{1000}{97.5} \approx 0.37, \qquad \theta_{3,2} = +0.37(97.5) = 35.90$$

Agent 3's cash and wealth become, $c_{3,2} = 1000 - 35.90 = 964.1$, $\omega_{3,2} = 964.1 + 35.9 = 1000$. The value of agent 3's wealth did not change because they just converted some cash to stock, XYZ. Their wealth will change once the value of XYZ changes. This can be seen below with agent 1's wealth.

Agent 1 has an open position from the last time step which means their wealth needs to be updated since the price of XYZ changed. The agent shorted 0.1 shares and now the stock is at 97.5 dollars per share. Agent 1's position now becomes $x_{1,2} = -0.1(97.5) = -9.75$. If the agent were to close their position and buy back the shares they would make 25 cents.

At the end of $t = 2$, agent attributes are tabulated below.

Agent 1 (M)	Agent 2 (M)	Agent 3 (F)	Agent 4 (F)
$p_{1,2} = 101$	$p_{2,2} = 100$	$p_{3,2} = 101$	$p_{4,2} = 0$
$\theta_{1,2} = 0$	$\theta_{2,2} = 0$	$\theta_{3,2} = 35.90$	$\theta_{4,2} = 0$
$\omega_{1,1} = 1009.75$	$\omega_{2,1} = 1000$	$\omega_{3,1} = 1000$	$\omega_{4,1} = 1000$

$\vdots$

t = 50

Skip to $t = 50$ where all agents have been participating in market activity. Agent attributes from the previous time step are given below. Agents wealths have been updated from the previous time step.

Agent 1 (M)	Agent 2 (M)	Agent 3 (F)	Agent 4 (F)
$p_{1,49} = 105$	$p_{2,49} = 98$	$p_{3,49} = 101$	$p_{4,49} = 100$
$\theta_{1,49} = 0$	$\theta_{2,49} = 0$	$\theta_{3,49} = 0$	$\theta_{4,49} = 0$
$\omega_{1,49} = 1022$	$\omega_{2,49} = 1003$	$\omega_{3,49} = 984$	$\omega_{4,49} = 1050$

with market price $S_{50} = 102$ and moving average $M_{50} = 97$.

Agent 2 is checks social media and sees that agent 1 posted about stock XYZ. Both agents have

nonzero perceived prices.

$$A_{2,50} = 1, \qquad J_2 = \{n_1, n_4\}$$

$$p_{2,50} = \nu_2 \left(\gamma_{2,49} p_{2,49} + (1 - \gamma_{2,49}) p_{1,49}\right) + (1 - \nu_2) M_{50}$$

$$\pi = \frac{\omega_{2,49} - \omega_{1,49}}{\omega_{1,49}}, \qquad \gamma_{2,49} = \frac{e^{\pi}}{1 + e^{\pi}},$$
$$\pi \approx -0.019, \qquad \gamma_{2,49} \approx 0.495$$

$$p_{2,50} = 0.5 \left(0.495(98) + (1 - 0.495)105\right) + 0.5 * 97 \approx 99.27$$

Agent 2 decides to buy stock since $p_{2,50} < S_{50}$. Then,

$$\theta_{2,50} = \pm \left(\alpha_{2,50} \frac{\omega_{2,49}}{S_{50}}\right) S_{50}$$
$$\theta_{2,50} = + \left(\frac{|99.27 - 102|}{102} \frac{1003}{102}\right) 102 \approx 26.85$$

Since agent 2 bought stock they transfer cash to stock but their wealth is still 1003 because the market price has not been updated.

Now, agent 3 checks social media and sees that agent 1 posted about XYZ.

$$A_{3,50} = 1, \qquad J_3 = \{n_1, n_4\}$$

These agents are not the same trading type therefore, the confidence probability, $\gamma_{3,49}$, is set to 0.75. Agent 3 believes that agent 1's wealth is based off luck and doesn't think that agent 1's trading strategy actually made them money. Agent 3 will not consider how "good" of a trader agent 1 is by checking their relative wealth difference.

$$\begin{aligned} p_{3,50} &= \nu_3 \left(\gamma_{3,49} p_{3,49} + (1 - \gamma_{3,49}) p_{1,49}\right) + (1 - \nu_3) V_{3,49} \\ &= 0.25 \left(0.75(101) + 0.25(105)\right) + 0.75(104) \\ p_{3,50} &= 103.5 \end{aligned}$$

Agent 3 decides to buy since they think the stock is undervalued, $p_{3,50} > S_{50}$.

$$\theta_{3,50} = \pm \left(\alpha_{3,50} \frac{\omega_{3,49}}{S_{50}}\right) S_{50}$$
$$\theta_{3,50} = + \left(\frac{|103.5 - 102|}{102} \frac{984}{102}\right) 102$$
$$\theta_{3,50} \approx 14.47$$

Agent 1 (M)	Agent 2 (M)	Agent 3 (F)	Agent 4 (F)
$p_{1,50} = 105$	$p_{2,50} = 99.27$	$p_{3,50} = 103.5$	$p_{4,50} = 100$
$\theta_{1,50} = 0$	$\theta_{2,50} = 26.85$	$\theta_{3,50} = 14.47$	$\theta_{4,50} = 0$
$\omega_{1,49} = 1022$	$\omega_{2,49} = 1003$	$\omega_{3,49} = 984$	$\omega_{4,49} = 1050$

The purpose of this example simulation was to demonstrate the dynamics of the model developed in the previous section. Traders update their perceived prices with information gained from social media and their own pricing techniques. Traders decided to place trades based on their trading rules and the market price is updated. The network used in this example is not a Torus or a Family network and is not the same as the networks used in the subsequent sections.

4 The Spread of Pricing Information

Section 4 explores the how traders are activated, interact, and how information flows across the network. An interaction is when a trader reads another trader's online post. Subsection 4.1 shows that the number of interactions per index step is not dependent on the size of the network. Subsection 4.2 explains how the perceived price spreads across the network. These sections provide intuition for the results attained in sections five and six.

4.1 Agent Activation and Interaction

Consider how agents interact on the network. The activation is designed such that, all agents have an equal probability of being chosen for activity. There is no activation bias coming from agent placement on the network.

At each time step, the algorithm choses randomly from the scheduler, $I = \{a_1, a_2, ..., a_N\}$. Once agent a_i is chosen there is a probability that a_i is activated, $P(A_{i,t} = 1)$. An agent can be chosen from the scheduler but not activated. If the agent is not activated, then the agent does not trade or interact on the network. It is unrealistic that every trader is activated at every time step which is why the activation probability is introduced. Once an agent is selected from the scheduler and activated they look for a neighboring node, or follower, to interact with. Neighbor selection is not dependent on the activation of the first agent. In other words, when an agent is activated there is no bias towards who they select to interact with.

Once agent a_i is activated agent a_j is chosen from the scheduler set without a_i, $I^{(-a_i)}$, with probability $\frac{1}{N-1}$. In order for a_j to be an adjacent node it must be an element of the set J_i. Since each agent has the same chance of being chosen from the scheduler the probability that, $a_j \in J_i$, is $\frac{1}{N-1}$ times the number of elements in J_i. For the Family and Torus networks all nodes have four neighbors thus, $P(a_j \in J_i) = \frac{4}{N-1}$.

The probability that the ith agent has an interaction is their activation probability, $P(A_{i,t} = 1)$, times the probability that a neighbor is chosen, $P(a_j \in J_i) = \frac{4}{N-1}$.

$$P(Interaction) = \frac{4P(A_{i,t} = 1)}{N-1}$$

To find the average number of agents that interact at each index step, Δ, take the probability that the ith agent has an interaction and multiply by the number of agents in the scheduler,

$$\Delta = \frac{4P(A_{i,t} = 1)}{N-1} N$$

$$\Delta \approx 4P(A_{i,t} = 1)$$

This means that the number of interactions per index step is not dependent on the size of the network. The larger the network the more steps it will take for the perceived price to reach everyone. This makes sense, just because the size of the network increases does not mean the rate at which information flows increases. However, Δ is dependent on the agent's activation probability and the size of the agent's set of adjacent nodes. For all of the simulations in this paper every agent's activation probability is 0.5.

4.2 Perceived Price Spread

The network structure has full affect when all agents are interacting with non-zero perceived prices. This leads the researcher to find the number of index steps that it takes for all agents to have non zero perceived prices. This can be done empirically. Considering that the flow of perceived prices is only dependent on $P(A_{i,t} = 1)$ and the size of J_i, it is vital that these values remain fixed. Fix $P(A_{i,t} = 1)$, J_i, network structure, trader demographics, and seed one agent with a nonzero perceived price. Trader demographics are the population of trader groups.

N	$P(A_{i,t} = 1)$	Network	Demographics
64	0.5	Torus	1:1

Table 2: Perceived price spread simulation characteristics.

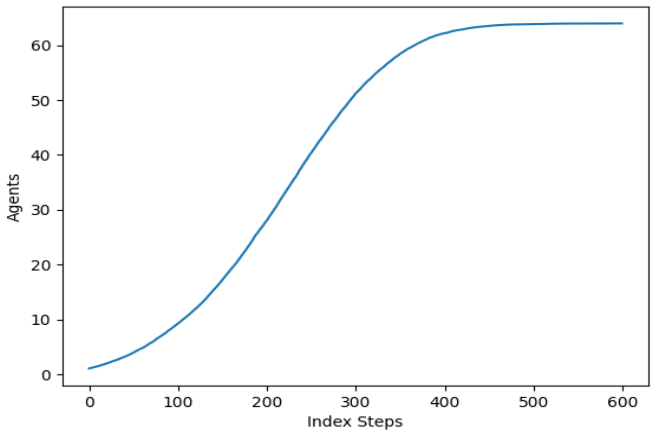


Figure 5: Average number of agents who have heard of the stock over 294 simulations. A Torus network with 64 agents and activation probability of 0.5.

The odds that a new person hears of the stock is a linear function of those who have already heard of the stock. The odds that a new person hears of the stock is dependent on the number of people who

have already heard of the stock. The odds increases linearly as more people hear of the stock. Figure 5 is a Monte Carlo simulation of the number of traders who have non zero perceived prices at each time step. The perceived price spreads the slowest at the beginning and end of the simulation when most trader have not heard of the stock and when most traders have heard of the stock. As more people hear of the stock, the rate of the perceived price spread increases but after a certain point, as the number of traders who have heard of the stock approaches the total network size the rate of the spread decreases. All of these characteristics can be described by a logistic differential equation. The spread of the perceived price on a network is similar to the spread of a virus where the traders who have heard of the stock are "infected" and those who have not heard of it are "susceptible". Often times, the spread of a virus follows the logistic curve. All of these reasons support the use of the logistic equation to model the spread of the perceived price.

Let R be the number of traders who have heard of the stock and N be the total number of traders on the network. For mathematical conveniences, assume the index step is sufficiently small such that the spread of the perceived price can be written continuously. Then the spread of the perceived price can be written in differential form.

$$\frac{dR}{dt} = rR\left(1 - \frac{R}{N}\right) \tag{13}$$

The differential logistic equation has solution,

$$R(t) = \frac{R_0 N}{(N - R_0)e^{-rt} + R_0} \tag{14}$$

with R_0 as the initial number of traders with nonzero perceived prices and r as the spread rate. To estimate the spread rate find the inflection point of equation 13 and substitute values from figure 2 into equation 14. The inflection point is found when the second derivative of equation 14 is equal to zero.

$$\frac{d^2R}{dt^2} = r - \frac{2rR}{N} \tag{15}$$

$$\begin{aligned} 0 &= r - \frac{2rR}{N} \\ \Rightarrow R &= \frac{N}{2} \end{aligned}$$

For the simulation in figure 2, the inflection point occurs at $R = 32$ and $t = 221$. Substitute $R = 32$ and $t = 221$ into equation 14 and solve for r.

$$\begin{aligned} 32 &= \frac{64}{63e^{-221r} + 1} \\ \Rightarrow r &\approx 0.0187 \end{aligned}$$

On average, the spread of perceived price on a network of 64 traders with activation probability of 0.5 is written below.

$$R(t) = \frac{64}{63e^{-0.0187t} + 1} \tag{16}$$

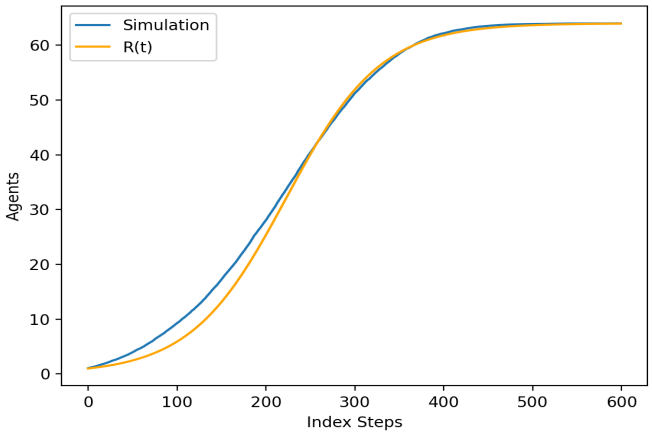


Figure 6: Parameterized logistic equation and simulation data for the perceived price spread on a Torus network of 64 traders.

To validate that the logistic equation is a good approximation for the perceived price spread plot the two curves together. Figure 6 shows that equation 16 approximates the data relatively well and can be used to determine the perceived spread throughout the simulation. A mathematical expression for how the perceived price spreads on the network is a powerful tool that can be used to understand and justify potential market behavior. Sections 5 and 6 show that once a percentage of the network has heard of the stock interesting market behavior occurs. The conclusions drawn from sections 5 and 6 would not be possible without the analysis done in this section.

5 Impact of Trader Demographics

Understanding how and why different trader types impact the market is valuable especially when an investor has the ability to interact with or influence traders. Two simulations are run each with different trader demographics. One simulation has a fundamental group majority with 75% of the network being fundamental traders. The other simulation has a momentum group majority of which 75% of the network are momentum traders. Traders are placed in an ordered fashion such that all momentum traders are grouped together and all fundamental traders are grouped together. This creates two group boundaries where the two group clusters meet. Most of the interactions between traders are intragroup since the groups are clustered together, e.g., a fundamentalist interacting with a fundamentalist. The intergroup interactions happen at the group boundaries where the fundamental group meets the momentum group.

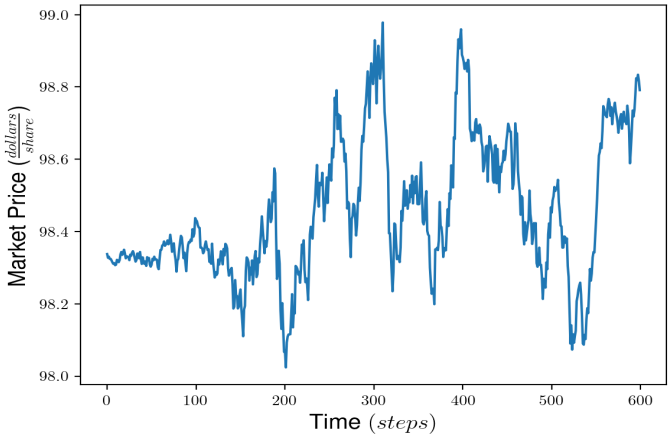


Figure 7: The market price generated on a Torus network where 75% of the 64 total traders are fundamentalists.

Figure 7 is a Monte Carlo simulation with 75% of the traders in the market being fundamental traders. The market price has a standard deviation of 0.192 dollars per share and never gets above 99 dollars per share or below 98 dollars per share. Relative to the market price with a momentum majority, (shown later in this section), the market price is stable with no extreme changes in price. The majority of fundamental traders provide stability to the market price. Fundamentalists are less influenced by the perceived price of other traders and make trades based on their fundamental financial analysis. Refer back to equation 10,

$$p_{i,t+1} = \begin{cases} 0 & (p_{i,t} = 0 \wedge p_{j,t} = 0) \\ p_{j,t} & (p_{i,t} = 0 \wedge p_{j,t} \neq 0) \\ V_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} = 0) \\ \nu_i \left(\gamma_{i,t} p_{i,t} + (1 - \gamma_{i,t}) p_{j,t}\right) + (1 - \nu_i) V_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} \neq 0) \end{cases}$$

ν_i is set to 0.25 which weights the trader's edge more heavily. Their perceived price stays true to their

fundamental financial analysis which causes the market price to reflect their fundamental valuation. The trading rule for fundamental traders is to buy when the stock is undervalued and sell when it is overvalued. Figure 7 demonstrates that the market price is mostly determined by the fundamentalists trading rule; they sell as the price rises to an overvalued level and buy when the price falls to an undervalued level. This keeps the price from moving too far in one direction and remains relatively stable.

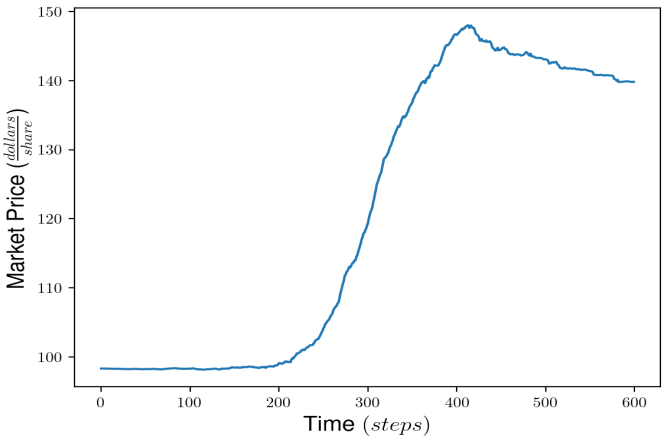


Figure 8: The market price generated by a Torus network with 64 total traders and 48 momentum traders.

Figure 8 is a Monte Carlo simulation with 75% of the traders in the market being momentum traders. Contrary to a market with a fundamental majority, the momentum majority causes a large change in price. Consider the trading rule for momentum traders, the momentum trader will buy when the price is greater than their perceived price and sell when the price is lower than their perceived price. This is the opposite of the fundamentalist strategy and causes the market price to move in one direction. The magnitude of the change in price is determined by the magnitude of the order imbalance, Θ_t, which is determined by the absolute relative difference between the traders perceived prices and the market price, $\alpha_{i,t}$.

Referring back to equation 1,

$$S_{t+1} = S_t + \frac{\Theta_t}{\lambda}$$

the order imbalance, Θ_t, is the sum of the market orders and a large Θ_t will cause a large move in S_{t+1}. Therefore, the jump in market price must be attributed to a large number of traders placing similar trades. If agents are placing similar trades then, their perceived prices must be similar. Referring back to the equations for a single market order,

$$\theta_{i,t} = \pm\left(\alpha_{i,t}\frac{\omega_{i,t}}{S_t}\right)S_t, \quad \alpha_{i,t} = \frac{|p_{i,t} - S_t|}{S_t}$$

if the agents have similar perceived prices then, they will have similar $\alpha_{i,t}$'s and place similar market orders, $\theta_{i,t}$. This means that as the perceived price spreads across the Torus network there is little variability between

agents' perceived prices. Most of the network sees the same perceived price which causes the traders to place the same trades. When traders imitate the investment decisions of others instead of relying on their own analysis it is known as herding [7]. Around index step 200, the market price begins to accelerate, i.e., the market price is growing at a faster rate than previously, and implies that some threshold has been crossed.

It is important to note that the algorithm generating this data drops simulations where the market price goes to zero. When the market price goes to zero, the company goes out of business and is no longer tradable. This means that figure 8 is only showing the upside price swings. For this reason, no conclusions about the *direction* of price movements can be drawn. It is not valid to claim that momentum traders will drive the price up, despite figure 8 showing a large upside move. However, the simulations that go to zero, (not shown here), have a price drop at the same index step that the price jumps in figure 8. Therefore, it is valid to claim that the momentum traders cause a large price change, up or down.

Using time steps to indicate the threshold does not provide a useful metric to compare across network sizes and structures. At time step $t = 200$, the price jumps on a Torus network of 64 traders with a momentum trader majority, but the price jump will not be at the same index step if the size of the network or structure is changed. This leads the trader to find a better indication for price jump which can be found in the perceived price spread. As previously mentioned, the change in price is driven by the order imbalance which is driven by the perceived price. Therefore, the threshold should be based off of the perceived price. Recall equation 16 for the spread of the perceived price across the network.

$$R(t) = \frac{64}{63e^{-0.0187t} + 1}$$

The inflection point for equation (16) is at $t = 221$ index steps while the price jump in figure 8 is at $t = 200$. This illustrates that the price jumps when a little less than half of the traders have heard of the stock. To find the number of traders who have heard of the stock at $t = 200$, use equation 16.

$$R(200) = \frac{64}{63e^{-(0.0187)200} + 1}$$

$$R(200) \approx 25$$

This means that once 25 traders, 40% of the network, have heard of the stock, there is about to be a large change in price. Denote the percentage of the network, who have heard of the stock when there is a significant change in price, as the Herd Threshold. To make substantial gains in a market on a Torus network with a momentum trader majority, the trader should enter before the Herd Threshold of 40% is reached.

6 Impact of Network Structure

This section analyzes the impact of the network structure on the market. To test the impact of network structure simulate a market on a Torus network and a Family network then compare results. First, the rate at which the perceived price spreads on a Family network must be determined. A Monte Carlo simulation is run with all of the same values as section 4.2 except the network will be a Family network.

N	$P(A_{i,t} = 1)$	Network	Demographics
64	0.5	Famiy	1:1

Table 3: Perceived price spread simulation characteristics.

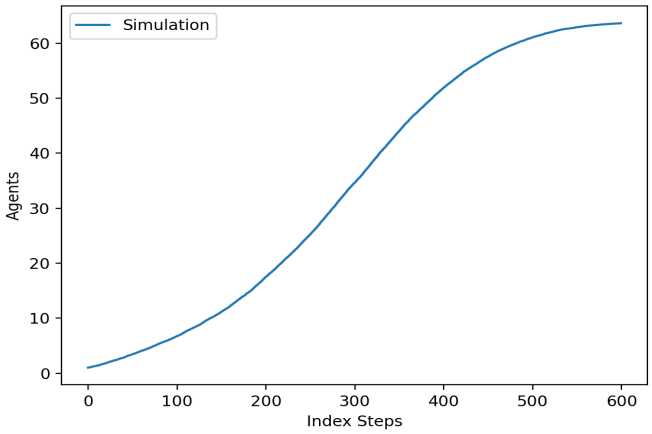


Figure 9: The number of agents who have heard of the stock averaged over 294 simulations. A Family network with 64 agents and activation probability of 0.5.

The inflection point and the rate of spread, r, must be found to construct the equation for perceived price spread. According to the data, the inflection point happens at $R = 32$ and $t = 286$. The spread rate can then be calculated using equation 14 and the inflection point.

$$\begin{aligned} R(t) &= \frac{R_0 N}{(N - R_0)e^{-rt} + R_0} \\ 32 &= \frac{64}{63e^{-286r} + 1} \\ \Rightarrow r &\approx 0.0145 \end{aligned}$$

For a Family network of 64 traders and activation probability of 0.5, the number of agents who have heard of the stock at a given index step is given below.

$$R(t) = \frac{64}{63e^{-0.0145t} + 1} \tag{17}$$

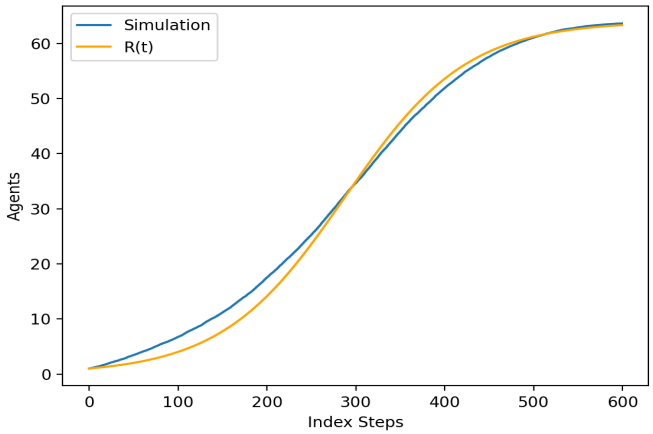


Figure 10: Parameterized logistic equation and Monte Carlo simulation for the perceived price spread on a Family network of 64 traders.

Again, the logistic equation is a relatively good fit for the Monte Carlo simulation and will be used to determine the Herd Threshold for a Family network.

Figure 10 compares the perceived price spread on the Family network with the spread on the Torus network.

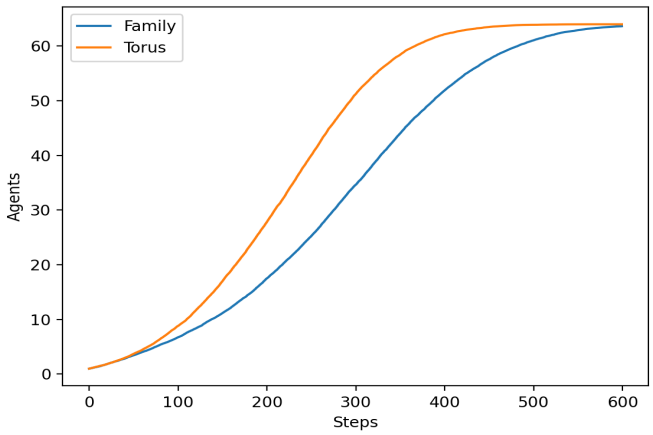


Figure 11: The average spread of perceived price on a Torus network and a Family network over 294 simulations. Each simulation has 64 traders and all agents have activation probabilities of 0.5.

Network	r	$t_{inflection}$
Family	0.0145	286
Torus	0.0187	221

Table 4: Difference between the Torus network and Family network's perceived price spread.

The spread rate, r, is approximately 28% greater on the Torus network than on the Family network. Figure 12 compares the Family network's market price to the Torus network's market price with a fundamental trader majority. A Family network has clustering coefficient of 0.5 while the Torus network has a clustering coefficient of zero. The clustering on a Family network increases the chances that a trader will interact with someone in their cluster. Consider four nodes in a Family network and four nodes in a Torus network. The four nodes in a Family network are more connected to each other than the four nodes in the Torus network. This means that when traders are interacting on the Family network there is more of a chance that they will interact with the same traders multiple times. Instead of the perceived price moving across the network it tends to stay within the clusters and it takes longer for the perceived price to spread to new traders. The perceived price gets "trapped" in the cluster and all of the traders in the cluster are caught in a feedback loop reinforcing their undiversified perceived price.

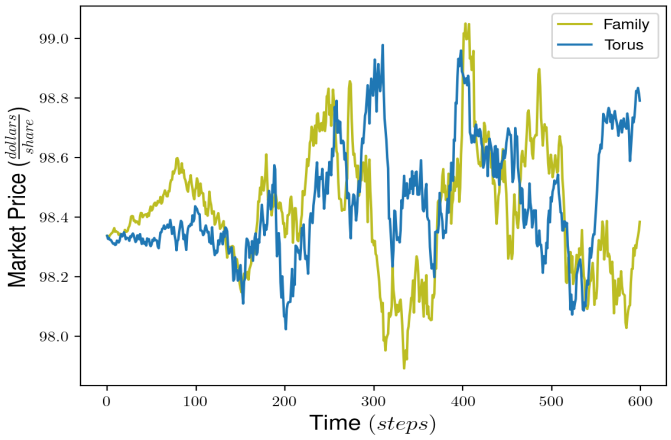


Figure 12: The market price on the Torus and Family networks with a trader demographic that is majority fundamentalists.

Figure 12 is a Monte Carlo simulation and shows that there is not much difference between the Family and Torus networks when the majority of traders are fundamentalists. This is expected since fundamental traders are not heavily influenced by other traders' perceived prices therefore the network structure does not have an impact.

When the trader demographics are flipped, 75% momentum traders, the results are very different.

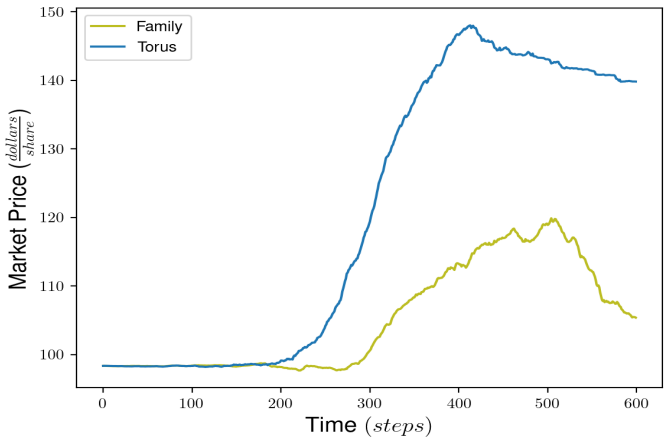


Figure 13: The market price on the Torus and Family networks with a majority of momentum traders.

Figure 13 is a Monte Carlo simulation and shows that the network structure *does* have an impact on the market price when there is a momentum trader majority. The two main differences between the Torus and Family network simulations are the magnitude of the price change and the index step when the change happens. The maximum market price for the Torus network is \$156 per share and the Family network is \$122 per share. The main structural difference between the two networks are their clustering coefficients. The Family network is highly clustered while the Torus network has no clustering. The clustering seems to reduce the magnitude of the price change. As mentioned previously, the clustering causes the perceived price within a cluster to stay within that cluster. This means once a cluster has a perceived price it won't often hear of a perceived price from outside of the cluster. The clusters that heard the perceived price at the beginning of the simulation won't receive newer more relevant perceived prices from outside their cluster. This causes the clusters that heard of the stock in the beginning of the simulation to remain at lower levels. If all of the traders within a cluster have low perceived prices then, they will place small market orders which causes small changes in price.

According to figure 13, the Herd Threshold is crossed at approximately $t = 270$. The inflection point of equation 16 on a Family network is at $t = 286$. At $t = 270$ the perceived price has spread to approximately 28 traders which is 44% of the network. The Family network simulation provides support that a Herd Threshold is exists across network structures.

7 Inferring Market Conditions

This section focuses on identifying the trading conditions that generate the market. Prior to this section the problem has been structured as a forward problem, i.e., generating market data by establishing assumptions about traders and market rules. When a trader is in a real trading environment they only see the market price data without seeing the demographics or network structure. The trader tries to identify what is causing the market behavior. The problem is now posed as, "Given this market data, what was the network and trader distribution?".

7.1 The Convolutional Neural Network

A convolutional neural network (CNN) is used to classify network structures and trader distributions. If the trader is given market data, then they should be able to classify the data by trader distribution and network structure. Market data is time series data and convolutional neural networks perform well on time series classification. CNNs extract features from the data and pool them together to get a better understanding of the uniqueness of each class. The methods proposed by [14] outperform other current leading time series classification models, such as k-nearest neighbors or support vector machines. A convolutional neural network consists of convolutional layers, pooling layers, a final feature layer, and an output layer [14]. In the convolution layer a filter, or kernel, is applied to the data that creates a feature map. The feature map represents the data in a different form that may be more useful for classification. The pooling layer divides the feature map into equal segments and then uses the segment's maximum or average value to represent that segment [14]. After the data passes through the convolution and pooling layers it reaches the final feature layer that represents the original time series data. The feature layer is then fed to the fully connected output layer that classifies the data [14].

The best time for a trader to enter the market is before the Herd Threshold. Found in section 5, this threshold is when approximately 50% of the network has heard of the stock. This means trader should identify the network demographics and structure before the perceived price spreads to 40% of the Torus network or 44% of the Family network. The Torus network passes its threshold before the Family network does. If the trader does not know the structure then, they must assume it is a Torus since the Torus is the limiting factor. This means the trader should identify the demographics and structure before $t = 200$. The data used for classification should be easily accessible for all traders which is why the CNN is trained on volume data. Volume data can be downloaded for free from various API's on the internet. The data set is structured as such, $\mathcal{D} = \{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), ..., (\mathbf{x_T}, \mathbf{y_T})\}$ where $\mathbf{x}_t \in \mathbb{R}^{Tx1}$ is the input time series and $\mathbf{y}_t \in \mathbb{R}^n$ is the respective classes, for $1 \leq t \leq T$ with n being the number of classes.

The convolution layers perform convolutions on the time series of the previous layer with a filter usually a

series of trained weights [14]. The activation function used in each convolution layer is the Rectified Linear Unit (ReLU) function. This function is used at each convolution layer but not at the output layer

$$f(x) = max(0, x), \tag{18}$$

where x is the time series. The ReLU function takes only the positive values of the inputs. The activation function for the output layer is the Softmax function since this is a classification problem. This function calculates the probability that time series belongs to one of the desired classes

$$\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{19}$$

where z is the time series after the data has passed through all convolution layers. The loss function used in training is the Categorical Cross-Entropy function. The softmax activation function returns the probability that the given data point belongs to each class. The Categorical Cross-Entropy minimizes the difference between the actual class, y_i, and the predicted class, $\hat{y}_i$,

$$L = -\sum_{i=1}^{n} y_i log(\hat{y}_i). \tag{20}$$

Below, figure 14, is an example of one of the time series inputs, $(\mathbf{x}_t, \mathbf{y}_t)$, that will be used in classifying both the trader majority and network structure. Figure 14 corresponds to a Torus network with a fundamental trader majority.

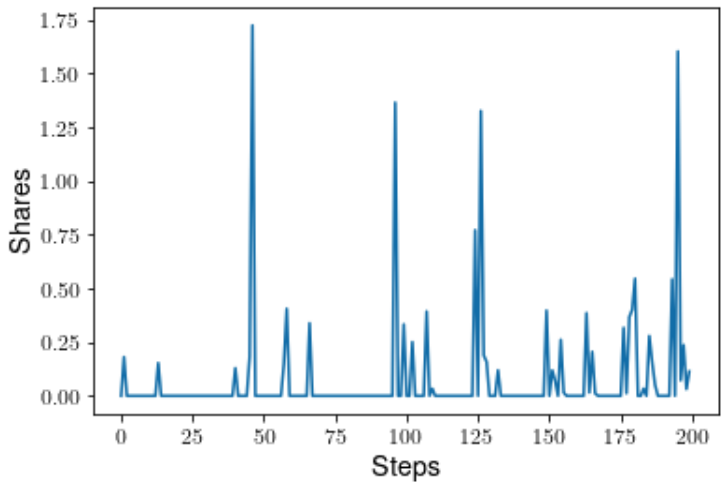


Figure 14: A example of a time series input data for the convolutional neural network.

Four market volume data sets are used for training and validation. To generate each data set 1000 simulations were run. Chebyshev's inequality is used to eliminate outliers. The mean of each time series is computed and pooled together to form one distribution. From the pooled distribution, outliers are filtered out if they are not within two standard deviations of the mean of the pooled distribution. The time series corresponding to the mean outliers is then discarded. Each data set has approximately 750 time series after eliminating outliers.

7.2 Training and Results

The classification problem is composed of two binary classification problems; the network structure and the trader demographic. The four classifications problems are; $N|D = momentum$, $N|D = fundamental$, $D|N = Torus$, and $D|N = Family$, where D is the demographics of traders, either a momentum majority or fundamental majority, and N is the network structure, either Family or Torus. The notation $N|D$ is read as, "Given the trader demographics, what is the network structure?". Four metrics are used to judge performance of the neural network which are recall, precision, false discovery rate, and accuracy. The metrics are calculated from a confusion matrix. Each classification problem has its own confusion matrix. The elements of a confusion matrix are true positive, false negative, true negative, and false positive. These are the values used in the calculations of the performance metrics.

True Positive (TP): Positive that was correctly classified as a positive.

False Negative (FN): Positive that was incorrectly classified as a negative.

True Negatives (TN): Negative that was correctly classified as a negative.

False Positives (FP): Negative that was incorrectly classified as a positive.

Below is the general structure of a confusion matrix.

	Predicted Negative	**Predicted Positive**
Actual Negative	TN	FP
Actual Positive	FN	TP

Table 5: Confusion matrix for a classification problem.

Recall is the number of correctly identified time series relative to the total number of relevant time series. In other words it is the number of true positives divided by the true positives plus false negatives.

Recall: $\frac{TP}{TP+FN}$

Precision is the number of correctly classified positives relative to the total classified positives. In other words it is the number of true positives divided by the true positives plus the false positives.

Precision: $\frac{TP}{TP+FP}$

False discovery rate, FDR, is the number of incorrectly classified positives relative to the total classified positives. It is the false positives divided by the true positives plus the false positives. It can also be calculated as, $1 - Precision$.

FDR: $\frac{FP}{FP+TP}$

Accuracy is the number of correctly classified positives and negatives relative to the total number of positives and negatives. The true positives plus true negatives divided by all of the positives plus all of the negatives, i.e., the total test set.

Accuracy: $\frac{TP+TN}{TP+FN+TN+FP}$

The Python library Tensorflow was used to build the convolutional neural networks, [10]. For all classification problems the neural networks had the same structure. This structure consistently produced the best results for each classification problem. Other parameter values and layer combinations were explored but this was consistently the best. The neural networks had three convolution layers each with 64 convolution filters, a kernel size of two, and a pooling size of two. The first layer had a dropout rate of 0.5 in order to prevent overfitting and all layers used max pooling. The network is trained with early stopping and one epoch of patience to prevent overfitting. The optimizer used in training is Adam [9] which is a stochastic gradient-based optimization.

D|N=Family

Given a Family network identify the trader demographics, $D|N = Family$. Below is the loss function for the validation and training set plotted for each training epoch. The plots of all loss functions for the training data and validation data only plot the loss values at the end of each epoch. Stochastic gradient based optimization may not follow a linear trend between epochs but the loss plots shown in this thesis only show the loss at the end of each epoch.

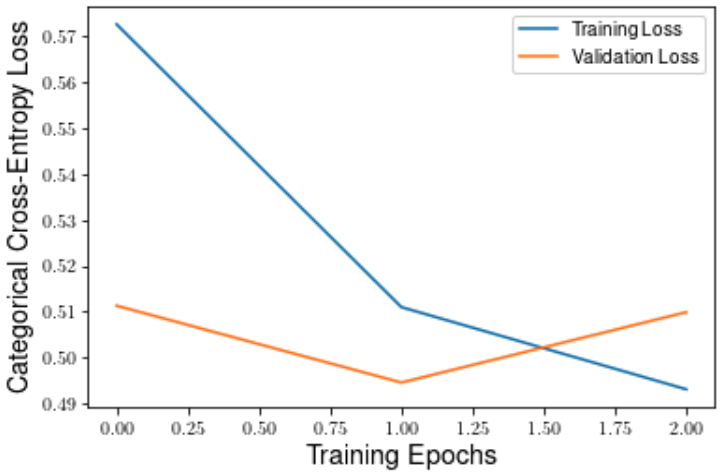


Figure 15: Cross-Entropy Loss function for a Convolutional Neural Network predicting the trader demographics given a Family network structure.

The network is able to achieve a minimum validation loss at the first training epoch, (starting at the zeroth epoch). Figure 15 shows that the neural network is not over fitting since it found a minimum in the validation loss.

A positive classification, corresponding to the definitions listed above, is the momentum majority and a negative is the fundamental majority. Table 6 shows the confusion matrix and classification statistics for classifying a momentum majority.

Total = 300	Fundamental	Momentum
Fundamental	113	48
Momentum	25	114

(a) Confusion Matrix

	Precision	Recall	FDR
Fundamental	0.82	0.70	0.18
Momentum	0.70	0.82	0.30

(b) Performance metrics

Table 6: Given a Family network identify the trader distribution.

The neural network attained an accuracy, (not shown in table 6), of 0.76. The metrics show that the network is relatively balanced in identifying the trader majority. It does a better job classifying the momentum majority than it does classifying the fundamental majority. The momentum recall metric is 0.82, which tells the trader that if the time series has a momentum trader majority, 82% of the time the neural network will correctly identify it as such. However, the recall is not the only metric to consider. A trader should know how much risk they are taking on when making the decision to trade. When classifying the trader distribution, the FDR is a good measure of prediction risk. Assuming a trader will enter the market when they identify the trader demographics as momentum majority because the Herd Threshold only exists for that demographic of traders. N.B. a trader could chose to enter the market with a fundamental trader majority and still profit however assume the trader is only interested in capitalizing on the Herd Threshold. Hence, the trader will not enter when they predict a fundamental majority. A false positive is the worst case scenario because the trader will enter the market expecting a large change in price and there will not be one. Therefore, minimizing the FDR minimizes the prediction risk of the neural network. Using this neural network the trader incurs a prediction risk of 30%.

D|N=Torus

Given a Torus network identify, the trader distribution, $D|N = Torus$. Below is the loss function for the validation and training set plotted for each training epoch.

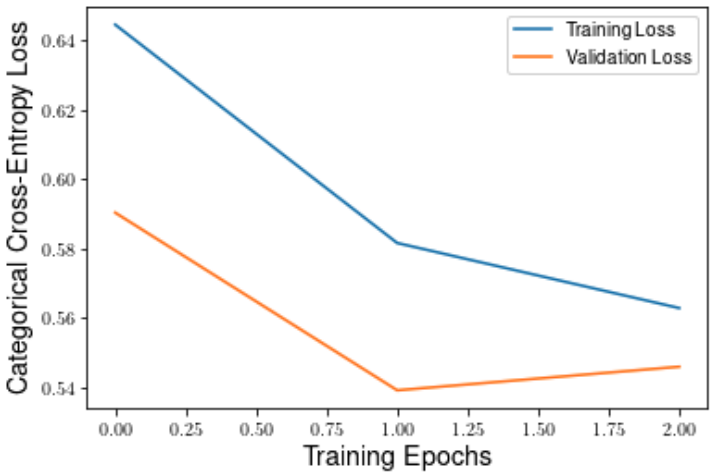


Figure 16: Cross-Entropy Loss function for a Convolutional Neural Network predicting the trader demographics given a Torus network structure.

Total = 300	Fundamental	Momentum
Fundamental	127	44
Momentum	29	100

(a) Confusion Matrix

	Precision	Recall	FDR
Fundamental	0.81	0.69	0.19
Momentum	0.69	0.78	0.31

(b) Performance metrics

Table 7: Given a Torus network identify the trader distribution.

Again, the neural network attained an accuracy of 0.76. The prediction results are very similar to those produced in the previous problem, table 6. The only difference between the training of this neural network and the previous neural network is the network structure that the traders were on while generating the volume data. This means that knowing the network is not vital to predicting the trader demographics. If there were significant differences between these results and the previous results it would be caused by the change in network structure.

N|D=momentum

Third, given a trader demographics of momentum majority, identify the network structure, $N|D = momentum$. Below is the loss function for the validation and training set plotted for each training epoch.

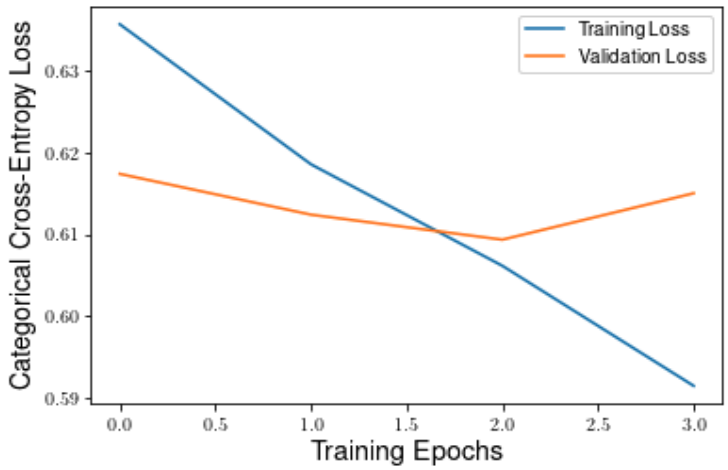


Figure 17: Cross-Entropy Loss function for a Convolutional Neural Network predicting the network structure given a momentum trader majority.

Total = 300	Family	Torus
Family	136	12
Torus	94	58

(a) Confusion Matrix

	Precision	Recall	FDR
Family	0.59	0.92	0.41
Torus	0.83	0.38	0.17

(b) Performance metrics

Table 8: Given a Momentum trader majority, identify the network structure.

The neural network attained an accuracy of 0.65. It is more difficult for the neural network to classify the network structure. This is expected since the results shown in section 5 indicate that the main differences between the Torus network and the Family network is that the Torus network has a larger and earlier change in price. The neural network can identify the Family structure well with a Family recall of 0.92. The neural network does not easily recognize the Torus structure with a Torus recall of 0.38. The neural network identifies most of the time series as Family structures. Classifying the network structure is not as important for the trader when placing trades. The trader can only lose out on the upside if the structure is incorrectly classified. The difference between the Family and Torus structures is the magnitude of the change in price therefore, if the trader identifies the structure to be a Family but is a Torus they could miss out on potential gains they would account for knowing the network structure is a Torus.

N|D=fundamental

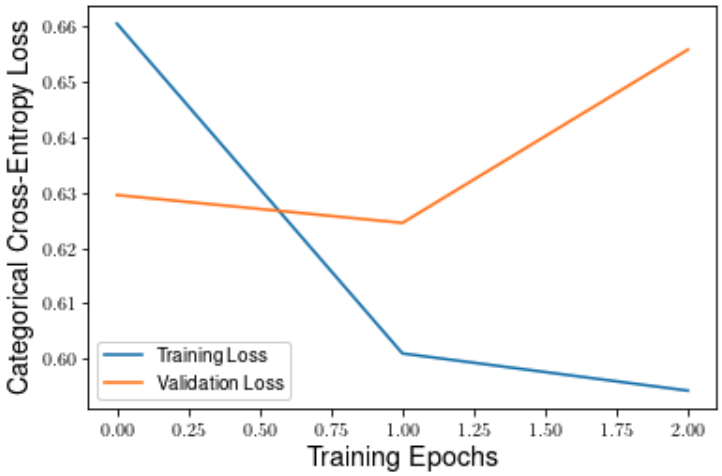


Figure 18: Cross-Entropy Loss function for a Convolutional Neural Network predicting the network structure given a fundamental trader majority.

Given a trader demographic of fundamental majority, identify the network structure, $N|D = Fundamental$. Below is the loss function for the validation and training set plotted for each training epoch.

Total = 300	Family	Torus
Family	135	10
Torus	98	57

(a) Confusion Matrix

	Precision	Recall	FDR
Family	0.58	0.93	0.42
Torus	0.85	0.37	0.15

(b) Performance metrics

Table 9: Given a Fundamental trader majority, identify the network structure.

The neural network attained an accuracy of 0.64. The results are as expected, similar to those in table 8. This means knowing the trader distribution is not vital to predicting the network structure.
Despite having relatively low classification success this was a proof of concept that the drivers of market behavior found in sections 5 and 6, i.e., trader demographics and network structure, can be classified with readily available market data. It is shown that knowing either the network structure or trader demographics is not necessary to classify the other. This section also confirms that the trader demographics are easier to classify than the network structures. This happens to be a good thing since demographics are the more important variable when making investment decisions.

8 Conclusion

Recent years have seen a revolution in both information and technology. The financial markets have been affected by the technological revolution. It has never been easier for the average person to become a market participant which has led to new challenges for investors. Social media platforms have changed the way human beings think, communicate, and invest. Retail traders have used social media platforms to manipulate stock prices, as seen in during January 2021, [13]. This thesis provides a mathematical proof of concept for how retail traders on social media can cause a large change in price and that it is possible to identify the factors that drive the price change before it is reflected in the market. An artificial market is created using an Agent Based Model that simulates traders on a network. Traders are placed on the network such that they exchange pricing information by reading each other's online postings. Two types of retail traders are used; a fundamental trader and a momentum trader. The fundamental trader relies mostly on fundamental financial analysis to make investments while the momentum trader is more influenced by what they hear on the network. It was found that when the network demographics are 75% momentum traders and 25% fundamental traders there is large change in price. There is a large change in price when approximately 40% of the Torus network has heard of the stock and 44% of the Family network has heard of the stock. These percentages are denoted as Herd Thresholds and are unique to this thesis. The Herd Threshold is the percentage of the network who have heard of the stock when there is a large change in price. This threshold is the mathematical explanation for the phenomena seen with stocks on social media such as GameStop, previously mentioned in the introduction. Two different network structures are used; a Torus network and a Family network. The Family network has a higher clustering coefficient than the Torus network. The higher clustering suppresses the magnitude of the price change. Higher clustering also reduces the rate at which pricing information spreads around the network. Once artificial market data was generated and analyzed, a convolutional neural network was used to predict the trader demographics and the network structure. The neural network was able to predict the trader demographics with an accuracy of 76% and the network structure with an accuracy of approximately 65%. The work done in this thesis can lead to a variety of future research to better understand the ever evolving financial markets.

9 Future Work

This thesis offers the opportunity for numerous areas of future work. Further research should be conducted into trading strategies of retail traders on social media. The trading strategies of the fundamental and momentum traders can be expanded. This thesis used a fundamental trader and a momentum trader, as they are grounded in the literature, but there are various other types of traders that can be explored. The impact of trader demographics can be developed further, (section 5 of this thesis). Increase the variability of trader demographics and larger networks are recommended. Note that employing an increased variability of trader demographics and larger networks requires greater computational power which was a limiting factor of this thesis. Characterize a more specific definition for the change in price corresponding to the Herd Threshold. It is recommended that once the stock's volatility surpasses a baseline volatility, over a defined time period, then there is considered a significant change in price. The Herd Threshold can then be defined as the percentage of the network that has heard of the stock once the stock's volatility has surpassed a baseline volatility. Higher network clustering suppresses the magnitude of the price change. Other network attributes and structures can be explored. This thesis showed that network structure and trader demographics are able to be predicted based off univariate time series volume data. More data, such as volatility data, and machine learning techniques should be explored to optimize predictive ability. This thesis is a proof of concept upon which to build.

10 Glossary

1) **Retail Trader:** An individual investor not associated with an investing institution and using their own money to invest.

2) **Financial Security:** A negotiable and tradable financial contract that holds value. For example, a stock, bond, commodity, etc..

3) **Market Maker:** An entity that provides liquidity in the market. Market makers are there to make up for the imbalance between buy orders and sell orders. The market maker will buy or sell excess market orders to ensure stability.

4) **Liquidity:** The ability to buy and sell. How easy it is to convert a financial security to cash. If there exists a buyer for every seller then, there is adequate liquidity.

5) **Market Order:** A request to buy or sell a financial security for the current market price.

6) **Limit Order:** A request to buy or sell a financial security at a specified price.

7) **Trading on Margin:** When an investor uses borrowed money to trade. Often times investors will borrow money from a broker to trade.

8) **Broker:** A financial entity that offers a medium to buy or sell a financial security. The "middle man" between the investor and entity that issues the financial security.

9) **Long Position:** The trader owns a certain amount of shares. The trader purchased shares from the seller.

10) **Short Position:** The trader does not currently own the stock but sold shares. The trader sold shares to a buyer that the seller does not currently own. Once the share price changes, (hopefully drops), the seller buys the shares from a third party, (hopefully cheaper than they originally sold them to the buyer for), and gives the shares to the buyer.

11) **Herd Threshold**: Given momentum traders on a network, if a certain percentage of the network has heard of the stock and is trading the stock, then there will be a large change in price.

11 Appendix

11.1 Model Summary

Market Price:

$$S_{t+1} = S_t + \frac{\Theta_t}{\lambda}$$

Market Orders:

$$\Theta_t = \sum_{i=1}^{n} \theta_{i,t}, \qquad \theta_{i,t} = \pm q_{i,t} S_t$$

Quantity:

$$q_{i,t} = \alpha_{i,t} \frac{w_{i,t-1}}{S_t}, \qquad \alpha_{i,t} = \frac{|p_{i,t} - S_t|}{S_t}$$

Agent Wealth:

$$\omega_{i,t} = c_{i,t} + |x_{i,t}|$$

Perceived Price:

$$p_{i,t+1} = \begin{cases} 0 & (p_{i,t} = 0 \wedge p_{j,t} = 0) \\ p_{j,t} & (p_{i,t} = 0 \wedge p_{j,t} \neq 0) \\ E_{i,t+1} & (p_{i,t} \neq 0 \wedge p_{j,t} = 0) \\ \nu_i \left(\gamma_{i,t} p_{i,t} + (1 - \gamma_{i,t}) p_{j,t} + (1 - \nu_i) E_{i,t+1}\right) & (p_{i,t} \neq 0 \wedge p_{j,t} \neq 0) \end{cases}$$

Confidence Probability:

$$\gamma_{i,t} = \begin{cases} \frac{exp(\beta_0 + \beta_1 \pi)}{1 + exp(\beta_0 + \beta_1 \pi)} & g_i = g_j \\ 0.75 & g_i \neq g_j \end{cases}$$

Relative Performance:

$$\pi = \frac{\omega_{i,t} - \omega_{j,t}}{\omega_{j,t}}$$

11.2 Code

```
import numpy as np
import pandas as pd
from mesa import Agent, Model
from mesa.time import RandomActivation
from mesa.space import NetworkGrid
from mesa.datacollection import DataCollector


def SMA(lis, price):

    lis.append(price) # add new price to the list
    lis.remove(lis[0]) # drop the first element since it is now out of the 20 day window
    moving_avg = np.mean(lis) # calculate the 20 day MA
    return moving_avg

def num_momentum(model):

    momentum = sum([1 for t in model.grid.get_all_cell_contents() if t.type == 'momentum'])
    return momentum

def num_fundamental(model):

    fundamental = sum([1 for t in model.grid.get_all_cell_contents() if t.type == 'fundame
    return fundamental

def momentum_perceived(model):

    p_list = [i.percieved for i in model.schedule.agents if i.type == 'momentum']

    while 0 in p_list:
        p_list.remove(0)

    return np.mean(p_list)
```

```
def fundamental_perceived(model):

    p_list = [i.percieved for i in model.schedule.agents if i.type == 'fundamental']

    while 0 in p_list:
        p_list.remove(0)

    return np.mean(p_list)

def momentum_wealth(model):

    profit = np.mean([i.wealth for i in model.grid.get_all_cell_contents() if i.type == 'm
    return profit

def fundamental_wealth(model):

    profit = np.mean([i.wealth for i in model.schedule.agents if i.type == 'fundamental'])
    return profit

def fundamental_trades(model):

    t = sum([i.trade_indicator for i in model.schedule.agents if i.type == 'fundamental'])
    return t

def momentum_trades(model):

    k = sum([t.trade_indicator for t in model.schedule.agents if t.type == 'momentum'])
    return k

def volume(model):
    return sum([abs(a.volume) for a in model.schedule.agents])

def excessDemand(model):

    d = sum([d.demand for d in model.schedule.agents])
```

```
    return d

def fundamental_shares(model):

    s_list = [s.shares for s in model.schedule.agents if s.type == 'fundamental']

    while 0 in s_list:
        s_list.remove(0)

    return np.mean(s_list)

def momentum_shares(model):

    s_list = [s.shares for s in model.schedule.agents if s.type == 'momentum']

    while 0 in s_list:
        s_list.remove(0)

    return np.mean(s_list)

def value_data(model):

    return np.mean([v.fund_value for v in model.grid.get_all_cell_contents() if v.type ==

def average_order_size(model):

    order_list = [i.order for i in model.schedule.agents]

    while 0 in order_list:
        order_list.remove(0)

    return np.mean(order_list)

def perceived_spread(model):

    agents = [a.percieved for a in model.schedule.agents]
```

```
    return all(agents)

def heard(model):
    agents = [a.percieved for a in model.schedule.agents]
    return len(model.G.nodes()) - agents.count(0)

def stockprice(model):
    return model.mktPrice.price


class market(Agent):

    def __init__(self, unique_id, model, num_shares):
        super().__init__(unique_id, model)

        path = '/Users/benweir/Documents/AgentBasedModel/Data/OrderedAgents/'
        lis = pd.read_excel(path + "StockInitializationData.xlsx")
        lis = np.log(lis.iloc[:,1])
        self.lis = list(lis)
        self.price = self.lis[-1]
        self.order = 0
        self.excessDemand = 0
        self.liquidity = num_shares
        self.sma = 0
        self.mu = 0

    def demand(self):

        self.excessDemand += self.order
        self.order = 0

    def update(self):

        self.price = np.log(self.price) + self.excessDemand/self.liquidity

        self.excessDemand = 0
```

```
        self.lis.append(self.price) # add new price to the list
        self.lis.remove(self.lis[0]) # drop the first element since it
                                     # is now out of the 20 day window
        self.sma = np.mean(self.lis)



class NetworkModel(Model): # Make sure to check length of family_node_list length
    def __init__(self,
                 noise,
                 network,
                 liquidity,
                 start_size = 1):

        self.noise = noise
        self.liquidity = liquidity
        self.G = network
        self.grid = NetworkGrid(self.G)
        self.schedule = RandomActivation(self)
        self.start_size = start_size
        self.running = True



        self.mktPrice = market(len(self.G.nodes())+1, self, self.liquidity)

        ############ FAMILY AGENT PLACEMENT CHUNK

        # family_node_list = [0]*int(len(self.G.nodes())/4) \
        # # number of families minus 1
        # family_node_list[0] = 1
        # first_family = 1

        # for k in range(1,len(family_node_list)):

        #     first_family += 4
        #     family_node_list[k] = first_family
```

```
# for i in range(len(family_node_list)):

#     if i < noise: # number of noise trader families

#         #random_node = int(np.random.choice(family_node_list,1))
#         #family_node_list.remove(random_node)
#         node = family_node_list[i]

#         for j in range(4): # create family of agents

#             a = momentumAgent(unique_id = node,
#                               model = self,
#                               mkt = self.mktPrice,
#                               enter = np.random.uniform(0.025, 0.05),
#                               exit = np.random.uniform(-0.01, 0.024),
#                               trade_proportion=trade_proportion)

#             self.schedule.add(a)
#             self.grid.place_agent(a,node)
#             #random_node += 1
#             node += 1

#     else:

#         #random_node = int(np.random.choice(family_node_list,1))
#         #family_node_list.remove(random_node)
#         node = family_node_list[i]

#         for j in range(4):

#             a = fundamentalAgent(unique_id = node,
#                                  model = self,
#                                  mkt = self.mktPrice,
#                                  enter = np.random.uniform(0.025, 0.05),
#                                  exit = np.random.uniform(-0.01, 0.024),
```

```
        #                                       trade_proportion=self.trade_proportion)

        #                 self.schedule.add(a)
        #                 self.grid.place_agent(a, node)
        #                 #random_node += 1
        #                 node += 1


        #########


        ############ TORUS AGENT PLACEMENT CHUNK

        node_list = list(self.G.nodes())

        for i, node in enumerate(self.G.nodes()):

            if node <= self.noise:

                a = momentumAgent(unique_id = i,
                                  model = self,
                                  mkt = self.mktPrice)

                self.schedule.add(a)
                #random_node = int(np.random.choice(node_list,1))
                self.grid.place_agent(a,node)

            else:

                a = fundamentalAgent(unique_id = i,
                                  model = self,
                                  mkt = self.mktPrice,
)
                #c = np.random.choice((a,b))

                self.schedule.add(a)
```

```
            #random_node = int(np.random.choice(node_list,1))
            self.grid.place_agent(a,node)

        # want to remove random_node that has been selected from node_list
        # node_list.remove(random_node)

############

begin = self.random.sample(self.G.nodes(), self.start_size)

for a in self.grid.get_cell_list_contents(begin):
    a.percieved = self.mktPrice.price + np.random.normal(0,1) # lis[-1]

self.data_collector = DataCollector(

    agent_reporters = {
        "Perceived Price": lambda x: x.percieved,
        "Wealth": lambda w: w.wealth,
        "Num_Conv": lambda c: c.conversations,
        "Conversation": lambda p: p.pair,
        "Order": lambda o: o.position
        },

    model_reporters= {
        "Market Price": stockprice,
        "Momentum Perceived": momentum_perceived,
        "Fundamental Perceived": fundamental_perceived,
        "Momentum Wealth": momentum_wealth,
        "Fundamental Wealth": fundamental_wealth,
        "Momentum Shares": momentum_shares,
        "Fundamental Shares": fundamental_shares,
        "Value": value_data,
        "Volume": volume,
        "Excess Demand": excessDemand,
        "20 Day SMA": lambda s: s.mktPrice.sma,
        "Perceived Spread": perceived_spread,
```

```
            "Nonzero Perceived": heard
            }
        )

        self.distribution = DataCollector(
            agent_reporters = {
                "Type": lambda t: t.type
            },
            model_reporters={
                "Momentum Traders": num_momentum,
                "Fundamental Traders": num_fundamental}
        )


        self.first_step = True

    def step(self):

        if(self.first_step == True):
            self.distribution.collect(self)

        if self.first_step == False:
            self.schedule.step()
            self.mktPrice.update()

        self.first_step = False

        self.data_collector.collect(self)

        agents = [a for a in self.schedule.agents]

        for i in range(len(agents)):
            agents[i].called = False
            #agents[i].pair = None
```

```
class momentumAgent(Agent):

    def __init__(self, unique_id, model, mkt):
        super().__init__(unique_id, model)

        self.type = str('momentum')
        self.wealth = 1000
        self.initial_wealth = self.wealth
        self.mktPrice = mkt
        self.cash = self.wealth
        self.percieved = 0
        self.conversations = 0
        self.talked = False
        self.shares = 0
        self.state = "EMPTY"
        self.pct_change = 0
        self.position = 0
        self.assets = 0
        self.first_trade = True
        self.num_trades = 0
        self.trade_indicator = 0
        self.demand = 0
        self.quantity = 0
        self.g = 0
        self.pair = None
        self.called = False
        self.volume = 0

    def talk(self):

        neighbors_node = self.model.grid.get_neighbors(self.pos, include_center = False)
        neighbors = [agent for agent in self.model.grid.get_cell_list_contents(neighbors_no
        all_agents = [agent for agent in self.model.schedule.agents]
        other = np.random.choice(all_agents)
```

```
if other in neighbors: # other.called == False is for 2 way conversations.
                       #It ensures that each trader can only
                       # have one convo per timestep

    self.talked = True
    self.conversations += 1
    other.conversations += 1
    self.pair = {"Self ID": self.unique_id,
                 "Other ID": other.unique_id}
    other.pair = {"Self ID": other.unique_id,
                  "Other ID": self.unique_id}

    if other.type == self.type:

        diff_wealth_s = (self.wealth - other.wealth) / other.wealth
       # diff_wealth_o = (other.wealth - self.wealth) / self.wealth
        gamma_s = np.exp(diff_wealth_s)/(1+np.exp(diff_wealth_s))
        #gamma_o = np.exp(diff_wealth_o) / (1+np.exp(diff_wealth_o))
        #gamma = 0.75
        if self.percieved == 0 and other.percieved != 0:
            self.percieved = other.percieved

        elif self.percieved != 0 and other.percieved != 0:
            self.percieved = (((1 - gamma_s) * other.percieved + gamma_s * self.pe
            #other.percieved = (((1 - gamma_o) * self.percieved \
            # + gamma_o * other.percieved) \
            # + other.mktPrice.sma)/2

        elif self.percieved != 0 and other.percieved == 0:
            #other.percieved = self.percieved
            self.percieved = self.mktPrice.sma

        elif self.percieved == 0 and other.percieved == 0:
            self.conversations -= 1 # This means they talked but neither
                                    #has heard of the stock so they didn't
                                # talk about the stock in their conversation
```

```
        elif other.type != self.type:

            gamma = 0.75

            if self.percieved == 0 and other.percieved != 0:
                    self.percieved = other.percieved

            elif self.percieved != 0 and other.percieved != 0:
                    self.percieved = (((1 - gamma) * other.percieved + gamma * self.pe
                    #other.percieved = (((1-gamma) * self.percieved + \
                    # gamma * other.percieved) + other.fund_value)/2

            elif self.percieved != 0 and other.percieved == 0:
                    #other.percieved = self.percieved
                    self.percieved = self.mktPrice.sma

            elif self.percieved == 0 and other.percieved == 0:
                    self.conversations -= 1



def trade(self):


    self.trade_indicator = 0

    self.pct_change = abs(self.percieved - self.mktPrice.price) / self.mktPrice.price
    self.g = self.pct_change

    if self.state == "EMPTY":
        # LONG BUY
        if self.percieved < self.mktPrice.price:

            self.quantity = (self.g * self.wealth / self.mktPrice.price)
```

```
            # checks if agent can afford to buy stock
            if ((self.quantity * self.mktPrice.price) < self.cash):
                # how much agent buys stock for
                self.mktPrice.order = (self.quantity * self.mktPrice.price)
                self.shares = self.quantity
                self.position = self.mktPrice.order

                self.first_trade = False
                self.state = "LONG"

        # SHORT SELL
        elif self.percieved > self.mktPrice.price:

            self.quantity = self.g * self.wealth / self.mktPrice.price
            # can only short if they have enough cash to "front" the trade
            if (abs(self.quantity) * self.mktPrice.price) < self.cash:

                self.mktPrice.order = -1*(self.quantity * self.mktPrice.price)
                self.shares = self.quantity
                self.position = self.mktPrice.order
                self.first_trade = False
                self.state = "SHORT"

    if self.state == "LONG": # Has long position and checks if they should close out

     # closes out long position
        if self.percieved > self.mktPrice.price:
            # self.position will be positive so needs to sell shares
            self.mktPrice.order = -1 * self.position
            self.shares = 0
            self.position = 0

            self.state = "EMPTY"

        # Will buy again, may want to take this out
        #elif self.percieved < self.mktPrice.price:
```

```
        #     # Still at long position

        #     # self.quantity = round(self.pct_change * \
        #       self.wealth) / self.mktPrice.price
        #     self.quantity = self.g * self.wealth / self.mktPrice.price
        #     self.mktPrice.order = self.quantity * self.mktPrice.price \
        #                                         - self.position
        #     self.shares += self.quantity
        #     self.position = self.mktPrice.order


    elif self.state == "SHORT":

        # Closes short position, "buys back shares"
        if self.percieved < self.mktPrice.price:
            # self.position should be negative so needs to buy back shares
            self.mktPrice.order = -1 * self.position
            self.shares = 0
            self.position = 0

            self.state = "EMPTY"

        # Will short again, may also want to take this out
        # elif self.percieved > self.mktPrice.price:

        #     # Still at short position

        #     # self.quantity = round(self.pct_change \
        #       * self.wealth) / self.mktPrice.price
        #     self.quantity = -self.g * self.wealth / self.mktPrice.price
        #     self.mktPrice.order = self.quantity * \
        #      self.mktPrice.price + self.position
        #     self.shares += self.quantity
        #     self.position = self.mktPrice.order
```

```
def portfolio(self):

    if self.mktPrice.order != 0:

        self.cash -= self.mktPrice.order
        self.assets += self.mktPrice.order
        self.wealth = self.cash + self.assets

    else:

        self.assets = self.shares * self.mktPrice.price
        self.wealth = self.cash + self.assets


def step(self):

    self.called = True

    if self.wealth > 1:
     # Call this first to update portfolio from trades made at last step
        self.portfolio()

        r = np.random.uniform(0,1)

        if r > .5: # 50% of the traders are chosen to trade and talk
            self.talk()

            if self.percieved > 0:
                self.trade()

        self.quantity = 0

        # if self.mktPrice.order != 0:
        #     self.num_trades += 1
        #     self.trade_indicator = 1
```

```
            self.volume = self.mktPrice.order/self.mktPrice.price
            self.demand = self.mktPrice.order

            self.mktPrice.demand()


class fundamentalAgent(Agent):


    def __init__(self, unique_id, model, mkt):
        super().__init__(unique_id, model)

        self.type = str('fundamental')
        self.wealth = 1000
        self.initial_wealth = self.wealth
        self.mktPrice = mkt
        self.conversations = 0
        self.position = 0
        self.cash = self.wealth
        self.assets = 0
        self.shares = 0
        self.state = "EMPTY"
        self.pct_change = 0
        self.first_trade = True
        self.num_trades = 0
        self.trade_indicator = 0
        self.demand = 0
        self.fund_value = self.mktPrice.price + np.random.uniform(-2,2)
        self.percieved = 0
        self.quantity = 0
        self.g = 0
        self.pair = None
        self.called = False
        self.volume = 0
```

```
def value(self):

    self.fund_value += np.random.normal(0,.1)

def talk(self):

    nu = 0.25

    neighbors_node = self.model.grid.get_neighbors(self.pos, include_center = False)
    neighbors = [agent for agent in self.model.grid.get_cell_list_contents(neighbors_no
    all_agents = [agent for agent in self.model.schedule.agents]
    other = np.random.choice(all_agents)

    if other in neighbors: # other.called == False is for 2 way conversations.
                #It ensures that each trader can only have one convo per timestep
        self.talked = True
        self.conversations += 1
        other.conversations += 1
        self.pair = {"Self ID: ", self.unique_id,
                     "Other ID: ", other.unique_id}
        other.pair = {"Self ID": other.unique_id,
                     "Other ID": self.unique_id}

        if other.type == self.type:

            diff_wealth_s = (self.wealth - other.wealth) / other.wealth
            #diff_wealth_o = (other.wealth - self.wealth) / self.wealth
            gamma_s = np.exp(diff_wealth_s)/(1+np.exp(diff_wealth_s))
            #gamma_o = np.exp(diff_wealth_o) / (1+np.exp(diff_wealth_o))

            if self.percieved == 0 and other.percieved != 0:
                self.percieved = other.percieved

            elif self.percieved != 0 and other.percieved != 0:
                self.percieved =  nu*((1 - gamma_s) * other.percieved + gamma_s * self
                #other.percieved = 0.5 * (((1 - gamma_o) * self.percieved \
```

```
                # + gamma_o * other.percieved) + other.fund_value)

            elif self.percieved != 0 and other.percieved == 0:
                #other.percieved = self.percieved
                self.percieved = self.fund_value

            elif self.percieved == 0 and other.percieved == 0:
                self.conversations -= 1 # This means they talked but neither \
                                        #has heard of the stock so they didn't
                                    # talk about the stock in their conversation

        elif other.type != self.type:

            gamma = 0.75

            if self.percieved == 0 and other.percieved != 0:
                    self.percieved = other.percieved

            elif self.percieved != 0 and other.percieved != 0:
                    self.percieved = nu*((1 - gamma) * other.percieved + gamma * self.
                    #other.percieved = 0.5 * (((1-gamma) * self.percieved \
                    # + gamma * other.percieved) + other.mktPrice.sma)

            elif self.percieved != 0 and other.percieved == 0:
                    #other.percieved = self.percieved
                    self.percieved = self.fund_value

            elif self.percieved == 0 and other.percieved == 0:
                    self.conversations -= 1

def trade(self):


    self.trade_indicator = 0

    self.pct_change = abs(self.percieved - self.mktPrice.price) / self.mktPrice.price
```

```
self.g = self.pct_change

if self.state == "EMPTY":
    # LONG BUY
    if self.percieved > self.mktPrice.price:

        self.quantity = (self.g * self.wealth / self.mktPrice.price)
        # checks if agent can afford to buy stock
        if ((self.quantity * self.mktPrice.price) < self.cash):
            # how much agent buys stock for
            self.mktPrice.order = (self.quantity * self.mktPrice.price)
            self.shares = self.quantity
            self.position = self.mktPrice.order

            self.first_trade = False
            self.state = "LONG"

    # SHORT SELL
    elif self.percieved < self.mktPrice.price:

        self.quantity = self.g * self.wealth / self.mktPrice.price

        # can only short if they have enough cash to "front" the trade

        if (abs(self.quantity) * self.mktPrice.price) < self.cash:
            # will be a negative number
            self.mktPrice.order = -1*(self.quantity * self.mktPrice.price)
            self.shares = self.quantity
            self.position = self.mktPrice.order
            self.first_trade = False
            self.state = "SHORT"

if self.state == "LONG": # Has long position and checks if they should close out

    # closes out long position
    if self.percieved < self.mktPrice.price:
```

```
            # self.position will be positive so needs to sell shares
            self.mktPrice.order = -1 * self.position
            self.shares = 0
            self.position = 0

            self.state = "EMPTY"

        # Will buy again, may want to take this out
        # elif self.percieved > self.mktPrice.price:

        #     # Still at long position

        #     # self.quantity = round(self.pct_change * \
        # self.wealth) / self.mktPrice.price
        #     self.quantity = self.g * self.wealth / self.mktPrice.price
        #     self.mktPrice.order = self.quantity * \
        # self.mktPrice.price - self.position
        #     self.shares += self.quantity
        #     self.position = self.mktPrice.order


    elif self.state == "SHORT":

        #self.pct_change = (self.percieved - \
        # self.mktPrice.price) / self.mktPrice.price

        # Closes short position, "buys back shares"
        if self.percieved > self.mktPrice.price:
            # self.position should be negative so needs to buy back shares
            self.mktPrice.order = -1 * self.position
            self.shares = 0
            self.position = 0

            self.state = "EMPTY"

        # elif self.pct_change < -1 * self.enter:  # Will short again, \
```

```
            #                           may also want to take this out
            # elif self.percieved < self.mktPrice.price:

            #     # Still at short position

            #     # self.quantity = round(self.pct_change * \
            # self.wealth) / self.mktPrice.price
            #     self.quantity = -self.g * self.wealth / self.mktPrice.price
            #     self.mktPrice.order = self.quantity * \
            # self.mktPrice.price + self.position
            #     self.shares += self.quantity
            #     self.position = self.mktPrice.order

    def portfolio(self):

        if self.mktPrice.order != 0:

            self.cash -= self.mktPrice.order
            self.assets += self.mktPrice.order
            self.wealth = self.cash + self.assets

        else:

            self.assets = self.shares * self.mktPrice.price
            self.wealth = self.cash + self.assets


    def step(self):

        self.called = True

        if self.wealth > 1:
            # Call this first to update portfolio from trades made at last step
            self.portfolio()

            r = np.random.uniform(0,1)
```

```
# 50% of the traders are chosen to trade and talk
if r > .5:

    self.value()
    self.talk()

    if self.percieved > 0:
        self.trade()

self.pair = None
#print("Q: ", self.quantity)
self.quantity = 0
#if self.conversations > 0:

# if self.mktPrice.order != 0:
#     self.num_trades += 1
#     self.trade_indicator = 1

self.volume = self.mktPrice.order/self.mktPrice.price
self.demand = self.mktPrice.order

self.mktPrice.demand()
```

References

1. F. Black, *The journal of finance* **41**, 528–543 (1986).
2. B. P. Brooks, N. DiFonzo, D. S. Ross, *Nonlinear Dynamics, Psychology, and Life Sciences* **17**, 269–293 (2013).
3. H. Chen, P. De, Y. J. Hu, B.-H. Hwang, *The Review of Financial Studies* **27**, 1367–1403 (2014).
4. J. B. De Long, A. Shleifer, L. H. Summers, R. J. Waldmann, *Journal of political Economy* **98**, 703–738 (1990).
5. J. D. Farmer, S. Joshi, *Journal of Economic Behavior & Organization* **49**, 149–171 (2002).
6. I. Giardina, J.-P. Bouchaud, *The European Physical Journal B-Condensed Matter and Complex Systems* **31**, 421–437 (2003).
7. S. Hwang, M. Salmon, *Journal of Empirical Finance* **11**, 585–616 (2004).
8. J. Kazil, D. Masad, A. Crooks, presented at the Social, Cultural, and Behavioral Modeling, ed. by R. Thomson, H. Bisgin, C. Dancy, A. Hyder, M. Hussain, pp. 308–317, ISBN: 978-3-030-61255-9.
9. D. P. Kingma, J. Ba, *arXiv preprint arXiv:1412.6980* (2014).
10. Martín Abadi *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015, (`https://www.tensorflow.org/`).
11. R. G. Palmer, W. B. Arthur, J. H. Holland, B. LeBaron, P. Tayler, *Physica D: Nonlinear Phenomena* **75**, 264–274 (1994).
12. Securities Act of 1933, 15 U.S.C. §§ 77a-77mm. 1934
13. Z. Umar, M. Gubareva, I. Yousaf, S. Ali, *Journal of Behavioral and Experimental Finance* **30**, 100501 (2021).
14. B. Zhao, H. Lu, S. Chen, J. Liu, D. Wu, *Journal of Systems Engineering and Electronics* **28**, 162–169 (2017).