

Forecasting Trade Direction and Size of Future Contracts Using Deep Belief Network

Anthony Lai (aslai), MK Li (lilemon), Foon Wang Pong (ppong)

Abstract

Algorithmic trading, high frequency trading (HFT) in particular, has drawn a lot of interests within the computer science community. Market making is one of the most common type of high frequency trading strategies. Market making strategies provide liquidity to financial markets, and in return, profit by capturing the bid-ask spread. The profitability of a market making strategy is dependent on its ability to adapt to demand fluctuations of the underlying asset. In this paper, we attempt to use deep belief network to forecast trade direction and size of future contracts. The ability to predict trade direction and size enables automated market maker to manage inventories more intelligently.

Introduction

Market making in a nutshell

According to Garman (1976), buyers and sellers arrive randomly according to a poisson process. [1] Assume that Alice wants to buy 100 shares of Google. There might not be someone who wants to sell 100 shares of Google right now. Market maker plays the role of an intermediary, quoting at the bid and ask price simultaneously. When a buyer or seller wants to make a trade, the market maker act as a counterparty, taking the other side of the trade. In return, the market maker profit from the bid-ask spread for providing liquidity to the market.

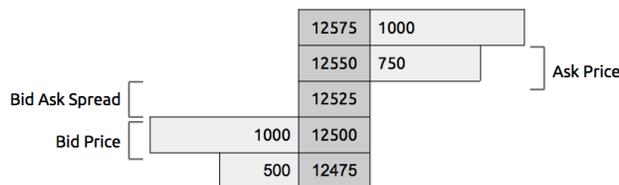


Figure 1. Illustration of Center Limit Order Book

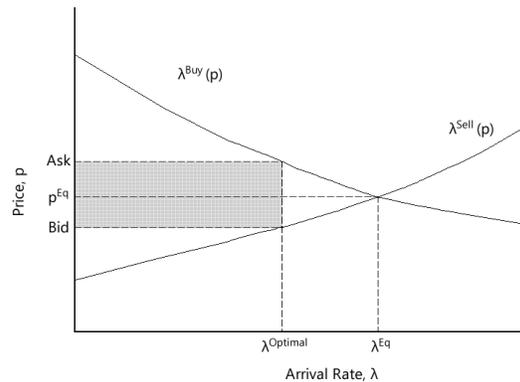


Figure 2. Bid Ask Price and Arrival Rates of Buyers and Sellers.

Automated market making strategy

An automated market maker is connected to an exchange; usually via the FIX protocol. The algorithm strategically places limit orders in the order book and constantly readjust the open orders based on market conditions and its inventory. As we have mentioned earlier, the market maker profits from the bid-ask spread. Let us formalize the average profit of a market maker over time. Let $\lambda^{\text{Buy}}(\text{Ask})$ and $\lambda^{\text{Sell}}(\text{Bid})$ be the arrival rate of the buyer and seller at the ask and bid price respectively. In the ideal world, $\lambda^{\text{Buy}}(\text{Ask})$ is equal to $\lambda^{\text{Sell}}(\text{Bid})$ and the average profit is given as:

$$\pi(\text{Bid}, \text{Ask}) = (\text{Ask} - \text{Bid})\lambda^{\text{Buy}}(\text{Ask}) = (\text{Ask} - \text{Bid})\lambda^{\text{Sell}}(\text{Bid})$$

However, in practice, this is rarely the case: buyers and sellers arrive randomly. Figure 2 illustrates how $\lambda^{\text{Buy}}(\text{Ask})$ and $\lambda^{\text{Sell}}(\text{Bid})$ is related to the bid and ask prices. In theory, assuming that there is only one market maker, he could control $\lambda^{\text{Buy}}(\text{Ask})$ and $\lambda^{\text{Sell}}(\text{Bid})$ by adjusting the bid and ask prices. In practice, market maker could also adapt by strategically adjust their open orders in the order book. The ability to accurately predict the future directions and

quantities of trades allow market maker to properly gauge the supply and demand in the marketplace and thereby to modify their open orders.

Deep Belief Network (DBN)

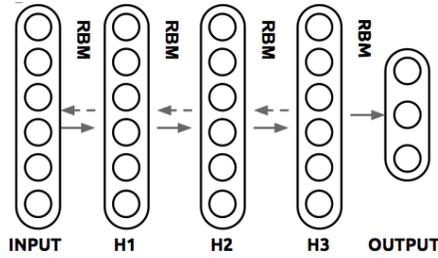


Figure 3. Structure of our deep belief network. Consists of an input layer, 3 hidden layers of RBM and an output layer of RBM.

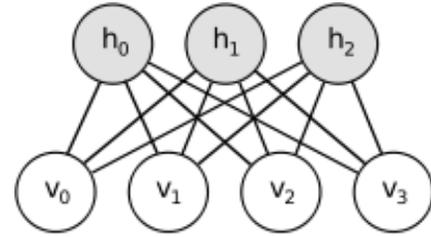


Figure 4. Diagram of a restricted Boltzmann machine.

Deep belief network is a probabilistic generative model proposed by Hinton [2]. It consists of an input layer, an output layer, and is composed of multiple layers of hidden variables stacked on top of each other. Using multiple hidden layers allows the network to learn higher order features. Hinton has developed an efficient, greedy layer-by-layer approach for learning the network weights. Each layer consists of a Restricted Boltzmann Machine (RBM). The deep belief network is trained in two steps, namely, the unsupervised pre-training and the supervised fine tuning.

Unsupervised pre-training

Traditionally, deep architectures is difficult to train because of their non-convex objective function. As a result, the parameters tend to converge at local optima. Performing unsupervised pre-training helps initialize the weights to better values and helps us find better local optima. [4] Unsupervised pre-training is performed by having each hidden layer greedily learns the identity function one at a time, using the output of the previous layer as the input.

Supervised fine tuning

Supervised fine tuning is performed after the unsupervised pre-training. The network is initialized using the weights from the pre-training. By performing unsupervised pre-training, we are able to find better local optima in the fine tuning phase than using randomly initialized weights.

Restricted Boltzmann Machine

Restricted Boltzmann Machines (RBMs) consist of visible and hidden units that form a bipartite graph (Figure 3). The visible and hidden units are conditionally independent given the other layer. The RBMs are then trained by maximizing the likelihood given by the following equation (Figure 5). RBM is trained using a method called contrastive divergence that is developed by Hinton. Further information can be found in Hinton's paper. [3]

$$\begin{aligned} \ell(\theta, \mathcal{D}) &= \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log p(x^{(i)}) \\ &= \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log \frac{\sum_h e^{-E(x^{(i)}, h)}}{\sum_{x^{(i)}} \sum_h e^{-E(x^{(i)}, h)}} \\ &= \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log \frac{\sum_h e^{a^T x^{(i)} + b^T h + h^T W x}}{\sum_{x^{(i)}} \sum_h e^{a^T x^{(i)} + b^T h + h^T W x}} \end{aligned}$$

Figure 5. Log likelihood of the restricted Boltzmann machine. In this case, $x^{(i)}$ refers to the visible units.

Dataset

To ensure the relevancy of our results, we acquired tick-by-tick data of S&P equity index Futures (ES) from the

Chicago Mercantile Exchange (CME). Tick-by-tick data includes all trade and order book events occurred at the exchange. It contains significantly more information than the typical information obtained from Google or Yahoo finance - which are summary of the tick-by-tick data. The data we obtained from CME is encoded in FIX format (Figure 5). We implemented a parser to reconstruct trade messages and order book data.

```
1128=9|9=361|35=X|49=CME|34=2279356|52=20120917185520880|75=20120917|268=3|279=1|22=8|48=10113|83=2476017|107=ES
Z2|269=0|270=145200|271=1607|273=185520000|336=2|346=240|1023=6|279=1|22=8|48=10113|83=2476018|107=ESZ2|269=0|270=
145325|271=421|273=185520000|336=2|346=95|1023=1|279=1|22=8|48=10113|83=2476019|107=ESZ2|269=0|270=145325|271=416|
273=185520000|336=2|346=94|1023=1|10=141|
```

Figure 6. A single FIX message encoding an incremental market data update message.

Normalization and data embedding

Our input vector is detailed in Figure 7. Prices and time values are first converted to relative values by subtracting the values of the trade_price and trade_time at t=0 respectively. For instance, if the trade price at t=0 and t=-9 are 12,575 and 12,550 respectively, then trade_price₀ = 0 and trade_price₉ = -25.

The neural network requires input to be in the range [0, 1]. We have therefore created an embedding by converting numerical values into binary form. For example, the final result for trade size would look as follows: Trade size=500 → round(log₂(500)) = 8 → [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]. After these data transformations, our input features becomes a 700x1 feature vector.

Event Time	t=0							...	t=-9								
Variable Name	bid_price<t>	bid_quantity<t>	ask_price<t>	ask_quantity<t>	trade_time<t>	trade_price<t>	trade_quantity<t>	trade_direction<t>	...	bid_price<t>	bid_quantity<t>	ask_price<t>	ask_quantity<t>	trade_time<t>	trade_price<t>	trade_quantity<t>	trade_direction<t>

Figure 7. Illustration of an Input vector.

Results

Baseline Approaches

For the purpose of this project, we evaluated several machine learning algorithms, in particular, logistic regression and support vector machines. Logistic regression was a natural choice because we wanted to make binary predictions about the directions of future trades. We also attempted to use SVM for both the trade direction and quantity prediction tasks. We did some field trials comparing the various learning algorithms. DBN was the most accurate out of the three, and therefore we decided to focus on DBN for the purpose of this project.

DBN

The structure of our deep belief network is shown in Figure 3. There are a total of 3 hidden layers, each with 100 nodes. This is the best configuration out of various other configurations of network topologies we explored.

Trade Direction Prediction

We want to predict the next 10 trades given the previous 10 trades. For this task, we trained 10 DBNs: the ith DBN is trained to predict the direction of the trade at t_i. The results are shown in Figure 8. For comparison, we used 2 different sets of input: one using only information from the past 10 trades and another using both trade and order book information.

Event Time	$t + 1$	$t + 2$	$t + 3$	$t + 4$	$t + 5$	$t + 6$	$t + 7$	$t + 8$	$t + 9$	$t + 10$
100-100-100 DBN	73%	72%	70%	68%	67%	65%	64%	65%	64%	63%
100-100-100 DBN + TOB	73%	72%	70%	68%	67%	65%	64%	65%	64%	63%

Figure 7. Accuracies of predicting trade direction. The predictions in the first row only uses trade information. The predictions in the second row uses both trade and top of the book information (namely the bid price, bid size, ask price, ask size).

Trade Quantity Prediction

Similar to the previous prediction task, we trained 10 DBNs to predict the sizes of the next 10 trades. Our earlier attempt uses a logistic regression in the output layer to predict a normalized value between $[0, 1]$. The prediction range is very narrow, between 0.2 and 0.3. We attribute this to the steepness of the sigmoid function. Posing this as a classification problem would perhaps be more appropriate. We therefore transformed the trade quantity into 10 discrete bins. The distribution of the trade quantity is extremely skewed as shown in Figure 8. As a preprocessing step, we log normalized the trade quantity, reducing the kurtosis from 80.52 to 2.9. Doing so created 10 discrete bins, each corresponded to a power of 2. Results are shown in Figure 10.

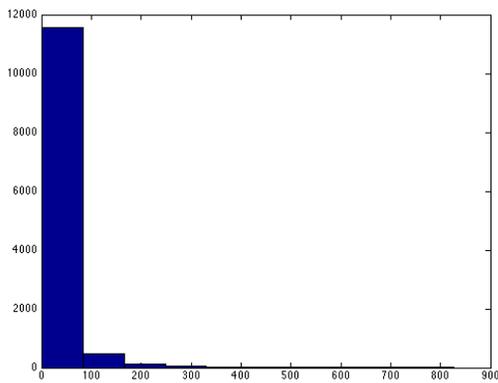


Figure 8. Histogram of trade quantities.

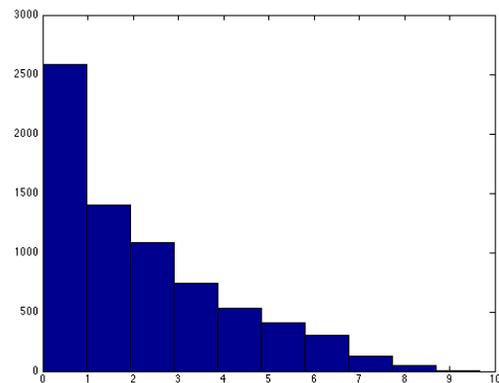


Figure 9. Histogram of log normalized trade quantities.

Event Time	$t + 1$	$t + 2$	$t + 3$	$t + 4$	$t + 5$	$t + 6$	$t + 7$	$t + 8$	$t + 9$	$t + 10$
100-100-100 DBN	2.72	2.76	2.81	2.79	2.83	2.80	2.80	2.82	2.81	2.82
100-100-100 DBN + TOB	2.77	2.80	2.82	2.82	2.82	2.83	2.81	2.83	2.83	2.83

Figure 10. RMSE of predicting trade quantities. The predictions in the first row only uses trade information. The predictions in the second row uses both trade and top of the book information (namely the bid price, bid size, ask price, ask size).

Error Analysis

Trade direction error analysis

Given the past 10 trades, our model yields an accuracy of over 60%. It appears that additional information, such as the order book data, did not improve the result. Although the unprocessed order book data does not seem very predictive, it could potentially be useful if we manually engineer features that is not currently captured by the DBN.

We are interested to understand why trade direction can be predicted. The graph of trade directions are shown

in figure 11. By inspection, we discovered that the trades tend to be of consecutive buys and sells, which explains it far from random and allow our algorithm to perform some predictions.

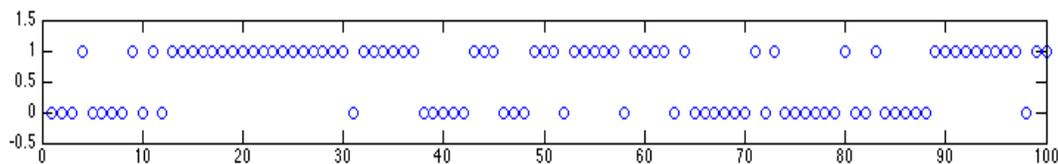


Figure 11. Time series of trade directions. Buy and sell orders are shown at the top and bottom respectively.

Trade quantity error analysis

From the result above, we can see that the prediction for the trade quantities are fairly inconsistent. It is unclear if there exists any correlations between our input data and the trade quantities. Perhaps, we may need additional information, other than the last 10 trades, to improve the prediction. We also tried to gain additional insights about the DBN by visualizing its weights, but it turned out not to be too informative either.

The RMSE of the results are on average over 2 bins different from the actual labels. When we look at the output of the DBN, it predicts the majority class most of the time - in this case it is bin 0. We attribute this to the skewness of the class labels. One way to deal with skewed class distribution is to resample from the underlying dataset to obtain a more evenly distributed set of training examples.

There are potentially multiple factors contributing to such unpredictability. First, when trading large quantities, hedge funds and institutional traders use several ways to minimize market impact. The simplest way is to divide large trades into smaller trades. More complex approaches involve further obscuring the trading patterns by introducing noise using multiple buying and selling transactions. [5] Moreover, other market makers may use similar algorithms to predict the demand and supply. As a result, any arbitrage opportunities would disappear quickly, leading to mixed signals mingled with each other.

Perhaps, the more interesting problem is the prediction of outliers -- trades with large quantities. The ability to predict these outliers is more important as these outliers would cause the market maker to over-buy or over-sell the asset, leading to increased risks and unbalance inventories.

Future Work

We would like to investigate further using other complementary tools and information and see if we can see improve our current results. One possibility is to include data from highly correlated financial product such as Nasdaq futures market(NQ) as additional features. The rationale behind this is similar to pairs trading. If a large quantity is traded in Nasdaq, it might inform us about trades related to ES (S&P) futures. Another possibility is to include other handcrafted statistical information such as moving averages as input features. There could be information not captured by the fundamental trade information that can help achieve better predictions. It may also be useful to create an automated process to experiment more comprehensively with our DBNs using different hyper parameters, such as the number of hidden layers, the dimension of different hidden layers, number of training iterations, learning rate and etc. Although we have done various tests and decide to settle on our current configurations, it would be beneficial if we could further fine tune our hyper parameters.

[1] Joel Hasbrouck. Empirical Market Microstructure: The Institutions, Economics, and Econometrics of Securities Trading.

[2] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, no. 7, pp. 1527–1554, 2006

[3] Geoffrey Hinton (2002). Training products of experts by minimizing contrastive divergence. Neural Computation 14:1771–1800.

[4] http://ufldl.stanford.edu/wiki/index.php/Deep_Networks:_Overview

[5] http://en.wikipedia.org/wiki/Algorithmic_trading